LUDWIGS-MAXIMILIANS UNIVERSITÄT
MÜNCHEN

FAKULTÄT FÜR MATHEMATIK, INFORMATIK UND STATISTIK
MATHEMATISCHES INSTITUT

Masterarbeit

Automatically verified program extraction from proofs with applications to constructive analysis

Nils Köpp

LUDWIGS-MAXIMILIANS UNIVERSITÄT
MÜNCHEN

FAKULTÄT FÜR MATHEMATIK, INFORMATIK UND STATISTIK
MATHEMATISCHES INSTITUT

Masterarbeit

Automatically verified program extraction from proofs with applications to constructive analysis

Autor: Nils Köpp
Betreuer: Prof. Dr. Helmut Schwichtenberg
Abgabedatum: 3.1.2018

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und noch nicht veröffentlicht.

München, 19.12.2017 Nils Köpp

Abstract

We utilize formal proofs in a theory allowing for (co)inductive definitions to extract computational content from proofs in constructive mathematics. We prove that these extracted terms are correct with respect to a realizability interpretation, so that we can automatically generate proofs of soundness. We then apply the above methods to some examples in constructive analysis, namely the principle of least upper bound and the representation of reals by sd-code, using the proof assistant Minlog.

Zusammenfassung

Wir benutzen formale Beweise in einer Theorie, die (co)induktive Definitionen gestattet, um den rechnerischen Gehalt aus Beweisen in der konstruktiven Mathematik zu extrahieren. Wir beweisen, dass diese extrahierten Terme bezüglich einer Realisierbarkeitsinterpretation korrekt sind, was es ermöglicht automatisch solche Korrektheitsbeweise zu erzeugen. Wir wenden diese Methoden unter Verwendung des Beweisassistenten Minlog auf einige Beispiele in der konstruktiven Analysis an: Das Prinzip der kleinsten oberen Schranke und die Darstellung von Reellen Zahlen mittels SD-Code.

Contents

1 Introduction

Our goal is to extract computational content from constructive proofs, specifically formal proofs in intuitionistic logic, and automatically generate correctness-proofs for these. Consider a specification of the form

$$\forall_x \exists_y A(x, y)\,.$$

One approach to solve this problem is to devise some algorithm f and prove its correctness, i.e. $\forall_x A(x, f(x))$, manually. In practice however, there is no method of automatically checking that some algorithm actually fulfills the specification. Formal proofs on the other hand can easily be automatically checked. Furthermore mathematicians are quite proficient in giving (complex) proofs while giving direct solutions to such problems can be quite tough.
We thus utilize formal proofs to obtain the "hidden algorithm" or *extracted terms* of proofs. Further we give an internal soundness-proof, which states that this extracted term is correct. This (constructive!) proof gives rise to an algorithm which - given a proof - generates a new proof stating the correctness of its extracted term.

2 The Formal System

The main goal is to formalize constructive mathematics in such a way, that we can easily identify the construction made in a mathematical proof. The strategy to achieve this is roughly as follows.
We regard proofs of mathematical statements as derivations in natural deduction with suitable axioms. Using the *Curry-Howard-Correspondence* we represent proofs in typed lambda calculus. By removing computationally irrelevant parts and substituting meaningful terms for axioms we define the *extracted term* of a derivation M, denoted by $et(M)$. Additionally we assign types to propositions, i.e for every formula A we define a *type* $\tau(A)$, which corresponds to the type of possible *realizers* (or "solutions") of a formula.
We then define a *realizability interpretation* z **r** A, where z is a term in the same language of type $\tau(A)$. This relation is modeled after Kreisels *modified realizability*, see [7]. Intuitively z **r** A means that z "solves the problem posed by the formula A". Finally, we give a soundness proof that claims that a proposition is realized by the extracted term of its proof, i.e. the proof extraction method is correct.
We introduce a term language which will be an extension of Gödels T (see [5]) and a theory allowing for (co)inductive definitions. The intended model for the theory will be the Scott-Ershov model for partial continuous functionals (see [4]). We will follow the formulation by Schwichtenberg and Wainer in [11] (also see [9]). We provide an outline with special emphasis on soundness.

2.1 Partial continuous functionals

This chapter will be organized as follows. In Section 2.1.1 we define (free) algebras and types. In Section 2.1.2 we study Scotts informations systems (see [14]) and concrete information systems with (free) algebras as base domains. In 2.1.3 we take a closer look at some special ideals in the concrete information systems previously defined.

2.1.1 Algebras and types

We start by defining *types*. They will be be built up from certain base types, namely (non-flat) free algebras, and the formation of function types $\rho \to \sigma$.

Definition 1 (Algebras and types)**.** Let α, ξ be type variables then we inductively define *type forms* by

$$\rho, \sigma := \alpha \mid \rho \to \sigma \mid \mu_\xi \left((\rho_{iv})_{v<n_i} \to \xi \right)_{i<k}$$

Furthermore we inductively define $Sp(\alpha, \rho)$, meaning that α occurs at most *strictly positive* in ρ.

$$Sp(\alpha, \beta) \qquad \frac{\alpha \notin TV(\rho) \qquad Sp(\alpha, \sigma)}{Sp(\alpha, \rho \to \sigma)} \qquad \frac{Sp(\alpha, \rho_{i\nu}) \qquad (\text{for all } i < k, \nu < n_i)}{Sp\left(\alpha, \mu_\xi\left((\rho_{i\nu})_{\nu<n_i} \to \xi\right)_{i<k}\right)}$$

Where $TV(\rho)$ denotes the set of type variables in ρ. We inductively define $Ty(\rho)$, meaning that ρ is a *type*.

$$Ty(\alpha) \qquad \frac{Ty(\rho) \qquad Ty(\sigma)}{Ty(\rho \to \sigma)} \qquad \frac{Ty\left(\rho_{i\nu}\right) \qquad Sp\left(\xi, \rho_{i\nu}\right) \qquad (\text{for all } i < k, \nu < n_i)}{Ty\left(\mu_\xi\left((\rho_{i\nu})_{\nu<n_i} \to \xi\right)_{i<k}\right)}$$

Then

$$\iota := \mu_\xi\left((\rho_{i\nu})_{\nu<n_i} \to \xi\right)_{i<k}$$

is called an *algebra*. We also write

$$\iota := \mu_\xi\left(\kappa_0(\xi), \ldots, \kappa_{k-1}(\xi)\right)$$

with $\kappa_i\left(\xi\right) = \left(\rho_{i\nu}\left(\xi\right)\right)_{\nu<n_i} \to \xi$ the *constructor types* of the algebra. If $\xi \notin TV\left(\rho_{i\nu}\right)$ then $\rho_{i\nu}$ is called a *parameter argument*. Otherwise $\rho_{i\nu}$ is called a *recursive argument*. In case $\rho_{i\nu}$ is a previously defined algebra with ξ in a strictly positive position, then $\rho_{i\nu}$ is called a *nested argument* and the algebra is called *nested*. We call an algebra *finitary* if all the argument types $\rho_{i\nu}$ are not of arrow form and do not contain type variables.

For each constructor type κ_i of an algebra ι we provide a constructor symbol C_i^ι of type $\kappa_i(\iota)$. For specific algebras we will have special constructor names (e.g. 0, S for the algebra of natural numbers, see below).

Example 1. (1) Some finitary algebras are

Unit	$\mathbf{U} := \mu_\xi(\xi)$	
Boole	$\mathbf{B} := \mu_\xi\left(\xi, \xi\right)$	tt, ff
Signed digits	$\mathbf{SD} := \mu_\xi\left(\xi, \xi, \xi\right)$	$-1, 0, 1$
Natural numbers	$\mathbf{N} := \mu_\xi\left(\xi, \xi \to \xi\right)$	$0, \mathsf{S}$
SD-streams	$\mathbf{Str} := \mu_\xi\left(\mathbf{SD} \to \xi \to \xi\right)$	C_d
Positive numbers	$\mathbf{P} := \mu_\xi\left(\xi, \xi \to \xi, \xi \to \xi\right)$	$1, \mathsf{S}_0, \mathsf{S}_1$
Integers	$\mathbf{Z} := \mu_\xi\left(\xi, \mathbf{P} \to \xi, \mathbf{P} \to \xi\right)$	$0, \mathsf{P}, \mathsf{N}$
Rational numbers	$\mathbf{Q} := \mu_\xi\left(\mathbf{Z} \to \mathbf{P} \to \xi\right)$	$\#$

U is a singleton. **B** is given by two constructors which are interpreted as true and false. **SD** and **Str** will be used to represent real numbers (see chapter 3). **N** is given by two constructors which are interpreted as 0 and successor. **P** represents positive binary numbers, e.g. $S_{b_0} \ldots S_{b_n} 1$ is interpreted as the (positive) binary number $1 b_n \ldots b_0$. **Z** is given by specifiying 0 and injection of positive numbers onto positive and negative

integers respectively. Rational numbers **Q** are given as pairs of integers and positive numbers.
(2) Some algebras with parameter arguments are

Lists	$\mathbf{L}(\alpha) := \mu_\xi\,(\xi, \alpha \to \xi \to \xi)$	Nil, cons
Type products	$\alpha \times \beta := \mu_\xi\,(\alpha \to \beta \to \xi)$	$\langle \cdot, \cdot \rangle$
Type sums	$\alpha + \beta := \mu_\xi\,(\alpha \to \xi, \beta \to \xi)$	InL, InR

$\mathbf{L}(\alpha)$ represents lists over some arbitrary type α. $\alpha \times \beta$ represents pairs of objects of type α and β. The sum algebra $\alpha + \beta$ is given by two constructors interpreted as left and right injection.
(3) An example of a nested algebra is

Nested trees	$\mathbf{Nt} := \mu_\xi\,(\mathbf{L}\,(\xi) \to \xi)$	Branch

Nt represent finitely branching trees. The constructor `Branch` takes a list of nested trees as argument. The result is then a new tree with the elements of the list as branches.
(4) Finally, examples for non-finitary algebras are given by T_n $(1 < n)$

Trees	$\mathbf{T}_0 := \mathbf{N},\ \mathbf{T}_{n+1} := \mu_\xi\,(\xi, (\mathbf{T}_n \to \xi) \to \xi)$

2.1.2 Information Systems

As we want to extract programs from proofs we need some conception of *higher type computability*. This notion will be specified by two principles, the *principle of finite support* and the *monotonicity principle*.
Regarding the former, consider a functional Φ of higher type. Then in any evaluation $\Phi(\phi)$ the argument can only be called upon finitely many times and in addition these calls must be presented to Φ in finite form, i.e. as "finite approximations" of ϕ.
The monotonicity principle says that if $\Phi(\phi)$ evaluates to some value and there is an "extension" ψ of ϕ, i.e. ψ carries more information than ϕ, then $\Phi(\psi)$ must evaluate to the same value.
A *finite approximation* of some functional Φ is a set of pairs (ϕ, σ) of finite functions and values such that $\Phi(\phi) = \sigma$. We then call Φ *computable* if its set of finite approximations is recursively enumerable. The above principle will be fulfilled by *consistent* and *deductively closed* sets in Scotts *information systems*.
In this section we study the latter as a way of constructing domains (see [11] for a detailed exposition). We take an arbitrary countable set A whose elements we call *tokens*. For finite subsets of A we define the notion of *consistency*, meaning that finite sets of tokens, or information, are consistent with each other. On top of that we define an entailment relation between consistent sets U and tokens a, written $U \vdash a$, which intuitively means that the information U is sufficient to compute a.

Definition 2 (information system)**.** An *information system* is a triple $(A, Con, \vdash)$, where A is a countable set, $Con \subset A$ is a set of finite subsets and $\vdash \subset Con \times A$ satisfying

$$\begin{aligned}
&U \subseteq V \in Con \to U \in Con\\
&a \in Con\\
&U \vdash a \to U \cup a \in Con\\
&a \in U \in Con \to U \vdash a\\
&U, V \in Con \to \forall_{a \in V}(U \vdash a) \to V \vdash b \to U \vdash b
\end{aligned}$$

The *ideals* of an information system are those subsets $x \subseteq A$ such that

$$\begin{aligned}
U \subseteq x \text{ finite} &\to U \in Con\\
U \vdash x &\to U \subseteq x \to a \in x
\end{aligned}$$

i.e. consistent and deductively closed subsets. We denote by $|\mathbf{A}|$ the set of ideals of an information system $\mathbf{A}$. An example of an ideal is the *deductive closure* for some $U \in Con$:

$$\overline{U} := \{a \in A \mid U \vdash a\}$$

Let $\mathbf{A} = (A, Con_A, \vdash_A)$,$\mathbf{B} = (B, Con_B, \vdash_B)$ be two information systems. We define the *function space* $\mathbf{A} \to \mathbf{B} = (C, Con, \vdash)$ by

$$\begin{aligned}
&C := Con_A \times B\\
&\{(U_i, b_i) | i \in I\} \in Con := \forall_{J \subseteq I}\left(\bigcup_{j \in J} U_j \in Con_A \to \{b_j | j \in J\} \in Con_B\right)\\
&\{(U_i, b_i) | i \in I\} \vdash (U, b) := \{b_i | U \vdash_A U_i\} \vdash_B b
\end{aligned}$$

Note that $\mathbf{A} \to \mathbf{B}$ is then itself an information system. We call the ideals of $\mathbf{A} \to \mathbf{B}$ *approximable maps*.

We can also characterize approximable maps in another way, namely as continuous functions w.r.t. the *Scott topology*.

Definition 3 (Scott topology)**.** Let $\mathbf{A} = (A, Con, \vdash)$ be an information system and $U \in Con$. We define

$$\mathcal{O}_U := \{x \in |\mathbf{A}| \mid U \subseteq x\}$$

The system $\{\mathcal{O}_U \mid U \in Con\}$ forms a topology on $|A|$. It is called the *Scott topology*.

We get the following characterization of continuous functions, recovering the principles of monotonicity and finite support.

Lemma 1. *Let* $\mathbf{A}$,$\mathbf{B}$ *be information systems and* $f : |\mathbf{A}| \to |\mathbf{B}|$*. The following are equivalent.*
(a) f *is continuous w.r.t. the Scott topology*
(b) f *is monotone, i.e.* $x \subseteq y \to f(x) \subseteq f(y)$*, and satisfies the principle of finite support, i.e. if* $b \in f(x)$ *then* $b \in f\left(\overline{U}\right)$ *for some* $U \subseteq x$*.*

We can identify approximable maps with continuous functions in the following way.

Theorem 1. *Let* $\mathbf{A}$,$\mathbf{B}$ *be information systems. Then we have a bijective correspondence*

$$\{r \in |\mathbf{A} \to \mathbf{B}|\} \quad \longleftrightarrow \quad \{f : |\mathbf{A}| \to |\mathbf{B}| \text{ continuous}\}$$

given by

$$\begin{aligned} & r \mapsto |r| \\ & |r|(z) := \{b \in B | r(U, b) \text{ for some } U \subseteq z\} \\ & f \mapsto \hat{f} \\ & \hat{f}(U, b) := (b \in f\left(\overline{U}\right)) \end{aligned}$$

Furthermore we obtain the following result.

Lemma 2 (Extensionality)**.** *Let* $\mathbf{A}$, $\mathbf{B}$ *be information systems and* $r, s \in |\mathbf{A} \to \mathbf{B}|$ *be approximable maps, then*

$$r = s \iff \forall_{z \in |\mathbf{A}|} \left(|r|(z) = |s|(z)\right)$$

For every type ρ we define an information system C_ρ.

Definition 4 (Information system of type ρ)**.** The information system $\mathbf{C}_\tau = (C_\tau, Con_\tau, \vdash_\tau)$ for type τ is defined by recursion in the following way. For simplicity we assume that base types are only finitary algebras.
(a) The tokens $a \in C_\iota$ are type-correct constructor expressions $\mathsf{C}a_1^* \dots a_n^*$ where the a_i^* are either tokens or the symbol $*$ (no information)
(b) A finite subset $U \subseteq C_\iota$ is consistent if all elements begin with the same constructor C, i.e.

$$U = \{\mathsf{C}a_{11}^* \dots a_{1n}^*, \dots, \mathsf{C}a_{j1}^* \dots a_{jn}^*\}$$

and for all $1 \leq i \leq n$ we have $U_i := \{a_{im}^* \mid 1 \leq m \leq j\} \in Con_{\tau_i}$
(c) $U \vdash_\tau *$
(d) $\{\mathsf{C}\vec{a}_1^* \dots \mathsf{C}\vec{a}_n^*\} \vdash_\iota \mathsf{C}'\vec{a}^*$ if and only if $\mathsf{C} = \mathsf{C}'$ and $U_i \vdash a_i^*$.
(e) $\mathbf{C}_{\rho\to\sigma} = \mathbf{C}_\rho \to \mathbf{C}_\sigma$ is defined according to definition 2
If $\mathbf{C}_\rho$ is the information system associated to some type ρ then we call the set of ideals $\left|\mathbf{C}_\rho\right|$ the *partial continuous functionals* of type ρ. We call $x \in \left|\mathbf{C}_\rho\right|$ *computable* if it is recursively enumerable when viewed as a set.

If C is some constructor of an algebra, then C generates an ideal in the function space $(\rho_{i\nu})_{\nu<n_i} \to \iota$, namely

$$r_\mathsf{C} := \left\{\left(\vec{U}, \mathsf{C}\vec{a}^*\right) \mid \vec{U} \vdash \vec{a}^*\right\}$$

According to theorem 1 the corresponding continuous function is given by

$$|r_\mathsf{C}|(\vec{x}) := \left\{\mathsf{C}\vec{a}^* \mid \exists_{\vec{U} \subseteq \vec{x}} \left(\vec{U} \vdash \vec{a}^*\right)\right\}$$

From the definitions we obtain the following

Lemma 3. *Let* $\mathsf{C}, \mathsf{C}_1 \neq \mathsf{C}_2$ *be constructors of some algebra* ι*, then*

$$|r_{\mathsf{C}}|(\vec{x}) \subseteq |r_{\mathsf{C}}|(\vec{y}) \;\leftrightarrow\; \vec{x} \subseteq \vec{y}$$
$$|r_{\mathsf{C}_1}|(\vec{x}) \neq |r_{\mathsf{C}_2}|(\vec{y})$$

i.e. constructors are injective and have disjoint range.

Remark 1. Note that according to the lemma above our base domains are *non-flat* information systems.

We define the *map operator*, which we later use in the conversion rules for the (co)recursion operators.

Definition 5 (Map operator)**.**

$$\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\rho(\vec{\alpha})} : \rho(\vec{\sigma}) \to (\vec{\sigma} \to \vec{\tau}) \to \rho(\vec{\tau})$$

is defined in the following way:
(a) If $\vec{\alpha}$ is not free in $\rho(\vec{\alpha})$, then

$$\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\rho(\vec{\alpha})} x\vec{f} := x$$

(b) If $\rho(\vec{\alpha}) = \iota(\vec{\alpha})$ is an algebra we write $\mathcal{M}^{\vec{\tau}}_{\iota(\vec{\sigma})}$ and define the operator by recursion on the algebra. For each constructor $\mathsf{C}^{\vec{\sigma}}_i : (\rho_\nu(\vec{\sigma}, \iota(\vec{\sigma})))_{\nu<n} \to \iota(\vec{\sigma})$ we have

$$\mathcal{M}^{\vec{\tau}}_{\iota(\vec{\sigma})}(\mathsf{C}^{\vec{\sigma}}_i \vec{x})\vec{f} := \mathsf{C}^{\vec{\tau}}_i \left(\mathcal{M}^{(\vec{\sigma},\iota(\vec{\sigma}))\to(\vec{\tau},\iota(\vec{\tau}))}_{\lambda_{\vec{\alpha},\beta}\rho_\nu(\vec{\alpha},\beta)} x_\nu \left(\vec{f}, \lambda_{\vec{x}} \mathcal{M}^{\vec{\tau}}_{\iota(\vec{\sigma})} \vec{x}\vec{f}\right)\right)_{\nu<n}$$

(c) In case $\rho(\vec{\alpha}) = \iota(\vec{\pi}(\vec{\alpha}))$ with $\vec{\alpha} \neq \vec{\pi}(\vec{\alpha})$

$$\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\iota(\vec{\pi}(\vec{\alpha}))} x\vec{f} := \mathcal{M}^{\vec{\pi}(\vec{\sigma})\to\vec{\pi}(\vec{\tau})}_{\iota} x \left(\lambda_{x_i} \mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\pi_i(\vec{\alpha})} x_i \vec{f}\right)_{i<|\vec{\pi}|}$$

(d) Otherwise we use outer recursion

$$\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\alpha_i} x\vec{f} := f_i x \qquad \mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}(\rho_1\to\rho_2)} h\vec{f} := \lambda_x \left(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\rho_2}(hx)\vec{f}\right)$$

Example 2. We give some examples for the map operator conversion:
(1) For the algebra of lists $\mathbf{L}(\alpha)$ we have

$$\mathcal{M}^{\beta}_{\mathbf{L}(\alpha)} \mathsf{Nil} f^{\alpha\to\beta} \;=\; \mathsf{Nil} \qquad \mathcal{M}^{\beta}_{\mathbf{L}(\alpha)}(\mathsf{cons}\ x\ l) f^{\alpha\to\beta} \;=\; \mathsf{cons}\ f(x)\ \mathcal{M}^{\beta}_{\mathbf{L}(\alpha)} l f$$

(2) For the sum type $\alpha + \beta$

$$\mathcal{M}^{\vec{\tau}}_{\sigma_1+\sigma_2}(\mathsf{InL}\ s_1)\vec{f} \;=\; \mathsf{InL}\left(\mathcal{M}^{(\vec{\sigma},\sigma_1+\sigma_2)\to(\vec{\tau},\tau_1+\tau_2)}_{\lambda_{\vec{\alpha},\beta}\alpha_1} s_1 \left(\vec{f}, \ldots\right)\right) \;=\; \mathsf{InL}\ f_1(s_1)$$

(3) Let

$$\iota := \mu_\xi \left((\sigma_{i\nu})_{\nu<n_i} \to (\rho_{i\nu} \to \xi)_{\nu<m_i} \to \xi\right)_{i<k}$$

with $\sigma_{i\nu}$ parameter arguments. Then for $u : \sigma_{i\nu}$

$$\mathcal{M}^{\iota\to\iota\times\tau}_{\lambda_\alpha\sigma_{i\nu}} u f \;=\; u \tag{2.1}$$

and for $v : \rho_{i\nu} \to \iota$

$$\mathcal{M}^{\iota\to\iota\times\tau}_{\lambda_\alpha(\rho_{i\nu}\to\alpha)} v f \;=\; \lambda_{x^{\rho_{i\nu}}} \left(\mathcal{M}^{\iota\to\iota\times\tau}_{\lambda_\alpha\alpha}(vx) f\right) \;=\; \lambda_x f(vx) \tag{2.2}$$

2.1.3 (Co)Total ideals

Let us take a closer look at some of the ideals in the information system $\mathbf{C}_\iota$ (ι some algebra). In particular we would like to single out the *total* and *cototal* ideals. In the following we will define those for finitary algebras. For simplicity we assume that no parameter arguments occur in the algebras considered.

Definition 6 ((co)total ideals)**.** Let $P(*)$ be some constructor tree with some distinguished occurrence of $*$, then we call $P(\mathsf{C}\vec{a^*})$ a *one-step extension* of $P(*)$ and write $P(\mathsf{C}\vec{a^*}) \succ_1 P(*)$. An ideal $x \in |\mathbf{C}_\iota|$ is called *cototal* if for every $P \in x$ and for all occurrences of $*$ in P there is $P(C\vec{a^*}) \in x$ with $P(\mathsf{C}\vec{a^*}) \succ_1 P(*)$. We call $x \in |\mathbf{C}_\iota|$ *total* if it is cototal and $\succ_1$ is a well-founded relation on x.

As an example consider the natural numbers $\mathbf{N}$. Total ideals are then given by the deductive closure of tokens $\mathsf{S}^n\mathsf{0}$. The set $\{\mathsf{S}^n * \mid n \geq 0\}$ is a cototal ideal. For the algebra **Str** there are no total ideals, since the algebra is not well-founded. The cototal ideals are the "well-defined" infinite streams, e.g. $\{\mathsf{C}_1{\mathsf{C}_1}^n * \mid n \geq 0\}$ which represents the infinite stream "$1(-1)(-1)\ldots$".

Remark 2. In case of parameter arguments we can define *structure-(co-)totality* in the same way as above by restricting occurrences of $*$ to non-parametric positions.

2.2 A Term language

This chapter will introduce the terms of our language in section 2.2.1. In 2.2.2 we study some special terms, namely the *(co)recursion operators* which will be of particular interest later.

2.2.1 Terms

There are several ways to introduce actual computation. One way would be to use constants together with defining equations. This is a very intuitive method to define mappings but it tends to lead to a lot of case distinctions in formal proofs. We can alleviate the latter by using recursors and a suitable term rewriting systems, however terms can become long and unreadable. For a maximum of flexibility we will use both (Actually the recursors will be instances of the former and we only add the term rewriting system).
The terms of our language will be built up from typed variables and constants C, D through (type-correct) application and abstraction.

Definition 7 (Terms)**.** The *Terms* T^+ are defined inductively by

$$M, N := \; x^\rho \mid \mathsf{C}^\rho \mid D^\rho \mid (\lambda_{x^\rho} M^\sigma)^{\rho\to\sigma} \mid (M^{\rho\to\sigma} N^\rho)^\sigma$$

Where the C are constructors and D are *(program) constants* defined by *computation rules* (see below). Moreover we identify terms according to the following conversion rules.

$$\begin{aligned}(\lambda_x M(x))\, N &\mapsto_\beta M(N)\\ \lambda_x\,(Mx) &\mapsto_\eta M \qquad \text{if } x \notin FV(M)\end{aligned}$$

Definition 8 (Constructor patterns)**.** We inductively define *Constructor patterns* $CP(\cdot)$ by
(a) The empty list is a constructor pattern: $CP()$.
(b) x is a constructor pattern: $CP(x)$.
(c) If $\vec{P}, Q$ are constructor patterns with disjoint variables, then $CP(\vec{P}, Q)$.
(d) If C is a constructor and $CP(\vec{P})$ then $CP(\mathsf{C}\vec{P})$.

Definition 9 (Computation rule)**.** Every constant D is defined by finitely many *computation rules* of the form

$$D\vec{P}_i\,(\vec{y}_i) \;=\; M_i \tag{2.3}$$

where the free variables of $\vec{P}_i$ and M_i are among $\vec{y}_i$ and the $\vec{P}_i$ are *constructor patterns*, such that $\vec{P}_i$ and $\vec{P}_j$ $(i \neq j)$ have disjoint free variables and are non unifiable. Moreover the *arity* of all $\vec{P}_i(\vec{y}_i)$ is the same. We identify terms according to

$$D\vec{P}_i\,(\vec{y}_i) \mapsto M_i \tag{2.4}$$

Example 3. We consider some examples for defined constants:
(1) We define the boolean connective "and" (andb) by

$$\begin{aligned} x \;\mathsf{andb}\; \mathsf{tt} &= x & \qquad x \;\mathsf{andb}\; \mathsf{ff} &= \mathsf{ff}\\ \mathsf{tt} \;\mathsf{andb}\; y &= y & \qquad \mathsf{ff} \;\mathsf{andb}\; y &= \mathsf{ff}\end{aligned}$$

(2) For finitary algebras ι we define the *decidable equality* $=_\iota\colon \iota \to \iota \to \mathbf{B}$ by

$$\begin{aligned}\left(\mathsf{C}_i\vec{x} =_\iota \mathsf{C}_j\vec{y}\right) &= \mathsf{ff} \quad \text{, if } i \neq j\\ (\mathsf{C} =_\iota \mathsf{C}) &= \mathsf{tt} \quad \text{, if } C \text{ nullary constructor}\\ (\mathsf{C}_i\vec{x} =_\iota \mathsf{C}_i\vec{y}) &= \operatorname*{Andb}_{\nu<n_i} \left(x_\nu =_{\rho_{i\nu}} y_\nu\right)\end{aligned}$$

e.g. for Natural numbers $\mathbf{N}$

$$\begin{aligned}(0 =_{\mathbf{N}} 0) &= \mathsf{tt} & \qquad (\mathsf{S}n =_{\mathbf{N}} \mathsf{S}m) &= n =_{\mathbf{N}} m\\ (0 =_{\mathbf{N}} \mathsf{S}n) &= \mathsf{ff} & \qquad (\mathsf{S}n =_{\mathbf{N}} 0) &= \mathsf{ff}\end{aligned}$$

(3) Let $\iota := \mu_\xi \left(\left(\rho_{i\nu}\,(\xi)_{\nu<n_i} \to \xi\right)\right)_{i<k}$. We define the *case-operator*

$$\mathcal{C}^\tau_\iota : \iota \to \left(\left(\rho_{i\nu}\,(\iota)_{\nu<n_i} \to \tau\right)\right)_{i<k} \to \tau$$

by the computation rules

$$\mathcal{C}^\tau_\iota(\mathsf{C}_i\vec{N})\vec{M} \;=\; M_i\vec{N} \qquad (i = 0, \ldots, k-1) \tag{2.5}$$

We also write (utilizing a programming language style)

$$\begin{aligned}\big[\mathsf{case}\ x^{\iota}\ \mathsf{of}\ (\mathsf{C}_0\vec{N}_0 \mapsto M_0\vec{N}_0 \mid \\ \vdots \\ \mathsf{C}_{k-1}\vec{N}_{k-1} \mapsto M_{k-1}\vec{N}_{k-1})\big]\end{aligned}$$

Using this we can e.g. define the *predecessor* for $\mathbf{N}$ by

$$\big[\mathsf{case}\ n^{\mathbf{N}}\ \mathsf{of}\ (0 \mapsto 0 \mid \mathsf{S}n \mapsto n)\big]$$

(4) For some algebra $\iota = \mu_\xi\,(\kappa_0, \ldots, \kappa_{k-1})$ with constructor types

$$\kappa_i(\iota) = (\rho_{i\nu}\,(\iota))_{\nu<n_i} \to \iota$$

We define the *destructor* D_ι. Its type is given by

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

The computation rules are

$$D_\iota(\mathsf{C}_j\vec{x}) \;=\; \mathrm{Inj}_j\langle x_1, \ldots, x_{n_i}\rangle \qquad (j = 1, \ldots k)$$

Where Inj_j is the j-th injection into the sum-type i.e.

$$\begin{aligned}&\mathrm{Inj}_j : \tau_j \to \sum_{i<k}\tau_i \\ &\mathrm{Inj}_j x := \mathsf{InL}^j\left(\mathsf{InR}^{k-j}x\right)\end{aligned}$$

E.g. for the algebra **Str** we have

$$D_{\mathbf{Str}}C_d v \;=\; \langle d, v\rangle$$

Remark 3. To make sense of the type of the destructor, note that for an algebra as above the product of the k constructor types is isomorphic to

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

2.2.2 (Co)Recursion

A special instance of the defined constants are the *(co-)recursion operators*. The recursion operators $\mathcal{R}^\tau_\iota$ are used to define maps $\iota \to \tau$ by recursion on the structure of the algebra ι. For every constructor C^ι_i there will be a computation rule. For instance $\mathcal{R}^\tau_{\mathbf{N}}$ is given by

$$\begin{aligned}&\mathcal{R}^\tau_{\mathbf{N}} : \mathbf{N} \to \tau \to (\mathbf{N} \to \tau \to \tau) \to \tau \\ &\mathcal{R}^\tau_{\mathbf{N}}0N^\tau M^{\mathbf{N}\to\tau\to\tau} \;=\; N^\tau \\ &\mathcal{R}^\tau_{\mathbf{N}}(\mathsf{N}n)NM \;=\; (Mn)^{\tau\to\tau}(\mathcal{R}^\tau_{\mathbf{N}}nNM)^\tau\end{aligned}$$

We can e.g. use $\mathcal{R}^{\mathbf{N}}_{\mathbf{N}}$ to define addition on natural numbers, namely

$$n + m \;=\; \mathcal{R}^{\mathbf{N}}_{\mathbf{N}} m n \lambda_{-,m} \mathsf{S} m$$

Indeed via the computation rules above

$$n + 0 \;=\; n \qquad n + \mathsf{S}m \;=\; \mathsf{S}(\mathcal{R} m n \lambda_{-,m} \mathsf{S} m) \;=\; \mathsf{S}(n + m)$$

For the general case let $\iota = \mu_\xi\,(\kappa_0, \ldots, \kappa_{k-1})$ then we define the i-th *step-type* w.r.t. τ as

$$\kappa_i(\iota, \tau) \coloneqq (\rho_{i\nu}\,(\iota \times \tau))_{\nu < n_i} \to \tau$$

The recursion operator for type ι (w.r.t τ) is then given by

$$\begin{aligned}
&\mathcal{R}^\tau_\iota : \iota \to (\kappa_i\,(\iota, \tau))_{i<k} \to \tau \\
&\mathcal{R}^\tau_\iota(\mathsf{C}^\iota_i \vec{N})\vec{M} \;=\; M_i \left(\mathcal{M}^{\iota \to \iota \times \tau}_{\lambda_\alpha \rho_{i\nu}(\alpha)} N_\nu \lambda_x \langle x, \mathcal{R}^\tau_\iota x \vec{M}\rangle\right)_{\nu < n_i} \qquad (i = 0, \ldots, k-1)
\end{aligned}$$

In case the algebra ι has the following form

$$\iota \coloneqq \mu_\xi \left((\sigma_{i\nu})_{\nu<n_i} \to (\rho_{i\nu} \to \xi)_{\nu<m_i} \to \xi\right)_{i<k}$$

with $\sigma_{i\nu}$ parameter arguments the conversion rule can be simplified (as in example 2):

$$\begin{aligned}
\mathcal{R}^\tau_\iota(\mathsf{C}^\iota_i \vec{N}^p \vec{N}^r)\vec{M} \;&=\; M_i \left(\mathcal{M}^{\iota \to \iota \times \tau}_{\lambda_\alpha \sigma_{i\nu}} N^p_\nu \lambda_x \langle x, \mathcal{R}^\tau_\iota x \vec{M}\rangle\right)_{\nu<n_i} \left(\mathcal{M}^{\iota \to \iota \times \tau}_{\lambda_\alpha(\rho_{i\nu} \to \alpha)} N^r_\nu \lambda_x \langle x, \mathcal{R}^\tau_\iota x \vec{M}\rangle\right)_{\nu<m_i} \\
&=\; M_i \vec{N}^p \left(\lambda_z \langle N^r_\nu z, \mathcal{R}^\tau_\iota (N^r_\nu z)\vec{M}\rangle\right)_{\nu<m_i}
\end{aligned}$$

Example 4. We give the types and computation rules of the recursion operator for some algebras.

$$\begin{aligned}
&\mathcal{R}^\tau_{\mathbf{B}} : \mathbf{B} \to \tau \to \tau \to \tau \\
&\mathcal{R}^\tau_{\mathbf{B}} \mathsf{tt} t_1 t_2 \;=\; t_1 \qquad \mathcal{R}^\tau_{\mathbf{B}} \mathsf{ff} t_1 t_2 \;=\; t_2 \\
&\mathcal{R}^\tau_{\alpha\times\beta} : \alpha \times \beta \to (\alpha \to \beta \to \tau) \to \tau \\
&\mathcal{R}^\tau_{\alpha\times\beta} \langle a, b\rangle f \;=\; f(a, b) \\
&\mathcal{R}^\tau_{\alpha+\beta} : \alpha + \beta \to (\alpha \to \tau) \to (\beta \to \tau) \to \tau \\
&\mathcal{R}^\tau_{\alpha+\beta}(\mathsf{InL}\ a) f g \;=\; f(a) \qquad \mathcal{R}^\tau_{\alpha+\beta}(\mathsf{InR}\ b) f g \;=\; g(b)
\end{aligned}$$

Example 5. We define some useful constants:
(1) We define for $M : \alpha \times \beta$ the *projections* onto its components by

$$M_0 \coloneqq \mathcal{R}^\alpha_{\alpha\times\beta} M \lambda_{x,y} x \qquad M_1 \coloneqq \mathcal{R}^\beta_{\alpha\times\beta} M \lambda_{x,y} y$$

(2) We write for $f : \alpha \to \tau$ and $g : \beta \to \tau$

$$[f, g](z) \coloneqq \mathcal{R}^\tau_{\alpha+\beta} z f g$$

then

$$[f, g](\mathsf{InL}\ a) = f(a) \qquad [f, g](\mathsf{InR}\ b) = g(b)$$

Examining the computation rules of the recursion operators, we see that in each step of the computation the argument loses its foremost constructor. If we apply the recursion operator to some total ideal, then the computation terminates, since a total ideal is in particular well-founded. To deal with cototal ideals (e.g. infinite streams) we can define a similar operator, namely the *corecursion operator* ${}^{co}\mathcal{R}_\iota^\tau$, which enables us to define maps $\tau \to \iota$ by corecursion on the structure of the algebra ι. As a first example we consider the corecursion operator for the type of lists $\mathbf{L}(\rho)$. Its type and computation rule are given by

$$\begin{aligned}
&{}^{co}\mathcal{R}^\tau_{\mathbf{L}(\rho)} : \tau \to (\tau \to \mathbf{U} + \rho \times (\mathbf{L}(\rho) + \tau)) \to \mathbf{L}(\rho)\\
&{}^{co}\mathcal{R}^\tau_{\mathbf{L}(\rho)} NM \;=\; \left[\lambda_\,\mathsf{Nil}, \lambda_{x^{\rho\times(\mathbf{L}(\rho)+\tau)}}\mathsf{cons}\; x_0\left[id, \lambda_y\left({}^{co}\mathcal{R}^\tau_{\mathbf{L}(\rho)} y M\right)\right]\right](MN)
\end{aligned}$$

Given some map $f : \rho \to \sigma$ we can extend the map to infinite lists $\overline{f} : \mathbf{L}(\rho) \to \mathbf{L}(\sigma)$ in the following way.

$$\overline{f}(l) \coloneqq {}^{co}\mathcal{R}^{\mathbf{L}(\rho)}_{\mathbf{L}(\sigma)}\; l\; \mathsf{InR}\,\langle f(\mathsf{head}(l)), \mathsf{InR}\;\mathsf{tail}(l)\rangle$$

Indeed via the computation rule above we get

$$\overline{f}(l) \;=\; \mathsf{cons}\;(f(\mathsf{head}\;l))\left({}^{co}\mathcal{R}^{\mathbf{L}(\rho)}_{\mathbf{L}(\sigma)}\;\;(\mathsf{tail}\;l)\;\;M\right) =\; \mathsf{cons}\; f(\mathsf{head}\;l)\;\overline{f}(\mathsf{tail}\;l)$$

Which is exactly what we want.
In the general case we again consider an algebra $\iota = \mu_\xi\,(\kappa_0, \ldots, \kappa_{k-1})$ with constructor types

$$\kappa_i(\iota) = (\rho_{i\nu}\,(\iota))_{\nu<n_i} \to \iota$$

Using the observation in remark 3 the type of the recursion operator $\mathcal{R}_\iota^\tau$ is isomorphic to

$$\iota \to \left(\sum_{i<k}\prod_{\nu<n_i}\rho_{i\nu}(\iota\times\tau)\to\tau\right)\to\tau$$

Then dually the type of the corecursion operator is

$${}^{co}\mathcal{R}_\iota^\tau : \tau \to \left(\tau \to \sum_{i<k}\prod_{\nu<n_i}\rho_{i\nu}(\iota+\tau)\right)\to\iota$$

We give the type for the algebras $\mathbf{N}$, $\mathbf{Str}$, $\mathbf{Nt}$.

$$\begin{aligned}
&{}^{co}\mathcal{R}^\tau_{\mathbf{N}} : \tau \to (\tau \to \mathbf{U} + (\mathbf{N} + \tau)) \to \mathbf{N}\\
&{}^{co}\mathcal{R}^\tau_{\mathbf{Str}} : \tau \to (\tau \to \mathbf{SD} \times (\mathbf{Str} + \tau)) \to \mathbf{Str}\\
&{}^{co}\mathcal{R}^\tau_{\mathbf{Nt}} : \tau \to (\tau \to \mathbf{L}\,(\mathbf{Nt} + \tau)) \to \mathbf{L}(\mathbf{Nt})
\end{aligned}$$

The computation rule for ${}^{co}\mathcal{R}_\iota^\tau NM$ will work as follows: Since $MN : \sum_{i<k}\prod_{\nu<n_i}\rho_{i\nu}(\iota+\tau)$, we have for some j that

$$MN = \mathrm{Inj}_j\,\vec{x}$$

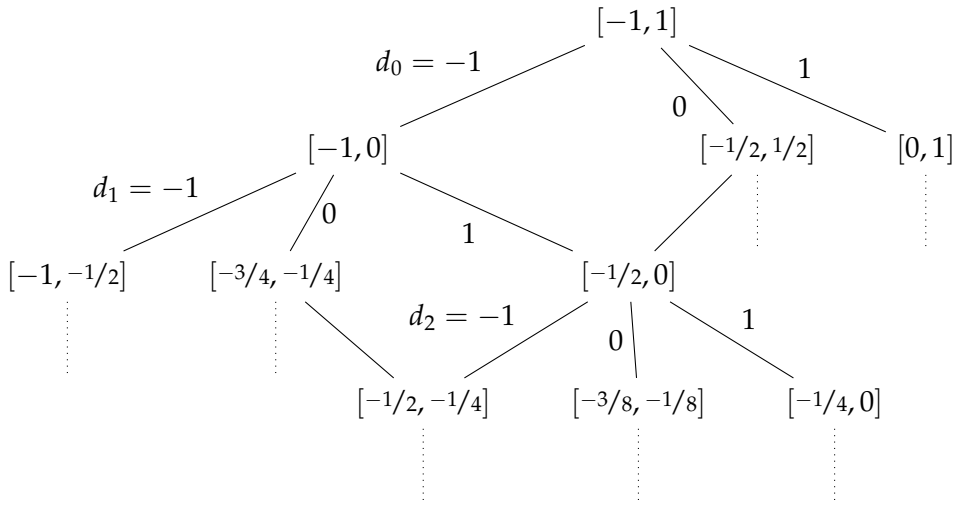
Where $\vec{x}$ is of type $\prod_{\nu<n_j} \rho_{j\nu}(\iota+\tau)$. The result is then the application of the constructor C_j^ι to $\vec{y}$, where

$$\begin{aligned} y_\nu \coloneqq \mathcal{M}^{\iota+\tau\to\iota}_{\lambda_\alpha \rho_{j\nu}(\alpha)} x_\nu \big(\lambda_p \big[&\mathsf{case}\ p^{\iota+\tau}\ \mathsf{of} \\ &\mathsf{InL}\ z^\iota \mapsto z \\ &\mathsf{InR}\ w^\tau \mapsto {}^{co}\mathcal{R}^\tau_\iota w M\big]\big) \end{aligned}$$

For the types above the computation rules are hence given by the following.

$$\begin{aligned} {}^{co}\mathcal{R}^\tau_{\mathbf{N}} NM &= \left[\lambda_{\_}0, \lambda_x \left(\mathsf{S}\left([id, \lambda_y {}^{co}\mathcal{R}^\tau_{\mathbf{N}} y M] x\right)\right)\right](MN) \\ {}^{co}\mathcal{R}^\tau_{\mathbf{Str}} NM &= \left(\lambda_{d,x} \left(\mathsf{C}_d \left([id, \lambda_y {}^{co}\mathcal{R}^\tau_{\mathbf{Str}} y M] x\right)\right)\right)(MN) \\ {}^{co}\mathcal{R}^\tau_{\mathbf{Nt}} NM &= \mathsf{Br}\left(\mathcal{M}^{\mathbf{Nt}+\tau\to\mathbf{Nt}}_{\lambda_\alpha \mathbf{L}(\alpha)}(MN)[id, \lambda_z({}^{co}\mathcal{R}^\tau_{\mathbf{Nt}} z M)]\right) \end{aligned}$$

Example 6. We want to use the corecursion operator ${}^{co}\mathcal{R}_{\mathbf{Str}}$ to represent real numbers, lying in the interval $[-1,1]$, by cototal ideals in **Str**, i.e. infinite strings of signed digits. We interpret such an infinite string $d_0 d_1 \ldots$ as nested intervals in $[0,1]$ according to the tree below



i.e. in every step the preceding interval is trisected into three (overlapping) ones. For now we view real numbers as given by some type σ. If x is such an abstract real number then we say that $d_0 d_1 \ldots$ represents x if it lies in all of the nested intervals, i.e.

$$\begin{aligned} & x \in \left[\frac{-1}{2} + \frac{d_0}{2}, \frac{1}{2} + \frac{d_0}{2}\right] \\ & x \in \left[\frac{-1}{4} + \frac{d_0}{2} + \frac{d_1}{4}, \frac{1}{4} + \frac{d_0}{2} + \frac{d_1}{4}\right] \longleftrightarrow 2x - d_0 \in \left[\frac{-1}{2} + \frac{d_1}{2}, \frac{1}{2} + \frac{d_1}{2}\right] \\ & \vdots \end{aligned}$$

So that we can formulate our notion of representation by the following.

$$(d_0 d_1 \dots) \text{ represents } x \;:=\; x \in \left[\frac{-1}{2} + \frac{d_0}{2}, \frac{1}{2} + \frac{d_0}{2}\right] \wedge\; (d_1 d_2 \dots) \text{ represents } (2x - d_0)$$

Note that this representation is not unique, e.g. the streams $1(-1)(-1)\dots$ and $(-1)11\dots$ both represent the real number zero. Now assume that we can compare real numbers to rational intervals, i.e. we have some function

$$f : \mathbf{Q} \times \mathbf{Q} \to \sigma \to \mathbf{Boole}$$

such that for rational numbers $a,\ b$ with $a < b$ it holds that

$$f(\langle a, b\rangle, x) = \mathsf{tt} \longleftrightarrow x \leq b$$
$$f(\langle a, b\rangle, x) = \mathsf{ff} \longleftrightarrow a \leq x$$

Note that we can actually constructively prove such a result for reals represented by cauchy-sequences (see lemma 7 in chapter 3).
To construct the first digit we need to decide whether x is contained in $[-1, 0]$, $[{}^{-1}/_2, {}^1/_2]$ or $[0, 1]$. We can do this by comparing x to the intervals $0 < {}^1/_2$ and ${}^{-1}/_2 < 0$ respectively. We obtain the three cases

$$x \leq 0 \qquad {}^{-1}/_2 \leq x \leq {}^1/_2 \qquad 0 \leq x$$

Then $d_0 = -1, 0, 1$ accordingly. A similar comparison with $2x - d_0$ then yields the next digit. We define the function

$$g : \sigma \to (\mathbf{SD} \times (\mathbf{Str} + \sigma))$$

by

$$\begin{aligned} g(x) := \big[&\mathsf{case}\ (f(\langle {}^{-1}/_2, 0, \rangle, x))\ \mathsf{of} \\ &\quad \mathsf{tt} \to \langle -1, \mathsf{InR}(2x+1)\rangle \\ &\quad \mathsf{ff} \to \big[\mathsf{case}\ (f(\langle 0, {}^1/_2, \rangle, x))\ \mathsf{of} \\ &\qquad\quad \mathsf{tt} \to \langle 0, \mathsf{InR}(2x)\rangle \\ &\qquad\quad \mathsf{ff} \to \langle 1, \mathsf{InR}(2x-1)\rangle\big]\big] \end{aligned}$$

Using this function we then define the translation from σ to **Str** by

$$\mathsf{RealToStr} := \lambda_x {}^{co}\mathcal{R}^{\sigma}_{\mathbf{Str}}\ x\ g$$

According to the conversion rule for ${}^{co}\mathcal{R}_{\mathbf{Str}}$ we have

$$\mathsf{RealToStr}(x) \;=\; {}^{co}\mathcal{R}^{\sigma}_{\mathbf{Str}}\ x\ g \;=\; \mathsf{C}_{g(x)_0}\,{}^{co}\mathcal{R}^{\sigma}_{\mathbf{Str}}\ g(x)_1\ g \;=\; \mathsf{C}_{g(x)_0}\mathsf{RealToStr}(g(x)_1)$$

which is exactly what we wanted.

2.3 (Co)inductive definitions

Proofs are intended to ascertain the correctness of some statement, but why do they carry more information, namely some form of "computational content? To gain an insight we consider the *Brouwer-Heyting-Kolmogorov-interpretation* for intuitionistic logic. Formulas will be viewed as problems, and proofs as solutions, where the notion of proof is explained by the following:

(a) p proves $A \to B$ if and only if p is a construction that transforms any proof q of A into a proof $p(q)$ of B.
(b) p proves $\forall_x A$ if and only if p is a construction such that for all x $p(x)$ proves $A(x)$.
(c) $\bot$ does not have a proof.

We are clearly missing some notion of prime formula and proof thereof in the definition above. Furthermore it relies on the unexplained notion of *construction*. We propose to solve this by specifying a construction to be a computable functional. We will see prime formulas as (co-)inductively defined predicates and a proof of such a formula as a generation tree.
In this chapter we give a deductive calculus in 2.3.1 and introduce inductive and coinductive definitions in 2.3.2

2.3.1 Minimal logic with decoration

Our main concern is the notion of mathematical proof. We outline the basics of formalizing such proofs in a suitable way. We will view proofs as derivations in natural deduction using only $\to$ and $\forall$ as logical connectives. The other usual logical connectives will later be added as instances of inductively defined predicates (also see [8]).
The Curry-Howard isomorphism will give us a representation of proofs in typed lambda calculus. The logical rules of implication and the universal quantifier will correspond to the rules of lambda calculus. This representation will allow us to apply techniques of lambda calculus and easily work with proofs as objects. These proof terms tend to be very long however. This leads to the implementation of the theory in a proof assistant like Minlog, shifting the weight of dealing with these objects to a computer.
Suppose $A, B, \ldots$ and $x, y, \ldots$ are (for now abstract) formulas and variables, where formulas have a (possibly empty) "arity", denoted by $FV(A)$. A formal derivation is a tree-like structure with the conclusion at the root and assumptions at the leaves. To distinguish the latter we will use markers (u,v,...). We define the rules of natural deduction in the usual way.

Definition 10. (Rules of Minimal Logic)

$$u : A \qquad \frac{\begin{array}{c}[u:A]\\ \vdots\,\mathrm{M}\\ B\end{array}}{A \to B}\ \to^+ u \qquad \frac{\vdots\,\mathrm{M} \quad A \to B \qquad \vdots\,\mathrm{N} \quad A}{B}\ \to^- \qquad \frac{\vdots\,\mathrm{M}\ A}{\forall_x A}\ \forall^+ x\ (\mathrm{VC}) \qquad \frac{\vdots\,\mathrm{M}\ \forall_x A \quad r}{A(r)}\ \forall^-$$

Where in the $\forall^+$-rule we have the variable condition (VC), that x is not free in any open assumption of the derivation M.

Note that an application of the $\to^+$-rule with $u : A$ closes all assumptions marked with u in the tree above. This is denoted by putting square brackets around the assumption. We can now define what we mean by saying "A is derivable".

Definition 11 (Derivability)**.** A formula A is *derivable*, written $\vdash A$, if there is a derivation M using the rules of minimal logic, such that there are no free assumptions in M. A formula B is *derivable* from assumptions $A_1, A_2, \ldots, A_n$, written $A_1, A_2, \ldots, A_n \vdash B$ if there is a derivation M using the rules of minimal logic, such that the free assumptions of M are all among $A_1, \ldots A_n$.

As mentioned above we can axiomatically introduce the other logical connectives. As an example consider the disjunction, which we can characterize by the following axioms.

$$\begin{aligned}
&\vee_0^+ : A \to A \vee B\\
&\vee_1^+ : B \to A \vee B\\
&\vee^- : A \vee B \to (A \to C) \to (B \to C) \to C
\end{aligned}$$

This is actually a special case of an inductively defined predicate.
Up to now we have not dealt with negation. We will define it by

$$\neg A :\equiv A \to \bot$$

where $\bot$ is (for now) an arbitrary but distinct predicate variable. We can fix the meaning of $\bot$ by requiring *ex-falso-quodlibet* axioms of the form

$$\forall_x(\bot \to A)$$

If the set of all these axioms is denoted by Efq, then we say that a formula A is *intuitionistically derivable* if $\mathrm{Efq} \vdash A$.
The tree-like derivations can become complicated fast, especially if we want to operate with the derivations themselves. For this reason we would like a linear representation of derivations, particularly in conjunction with the computational aspects that we will consider later. The representation treated here is given by the Curry-Howard correspondence and goes back to [3]. The main idea is, that if M is a derivation of A, then M is a term of type A. The elimination rules correspond to application, while the introduction rules correspond to abstraction. We inductively define the translation from proof-trees to λ-terms.

$u : A$	$\dfrac{\begin{array}{c}[u:A]\\ \vdots\,\mathrm{M}\\ B\end{array}}{A \to B}\ \to^+ u$	$\dfrac{\begin{array}{cc}\vdots\,\mathrm{M} & \vdots\,\mathrm{N}\\ A \to B & A\end{array}}{B}\ \to^-$	$\dfrac{\vdots\,\mathrm{M}}{\begin{array}{c}A\\ \forall_x A\end{array}}\ \forall^+ x$	$\dfrac{\vdots\,\mathrm{M}}{\begin{array}{cc}\forall_x A & r\end{array}}\ \forall^-$
u^A	$\left(\lambda_{u^A}\left(M^B\right)\right)^{A\to B}$	$\left(M^{A\to B}N^A\right)^B$	$\left(\lambda_x M^A\right)^{\forall_x A}$	$\left(M^{\forall_x A}r\right)^{A(r)}$

As an example for the above let us consider the following formula.

$$(A \to \forall_x B) \to \forall_x (A \to B) \quad x \notin FV(A)$$

A derivation and its corresponding proof term are given by

$$\dfrac{\dfrac{\dfrac{\dfrac{[u : A \to \forall_x B] \quad [v : A]}{\forall_x B}\to^- \quad x}{B}\forall^-}{A \to B}\to^+ v}{\forall_x(A \to B)}\forall^+ x}{(A \to \forall_x B) \to \forall_x (A \to B)}\to^+ u \qquad \lambda_u \lambda_x \lambda_v ((uv)^{\forall_x B} x)^B$$

Remark 4. According to β-conversion we can normalize proofs. Consider the following derivations, which both have "detours".

$$\dfrac{\dfrac{[u : A]}{\vdots M}{B}}{A \to B}\to^+ u \qquad \dfrac{\vdots N}{A}{B}\to^- \quad , ((\lambda_u M)^{A\to B} N^A)^B \quad \mapsto_\beta \quad \dfrac{\vdots N}{A}{\vdots M}{B} \quad , M(N)$$

$$\dfrac{\dfrac{\vdots M}{A}}{\forall_x A(x)}\forall^+ x \qquad \dfrac{r}{A(r)}\forall^- \quad , ((\lambda_x M)^{\forall_x A(x)} r)^{A(r)} \quad \mapsto_\beta \quad \dfrac{\vdots M(r)}{A(r)} \quad , M(r)^{A(r)}$$

One can prove that any reduction sequence by β-conversion terminates and that the normal form is unique (see [11]).

We adapt our deductive calculus by introducing *decorated* connectives $\to^{nc}$, $\forall^{nc}$. They are *non-computational* (n.c.) variants of the connectives, whereas we view the un-decorated $\forall$, $\to$ as computational. The intended meaning of $\forall_x^{nc}$ is that the variables x are only viewed as "input", but in contrast to $\forall_x$ no construction is supplied. The meaning of $A \to^{nc} B$ is that we do not want to use any construction made in a proof of A. We need to take some care in the introduction rules for the non-computational connectives. We need the following definition.

Definition 12 (computational assumptions/variables)**.** We define the *computational assumptions* of derivations by

$$
\begin{aligned}
&CA(c^A) := \emptyset \qquad c \text{ Axiom}\\
&CA(u^A) := \{u\}\\
&CA((\lambda_{u^A} M^B)^{A\to B}) = CA((\lambda_{u^A} M^B)^{A\to^{nc} B}) := CA(M) - \{u\}\\
&CA(M^{A\to B} N^A) := CA(M) \cup CA(N)\\
&CA(M^{A\to^{nc} B} N^A) := CA(M)\\
&CA((\lambda_x M^A)^{\forall_x A}) = CA((\lambda_x M^A)^{\forall^{nc}_x A}) := CA(M)\\
&CA((M^{\forall_x A} r)^{A(r)}) = CA((M^{\forall^{nc}_x A} r)^{A(r)}) := CA(M)
\end{aligned}
$$

If A is n.c., $\tau(A) = \circ$ (see later), then we set $CA(M^A) = \emptyset$. Similarly we define the *computational variables* of derivations by

$$
\begin{aligned}
&CV(c^A) := \emptyset \qquad c \text{ Axiom}\\
&CV(u^A) := \emptyset\\
&CV((\lambda_{u^A} M^B)^{A\to B}) = CV((\lambda_{u^A} M^B)^{A\to^{nc} B}) := CV(M)\\
&CV(M^{A\to B} N^A) := CV(M) \cup CV(N)\\
&CV(M^{A\to^{nc} B} N^A) := CV(M)\\
&CV((\lambda_x M^A)^{\forall_x A}) = CV((\lambda_x M^A)^{\forall^{nc}_x A}) := CV(M) - \{x\}\\
&CV((M^{\forall_x A} r)^{A(r)}) := CV(M) \cup FV(r)\\
&CV((M^{\forall^{nc}_x A} r)^{A(r)}) := CV(M)
\end{aligned}
$$

And again $CV(M^A) = \emptyset$ if A is non-computational. Now the elimination rules for $\forall^{nc}$ and $\to^{nc}$ will be the same as for their computational variants. Using the above definition we can give the introduction rules:

(i) If M^B and $u^A \notin CA(M)$ then $\lambda_{u^A} M : A \to^{nc} B$.

(ii) If M^A, x not free in open assumptions of M and $x \notin CV(M)$ then $\lambda_x M : \forall^{nc}_x A$.

2.3.2 (Co)Inductively defined predicates

To make mathematical statements about computable functionals we will take (co)inductively defined predicates as initial propositions. As a first example consider the algebra of natural numbers **N**. We can inductively define the statement "n is even" by the clauses

$$\mathrm{Ev}(0) \qquad \forall^{nc}_n (\mathrm{Ev}(n) \to \mathrm{Ev}(SSn)) \tag{2.6}$$

Furthermore we express that Ev is the least predicate satisfying (2.6) by

$$\forall^{nc}_n(\mathrm{Ev}n \to P0 \to \forall^{nc}_n(\mathrm{Ev}(n) \to Pn \to P(Sn)) \to Pn)$$

i.e. any "competitor predicate" P that satisfies the same clauses contains Ev. We will define formulas by arrow-formation $A \to^{nc} B$, $A \to B$ and all-quantification $\forall_x^{nc} A$, $\forall A_x$ of (co)inductively defined predicates whose meaning will be fixed by (introduction- and elimination-) axioms. We restrict our attention to inductively defined predicates for now.

Definition 13 (Predicates and formulas)**.** Let X denote a predicate variable. By simultaneous induction we define *predicate forms*

$$P, Q := X \;\Big|\; X^{nc} \;\Big|\; \{\vec{x} \mid A\} \;\Big|\; \mu_X \left(\forall_{\vec{x}_i}^{c/nc} \left((A_{i\nu})_{\nu<n_i} \to^{c/nc} X\vec{r}_i\right)\right)_{i<k} \;\Big|\; \mu_X^{nc} \left(\forall_{\vec{x}_i} \left((A_{i\nu})_{\nu<n_i} \to X\vec{r}_i\right)\right)_{i<k}$$

and *formula forms*

$$A, B := P\vec{r} \mid A \to B \mid A \to^{nc} B \mid \forall_x A \mid \forall_x^{nc} A$$

Where the free variables of $(A_{i\nu})_{\nu<n_i} \to^{c/nc} X\vec{r}_i$ range over $\vec{x}_i$. Let C be a predicate or formula form and $PV(C)$ the set of predicate variables in C. We define $SP(Y, C)$, meaning that the predicate variable Y occurs at most strictly positive in C.

$$SP(Y, X) \qquad \frac{SP(Y, A)}{SP(Y, \{\vec{x} \mid A\})} \qquad \frac{SP(Y, A_{i\nu}) \quad (\text{for all } i < k,\ \nu < n_i)}{SP\left(Y, \mu_X^{c/nc} \left(\forall_{\vec{x}_i}^{c/nc} \left((A_{i\nu})_{\nu<n_i} \to^{c/nc} X\vec{r}_i\right)\right)_{i<k}\right)}$$

$$\frac{SP(Y, P)}{SP(Y, P\vec{r})} \qquad \frac{Y \notin PV(A) \quad SP(Y, B)}{SP(Y, A \to^{c/nc} B)} \qquad \frac{SP(Y, A)}{SP(Y, \forall_x^{c/nc} A)}$$

By simultaneous induction we can finally define formulas $\mathbf{F}$ and predicates $\mathbf{P}$

$$\mathbf{P}(X^{c/nc}) \qquad \frac{\mathbf{F}(A)}{\mathbf{P}(\{\vec{x} \mid A\})} \qquad \frac{\mathbf{F}(A_{i\nu}) \quad SP(X, A_{i\nu}) \quad (\text{for all } i < k,\ \nu < n_i)}{\mathbf{P}\left(\mu_X^{c/nc} \left(\forall_{\vec{x}_i}^{c/nc} \left((A_{i\nu})_{\nu<n_i} \to^{c/nc} X\vec{r}_i\right)\right)_{i<k}\right)}$$

$$\frac{\mathbf{P}(P)}{\mathbf{F}(P\vec{r})} \qquad \frac{\mathbf{F}(A) \quad \mathbf{F}(B)}{\mathbf{F}(A \to^{c/nc} B)} \qquad \frac{\mathbf{F}(A)}{\mathbf{F}(\forall_x^{c/nc} A)}$$

We call

$$I := \mu_X^{c/nc} \left(\forall_{\vec{x}_i}^{c/nc} \left((A_{i\nu})_{\nu<n_i} \to^{c/nc} X\vec{r}_i\right)\right)_{i<k}$$

an *inductively defined predicate*. In case $X \notin PV\left(A_{i\nu}(X)\right)$ we call $A_{i\nu}$ a *parameter premise* and a *recursive premise* otherwise. To every predicate or formula we assign its *final predicate* $fp(\cdot)$

$$fp(X^{c/nc}) := X^{c/nc} \qquad fp(P\vec{r}) = fp(P) \qquad fp(\{\vec{x} \mid A\}) := fp(A)$$

$$fp(A \to^{c/nc} B) := fp(B) \quad fp\left(\mu_X^{c/nc}(\vec{K})\right) := \mu_X^{c/nc}(\vec{K}) \quad fp(\forall_x^{c/nc} A) := fp(A)$$

We call a formula A *non-computational* (or n.c.) if $fp(A)$ is X^{nc} or I^{nc}. Otherwise we call it *computationally relevant* (short c.r.). Furthermore for every predicate or formula A we define its *non-computational variant* A^{nc} by

$$\begin{aligned} \{\vec{x} \mid A\}^{nc} &:= \{\vec{x} \mid A^{nc}\} & (P\vec{r})^{nc} &:= P^{nc}\vec{r} \\ (A \to^{c/nc} B)^{nc} &:= A \to B^{nc} & (\forall_x^{c/nc} A)^{nc} &:= \forall_x A^{nc} \end{aligned}$$

Remark 5. Note that in n.c. parts of proofs, i.e. partial proofs with a non-computational conclusion, we can ignore the decoration. This is due to the definition of computational variables and assumptions, which are empty by definition if the conclusion is n.c..

We now fix the meaning of the inductively defined predicates by adding introduction and elimination axioms.

Definition 14 (Axioms)**.** Let

$$I := \mu_X^{c/nc} \left(\forall_{\vec{x}_i}^{c/nc} \left((A_{i\nu})_{\nu<n_i} \to^{c/nc} X\vec{r}_i \right) \right)_{i<k}$$

a c.r. inductive predicate. Then we have for every $i < k$ an *introduction axiom*

$$I_i^+ : \forall_{\vec{x}_i}^{c/nc} \left((A_{i\nu}(I))_{\nu<n_i} \to^{c/nc} I\vec{r}_i \right)$$

and an *elimination axiom*

$$I^- : \forall_{\vec{x}}^{nc} \left(I\vec{x} \to \left(\forall_{\vec{x}_i}^{c/nc} \left((A_{i\nu}\,(I \cap X))_{\nu<n_i} \to^{c/nc} X\vec{r}_i \right) \right)_{i<k} \to X\vec{x} \right)$$

For a n.c. inductive predicate I^{nc} the axioms $(I^{nc})_i^+$, $(I^{nc})^-$ are formed in the same way, but for the elimination axiom we only allow non-computational predicates to be substituted as competitors. The only exception to this are so-called *one-clause-nc* inductive predicates (i.e. $I := \mu_X\,(K(X))$ with $K(X)$ only with $\to^{nc}$, $\forall^{nc}$). In this case we require no restriction on the competitors.

We define the *Leibniz equality* for type ρ by

$$\mathsf{EqD} := \mu_X\,(\forall_{x^\rho}^{nc} Xxx)$$

To enhance readability we (exclusively) write $x \equiv y$ instead of $\mathsf{EqD}\ x\ y$. Then the introduction and elimination axioms read

$$\begin{aligned} &(\mathsf{EqD})^+ :\ \forall_x^{nc} x \equiv x \\ &(\mathsf{EqD})^- :\ \forall_{x,y}^{nc}\,(x \equiv y \to \forall_x^{nc} Xxx \to Xxy) \end{aligned}$$

We can prove the following compatibility result (justifying the "Leibniz"-part of the name).

Lemma 4 (Compatibility of EqD)**.** $\forall_{x,y}\,(x \equiv y \to A(x) \to A(y))$

Proof. Immediatly from $(\mathsf{EqD})^-$ with $Xxy := A(x) \to A(y)$ as competitor. □

We now define *falsity* by $\mathbf{F} := \mathsf{tt} \equiv \mathsf{ff}$. We can prove the following.

Theorem 2 (Ex-falso-quodlibet)**.** *For formulas A we can derive*

$$\mathbf{F} \to A$$

from assumptions

$\mathrm{EfQ}_Y : \forall_{\vec{x}}\,(\mathbf{F} \to Y\vec{x})$	Y predicate variable with $SP(Y, A)$
$\mathrm{EfQ}_I : \forall_{\vec{x}}\,(\mathbf{F} \to I\vec{x})$	I inductive predicate without nullary clause

Proof. See [11]. □

We now add the other usual logical connectives ($\exists$, $\wedge$, $\vee$) as inductively defined predicates. Due to the decoration we obtain several variants. For the existential quantifier we define

$$ExD_Y := \mu_x\,(\forall_x(Yx \to X)) \qquad ExL_Y := \mu_x\,(\forall_x(Yx \to^{nc} X))$$
$$ExR_Y := \mu_x\,(\forall^{nc}_x(Yx \to X)) \qquad ExNc_Y := \mu^{nc}_x\,(\forall_x(Yx \to X))$$

We use the abbreviations

$$\exists^d_x A := ExD_{\{x|A\}} \qquad \exists^l_x A := ExL_{\{x|A\}}$$
$$\exists^r_x A := ExR_{\{x|A\}} \qquad \exists^{nc}_x A := ExNc_{\{x|A\}}$$

For conjunction we define

$$CapD_{Y,Z} := \mu_X\,(Y \to Z \to X) \qquad CapL_{Y,Z} := \mu_X\,(Y \to Z \to^{nc} X)$$
$$CapR_{Y,Z} := \mu_X\,(Y \to^{nc} Z \to X) \qquad CapNc_{Y,Z} := \mu^{nc}_X\,(Y \to Z \to X)$$

and use the abbreviations

$$A \wedge^d B := CapD_{\{|A\},\{|B\}} \qquad A \wedge^l B := CapL_{\{|A\},\{|B\}}$$
$$A \wedge^r B := CapR_{\{|A\},\{|B\}} \qquad A \wedge^{nc} B := CapNc_{\{|A\},\{|B\}}$$

Finally for disjunction we define

$$CuPD_{Y,Z} := \mu_X\,(Y \to X, Z \to X) \qquad CuPL_{Y,Z} := \mu_X\,(Y \to X, Z \to^{nc} X)$$
$$CuPR_{Y,Z} := \mu_X\,(Y \to^{nc} X, Z \to X) \qquad CuPU_{Y,Z} := \mu_X\,(Y \to^{nc} X, Z \to^{nc} X)$$
$$CuPNc_{Y,Z} := \mu^{nc}_X\,(Y \to X, Z \to X)$$

and the abbreviations

$$A \wedge^d B := CuPD_{\{|A\},\{|B\}} \qquad A \wedge^l B := CuPL_{\{|A\},\{|B\}}$$
$$A \wedge^r B := CuPR_{\{|A\},\{|B\}} \qquad A \wedge^u B := CuPU_{\{|A\},\{|B\}}$$
$$A \wedge^{nc} B := CuPNc_{\{|A\},\{|B\}}$$

Some of the axioms associated to these predicates are given below.

$$(\exists^d)^+ : \forall_x(A \to \exists_x^d A) \qquad (\exists^d)^- : \exists_x^d A \to \forall_x(A \to B) \to B \quad \text{if } x \notin FV(B)$$
$$(\wedge^l)^+ : A \to B \to^{nc} A \wedge^l B \qquad (\wedge^l)^- : A \wedge^l B \to (A \to B \to^{nc} C) \to C$$
$$(\vee^u)_i^+ : A_i \to^{nc} A_1 \vee^u A_2 \qquad (\vee^u)^- : A \vee^u B \to (A \to^{nc} C) \to (B \to^{nc} C) \to C$$
$$(\vee^{nc})_i^+ : A_i \to^{nc} A_1 \vee^u A_2 \qquad (\vee^{nc})^- : A \vee^{nc} B \to (A \to^{nc} C) \to (B \to^{nc} C) \to C \quad \text{if } C \text{ n.c.}$$

Example 7. We give further useful inductively defined predicates:
(1) *Totality*. Let $\iota := \mu_\xi\left((\rho_{i\nu})_{\nu<n_i} \to \xi\right)_{i<k}$ be a finitary algebra without nested parameters. Then we define

$$T_\iota := \mu_X\left(\forall_{\vec{x}}^{nc}\left((T_{\tau_j} x_j^p)_{j<m_i} \to (Xx_j^r)_{j<k_i} \to X\mathsf{C}_i^\iota \vec{x}\right)\right)_{i<k} \tag{2.7}$$

Where $\vec{x}_j^p$ corresponds to the parameter arguments and $\vec{x}_j^r$ to the recursive arguments of the algebra.
As example consider the algebra $\mathbf{P}$. We have the axioms

$$(T_{\mathbf{P}})_0^+ : T_{\mathbf{P}}1 \qquad (T_{\mathbf{P}})_1^+ : \forall_p^{nc}(T_{\mathbf{P}}p \to T_{\mathbf{P}}\mathsf{S}_0 p) \qquad (T_{\mathbf{P}})_2^+ : \forall_p^{nc}(T_{\mathbf{P}}p \to T_{\mathbf{P}}\mathsf{S}_1 p)$$
$$(T_{\mathbf{P}})^- : \forall_p^{nc}(T_{\mathbf{P}}p \to X1 \to \forall_q^{nc}(T_{\mathbf{P}}q \to Xq \to X\mathsf{S}_0 q) \to \forall_q^{nc}(T_{\mathbf{P}}q \to Xq \to X\mathsf{S}_1 q) \to Xp)$$

Note that the elimination axiom corresponds to induction on positive numbers.
(2) *Pointwise equality*. Let ι be as is (1). Then we define

$$=_\iota := \mu_X\left(\forall_{\vec{y}_i,\vec{z}_i}^{nc}\left(\left(y_j^p =_{\vec{\tau}_j} z_j^p\right)_{j<m_i} \to \left(\forall_{\vec{x}_\nu}^{nc}(X(y_{i\nu}^r \vec{x}_\nu)(z_{i\nu}^r \vec{x}_\nu))\right)_{\nu<k_i} \to X(\mathsf{C}_i^\iota \vec{y}_i)(\mathsf{C}_i^\iota \vec{z}_i)\right)\right)_{i<k}$$

For natural numbers we have

$$(=_{\mathbf{N}})_0^+ : 0 =_{\mathbf{N}} 0 \qquad (=_{\mathbf{N}})_1^+ : \forall_{n,m}^{nc}(n =_{\mathbf{N}} m \to \mathsf{S}n =_{\mathbf{N}} \mathsf{S}m)$$
$$(=_{\mathbf{N}})^- : \forall_{n,m}^{nc}(n =_{\mathbf{N}} m \to P0 0 \to \forall_{n,m}^{nc}(n =_{\mathbf{N}} m \to Pnm \to P(\mathsf{S}n)(\mathsf{S}m)) \to Pnm)$$

We now consider coinductive definitions. In general a *coinductively defined predicate* J will be given by one clause

$$\forall_{\vec{x}}^{nc}(J\vec{x} \to \bigvee_{i<k} \exists_{\vec{y}_i}^r \bigwedge_{\nu<n_i} A_{i\nu}(J))$$

and by stipulating that J is the *greatest fix-point* satisfying the clause, i.e. any competitor predicate that satisfies the clause is already contained in J. We extend the definition of

formulas and predicates by

$$\frac{\mathbf{F}(A_{i\nu}) \qquad SP(X, A_{i\nu}) \qquad (\text{for all } i < k \text{ and } \nu < n_i)}{\mathbf{P}\left(\nu_X\left(\forall^{nc}_{\vec{x}}\left(X\vec{x} \to \bigvee_{i<k} \exists^{r}_{\vec{y}_i} \bigwedge_{\nu<n_i} A_{i\nu}(X)\right)\right)\right)}$$

For every coninductively defined predicate J we add the *closure axiom*

$$J^- : \forall^{nc}_{\vec{x}}\Big(J\vec{x} \to \bigvee_{i<k} \exists^{r}_{\vec{y}_i} \bigwedge_{\nu<n_i} A_{i\nu}(J)\Big)$$

and the *greatest-fixed-point* theorem

$$J^+ : \forall^{nc}_{\vec{x}}\Big(P(\vec{x}) \to \forall^{nc}_{\vec{x}}\Big(P\vec{x} \to \bigvee_{i<k} \exists^{r}_{\vec{y}_i} \bigwedge_{\nu<n_i} A_{i\nu}(J \cup P)\Big) \to J\vec{x}\Big)$$

to our theory. The most important case (and the only case dealt with henceforth) for us will be the *dual* ${}^{co}I$ of an inductively defined predicate I. We consider some inductive predicate (where computational $\forall$ is omitted for simplicity)

$$I \coloneqq \mu_X\left(\forall^{nc}_{\vec{x}_i}\left((A_{i\nu})_{\nu<n_i} \to^{c/nc} X\vec{r}_i\right)\right)_{i<k}$$

Then the conjunction of the clauses is equivalent to

$$\forall^{nc}_{\vec{x}}\Big(\bigvee_{i<k} \exists^{r}_{\vec{x}_i}\Big(\vec{x} \equiv \vec{r}_i \wedge \bigwedge_{\nu<n_i} A_{i\nu}(I)\Big) \to I\vec{x}\Big)$$

We define

$${}^{co}I \coloneqq \nu_X\Big(\forall^{nc}_{\vec{x}}\Big(X\vec{x} \to \bigvee_{i<k} \exists^{r}_{\vec{x}_i}\Big(\vec{x} \equiv \vec{r}_i \wedge \bigwedge_{\nu<n_i} A_{i\nu}(X)\Big)\Big)\Big)$$

The *closure* and *greatest-fixed-point* axioms are then given by

$${}^{co}I^- : \forall^{nc}_{\vec{x}}\Big({}^{co}I\vec{x} \to \bigvee_{i<k} \exists^{r}_{\vec{x}_i}\Big(\vec{x} \equiv \vec{r}_i \wedge \bigwedge_{\nu<n_i} A_{i\nu}({}^{co}I)\Big)\Big)$$
$${}^{co}I^+ : \forall^{nc}_{\vec{x}}\Big(X\vec{x} \to \forall^{nc}_{\vec{x}}\Big(X\vec{x} \to \bigvee_{i<k} \exists^{r}_{\vec{x}_i}\Big(\vec{x} \equiv \vec{r}_i \wedge \bigwedge_{\nu<n_i} A_{i\nu}({}^{co}I \cup X)\Big)\Big) \to {}^{co}I\vec{x}\Big)$$

Example 8. (i) Totality for **SD** and **Str** is given by

$$T_{\mathbf{SD}}{-}1 \qquad T_{\mathbf{SD}}0 \qquad T_{\mathbf{SD}}1$$
$$T_{\mathbf{Str}} \coloneqq \mu_X\left(\forall^{nc}_{d,u}\left(T_{\mathbf{SD}}d \to Xu \to X\mathsf{C}_d u\right)\right)$$

Then *cototality* for **Str** is given by the clause

$$\forall^{nc}_{u}\left({}^{co}T_{\mathbf{Str}}u \to \exists^{r}_{d,v}\left(u \equiv \mathsf{C}_d v \wedge^{r} \left(T_{\mathbf{SD}}d \wedge^{d} {}^{co}T_{\mathbf{Str}}v\right)\right)\right)$$

Note that here we can directly recover the notion of cototal ideal as given in definition 6.
(ii) We consider *bisimilarity*, the dual to point-wise equality

$$ {}^{co}{=_\iota} := \nu_X\Big(\forall^{nc}_{\vec{y},\vec{z}}\Big(X\vec{y}\vec{z} \to \bigvee_{i<k} \exists^r_{\vec{x}_1,\vec{x}_2}\Big(\vec{y} \equiv (\mathsf{C}_i\vec{x}_1) \wedge \vec{z} \equiv (\mathsf{C}_i\vec{x}_2) \\ \wedge \vec{x}_1^p =_{\vec{\tau}} \vec{x}_2^p \wedge \bigwedge_{\nu<n_i} \forall^{nc}_{\vec{x}_\nu}\big(X(x^r_{1\nu}\vec{x}_\nu)(x^r_{2\nu}\vec{x}_\nu)\big)\Big)\Big)\Big) $$

E.g. for the algebra **Str** we have

$$ \big({}^{co}{=_{\mathbf{Str}}}\big)^- : \forall^{nc}_{v,w}\Big(v\ {}^{co}{=_{\mathbf{Str}}}\ w \to \exists^r_{v_1,v_2,d_1,d_2}\big(v \equiv (\mathsf{C}_{d_1}v_1) \wedge^{nc} w \equiv (\mathsf{C}_{d_2}v_2) \\ \wedge^l d_1 =_{\mathbf{SD}} d_2 \wedge^d (C_{d_1}v_1)\ {}^{co}{=_{\mathbf{Str}}}\ (C_{d_2}v_2)\big)\Big) $$

$$ \big({}^{co}{=_{\mathbf{Str}}}\big)^+ : \forall^{nc}_{v,w}\Big(Xvw \to \forall^{nc}_{v,w}\Big(Xvw \to \exists^r_{v_1,v_2,d_1,d_2}\Big(v \equiv (\mathsf{C}_{d_1}v_1) \wedge w \equiv (\mathsf{C}_{d_2}v_2) \\ \wedge d_1 =_{\mathbf{SD}} d_2 \wedge \big(\mathsf{C}_{d_1}v_1\ {}^{co}{=_{\mathbf{Str}}}\ \mathsf{C}_{d_2}v_2 \vee X(\mathsf{C}_{d_1}v_1)(\mathsf{C}_{d_2}v_2)\big)\Big)\Big) \to v\ {}^{co}{=_{\mathbf{Str}}}\ w\Big) $$

Remark 6. Another way to introduce predicates and formulas is to drop the distinction between n.c. and c.r. $\forall$, $\to$ and always view $\forall$ as non-computational and $\to$ as computational. This alleviates some troubles arising from the decoration and brings the theory closer to comparable theories while still retaining the exact distinction between n.c. and c.r.. Also we can just use minimal logic without decorations which further simplifies matters. This will be done by passing the distinction between computational and non-computational entirely to the inductively defined predicates. Definition 13 can be adapted to not include $\forall^{nc}$, $\to^{nc}$ and by defining instead

$$ \frac{\mathbf{F}(A_{i\nu}) \qquad SP(X, A_{i\nu}) \qquad (\text{for all } i<k,\ \nu<n_i)}{\mathbf{P}\left(\mu_X^{c/nc}\left(\forall_{\vec{x}_i}\left(\left(A_{i\nu}\right)_{\nu<n_i} \to X\vec{r}_i\right)\right)_{i<k}\right)} $$

and only allowing c.r. (respectively n.c.) predicates to be substituted for c.r. (respectively n.c.) predicate variables. The introduction and elimination axioms will be formed as before, where in case of $I = \mu^{nc}_X(\vec{K})$ we again only allow n.c. predicates and formulas to be substituted in I^-. We can still define variants for the logical connectives, e.g. for conjunction

$$ A \wedge^d B := \mu_X\left(\{|A\} \to \{|B\} \to X\right) \qquad A \wedge^l B := \mu_X\left(\{|A\} \to \{|B\}^{nc} \to X\right) $$
$$ A \wedge^r B := \mu_X\left(\{|A\}^{nc} \to \{|B\} \to X\right) \qquad A \wedge^{nc} B := \mu^{nc}_X\left(\{|A\} \to \{|B\} \to X\right) $$

But since the decoration is determined by whether the formulas involved are n.c. or c.r. the decoration can be dropped in most cases.

2.4 Proof extraction

We will define the method by which we intend to extract programs from proofs. To do this we assign types to formulas in 2.4.1 and define a *realizability interpretation* in 2.4.2. This will be a relation between terms in T^+ and formulas. Finally in 2.4.3 we define the *extracted term* of a derivation as our program extraction method.

2.4.1 Types of formulas

We define the *type* $\tau(A)$ of formulas A. If A is c.r. then $\tau(A)$ tells us what the type of the extracted program of a potential proof will be. To take care of n.c. formulas we introduce the *null-type* $\circ$ and extend the arrow formation of types by the following rules

$$\rho \to \circ := \circ \quad \circ \to \sigma := \sigma$$

Definition 15 (Types of formulas and predicates)**.** Given a global injective assignment of type-variables ξ to c.r. predicate variables X we define the *type* of formulas and predicates, written $\tau(\cdot)$, by

$$\tau(A) := \circ \quad \text{if } A \text{ is n.c.} \qquad \tau(X) := \xi \qquad \tau\left(\{\vec{x}|A\}\right) := \tau(A) \qquad \tau\left(P\vec{r}\right) := \tau(P)$$
$$\tau\left(\mu_X\left(\forall^{nc}_{\vec{x}_i}\forall_{\vec{y}_i}\left(\vec{A}_i \to^{nc} \vec{B}_i \to X\vec{r}_i\right)\right)_{i<k}\right) := \mu_\xi\left(\left(\tau\left(y_{i\nu}\right)\right)_{\nu<m_i} \to \left(\tau\left(B_{i\nu}\right)\right)_{\nu<k_i} \to \xi\right)_{i<k}$$
$$\tau(A \to B) := \tau(A) \to \tau(B) \qquad \tau(A \to^{nc} B) := \tau(B) \qquad \tau\left(\forall^{nc}_x A\right) := \tau(A)$$
$$\tau\left(\forall_{x^\rho} A\right) := \rho \to \tau(A) \qquad \tau(^{co}I) := \tau(I)$$

Where in the definition of $\tau(I)$ we have that $\tau(B_{i\nu})$ is missing in case $B_{i\nu}$ is n.c.. The type associated to an inductively defined predicate I is also called the associated algebra ι_I, i.e. $\tau(I) = \iota_I$.

2.4.2 Realizability

We define the notion of *realizability*. To deal with n.c. formulas we introduce a *nullterm* ϵ as a "realizer". We extend our term application rules by

$$\epsilon t := \epsilon \qquad t\epsilon := t$$

Definition 16 (Realizability)**.** Assume that we have an injective assignment of c.r. predicate variables X to n.c. predicate variables $X^{\mathbf{r}}$. If C is a n.c. predicate or formula then we define $C = C^{\mathbf{r}}(\epsilon)$. Otherwise we assign to c.r. predicates P a n.c. predicate $P^{\mathbf{r}}$. In case of a c.r. formula A we write $z\ \mathbf{r}\ A$ instead of $A^{\mathbf{r}}z$. We define

$$\{\vec{x} \mid A\}^{\mathbf{r}} := \{z, \vec{x} \mid z\ \mathbf{r}\ A\}$$

Let I be a c.r. inductive predicates given by

$$I := \mu_X\left(\forall^{nc}_{\vec{x}_i}\forall_{\vec{y}_i}\left(\left(A_{i\nu}\right)_{\nu<n_i} \to^{nc} \left(B_{i\nu}\right)_{\nu<n_i} \to X\vec{r}_i\right)\right)_{i<k}$$

Then we define its *witnessing predicate* $I^{\mathbf{r}}$ by

$$I^{\mathbf{r}} := \mu^{nc}_{X^{\mathbf{r}}}\left(\forall_{\vec{x}_i,\vec{y}_i,\vec{z}_i}\left((A_{i\nu})_{\nu<n_i} \to (z_{i\nu}\ \mathbf{r}\ B_{i\nu})_{\nu<n_i} \to \mathsf{C}^{\iota_I}_i\vec{y}_i\vec{z}_i\ \mathbf{r}\ X\vec{r}_i\right)\right)_{i<k}$$

Where we assume all $B_{i\nu}$ to be c.r. (in case $B_{i\nu}$ is n.c. then $z_{i\nu}$ and the $z_{i\nu}$ $\mathbf{r}$ -part is omitted). For c.r. formulas we define

$$\begin{aligned}
&z\ \mathbf{r}\ P\vec{r} := P^{\mathbf{r}}(z,\vec{r}) \qquad z\ \mathbf{r}\ (A \to^{nc} B) := A \to z\ \mathbf{r}\ B\\
&z\ \mathbf{r}\ (A \to B) := \begin{cases}\forall_w(w\ \mathbf{r}\ A \to zw\ \mathbf{r}\ B) & \text{if } A \text{ c.r.}\\ A \to z\ \mathbf{r}\ B & \text{if } A \text{ n.c.}\end{cases}\\
&z\ \mathbf{r}\ \forall_x A := \forall_x\,(zx\ \mathbf{r}\ A) \qquad z\ \mathbf{r}\ \forall^{nc}_x A := \forall_x\,(z\ \mathbf{r}\ A)
\end{aligned}$$

The witnessing predicate for the dual ${}^{co}I$ is given by $({}^{co}I)^{\mathbf{r}} := {}^{co}(I^{\mathbf{r}})$, i.e. if

$${}^{co}I := \nu_X\Big(\forall^{nc}_{\vec{x}}\Big(X\vec{x} \to \bigvee_{i<k}\exists^{r}_{\vec{x}_i}\Big(\vec{x} \equiv \vec{r}_i \wedge \bigwedge_{\nu<n_i} A_{i\nu}(X)\Big)\Big)\Big)$$

then

$${}^{co}I^{\mathbf{r}} := \nu_{X^{\mathbf{r}}}\Big(\forall_{\vec{x},\vec{z}}\Big(\vec{z}\ \mathbf{r}\ X\vec{x} \to \bigvee_{i<k}\exists^{r}_{\vec{x}_i,\vec{z}_i}\Big(\vec{x} \equiv \vec{r}_i \wedge \vec{z} \equiv \mathsf{C}_i\vec{z}_i \wedge \bigwedge_{\nu<n_i}(z_{i\nu}\ \mathbf{r}\ A_{i\nu})\Big)\Big)\Big)$$

Remark 7. We motivate the definition for the witnessing predicate $I^{\mathbf{r}}$. As an example we consider totality on $\mathbf{P}$ which has the introduction axioms

$$(T_{\mathbf{P}})^+_0 : T_{\mathbf{P}}1 \qquad (T_{\mathbf{P}})^+_1 : \forall^{nc}_p\,(T_{\mathbf{P}}p \to T_{\mathbf{P}}\mathsf{S}_0p) \qquad (T_{\mathbf{P}})^+_2 : \forall^{nc}_p\,(T_{\mathbf{P}}p \to T_{\mathbf{P}}\mathsf{S}_1p)$$

What is a solution to $T(p)$ if viewed as a computational problem? The only way to arrive at $T(p)$ is by using an introduction axiom. Then a solution to e.g. $T(\mathsf{S}_1\mathsf{S}_0 1)$ should be a generation tree $T(1) - T(\mathsf{S}_0 1) - T(\mathsf{S}_1\mathsf{S}_0 1)$. Or more generally

$$\begin{aligned}
&1 \text{ solves } T(1)\\
&t \text{ solves } T(\mathsf{S}_0p) \leftrightarrow t \cong s - T(\mathsf{S}_0p) \text{ and } s \text{ solves } T(p)\\
&t \text{ solves } T(\mathsf{S}_1p) \leftrightarrow t \cong s - T(\mathsf{S}_1p) \text{ and } s \text{ solves } T(p)
\end{aligned}$$

This directly corresponds to the introduction axioms for $T^{\mathbf{r}}$ which are

$$\begin{aligned}
&1\ \mathbf{r}\ T_{\mathbf{P}}1\\
&\forall_{q,p}\,(q\ \mathbf{r}\ T_{\mathbf{P}}p \to \mathsf{S}_0q\ \mathbf{r}\ \mathsf{S}_0p)\\
&\forall_{q,p}\,(q\ \mathbf{r}\ T_{\mathbf{P}}p \to \mathsf{S}_1q\ \mathbf{r}\ \mathsf{S}_1p)
\end{aligned}$$

Example 9. For the inductive predicate ExD_Y we have

$$\begin{aligned}
&(ExD^{\mathbf{r}}_{Y^{\mathbf{r}}})^+ : \forall_{x,z}\,(z\ \mathbf{r}\ Yx \to \langle x,z\rangle\ \mathbf{r}\ ExD_Y)\\
&(ExD^{\mathbf{r}}_{Y^{\mathbf{r}}})^- : \forall_{p^{\rho\times\tau(Y)}}\,(ExD^{\mathbf{r}}_{Y^{\mathbf{r}}}p \to \forall_{x,z}\,(z\ \mathbf{r}\ Yx \to X\langle x,z\rangle) \to Xp)
\end{aligned}$$

For the witnessing predicates of the logical connectives we can recover some equivalences that are closer to a more intuitive definition of realizability for them. We have the following lemma, giving an alternative way of eliminating or introducing the witnessing predicate of the existential quantifiers.

Lemma 5 ($\exists^{\mathbf{r}}$-Elim/Intro)**.**
a. $\langle x,z\rangle \ \boldsymbol{r}\ \exists^d_x A \leftrightarrow z\ \boldsymbol{r}\ A(x)$ for A c.r.
b. $x\ \boldsymbol{r}\ \exists^l_x A \leftrightarrow A(x)$ for A n.c.
c. $z\ \boldsymbol{r}\ \exists^r_x A \leftrightarrow \exists^{nc}_x(z\ \boldsymbol{r}\ A)$ for A c.r.

Proof. (a). Substituting $Y^{\mathbf{r}} := \{z,x \mid z\ \mathbf{r}\ A(x)\}$ in $(ExD^{\mathbf{r}}_{Y^{\mathbf{r}}})^+$ (see example above) we get

$$\forall_{x,z}\left(z\ \mathbf{r}\ A \to \langle x,z\rangle\ \mathbf{r}\ \exists^d_x A\right)$$

For the other direction we substitute

$$\begin{aligned} Y^{\mathbf{r}} &:= \{z,x \mid z\ \mathbf{r}\ A(x)\} \\ X &:= \{p^{\rho\times\tau(Y)} \mid p_1\ \mathbf{r}\ A(p_0)\} \end{aligned}$$

in $(ExD^{\mathbf{r}}_{Y^{\mathbf{r}}})^-$ to obtain

$$\forall_p\left(p\ \mathbf{r}\ \exists^d_x A \to \forall_{x,z}(z\ \mathbf{r}\ A(x) \to z\ \mathbf{r}\ A(x)) \to p_1\ \mathbf{r}\ A(p_0)\right)$$

and the middle part is provable.
The other cases are proven in a similar fashion. □

Similar result can be obtained for the other connectives, e.g. for conjunction

$$\begin{aligned} &z\ \mathbf{r}\ (A\wedge^l B) \to z\ \mathbf{r}\ A\wedge^{nc} Bj \\ &z\ \mathbf{r}\ A \to B \to z\ \mathbf{r}\ (A\wedge^l B) \end{aligned} \qquad \text{for } A \text{ c.r. } B \text{ n.c.}$$

and for disjunction

$$\begin{aligned} &z\ \mathbf{r}\ A \to \mathsf{InL}(z)\ \mathbf{r}\ (A\vee^d B) \\ &w\ \mathbf{r}\ B \to \mathsf{InR}(w)\ \mathbf{r}\ (A\vee^d B) \qquad\qquad \text{for } A,B \text{ c.r. and } C \text{ n.c.} \\ &u\ \mathbf{r}\ (A\vee^d B) \to \forall_z(z\ \mathbf{r}\ A \to C) \to \forall_w(w\ \mathbf{r}\ B \to C) \to C \end{aligned}$$

Remark 8. Note that lemma 5 and the results about conjunction and disjunction raise an interesting point: While the inductive definition of the witnessing predicate is very useful for the general theory, in practice they are somewhat unwieldy. In many cases there is a more intuitive way of expressing that $z\ \mathbf{r}\ I$. Consider for example the predicate Even defined by

$$\mathsf{Even} := \mu_X\,(X0, \forall^{nc}_n\,(Xn \to X(\mathsf{S}\mathsf{S}n)))$$

with the realizability predicate

$$\mathsf{Even}^{\mathbf{r}} := \mu^{nc}_{X^{\mathbf{r}}}\,(X^{\mathbf{r}}00, \forall_{n,m}\,(X^{\mathbf{r}}nm \to X^{\mathbf{r}}(\mathsf{S}m)(\mathsf{S}\mathsf{S}n)))$$

A more intuitive (and equivalent) way to describe the realizability would be

$$m \ \mathbf{r} \ \mathsf{Even}(n) := T^{nc} m \wedge 2m = n$$

where T^{nc} is the non-computational version of totality.

We axiomatically stipulate that the existence of a realizer of some formula A is equivalent to the formula itself. Intuitively: Solvability of the problem associated to a proposition is equivalent to the proposition being provable.

Definition 17 (Invariance Axioms)**.**

$$\mathrm{Inv}_A : A \leftrightarrow \exists^l_z \ z \ \mathbf{r} \ A \tag{2.8}$$

Lemma 6. *The identity is a realizer of the invariance axioms, i.e.*

$$\lambda_w w \ \mathbf{r} \ (A \to \exists^l_z z \ \mathbf{r} \ A)$$
$$\lambda_w w \ \mathbf{r} \ (\exists^l_z z \ \mathbf{r} \ A \to A)$$

Proof. Unfolding the definition and using Lemma 5 we obtain

$$\forall_w (w \ \mathbf{r} \ A \leftrightarrow \underbrace{w \ \mathbf{r} \ \exists^l_z z \ \mathbf{r} \ A}_{w \ \mathbf{r} \ A})$$

□

Using the Invariance axioms we can actually prove the axiom of choice.

Theorem 3 (Choice)**.** *For $s \in \{l, d\}$ and A, B n.c. in case $s = l$ we can derive*

$$\forall_x \exists^s_y A(x, y) \to \exists^s_f \forall_x A(x, fx)$$

Proof. By Invariance it suffices to find a realizer. In case $s = l$ we claim that the identity is a realizer. Unfolding definitions and using lemma 5 we obtain:

$$\begin{aligned}
&\lambda_w w \ \mathbf{r} \ \left(\forall_x \exists^l_y A(x, y) \to \exists^l_f \forall_x A(x, fx)\right) \\
&\longleftrightarrow \forall_w \left(\forall_x wx \ \mathbf{r} \ \exists^l_y A(x, y) \to w \ \mathbf{r} \ \exists^l_f \forall_x A(x, fx)\right) \\
&\longleftrightarrow \forall_w \left(\forall_x A(x, wx) \to \forall_x A(x, wx)\right)
\end{aligned}$$

In case $s = d$ the realizer is $\lambda_w \langle \lambda_x (wx)_0, \lambda_x (wx)_1 \rangle$. □

Remark 9. Note that this is not surprising if we read the premise, i.e. $\forall_x \exists_y A(x, y)$, constructively. Then our assumption says that for any x we can construct $y(x)$ such that $A(x, y)$ holds. The supposed f is then just given by $\lambda_x y(x)$.

2.4.3 Extracted terms

We now define our proof extraction method, namely the *extracted term* of a derivation. The "algorithm" will essentially work by dropping n.c. parts and substituting meaningful terms for axioms.

Definition 18 (Extracted term)**.** Assume we have a fixed assignment of assumption variables u^A (A c.r.) to object variables z_u of type $\tau(A)$, then we define the *extracted term* $et(\cdot)$ of derivations M^A (A c.r.) by

$$et((\lambda_{u^A} M^B)^{A\to B}) := \begin{cases} \lambda_{z_u}^{\tau(A)} et(M)\,, & \text{if } A \text{ c.r.} \\ et(M)\,, & \text{if } A \text{ n.c.} \end{cases} \qquad et((\lambda_{u^A} M^B)^{A\to^{nc} B}) := et(M)$$

$$et((M^{A\to B} N^A)^B) := \begin{cases} et(M)et(N)\,, & \text{if } A \text{ c.r.} \\ et(M)\,, & \text{if } A \text{ n.c.} \end{cases} \qquad et((M^{A\to^{nc} B} N^A)^B) := et(M)$$

$$et((\lambda_{x^\rho} M^A)^{\forall_x A}) := \lambda_{x^\rho} et(M) \qquad et((\lambda_{x^\rho} M^A)^{\forall^{nc}_x A}) := et(M)$$

$$et((M^{\forall_x A} r)^{A(r)}) := et(M)r \qquad et((M^{\forall^{nc}_x A} r)^{A(r)}) := et(M)$$

For a c.r. inductive predicate I we define

$$et(I_i^+) := C_i^{\iota_I} \qquad et(I^-) := \mathcal{R}_{\iota_I}$$
$$et({}^{co}I^-) := D_{\iota_I} \qquad et({}^{co}I^+) := {}^{co}\mathcal{R}_{\iota_I}$$

In case A is n.c. we define $et(M^A) := \epsilon$.

Remark 10. Following remark 6 we can also develop the theory further without the usage of decorated connectives.
The types of formulas can be adapted by dropping the cases involving decoration and $\forall$ and instead defining

$$\tau(X^{nc}) := \circ \qquad \tau\big(\mu_X^{nc}(\vec{K})\big) := \circ \qquad \tau\left(\forall_x A\right) := \tau(A)$$

The definition of the associated algebra ι_I then simplifies to

$$\tau\big(\mu_X\big(\forall_{\vec{x}_i}\big(\vec{A}_i \to X\vec{r}_i\big)\big)_{i<k}\big) := \mu_\xi\big(\left(\tau\left(A_{i\nu}\right)\right)_{\nu<k_i} \to \xi\big)_{i<k}$$

with $\tau(A_{i\nu})$ missing in case $A_{i\nu}$ is non-computational. The definition of realizability is the same as in definition 16 without the cases for decoration and all-quantifier which is now defined by

$$z\ \mathbf{r}\ \forall_x A := \forall_x(z\ \mathbf{r}\ A)$$

The witnessing predicate of some inductive predicate $I := \mu_X\big(\forall_{\vec{x}_i}\big(\vec{A}_i \to X\vec{r}_i\big)\big)_{i<k}$ is defined by

$$I^{\mathbf{r}} := \mu_{X^{\mathbf{r}}}\left(\forall_{\vec{x}_i,\vec{z}_i}\left(\left(z_{i\nu}\ \mathbf{r}\ A_{i\nu}\right)_{\nu<n_i} \to \mathsf{C}_i^{\iota_I}\vec{z}_i\ \mathbf{r}\ X\vec{r}_i\right)\right)_{i<k}$$

where $z_{iv}\ \mathbf{r}$ is omitted in case A_{iv} is n.c.
Furthermore the definition of the extracted term (for c.r. formulas) simplifies to

$$et(u^A) := z_u$$
$$et(\lambda_{u^A} M^B) := \begin{cases} \lambda_{z_u} et(M) & \text{if } A \text{ is c.r.} \\ et(M) & \text{if } A \text{ is n.c.} \end{cases}$$
$$et(M^{A\to B} N^A) := \begin{cases} et(M)et(N) & \text{if } A \text{ c.r.} \\ et(M) & \text{if } A \text{ is n.c.} \end{cases}$$
$$et(\lambda_x M^A) := et(M)$$

We obtain the same results as above (and below) and the proofs become much simpler due to missing case distinctions.

2.5 Soundness

We prove that the extracted term of a derivation is a realizer, hence justifying the method of program extraction.

Theorem 4 (Soundness)**.** *Let M be a derivation of some c.r. formula A from assumptions $u_i : C_i$ $(i < n)$. Then we can derive $et(M)\ \mathbf{r}\ A$ from assumptions*

$$\begin{cases} v_i : z_{u_i}\ \mathbf{r}\ C_i\ , & \text{if } C_i \text{ c.r.} \\ v_i : C_i\ , & \text{if } C_i \text{ n.c.} \end{cases} \quad (i < n) \tag{2.9}$$

Proof. The proof goes by induction on M.
Case $u : A$. We have $et(M) = z_u$ and $z_u\ \mathbf{r}\ A$ by assumption.
Case $(\lambda_{u^A} M^B)^{A\to B}$. We have to derive

$$et(\lambda_u M)\ \mathbf{r}\ (A \to B) = \begin{cases} \forall_z\,(z\ \mathbf{r}\ A \to (\lambda_{z_u} et(M))z\ \mathbf{r}\ B)\ , & \text{if } A \text{ c.r.} \\ A \to et(M)\ \mathbf{r}\ B\ , & \text{if } A \text{ n.c.} \end{cases}$$

In case A is n.c., we have by induction hypothesis a derivation of $et(M)\ \mathbf{r}\ B$ from assumption $v : A$. We obtain the desired derivation by $(\to)^+$. If A is c.r. we derive

$$\dfrac{\dfrac{\begin{array}{c}[v : z_u\ \mathbf{r}\ A] \\ \vdots\ \text{IH} \\ et(M)\ \mathbf{r}\ B\end{array}}{z_u\ \mathbf{r}\ A \to et(M)\ \mathbf{r}\ B}\ \to^+ v}{\forall_{z_u}(z_u\ \mathbf{r}\ A \to et(M)\ \mathbf{r}\ B)}\ \forall^+ z_u$$

and $et(M) = (\lambda_{z_u} et(M))z$ by $\mapsto_\beta$.
Case $(\lambda_{u^A} M^B)^{A\to^{nc} B}$. We have to derive

$$et(\lambda_u M)\ \mathbf{r}\ (A \to^{nc} B) := A \to et(M)\ \mathbf{r}\ B$$

If A is n.c., then by induction hypothesis we have a derivation of $et(M)\ \mathbf{r}\ B$ from assumption $v : A$. This is what we want. In case A is c.r. we have the following derivation.

$$\dfrac{\dfrac{Inv : A \to \exists^l_z(z\ \mathbf{r}\ A) \qquad [u : A]}{\exists^l_z(z\ \mathbf{r}\ A)}\ {\to}^- \qquad \dfrac{\begin{array}{c}[v : z_u\ \mathbf{r}\ A]\\ \vdots\ \mathrm{IH}\\ et(M)\ \mathbf{r}\ B\end{array}}{z_u\ \mathbf{r}\ A \to^{nc} et(M)\ \mathbf{r}\ B}\ (\to^{nc})^+ v}{et(M)\ \mathbf{r}\ B}\ (\exists^l)^-$$
$$\dfrac{et(M)\ \mathbf{r}\ B}{A \to et(M)\ \mathbf{r}\ B}\ {\to}^+ u$$

Case $\left(M^{A\to B}N^A\right)^B$. We have to derive

$$et(MN)\ \mathbf{r}\ B \coloneqq \begin{cases} et(M)et(N)\ \mathbf{r}\ B\,, & \text{if } A \text{ c.r.}\\ et(M)\ \mathbf{r}\ B\,, & \text{if } A \text{ n.c.}\end{cases}$$

If A is n.c. then we have by assumption derivations of A and $A \to et(M)\ \mathbf{r}\ B$. By $(\to)^-$ we obtain a derivation of $et(M)\ \mathbf{r}\ B$. For A c.r. we do the following

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \mathrm{IH}_2\\ \forall_z(z\ \mathbf{r}\ A \to et(M)z\ \mathbf{r}\ B) \qquad et(N)\end{array}}{et(N)\ \mathbf{r}\ A \to et(M)et(N)\ \mathbf{r}\ B}\ \forall^- \qquad \begin{array}{c}\vdots\ \mathrm{IH}_1\\ et(N)\ \mathbf{r}\ A\end{array}}{et(M)et(N)\ \mathbf{r}\ B}\ {\to}^-$$

Case $\left(M^{A\to^{nc}B}N^A\right)^B$. We have to derive $et(MN)\ \mathbf{r}\ B \coloneqq et(M)\ \mathbf{r}\ B$. In case A n.c. the derivation is the same as above. If A is c.r. then the derivation is as follows

$$\dfrac{\begin{array}{c}\vdots\ \mathrm{IH}_1\\ A \to et(M)\ \mathbf{r}\ B\end{array} \qquad \dfrac{\dfrac{Inv : \forall_z(z\ \mathbf{r}\ A \to A) \qquad et(N)}{et(N)\ \mathbf{r}\ A \to A}\ \forall^- \qquad \begin{array}{c}\vdots\ \mathrm{IH}_2\\ et(N)\ \mathbf{r}\ A\end{array}}{A}\ {\to}^-}{et(M)\ \mathbf{r}\ B}\ {\to}^-$$

Case $(M^{\forall_x A}t)^{A(t)}$. We have to derive $et(Mt)\ \mathbf{r}\ A(t) \coloneqq et(M)t\ \mathbf{r}\ A(t)$. By assumption we have a derivation of $et(M)\ \mathbf{r}\ \forall_x A \coloneqq \forall_x(et(M)x\ \mathbf{r}\ A(x))$. $\forall^-$ with t yields the claim.
Case $(M^{\forall^{nc}_x A}t)^{A(t)}$. We have to derive $et(Mt)\ \mathbf{r}\ A(t) \coloneqq et(M)\ \mathbf{r}\ A(t)$. By induction hypothesis we have a derivation of $\forall_x(et(M)\ \mathbf{r}\ A)$. $\forall^-$ with t again yields the claim.
Case $(\lambda_x M^A)^{\forall_x A}$. We need to derive $et(\lambda_x M)\ \mathbf{r}\ \forall_x A \coloneqq \forall_x(et(M)\ \mathbf{r}\ A)$. By assumption we have a derivation of $et(M)\ \mathbf{r}\ A$, so the claim follows by $(\forall)+$.
Case $(\lambda_x M^A)^{\forall^{nc}_x A}$. We have to derive $et(\lambda_x M)\ \mathbf{r}\ \forall^{nc}_x A \coloneqq \forall_x(et(M)\ \mathbf{r}\ A)$ which we get from our assumption $et(M)\ \mathbf{r}\ A$ and $(\forall)^+$.

Case c Axiom.
Subcase I_i^+. We have to show that $\mathsf{C}_i^{\iota_I}\ \mathbf{r}\ I_i^+$. Let I be a c.r. inductive predicate and let us assume that the clauses have the following form

$$I := \mu_X\left(\forall^{nc}_{\vec{x}_i}\forall_{\vec{y}_i}\left((A_{i\nu})_{\nu<n_i} \to^{nc} (B_{i\nu})_{\nu<n_i} \to X\vec{r}_i\right)\right)_{i<k}$$

with $A_{i\nu}$ n.c. and $B_{i\nu}$ c.r.. Then we have

$$I^{\mathbf{r}} := \mu^{nc}_{X^{\mathbf{r}}}\left(\forall_{\vec{x}_i,\vec{y}_i,\vec{z}_i}\left((A_{i\nu})_{\nu<n_i} \to (z_{i\nu}\ \mathbf{r}\ B_{i\nu})_{\nu<n_i} \to \mathsf{C}_i^{\iota_I}\vec{y}_i\vec{z}_i\ \mathbf{r}\ X\vec{r}_i\right)\right)_{i<k}$$

Unfolding the definitions yields

$$\begin{aligned}&\mathsf{C}_i\ \mathbf{r}\ I_i^+ := C_i\ \mathbf{r}\ \forall^{nc}_{\vec{x}_i}\forall_{\vec{y}_i}\left((A_{i\nu})_{\nu<n_i} \to^{nc} (B_{i\nu})_{\nu<n_i} \to X\vec{r}_i\right) \ =\\ &\forall_{\vec{x}_i,\vec{y}_i,\vec{z}_i}\left((A_{i\nu})_{\nu<n_i} \to (z_{i\nu}\ \mathbf{r}\ B_{i\nu})_{\nu<n_i} \to \mathsf{C}_i^{\iota_I}\vec{y}_i\vec{z}_i\ \mathbf{r}\ X\vec{r}_i\right)\end{aligned}$$

which is exactly the i-th clause of $I^{\mathbf{r}}$.
Subcase I^-. We need to derive $\mathcal{R}_{\iota_I}\ \mathbf{r}\ I^-$, that is

$$\begin{aligned}\mathcal{R}_{\iota_I}\ \mathbf{r}\ \forall^{nc}_{\vec{x},\vec{y}}\Big(I\vec{x}\vec{y} \to \Big(\overbrace{\forall^{nc}_{\vec{x}_i}\forall_{\vec{y}_i}\Big((A_{i\nu})_{\nu<n_i} \to^{nc} (B_{i\nu}(I\cap X))_{\nu<n_i} \to X\vec{r}_i\Big)}^{K_i(I,X)}\Big)_{i<k} \to X\vec{x}\vec{y}\Big)\\ = \ \forall_{\vec{x},\vec{y},z,\vec{w}}\Big(z\ \mathbf{r}\ I\vec{x}\vec{y} \to \vec{w}\ \mathbf{r}\ \vec{K}(I,X) \to \mathcal{R}z\vec{w}\ \mathbf{r}\ X\vec{x}\vec{y}\Big)\end{aligned}$$

Let $\vec{x}$, $\vec{y}$, z with $z\ \mathbf{r}\ I\vec{x}\vec{y}$ and $\vec{w}$ with $w_i\ \mathbf{r}\ K_i(I,X)$ be given. We use the elimination axiom of $I^{\mathbf{r}}$ with competitor $Qz\vec{x}\vec{y} := \mathcal{R}z\vec{w}\ \mathbf{r}\ X\vec{x}\vec{y}$, i.e.

$$\begin{aligned}(I^{\mathbf{r}})^- : z\ \mathbf{r}\ I\vec{x}\vec{y} \to \Big(\forall_{\vec{x}_i,\vec{y}_i,\vec{z}_i}\Big((A_{i\nu})_{\nu<n_i} \to (B^{\mathbf{r}}_{i\nu}(I^{\mathbf{r}}\cap Q)(z_{i\nu}))_{\nu<n_i} \to Q(C_i\vec{y}_i\vec{z}_i)\vec{r}_i\Big)\Big)_{i<k}\\ \to Qz\vec{x}\vec{y}\end{aligned} \tag{2.10}$$

So in order to prove our goal we only need to prove the premises in (2.10). Let $i<k$ and assume $\vec{x}_i$, $\vec{y}_i$, $\vec{z}_i$ with $A_{i\nu}$ and $B^{\mathbf{r}}_{i\nu}(I^{\mathbf{r}}\cap Q)(z_{i\nu})$ for all $\nu<n_i$. We need to show that $Q(C_i\vec{y}_i\vec{z}_i)\vec{r}_i$, i.e.

$$\mathcal{R}(C_i\vec{y}_i\vec{z}_i)\vec{w}\ \mathbf{r}\ X\vec{r}_i \ = \ w_i\vec{y}_i\left(\mathcal{M}^{\iota\to\iota\times\tau}_{\lambda_\alpha\rho_{i\nu}(\alpha)}z_{i\nu}\lambda_z\langle z,\mathcal{R}^\tau_\iota z\vec{w}\rangle\right)_{\nu<n_i}\ \mathbf{r}\ X\vec{r}_i \tag{2.11}$$

by the conversion for $\mathcal{R}$, where $\iota_I := \mu_\xi\left((\tau(y_{i\nu}))_{\nu<m_i} \to (\rho_{i\nu}(\xi))_{\nu<n_i} \to \xi\right)_{i<k}$. By assumption we have $w_i\ \mathbf{r}\ K_i(I,X)$ which unfolds to

$$\forall_{\vec{x}_i,\vec{y}_i,\vec{z}}\left((A_{i\nu})_{\nu<n_i} \to (z_\nu\ \mathbf{r}\ B_{i\nu}(I\cap X))_{\nu<n_i} \to w_i\vec{y}_i\vec{z}\ \mathbf{r}\ X\vec{r}_i\right) \tag{2.12}$$

Substituting $\vec{x}_i$, $\vec{y}_i$ and $\vec{z} := \left(\mathcal{M}^{\iota\to\iota\times\tau}_{\lambda_\alpha\rho_{i\nu}(\alpha)}z_{i\nu}\lambda_z\langle z,\mathcal{R}^\tau_\iota z\vec{w}\rangle\right)_{\nu<n_i}$ the conclusion of (2.12) is exactly (2.11). It remains to prove the premises, i.e $A_{i\nu}$ and $z_\nu\ \mathbf{r}\ B_{i\nu}(I\cap X)$ for all $\nu<n_i$.

We already have $A_{i\nu}$ by assumption. For the other claim we distinguish cases for all $\nu < n_i$.
(i) If $B_{i\nu}$ is a parameter premise then $\rho_{i\nu}$ is a parameter argument and we have $z_\nu = z_{i\nu}$. Now by assumption $B^{\mathbf{r}}_{i\nu}(I^{\mathbf{r}} \cap Q)(z_{i\nu}) = z_\nu \ \mathbf{r}\ B_{i\nu}$ which is what we want.
(ii) If $B_{i\nu}(X) \ = \ C_{i\nu} \to X$ is a recursive premise, then since X is strictly positive, $\rho_{i\nu}$ must have the form $\rho_{i\nu}(\iota) = \sigma_{i\nu} \to \iota$. Hence

$$z_\nu = \lambda_{s^{\sigma_{i\nu}}} \langle z_{i\nu} s, \mathcal{R}(z_{i\nu} s)\vec{w} \rangle$$

Then by assumption we have

$$z_\nu \ \mathbf{r}\ B_{i\nu}(I \cap X) \longleftrightarrow \forall_{\vec{y}_{i\nu},\vec{x}_{i\nu}} \forall_t \left(t \ \mathbf{r}\ C_{i\nu} \to z_{i\nu}\vec{y}_{i\nu} t \ \mathbf{r}\ I\vec{s}_{i\nu} \to \mathcal{R}(z_{i\nu}\vec{y}_{i\nu} t)\vec{w} \ \mathbf{r}\ X\vec{s}_{i\nu} \right)$$

and we need to prove

$$B^{\mathbf{r}}_{i\nu}\left(I^{\mathbf{r}} \cap Q\right)(z_{i\nu}) \longleftrightarrow \forall_{\vec{y}_{i\nu},\vec{x}_{i\nu}} \forall_t \big(C^{\mathbf{r}}_{i\nu}(t) \to I^{\mathbf{r}}(z_{i\nu}\vec{y}_{i\nu} t)\vec{s}_{i\nu} \to \underbrace{Q(z_{i\nu}\vec{y}_{i\nu} t)\vec{s}_{i\nu}}_{\mathcal{R}(z_{i\nu}\vec{y}_{i\nu} t)\vec{w} \ \mathbf{r}\ X\vec{s}_{i\nu}} \big)$$

But they are actually equivalent.
Subcase ${}^{co}I^-$. We need to show that $D \ \mathbf{r}\ {}^{co}I^-$, i.e.

$$\forall_{\vec{x},v} \big(v \ \mathbf{r}\ {}^{co}I\vec{x} \to D_\iota v \ \mathbf{r} \bigvee_{i<k} \exists^r_{\vec{x}_i} \big(\vec{x} \equiv \vec{r}_i \wedge \bigwedge_{\nu<n_i} A_{i\nu}({}^{co}I) \big) \big) \tag{2.13}$$

Assume $\vec{x}$, v and $v \ \mathbf{r}\ {}^{co}I\vec{x}$. By $({}^{co}I^{\mathbf{r}})^-$ there is some j and $\vec{x}_j$, $\vec{z}_j$ such that

$$\vec{x}_j \equiv \vec{r}_j \ \wedge v \equiv C_j\vec{z}_j \wedge \bigwedge_{\nu<n_j} z_{j\nu} \ \mathbf{r}\ A_{j\nu}$$

so that $D_\iota v = \mathrm{Inj}_j \vec{z}_j$ and it remains to prove

$$\vec{z}_j \ \mathbf{r}\ \exists^r_{\vec{x}_j} \big(\vec{x} \equiv \vec{r}_j \wedge \bigwedge_{\nu<n_j} A_{j\nu}({}^{co}I) \big)$$

But by Lemma 5 this is equivalent to

$$\exists^{nc}_{\vec{x}_j} \big(\vec{x} \equiv \vec{r}_j \ \wedge \bigwedge_{\nu<n_j} z_{j\nu} \ \mathbf{r}\ A_{j\nu} \big)$$

which we have by assumption.
Subcase ${}^{co}I^+$. We need to derive ${}^{co}\mathcal{R} \ \mathbf{r}\ ({}^{co}I)^+$. Recall that

$$({}^{co}I)^+ : \forall^{nc}_{\vec{x}} \Big(X\vec{x} \to \forall^{nc}_{\vec{x}} \Big(X\vec{x} \to \bigvee_{i<k} \exists^r_{\vec{x}_i} \Big(\vec{x} \equiv \vec{r}_i \wedge \bigwedge_{\nu<n_i} A_{i\nu} \left({}^{co}I \cup X\right) \Big) \Big) \to {}^{co}I\vec{x} \Big)$$

Our goal unfolds to

$$\forall_{\vec{x},u} \Big(X^{\mathbf{r}} u\vec{x} \to \forall_w \Big(\forall_{\vec{x},u} \Big(X^{\mathbf{r}} u\vec{x} \to wu \ \mathbf{r} \bigvee_{i<k} \exists^r_{\vec{x}_i} \Big(\vec{x} \equiv \vec{r}_i \wedge \bigwedge_{\nu<n_i} A_{i\nu} \left({}^{co}I \cup X\right) \Big) \Big) \\ \to {}^{co}\mathcal{R}uw \ \mathbf{r}\ {}^{co}I\vec{x} \Big) \Big)$$

Assume $\vec{x}$ and u with $X^{\mathbf{r}}u\vec{x}$ and w with

$$\forall_{\vec{x},u}\Big(X^{\mathbf{r}}u\vec{x} \to wu \;\mathbf{r}\; \bigvee_{i<k} \exists^{r}_{\vec{x}_i}\Big(\vec{x} \equiv \vec{r}_i \wedge \bigwedge_{\nu<n_i} A_{i\nu}\,({}^{co}I \cup X)\Big)\Big) \tag{2.14}$$

In order to prove ${}^{co}\mathcal{R}uw \;\mathbf{r}\; {}^{co}I\vec{x}$ we use $({}^{co}I^{\mathbf{r}})^{+}$ with the competitor

$$Qt\vec{x} := \exists_v\,(t \equiv {}^{co}\mathcal{R}vw \wedge X^{\mathbf{r}}v\vec{x})$$

Thus we need to prove the premises in $({}^{co}I^{\mathbf{r}})^{+}$, which are

$$Q\,({}^{co}\mathcal{R}uw)\,\vec{x} := \exists_v\,({}^{co}\mathcal{R}uw \equiv {}^{co}\mathcal{R}vw \wedge X^{\mathbf{r}}v\vec{x}) \tag{2.15}$$

$$\forall_{\vec{x},t}\Big(Qt\vec{x} \to \bigvee_{i<k} \exists^{nc}_{\vec{y},\vec{z}}\Big(t \equiv \mathsf{C}^{\iota}_i\vec{z} \wedge \bigwedge_{\nu<n_i} A^{\mathbf{r}}_{i\nu}\,({}^{co}I^{\mathbf{r}} \cup Q)\,z_\nu\Big)\Big) \tag{2.16}$$

We have (2.15) by $(\exists)^{+}$ with u, reflexivity of $\equiv$ and the assumption $X^{\mathbf{r}}u\vec{x}$. For (2.16) assume $\vec{x}$ and $\tilde{t}$ with $Q\tilde{t}\vec{x}$. From $Q\tilde{t}\vec{x}$ we get a $\tilde{v}$ with

$$\tilde{t} \equiv {}^{co}\mathcal{R}\tilde{v}w \wedge X^{\mathbf{r}}\tilde{v}\vec{x}$$

Using $(\forall)^{-}$ twice with $\vec{x}$, $\tilde{v}$ on (2.14) and $(\to)^{-}$ with $X^{\mathbf{r}}\tilde{v}\vec{x}$ yields

$$w\tilde{v} \;\mathbf{r}\; \bigvee_{i<k} \exists^{r}_{\vec{x}_i}\Big(\vec{x} \equiv \vec{r}_i \wedge \bigwedge_{\nu<n_i} A_{i\nu}\,({}^{co}I \cup X)\Big) \tag{2.17}$$

Now we have for some j that $w\tilde{v} = \mathrm{Inj}_j\vec{z}_j$, hence from (2.17) with lemma 5 and similar statements for $\wedge$ and $\vee$ we get

$$\exists^{nc}_{\vec{x}_j}\Big(\vec{x} \equiv \vec{r}_j \wedge \bigwedge_{\nu<n_j} z_{j\nu} \;\mathbf{r}\; A_{j\nu}\,({}^{co}I \cup X)\Big)$$

We use $(\exists)^{-}$ to obtain $\vec{x}_j$ which satisfies the kernel, i.e. $\vec{x} \equiv \vec{r}_j$ and for all $\nu < n_j$

$$z_{j\nu} \;\mathbf{r}\; A_{j\nu}\,({}^{co}I \cup X)\,\vec{x}_j \tag{2.18}$$

We prove

$$\exists^{nc}_{\vec{y},\vec{z}}\Big(\tilde{t} \equiv \mathsf{C}^{\iota}_j\vec{z} \wedge \bigwedge_{\nu<n_j} A^{\mathbf{r}}_{j\nu}\,({}^{co}I^{\mathbf{r}} \cup Q)\,z_j\vec{y}\Big) \tag{2.19}$$

which yields the claim by applications of $(\vee)^{+}$. By the corecursion conversion we have

$${}^{co}\mathcal{R}\tilde{v}w \;=\; \mathsf{C}^{\iota}_j\Big(\underbrace{\mathcal{M}^{\iota+\tau\to\iota}_{\lambda_\alpha\rho_{j\nu}(\alpha)} z_{j\nu}[id, {}^{co}\mathcal{R}\cdot w]}_{=:z_\nu}\Big)_{\nu<n_j}$$

We use $(\exists)^{+}$ with $\vec{x}_j$ and $\vec{z}$ in order to prove (2.19). Thus we need to prove $\tilde{t} \equiv \mathsf{C}\vec{z}$, which we have by definition of $\vec{z}$ and transitivity of EqD (Note that by assumption $\tilde{t} \equiv {}^{co}\mathcal{R}\tilde{v}w$). Further we need to prove for all $\nu < n_j$

$$A^{\mathbf{r}}_{j\nu}({}^{co}I^{\mathbf{r}} \cup Q)z_\nu\vec{x}_j$$

We get this by a case-distinction on $A_{j\nu}$. In case $A_{j\nu}(X) = A_{j\nu}$ is a parameter premise then $z_\nu = z_{j\nu}$ by the map-operator conversion. Since X doesn't occur in $A_{j\nu}(X)$ we simply get

$$z_{j\nu}\ \mathbf{r}\ A_{j\nu}({}^{co}I \cup X) \leftrightarrow A^{\mathbf{r}}_{j\nu}({}^{co}I^{\mathbf{r}} \cup Q)z_\nu$$

Otherwise $A_{j\nu}(X) = C_{j\nu} \to X$ is a recursive premise and we have

$$z_\nu = \lambda_s([id, {}^{co}\mathcal{R} \cdot\ w]z_{j\nu}s) \tag{2.20}$$

Then by definition of realizability our assumption is

$$z_{j\nu}\ \mathbf{r}\ A_{j\nu}({}^{co}I \cup X) = \forall_s \left(s\ \mathbf{r}\ C_{j\nu} \to z_{j\nu}s\ \mathbf{r}\ ({}^{co}I \vee X)\right) \tag{2.21}$$

Similarly our goal unfolds to

$$A^{\mathbf{r}}_{j\nu}({}^{co}I \cup Q)z_\nu = \forall_s \left(C^{\mathbf{r}}_{j\nu}(s) \to {}^{co}I^{\mathbf{r}}(z_\nu s)\vec{x}_j \vee Q(z_\nu s)\vec{x}_j\right) \tag{2.22}$$

Assume s with $C^{\mathbf{r}}_{j\nu}(s)$. In case $z_{j\nu}s = \mathsf{InL}\ t_0$ we have by assumption, i.e. (2.21) that $z_{j\nu}s\ \mathbf{r}\ ({}^{co}I \vee X)$ and hence $u\ \mathbf{r}\ {}^{co}I$. By $(\vee)^+_0$ we obtain

$${}^{co}I^{\mathbf{r}}([id, {}^{co}\mathcal{R} \cdot\ w]\mathsf{InL}\ t_0) \vee Q([id, {}^{co}\mathcal{R} \cdot\ w]\mathsf{InL}\ t_0)$$

In case $z_{j\nu}s = \mathsf{InR}\ u$ we have from (2.21) that $X^{\mathbf{r}}u\vec{x}_j$ and we get

$$Q({}^{co}\mathcal{R}uw)\vec{x}_j := \exists_v \left({}^{co}\mathcal{R}uw \equiv {}^{co}\mathcal{R}vw \wedge X^{\mathbf{r}}v\vec{x}_j\right)$$

by using $(\exists)^+$ with u and reflexivity of EqD. We use $(\vee)^+_1$ to obtain our goal.
Subcase Inv_A. See lemma 6. □

Remark 11. (i) In case A is n.c. and M^A then $et(M)\ \mathbf{r}\ A = A$ and by invariance for c.r. assumptions $u : C$ we have $z_u\ \mathbf{r}\ C \to C$, so the above theorem also holds for n.c. formulas.
(ii) Note that in the case for $({}^{co}I)^+$ above only the reflexivity and transitivity of EqD where used. Given some predicate I of arity σ we can actually define the companion ${}^{co}I$ w.r.t. any reflexive and transitive relation on σ and the theorem above still holds.
(iii) Also note that the soundness-proof for $({}^{co}I)^+$ as stated above is <u>not</u> correct. In two places an argument of the form

$$w^{\sigma+\tau}\ \mathbf{r}\ A \vee B \Rightarrow w \equiv \mathsf{InL}\ S \vee w \equiv \mathsf{InR}\ T$$

was used. This is not allowed in our setting since t is in general a partial variable. This trouble can be alleviated if we require realizers to be *cototal*. In that case we get a case-distinction as above from the cototality-clause for the type $\sigma + \tau$.

2.6 Totality

We briefly discuss totality predicates. Totality was already defined for finitary algebras in example 7. For arrow types the definition is given by

$$T_{\rho\to\sigma} := \mu_X \left(\forall^{nc}_f \left(\forall^{nc}_x \left(T_\rho x \to T_\sigma f x\right) \to Xf\right)\right) \tag{2.23}$$

The introduction and elimination axioms are then simply given by

$$\forall^{nc}_f \left(\forall^{nc}_x \left(T_\rho x \to T_\sigma f x\right) \to T_{\rho\to\sigma} f\right)$$
$$\forall^{nc}_f \left(T_{\rho\to\sigma} f \to \forall^{nc}_x \left(T_\rho x \to T_\sigma f x\right)\right)$$

Since in many practical cases we only work with the total part of an algebra we axiomatically introduce abbreviation.

Definition 19 (AllTotalIntro/Elim)**.** We add the following axioms to our theory.

$$(\forall T)^+ : \forall^{nc}_{\hat{x}} \left(T\hat{x} \to A\hat{x}\right) \to \forall_x A$$
$$(\forall T)^- : \forall_x A \to \forall^{nc}_{\hat{x}} \left(T\hat{x} \to A\hat{x}\right)$$

Remark 12. (i) We strictly differentiate between partial variables $\hat{x}$, which will be written with a hat, and total variables x without hat.
(ii) Similarly we axiomatically define total versions of $\exists$, e.g. for A n.c.

$$\exists^l_x A(x) \leftrightarrow \exists^r_{\hat{x}} \left(T(\hat{x}) \wedge^l A(\hat{x})\right)$$

Similarly we could also define $\exists^l_x$ by

$$\exists^l_x A(x) \leftrightarrow \exists^l_{\hat{x}} \left(T^{nc}(\hat{x}) \wedge^{nc} A(\hat{x})\right)$$

Note that at least for finitary algebras both versions are provably equivalent.
(iii) Since we added new axioms we also need to extend the soundness-proof to these. One can easily proof that the identity is a realizer of $(\forall T)^\pm$ in case of a finitary algebra. Furthermore one can prove soundness for arrow types built up from finitary algebras under the additional assumptions

$$\mathrm{Ext}_{\hat{f},\hat{g}} : \ \forall_{\hat{x}}(T^{nc}\hat{x} \to \hat{f}\hat{x} \equiv \hat{g}\hat{x}) \to \hat{f} \equiv \hat{g}$$

It is rather unfortunate that the soundness of the $(\forall T)^{+/-}$-axioms is not easily provable and it will possibly lead to a different definition of totality. In the Implementation in Minlog this is at the moment overcome by axiomatically stipulating the correctness of the $(\forall T)^\pm$-axioms. In any case, the realizer should just be the identity.

2.7 Minlog

The theory above has in most parts been implemented in the proof-assistant Minlog, see `http://www.mathematik.uni-muenchen.de/~logik/minlog/index.php` for more information (also see the minlog reference [13]). Minlog is written in the scheme-language and commands can be directly parsed to a scheme session, it is however beneficial to use an editor (like emacs, see the installation instructions on the homepage).
Proofs are built by backward chaining. Some important commands to build proofs are described below.
set-goal. Expects a formula, which is then set as the new goal.
assume. Expects variable or assumption names. Moves universally quantified variables or hypotheses into the context. Corresponds to $\forall$, $\to$ introduction
use. Expects either a hypothesis from the context, a formula or a theorem name. The premises or the new formula will be the new goal. Corresponds to $\to$ elimination
intro. Expects a number i as argument and an inductively defined predicate I as goal. It will use the i-th introduction axiom of I via the use-command. Alternatively it expects a term and an existence statement as goal. In this case it will use the elimination-axiom of $\exists$ instantiated with the term.
assert. Expects some formula `"A"`. If `"B"` is the goal, then the new goals will be `"A"` and `"A->B"`.
ng. Normalizes all terms in the goal and the context. `"(ng #t)"` will only normalize the goal.
split. Expects a conjunction `"A and B"` as goal, which is replaced by the new goals `"A"` and `"B"`.
elim. Expects an inductively defined predicate I as argument. It will use the elimination axiom of I with the current goal as competitor.
coind. Expects some formula A and a coinductively defined predicate ${}^{co}I$ as goal. It will use the greatest-fixed-point axiom of ${}^{co}I$ with competitor A.
cdp. Checks and displays the current proof.
save. Expects a name and a finished proof. Saves the proof under the name.
proof-to-extracted-term. Generates the extracted term of the current proof.
proof-to-soundness-proof. Generates the soundness-proof corresponding to the current proof.
The generation of the soundness-proofs is according to theorem 4. The algorithm to generate such soundness-proofs was explicitly given in the proof. Since the proof is by induction, the algorithm will work by recursion on the structure of derivations. The cases for the logical rules are direct, consider e.g. the case $N := (\lambda_{u^A} M^B)^{A\to B}$ for A c.r.. Then

$$\text{proof-to-soundness-proof}(N) \;=\; \lambda_{z_u}\lambda_v(\text{proof-to-soundness-proof}(M))$$

where $v : z_u \;\mathbf{r}\; A$. The cases for the introduction respectively elimination axioms are more involved, but again explicitly given in the proof.

3 Examples from constructive analysis

3.1 Principle of least upper bound

3.1.1 Notes on constructive analysis

Following [12], [2] we can develop constructive analysis. We view real numbers as Cauchy sequences of rational numbers with a concrete modulus of convergence.

Definition 20 (Real numbers)**.** A *real number* x is a pair $((a_n)_n, M)$ consisting of a sequence of rational numbers a_n and a map $M : \mathbb{Z}^+ \to \mathbb{N}$ such that

$$\forall_p \left(Mp \leq n, m \to |a_n - a_m| \leq \frac{1}{2^p} \right) \tag{3.1}$$

and M is weakly increasing (i.e. $p \leq q \to Mp \leq Mq$).
Equality of real numbers $x = ((a_n), M)$, $y = ((b_n), M)$, written $x = y$, is defined by

$$\forall_p \left(\left| a_{M(p+1)} - b_{M(p+1)} \right| \leq \frac{1}{2^p} \right) \tag{3.2}$$

It is easy to prove that the relation defined by 3.2 is an equivalence relation on the set of real numbers. Like in [12] we can define rational sequences $x + y$, $x - y$, xy, $\frac{1}{x}$, $|x|$ and prove that they fulfill the condition 3.1 as well as the properties of a field. Furthermore we define the notions of positivity and non-negativity for reals and, with these, relations $\leq$ and $<$.

Definition 21. Let $x = ((a_n), M)$. We say that x is *non-negative*, written $x \in \mathbb{R}^{0+}$, if

$$\forall_p \left(-\frac{1}{2^p} \leq a_{M(p+1)} \right) \tag{3.3}$$

We say that x is (p-)*positive*, written $x \in \mathbb{R}^+$ or $x \in_p \mathbb{R}^+$if

$$\frac{1}{2^p} \leq a_{M(p+1)} \tag{3.4}$$

Now let $y = ((b_n), N)$, then we write $x \leq y$ for $y - x \in \mathbb{R}^{0+}$ and $x < y$ (or $x <_p y$) for $y - x \in \mathbb{R}^+$. Then $\leq$ and $<$ are order relations on the real numbers.

We will need the following property (see [12]).

Lemma 7 (Approximate Split)**.** $\forall_{x,y,z} (x < y \to z \leq y \vee x \leq z)$ □

Remark 13. Note that $\leq$ is not decidable, i.e. we cannot prove for arbitrary x, y that $x \leq y \vee y \leq x$. We are however able to compare reals to intervals, see the lemma above.

3.1.2 Principle of least upper bound (LUB)

We want to prove that any bounded set of reals has a supremum. Classically this can be proven, but in our constructive setting we need an additional condition, namely that the set in question is *located from above*. We will follow [6], also see [2].
We will need the following definitions.

Definition 22 (Upper bound)**.** A real number y is called *upper bound* of the set $\mathbf{S} \subset \mathbb{R}$ if

$$\forall_x (x \in \mathbf{S} \to x \leq y) \tag{3.5}$$

Definition 23 (Supremum)**.** A real number y is called *supremum* of a set $\mathbf{S} \subset \mathbb{R}$, written $y = Sup\ \mathbf{S}$, if

$$\forall_x (x \in \mathbf{S} \to x \leq y) \wedge \forall_a (a < y \to \exists_x (x \in \mathbf{S} \wedge a \leq x)) \tag{3.6}$$

i.e. y is the least upper bound.

Definition 24 (Located from above)**.** A set $\mathbf{S} \subset \mathbb{R}$ is called *located from above* if

$$\forall_{a,b} (a < b \to \forall_x (x \in \mathbf{S} \to x \leq b) \vee \exists_x (x \in \mathbf{S} \wedge a \leq x)) \tag{3.7}$$

Theorem 5. *Let $\mathbf{S} \subset \mathbb{R}$ be an inhabited set with an upper bound. Then S has a supremum if and only if it is located from above.*

Proof. Assume $\mathbf{S}$ has a supremum y. We have to show that $\mathbf{S}$ is located from above. Let a, b with $a < b$ be given. By lemma 7 we get $y \leq b \vee a \leq y$. In the first case we have for any $x \in S$ that $x \leq y$, so $x \leq b$ by transitivity. In the second case we have $y \in \mathbf{S} \wedge a \leq y$ so in particular $\exists_x (x \in \mathbf{S} \wedge a \leq x)$.

Now Let $x_0 \in \mathbf{S}$, $x_0 = \left((a_n)_{n \in \mathbb{N}}, M\right)$ and b an upper bound. Assume that $\mathbf{S}$ is located from above. We need to construct a Supremum for the set $\mathbf{S}$. We inductively define two sequences of rational numbers (c_n) and (d_n), such that for all $n \in \mathbb{N}$

$$c_0 \leq c_1 \cdots \leq c_n < d_n \leq \cdots \leq d_1 \leq d_0 \tag{3.8}$$
$$\forall_x (x \in \mathbf{S} \to x \leq d_n) \wedge \exists_x (x \in \mathbf{S} \wedge c_n \leq x) \tag{3.9}$$
$$d_n - c_n \leq (2/3)^n (d_0 - c_0) \tag{3.10}$$

The initial terms are given by

$$c_0 = a_{M(1)} - 1, \quad d_0 = b$$

which clearly satisfy the three conditions above. Now suppose the first n terms of the sequences are constructed, then let $c = c_n + \frac{1}{3}(d_n - c_n)$, $d = c_n + \frac{2}{3}(d_n - c_n)$. We have $c < d$ and since S is located above we obtain

$$\forall_x (x \in \mathbf{S} \to x \leq d) \vee \exists_x (x \in \mathbf{S} \wedge c \leq x)$$

Case $\forall_x(x \in \mathbf{S} \to x \leq d)$.
Let $c_{n+1} = c_n$ and $d_{n+1} = d$. The conditions (3.8) and (3.10) are satisfied by definition. Now by induction hypothesis we have $\exists_x(x \in \mathbf{S} \wedge c_n \leq x)$ and $c_n = c_{n+1}$, so we also have (3.9).
Case $\exists_x(x \in \mathbf{S} \wedge c \leq x)$.
Let $c_{n+1} = c$ and $d_{n+1} = d_n$. The conditions (3.8) and (3.10) are again satisfied by definition. By induction hypothesis we have $\forall_x(x \in \mathbf{S} \to x \leq d_n)$ and $d_{n+1} = d_n$ so (3.9) again holds.
Now by an elementary property of rational numbers there exists positive p_0 such that $d_0 - c_0 \leq 2^{p_0}$. Let $M = \lambda_p\,(2(p + p_0))$ then $((c_n), M)$ and $((d_n), M)$ are real numbers. To see this, let $n, m \geq 2(p + p_0)$ and assume wlog $n \leq m$. Then

$$|c_m - c_n| \leq |d_n - c_n| \leq \left(\frac{2}{3}\right)^n 2^{p_0} = \frac{2^{n+p0}}{3^n}$$

Using the elementary inequality

$$\forall_{n,m}\left(2m \leq n \to 2^{n+m} \leq 3^n\right)$$

with $n \geq 2\,(p + p_0)$ we obtain $2^{n+p+p_0} \leq 3^n$ and

$$|c_m - c_n| \leq \frac{1}{2^p}$$

A similar argument yields that $((d_n), M)$ is a real number.
Condition (3.10), by some calculation, implies that we actually have $((d_n), M) = ((c_n), M)$.
Let $x = ((d_n), M)$. We show that x is a Supremum of the set $\mathbf{S}$. For the first part of the conjunction let $y \in \mathbf{S}$. By the left part of the conjunction in (3.9) we get $y \leq d_n$ for all n, which implies $y \leq x$.
For the second part let a, p be given with $a <_p x$, i.e. $\frac{1}{2^p} \leq c_{M(p+2)} - a$. The latter implies that $a \leq c_n$ for some n, so that by the right part of the conjunction (3.9) we get $\tilde{x}$ with $\tilde{x} \in \mathbf{S} \wedge c_n \leq \tilde{x}$. Transitivity of $\leq$ implies $a \leq \tilde{x}$. □

3.1.3 Implementation and soundness

We now discuss the implementation of the theorem above in the proof assistant Minlog. We use the libraries `nat.scm`, `pos.scm`, `int.scm`, `rat.scm` and `rea.scm` from `/git/minlog/lib/`. The other relevant files for the implementation are

`lub.scm` `lub_lemma.scm` `lub_sound.scm` `lub_sqrt.scm`

which are located in the `scm` folder, on the disc supplied with this document, together with a copy of Minlog.
We start by loading the file `lub_lemma.scm` which contains additional theorems, mostly concerning real numbers, and `lub_sound` which contains some propositions needed for

the soundness proof (note that the soundness of the `ExLTotalIntro` axiom is stipulated axiomatically here, with the identity as a realizer).
We will take a look at code snippets from `lub.scm` which will be framed and output from the minlog-session which will not be framed. It always starts with a `">"` followed by some command and the output of the command on the next line.
Note that real numbers in Minlog are initially defined as the algebra

$$\text{Reals} \qquad \mathbf{R} := \mu_\xi\,((\mathbf{N} \to \mathbf{Q}) \to (\mathbf{P} \to \mathbf{N}) \to \xi) \qquad \text{RealConstr}$$

i.e. as arbitrary pairs of rational sequences and moduli. The constriction to actual reals is then realized by the predicate `Real` which is defined according to definition 20.
The proof is separated into several pieces. The construction of the rational sequences will be separated into the theorems `"Init"` and `"Step"`, where `"Init"` gives the start of the inductive definition and `"Step"` gives the inductive construction. The Set `"S"` is added as a predicate variable with arity `"rea"` using `"(add-pvar-name ...)"`. In the formulas `"S"` is assumed to be non-computational, which is denoted by the hat behind it. Note that for simplicity the bound of the set is assumed to be rational.
We first look at the theorem `"Init"`. Given an inhabitant and a bound of the set it asserts the existence of the initial interval.

```
(add-pvar-name "S" (make-arity (py "rea")))

;;Init
(set-goal
"all x,b(Real x -> S^ x --> all y(Real y -> S^ y -> y<<=b) ->
            exd c exl d(c<d andi
                                all y(Real y -> S^ y -> y<<=d) andi
                                exnc y(Real y andnc S^ y andnc c<<=y)))")
(cases)
(assume "as" "M" "b" "Rx" "Sx" "bBd")
(intro 0 (pt "as(M 1)-1"))
(intro 0 (pt "b"))

                                                ⋮

(save "Init")
(define sproof (proof-to-soundness-proof))
(add-theorem "InitSound" sproof)
```

After assuming the context, the start of the inductive definition is directly given by the `"(intro ...)"` command. The rest of the proof is straightforward. We can take a look at the (normalized) extracted term of the proof.

```
> (define eterm (proof-to-extracted-term))
> (define neterm (rename-variables (nt eterm)))
> (pp neterm)
[x,a][if x ([as,M]as(M 1)+IntN 1 pair a)]
```

Which is, as expected, exactly the same as in the paper proof. Furthermore we generate the soundness proof using `(proof-to-soundness-proof)` and add a theorem `"InitSound"`. We can take a look at the soundness-formula:

```
> (pp "InitSound")
all x,b(Real x -> S^ x -> all y(Real y -> S^ y -> y<<=b) ->
            (ExDTMR (cterm (a^,c)
                  (ExLTMR (cterm (d)
                        c<d andnc
                        all y(Real y -> S^ y -> y<<=d) andnc
                        exnc y(Real y andnc S^ y andnc c<<=y)))
                  a^))
            ([if x ([as,M,b0]as(M 1)-1 pair([a^]a^)b0)]b))
```

It is somewhat unreadable due to the witnessing predicates for the existence quantifiers. In the proof of LUB later we do not intend to use dependent choice to construct the two sequences of rational numbers, rather we directly introduce them using the recursion operator and the extracted terms of `"Init" "Step"`. For ease of use we prove different versions of the soundness proofs without the realizability predicates. For `"Init"` this looks like the following.

```
(set-goal "all x,b(Real x -> S^ x -> all y(Real y -> S^ y -> y<<=b) ->
                        lft (cInit x b)<rht (cInit x b) andnc
                        all y(Real y -> S^ y -> y<<= rht (cInit x b)) andnc
                        exnc y(Real y andnc S^ y andnc lft (cInit x b)<<=y))")
(cases)
(assume "as" "M" "b1" "Rx1" "Sx1" "b1Bd")

                                          ⋮

(use "InitSound")
(auto)
(save "InitReal")
```

The proof makes use of lemma 5, the corresponding theorems in Minlog are called `"ExLTMRElim"` and `"ExLDMRElim"`.
For convenience we add some variable names, namely `"ola"` for variables of type `"rat=>rat=>boole"`, `"cd"` for type `"rat yprod rat"` and finally `"cds"` for the type `"nat=>rat yprod rat"`. Note that for pairs, the projections in Minlog are called `"rht"` and `"lft"`. Now we proceed with the theorem `"Step"`. It asserts the existence of the next interval, given the previous one.

```
;;Step
(set-goal "all c,d(c<d -> all x(Real x -> S^ x -> x<<=d) oru
                                  exnc x(Real x andnc S^ x andnc c <<=x)) ->
                  all cd(lft cd<rht cd -> all x(Real x -> S^ x -> x<<=rht cd) ->
                        exnc x(Real x andnc S^ x andnc lft cd<<=x) ->
                              exl cd1(all x(Real x -> S^ x -> x<<=rht cd1) andi
                              exnc x(Real x andnc S^ x andnc lft cd1<<=x) andi
                              lft cd<=lft cd1 andi
                              rht cd1<=rht cd andi
```

```
                              rht cd1-lft cd1==(2#3)*(rht cd-lft cd) andnc
                              lft cd1 < rht cd1))")
(assume "OLA" "cd" "c<d" "S<=d" "c<=S")

                                                  ⋮

(save "Step")
(define sproof (proof-to-soundness-proof(theorem-name-to-proof "Step")))
(animate "Step")
(save "StepSound" sproof)
```

Again we automatically generate a soundness-proof and add a theorem `"StepSound"`. Now as for `"Init"` we prove another version, eliminating the witnessing predicate for $\exists$, the corresponding theorem is called `"StepReal"`.
The last auxiliary statement before the actual proof of LUB is `"SeqProps"` which states that rational sequences with the properties (3.8)-(3.10) are equal real numbers. The proof is quite long, but doesn't differ from the paper proof.

```
;;SeqProps
(set-goal
 "all cs,ds,m(all n(cs n<ds n andnc
                        cs n<=cs(Succ n) andnc
                        ds(Succ n)<=ds n andnc
                        ds(Succ n)-cs(Succ n)==(2#3)*(ds n-cs n)) ->
            ds Zero-cs Zero<=2**m ->
                  Real(RealConstr cs([q]2*(m+(PosToNat q)))) andnc
                  Real(RealConstr ds([q]2*(m+(PosToNat q)))) andnc
                  RealConstr cs([q]2*(m+(PosToNat q)))===
                  RealConstr ds([q]2*(m+(PosToNat q))))")
(assume "cs" "ds" "m0" "SeqProp" "InitBound")

                                                  ⋮

(save "SeqProps")
```

Before looking at the proof of LUB let us consider how we want to introduce the rational sequences in the proof. We have the extracted terms `"cInit" "cStep"`. They have the following types.

```
> (pp (term-to-type (pt "cInit")))
rea=>rat=>rat yprod rat
> (pp (term-to-type (pt "cStep")))
(rat=>rat=>boole)=>rat yprod rat=>rat yprod rat
```

Furthermore the type of the formula `"OLA"`, expressing that the set is located from above, is given by

```
> (pp (formula-to-et-type (pf "all c,d(c<d ->
      all x(Real x -> S^ x -> x<<=d) oru
      exnc x(Real x andnc S^ x andnc c<<=x))")))
rat=>rat=>boole
```

Now if "ola" is a realizer of "OLA", "x" is an inhabitant and "b" and upper bound, then from the theorems "Init/StepReal" we know that cInit x b and "cStep ola cd" are according to the conditions (3.8)-(3.10).
Now we want to use the recursion operator $\mathcal{R}_{\mathbf{N}}^{\mathbf{Q}\times\mathbf{Q}}$, recall that its type is given by

$$\mathcal{R}_{\mathbf{N}}^{\mathbf{Q}\times\mathbf{Q}} : \mathbf{N} \to \mathbf{Q}\times\mathbf{Q} \to (\mathbf{N} \to \mathbf{Q}\times\mathbf{Q} \to \mathbf{Q}\times\mathbf{Q}) \to \mathbf{Q}\times\mathbf{Q}$$

Given a (total!) realizer "ola" of "OLA" we define the rational sequences by:

```
[n](Rec nat=>rat yprod rat)n(cInit x b)([n](cStep ola))
```

We finally turn to the proof of LUB:

```
;;Lub
(set-goal "all b,x(Real x -> S^ x --> all y(Real y -> S^ y -> y<<=b) ->
                  all c,d(c<d -> all x(Real x -> S^ x -> x<<=d) oru
                                 exnc x(Real x andnc S^ x andnc c<<=x)) ->
                  exl x(Real x andi
                        all y(Real y -> S^ y -> y<<=x) andi
                        all a,p(RealLt a x p -> exnc x(S^ x andi a<<=x))))")
(assume "b" "x" "Rx" "Sx" "bSup" "OLA")
;we obtain a realizer of OLA via invariance
(assert "exl ola^ all c,d(c<d -> (OrUMR
                                  (cterm () all x(Real x -> S^ x -> x<<=d))
                                  (cterm () exnc x(Real x andnc S^ x andnc
    c<<=x)))(ola^ c d))")

                                                    ⋮

(assume "ExOlaMR")
(by-assume "ExOlaMR" "ola^" "OlaMR")
```

The first thing we do is to obtain a realizer ola^ for the assumption "OLA". Note that it is not total, which we would need for the introduction of the sequences via the recursion operator. But we can make it total by modifying it on the domain where $(a < b) \to \mathbf{F}$. Namely we introduce "ola:=[c,d][if (a<b) ola^(c d) False]" which is total and still a realizer of "OLA".

```
(assert "exl ola all c,d(c<d -> (OrUMR
                                  (cterm () all x(Real x -> S^ x -> x<<=d))
                                  (cterm () exnc x(Real x andnc S^ x andnc
    c<<=x)))(ola c d))")

                                                    ⋮
```

```
 (intro 0 (pt "[c,d][if (c<d) (ola^ c d) False]"))

                                                  ⋮

(assume "ExOlaMRTotal")
(drop "OlaMR")
(by-assume "ExOlaMRTotal" "ola" "OlaMRTotal")
```

Now with this total realizer `ola` we introduce our rational sequences and after that the intended upper bound of `"S^"`.

```
(assert "exl cds(
            (cds Zero) eqd cInit x b andnc
            all n(all x(Real x -> S^ x -> x<<=rht (cds (Succ n))) andnc
            exnc x(Real x andnc S^ x andnc lft (cds (Succ n))<<=x) andnc
            lft (cds n)<=lft (cds (Succ n)) andnc
            rht (cds (Succ n))<=rht (cds n) andnc
            rht (cds (Succ n))-lft (cds (Succ n))==(2#3)*(rht (cds n)-lft (cds n)) andnc
            lft (cds (Succ n))<rht (cds (Succ n))))")
(intro 0 (pt "[n](Rec nat=>rat yprod rat)n(cInit x b)([n](cStep ola))"))

                                                  ⋮

(assume "exlseq")
(by-assume "exlseq" "cds" "seqprops")
(cut "all m(m=cRatLeAbsBound(rht(cInit x b)-lft(cInit x b)) ->
            exl x(Real x andnc all y(Real y -> S^ y -> y<<=x) andi
            all a,p(RealLt a x p -> exnc x0(S^ x0 andnc a<<=x0))))")
(assume "AllHyp")
(use "AllHyp" (pt "cRatLeAbsBound(rht(cInit x b)-lft(cInit x b))"))
(use "Truth")
(assume "m" "mDef")
(intro 0 (pt "RealConstr ([n](lft(cds n)))([p]2*(m+p))"))
```

We then proceed by using the theorem `"SeqProps"` to obtain that `"[n]lft(cds n)"` and `"[n]rht(cds n)"` with the proper modulus are real numbers and actually equal.

```
(assert "Real(RealConstr([n]lft(cds n))([p]2*(m+p))) andnc
                 Real(RealConstr([n]rht(cds n))([p]2*(m+p))) andnc
                 RealConstr([n]lft(cds n))([p]2*(m+p))===RealConstr([n]rht(cds
    n))([p]2*(m+p))")
(use "SeqProps")

                                                  ⋮

(save "LUB")
(define sproof (proof-to-soundness-proof))
```

The rest of the proof proceeds as in the paper proof. As before we can generate a soundness-proof and look at the extracted term.

```
> (pp (proof-to-formula sproof))
all b,x(
 Real x ->
 S^ x ->
 all y(Real y -> S^ y -> y<<=b) ->
 all ola^39125(
  all c,d(
   c<d ->
   (OrUMR (cterm () all x(Real x -> S^ x -> x<<=d))
       (cterm () exnc x(Real x andnc S^ x andnc c<<=x)))
   (ola^39125 c d)) ->
  (ExLTMR (cterm (x)
                 Real x andnc
                 all y(Real y -> S^ y -> y<<=x) andnc
                 all a,p(RealLt a x p -> exnc x(S^ x andnc a<<=x))))
  (cLUB b x ola^39125)))
> (define eterm (proof-to-extracted-term (theorem-name-to-proof "LUB")))
> (animate "ExLTotalIntro")
> (animate "BooleTotalNcToTotal")
> (pp (rename-variables (nt eterm)))
[a,x,ola]
 RealConstr
([n]
 lft((Rec nat=>rat yprod rat)n(cInit x a)
     ([n0]cStep([a0,a1][if (a0<a1) (ola a0 a1) False])))
 ([p]
cRatLeAbsBound(rht(cInit x a)+ ~lft(cInit x a))+p+
cRatLeAbsBound(rht(cInit x a)+ ~lft(cInit x a))+
p)
```

Remark 14. Another possibility to introduce the rational sequences would be by defining a program constant. The inductive definition depends on the *locatedness* as well as the bound and inhabitant of the set in question. We can define a program constant ratseqs of type

$$(\mathbf{Q}\times\mathbf{Q}\to\mathbf{B})\times(\mathbf{R}\times\mathbf{Q})\to\mathbf{N}\to\mathbf{Q}\times\mathbf{Q}$$

Again writing *ola* for variables of type $\mathbf{Q}\times\mathbf{Q}\to\mathbf{B}$ we define

$$\mathsf{ratseqs}\langle ola,\langle(\mathsf{RealConstr}\ as\ M),b\rangle\rangle 0\ =\ \langle as_{M(1)}-1,b\rangle$$

Now writing $\mathsf{ratseqs}\langle ola,\langle(\mathsf{RealConstr}\ as\ M),b\rangle\rangle n := \langle f(n),g(n)\rangle$ we further define

$$\begin{aligned}\mathsf{ratseqs}\langle ola,\langle(\mathsf{RealConstr}\ as\ M,b)\rangle\rangle \mathsf{S}n\ =\ \big[\mathsf{case}\ &\big(ola\ {}^{(2f(n)+g(n))}/_3\ {}^{(f(n)+2g(n))}/_3\big)\\ &\big(\mathsf{tt}\to\langle f(n),{}^{(f(n)+2g(n))}/_3\rangle\big)\\ &\big(\mathsf{ff}\to\langle{}^{(2f(n)+g(n))}/_3,g(n)\rangle\big)\big]\end{aligned}$$

3.1.4 Extracting square roots

We want to use LUB to extract for some real number $0\le y$ its square root $\sqrt{y}$. To do so, we consider the set

$$R := \{x\in\mathbb{R}\mid x^2\le y\}$$

We prove that R is located from above.

Lemma 8. *Let $0 \leq y$, then for all c, d with $c < d$ we have*

$$\forall_x(x^2 \leq y \to x \leq d) \vee \exists_x\left(x^2 \leq y \wedge c \leq x\right)$$

i.e. R is located from above.

Proof. Let

$$I := \forall_x(x^2 \leq y \to x \leq d) \qquad II := \exists_x\left(x^2 \leq y \wedge c \leq x\right)$$

We distinguish cases: $c^2 < d^2 \vee d^2 \leq c^2$
Case $c^2 < d^2$. using lemma 7 we compare y to the interval $c^2 < d^2$. We obtain

$$y \leq d^2 \vee c^2 \leq y$$

Subcase $y \leq d^2$. We again distinguish cases: $0 < d \vee d \leq 0$
If $0 < d$ we show I. Let x be given with $x^2 \leq y$, so by transitivity $x^2 \leq d^2$. But this, together with $0 < d$ implies $x \leq d$.
If $d \leq 0$ then also $c \leq 0$ and $0^2 \leq y \wedge c \leq 0$, so we have II.
Subcase $c^2 \leq y$. We have $c^2 \leq y \wedge c \leq c$, i.e. II
Case $d^2 \leq c^2$. We have

$$d^2 \leq c^2 \Rightarrow 0 \leq (c-d)(c+d)$$

and $c - d < 0$ by assumption, so that $c + d \leq 0$. Hence $c \leq 0$ and again $0^2 \leq y \wedge c \leq 0$, i.e. II □

Using the principle of least upper bound in conjunction with the above lemma we can extract a real number. The above proof was also implemented in Minlog, the proof is located in the file `lub_sqrt.scm`. Lemma 8 takes the form

```
;;S is located from above
(set-goal "all y,c,d(Real y -> 0<<=y -> c<d ->
                                    all x(Real x -> x*x<<=y -> x<<=d) oru
                                    exnc x(Real x andnc x*x<<=y andnc c<<=x)) ")
(assume "y" "c" "d" "Ry" "0<=y" "c<d")
;;case c^2<=d^2
(casedist (pt "c*c<d*d"))
(assume "c^2<d^2")
(assert "exl p RealLt (RealConstr([n]c*c)([p]Zero)) (RealConstr([n]d*d)([p]Zero)) p")
(use "RatToRealLt")
(use "c^2<d^2")
(assume "ExHypI")
(by-assume "ExHypI" "p" "pProp")
(assert "y<<=RealConstr([n]d*d)([p]Zero) oru RealConstr([n]c*c)([p]Zero)<<=y")
(use-with "ApproxSplit" (pt "RealConstr([n]c*c)([p]Zero)") (pt
    "RealConstr([n]d*d)([p]Zero)") (pt "y") (pt "p") "?" "?" "?" "pProp")
(use "RealRat")
(use "RealRat")
(realproof)
```

```
(assume "Disj")
(elim "Disj")
;;subcase y<=d^2
(assume "y<=d^2")
(casedist (pt "0<d"))

                                                    ⋮

;;subcase c^2<=y
(assume "c^2<=y")

                                                    ⋮

;;case d^2<=c^2
(assume "c^2<d^2 -> F")

                                                    ⋮

;;done
(save "OLASqrt")
```

The case distinctions in the proofs are realized by the `"(casedist ...)"` command and by disjunction elimination from the instantiated `"ApproxSplit"` theorem. We then apply `"OLASqrt"` to `"LUB"` in the case that $y = 2$:

```
;;LUB for x^2<=2
(set-goal "exl x(Real x andnc all y(Real y -> y*y<<=2 -> y<<=x)
                        andnc all a0,p(RealLt a0 x p -> exnc x0(x0*x0<<=2 andnc
    a0<<=x0)))")
(use "LUB" (pt "(2#1)") (pt "RealConstr([n](0#1))([p]Zero)"))

                                                    ⋮

(use "OLASqrt")

                                                    ⋮

(save "SqrtTwo")

(define eterm (proof-to-extracted-term))
(pp eterm)
;cLUB 2 0([c,d]cOLASqrt 2 c d)
```

Note that the inhabitant was chosen to be 0 and the upper bound to be 2. As extracted term we get $\sqrt{2}$. Animating all theorems with computational content involved we can take a look at the sequence of the real number extracted from the proof:

```
> (animate "Init")
> (animate "StepAux")
> (animate "Step")
> (animate "ExLTotalIntro")
> (animate "LUB")
```

```
> (animate "ApproxSplit")
> (animate "NatPos")
> (animate "Lemma8")
> (animate "RatToRealLt")
> (animate "OLASqrt")
> (define seq-term (pt "[n](cSqrtTwo seq)n"))
> (animate "SqrtTwo")
> (pp (nt (make-term-in-app-form seq-term (pt "10"))))
27422#19683
> (pp (nt (make-term-in-app-form seq-term (pt "20"))))
547756954#387420489
```

In decimal notation we have

$$27422\#19683 = 1.393181934$$
$$547756954\#387420489 = 1.413856442$$

3.2 Translation between signed digit streams and cauchy-sequences

In the last section we dealt with real numbers represented by cauchy-sequences of rational numbers. In the Interval $[-1,1]$ we can also represent reals by so-called *signed-digit-code*. Namely as infinite streams of *signed digits* $\mathbf{SD} = \{-1,0,1\}$. A stream $d_0 d_1 \dots$ is supposed to represent the number

$$\sum_i \frac{d_i}{2^{i+1}}$$

This chapter will deal with the translation between these two representations of real numbers. We will use a co-inductively defined predicate ${}^{co}I$, where ${}^{co}Ix$ is intended to mean that x has some sd-code representation. Using this we will prove that any real contained in the interval $[-1,1]$ has some sd-code representation, and conversely, that any sd-stream corresponds to some real number (see [1],[10]).

3.2.1 Signed digit streams

We consider the algebra

$$\mathbf{Str} := \mu_\xi \, (\mathbf{SD} \to \xi \to \xi)$$

of "standard rational intervals". As in chapter 3 we denote the constructor by C_d and elements of **Str** by u, v. This algebra is not well-founded, but we can still look at the cototal ideals, which correspond to infinite streams of **SD**.
We want to relate concrete reals, represented by cauchy-sequences, to such streams. For this purpose we define a predicate I of arity **R** by the clause

$$\forall^{nc}_{x,y} \forall_{d \in \mathbf{SD}} \Big(|x| \le 1 \to Ix \to y = \frac{x+d}{2} \to Iy \Big)$$

Note that $\forall_{d\in\mathbf{SD}}$ is short for $\forall^{nc}_d(Sd(d)\to\ldots$, where Sd is the set of integers inductively defined by

$$Sd := \mu_X\,(X(\mathsf{N1}), X(0), X(\mathsf{P1}))$$

Then the associated algebra is just **Str**. Furthermore we define its companion, the co-inductively defined predicate ${}^{co}I$ with ${}^{co}I$-clause

$$\forall^{nc}_x\Big({}^{co}Ix\to\exists^r_y\exists^d_{d\in\mathbf{SD}}\Big(\,|y|\le 1\wedge^r {}^{co}Iy\wedge^l x=\frac{y+d}{2}\Big)\Big)$$

The realizability predicate for ${}^{co}I$ is then given by the clause

$$\forall_{x,\hat u}\left(\hat u\;\mathbf{r}\;{}^{co}Ix\to\exists^{nc}_{d,y,\hat s}\left(\hat s\;\mathbf{r}\;Sd(d)\wedge|y|\le 1\wedge\exists^{nc}_{\hat v}\left(\hat v\;\mathbf{r}\;{}^{co}Iy\wedge x=\frac{y+d}{2}\wedge\hat u\equiv C_{\hat s}\hat v\right)\right)\right)$$

Now realizers of the proposition ${}^{co}Ix$ are infinite streams of signed digits, i.e. cototal ideals in **Str**. We can actually prove this:

Theorem 6. *Let $\hat u$ be a realizer of ${}^{co}Ix$, then $\hat u$ is cototal, i.e.*

$$\forall^{nc}_x\forall_{\hat u}\,(\hat u\;\mathbf{r}\;{}^{co}Ix\to{}^{co}T^{nc}\hat u)$$

Proof. Assume we have x and $\hat u$ with $\hat u\;\mathbf{r}\;{}^{co}Ix$. From the ${}^{co}I^{\mathbf{r}}$-clause we obtain

$$A(\hat u) := \exists^{nc}_{d,y,\hat s,\hat u_0}\,(\hat s\;\mathbf{r}\;Sd(d)\wedge^{nc}\hat u_0\;\mathbf{r}\;{}^{co}Iy\wedge^{nc}\hat u\equiv \mathsf{C}_{\hat s}\hat u_0) \tag{3.11}$$

We prove ${}^{co}T^{nc}\hat u$ by $({}^{co}T^{nc})^+$ with competitor $A(\hat u)$. Thus we only need to prove the premise, which is

$$\forall^{nc}_{\hat u}\,(A(\hat u)\to\exists^{nc}_{\hat s}\,(T\hat s\wedge\exists^{nc}_{\hat v}\,({}^{co}T^{nc}\hat v\vee^{nc}A(\hat v)\wedge^{nc}\hat u\equiv\mathsf{C}_{\hat s}\hat v)))$$

So let $\hat u$ and $d,\ y,\ \hat s,\hat u_0$ with $\hat s\;\mathbf{r}\;Sd(d)\wedge^{nc}\hat u_0\;\mathbf{r}\;{}^{co}Iy\wedge^{nc}\hat u\equiv\mathsf{C}_{\hat s}\hat u_0$ be given. We use $(\exists)^+$ with $\hat s$. From $\hat s\;\mathbf{r}\;Sd(d)$ we obtain $T\hat s$. Now using $(\exists)^+$ with $\hat u_0$ it remains to prove

$${}^{co}T^{nc}\hat u_0\vee^{nc}A(\hat u_0)\wedge^{nc}\hat u\equiv\mathsf{C}_{\hat s}\hat u_0$$

The second conjunct we have by assumption. For the first we choose the right side, which we get from $\hat u_0\;\mathbf{r}\;{}^{co}Iy$ as above. □

Now recall example 6 and the requirement on the representation by sd-code we had there. The first digit v_0 of the assigned code should indicate that

$$x\in[{}^{-1}\!/_2+{}^{v_0}\!/_2,{}^1\!/_2+{}^{v_0}\!/_2]\longleftrightarrow 2x-v_0\in[-1,1]$$

Now if $u\;\mathbf{r}\;{}^{co}Ix$ then by the ${}^{co}I$-clause we get $y,\ d,\ v$ with

$$|y|\le 1\wedge v\;\mathbf{r}\;{}^{co}Iy\wedge x=\frac{y+d}{2}\wedge u\equiv\mathsf{C}_d v$$

Hence $u_0\equiv d$ and $2x-u_0=y\in[-1,1]$ and the process is repeated via $v\;\mathbf{r}\;{}^{co}Iy$. We see that we have exactly the same requirement on the realizing sd-stream as we had in the example.

3.2.2 Translation between sd-code and cauchy-sequences

We now prove that any real contained in the interval $[-1,1]$, i.e. with modulus ≤ 1, lies in ${}^{co}I$. Conversely we prove that for any real that lies in ${}^{co}I$ there is a (c.r.!) real number equal to it. The computational content of these proofs will then give us the desired translations between cauchy-sequences and sd-codes.

Theorem 7. $\forall_x \left(|x| \leq 1 \to {}^{co}Ix\right)$

Proof. Let

$$X := \left\{x \mid |x| \leq 1|\right\} \qquad Y := \left\{x \mid \exists_{d,y}\left(|y| \leq 1 \wedge x = \frac{y+d}{2}\right)\right\}$$

We show that $X \subseteq Y \subseteq {}^{co}I$. For $X \subseteq Y$ let x with $|x| \leq 1$ be given. We need to prove that there exist y and $d \in \mathbf{SD}$, such that

$$|y| \leq 1 \wedge x = \frac{y+d}{2}$$

As in example 6 we obtain the same cases by comparing x to the intervals with $0 < 1/2$ and $-1/2 < 0$ via lemma 7. Naturally we then choose the following

$$\langle d, y\rangle = \begin{cases} \langle -1, 2x+1\rangle, & x \leq 0 \\ \langle 0, 2x\rangle, & -1/2 \leq x \leq 1/2 \\ \langle 1, 2x-1\rangle, & 0 \leq x \end{cases}$$

In all cases we clearly have $x = \frac{y+d}{2}$ and $|y| \leq 1$. Hence $X \subseteq Y$.
In order to show that $Y \subseteq {}^{co}I$ we must prove that Y satisfies the ${}^{co}I$-clause, i.e.

$$\forall_x^{nc}\left(Y(x) \to \exists_y^r \exists_{d\in\mathbf{SD}}^d \left(|y| \leq 1 \wedge^r \left({}^{co}Iy \vee^d Y(y)\right) \wedge^l x = \frac{y+d}{2}\right)\right)$$

So let x with $Y(x)$ e given. By definition of Y we have y, d with $|y| \leq 1$ and $x = \frac{y+d}{2}$. It remains to show that $Y(y)$, but we have $|y| \leq 1$, i.e. $X(y)$ and we already proved $X \subseteq Y$. □

Remark 15. (i) Using the corecursion operator we have already defined the conversion from (abstract) real numbers to cototal ideals in **Str** (see example 6). Note that there we assumed that we have some function that allows us to compare real numbers to proper rational intervals. In the context of concrete real numbers given as cauchy-sequences this function is given by the computational content of lemma 7.
(ii) We give the informal algorithm given by the proof. Let cAS be the computational content of lemma 7 of type $\mathbf{R} \to \mathbf{R} \to \mathbf{R} \to \mathbf{B}$. The first step in the proof is to compare x

to the intervals $[{}^{-1}/_2, 0]$, $[0, {}^1/_2]$ and choose a signed digit and a new real:

$$\begin{aligned} g(x) :\equiv \big[&\mathsf{case}\ (\mathsf{cAS}({}^{-1}/_2, 0, x))\ \mathsf{of} \\ &\mathsf{tt} \to \langle -1, 2x+1\rangle \\ &\mathsf{ff} \to \big[\mathsf{case}\ (\mathsf{cAS}(0, {}^1/_2, x))\ \mathsf{of} \\ &\qquad \mathsf{tt} \to \langle 0, 2x\rangle \\ &\qquad \mathsf{ff} \to \langle 1, 2x-1\rangle\big]\big] \end{aligned}$$

In the coinduction this step is repeated with the new real number $g(x)_1$. The informal algorithm is then given by

$$\mathsf{CsToStr}(x) :\equiv \mathsf{C}_{g(x)_0}\mathsf{CsToStr}(g(x)_1)$$

which is basically the same as in example 6.

We turn to the translation in the other direction, i.e. for some sd-string we want a real number represented by it. The formal specification is as follows.

Theorem 8. $\forall_x^{nc}\big({}^{co}Ix \to \exists_y^l\ (x = y)\big)$

Proof. Let x with ${}^{co}Ix$ be given. We inductively define sequences $(a_n)_n$ and $(y_n)_n$ such that

$$ {}^{co}Iy_n \wedge x = \frac{y_n}{2^{n+1}} + a_n \tag{3.12}$$

For $n = 0$ we use the ${}^{co}I$-clause for x to obtain y and $d \in \mathbf{SD}$ with

$$|y| \le 1 \wedge^{r\ co} Iy \wedge^l x = \frac{y+d}{2}$$

Then $y_0 = y$ and $a_0 = \frac{d}{2}$ satisfy (3.12). Suppose the first n terms of the sequences are constructed. We use the ${}^{co}I$-clause applied to y_n to obtain y and $d \in \mathbf{SD}$ with

$$|y| \le 1 \wedge^{r\ co} Iy \wedge^l y_n = \frac{y+d}{2}$$

Then by assumption

$$x = \frac{y_n}{2^{n+1}} + a_n = \frac{y}{2^{n+2}} + \left(\frac{d}{2^{n+2}} + a_n\right)$$

We thus choose $y_{n+1} = y$ and $a_{n+1} = \frac{d}{2^{n+2}} + a_n$ which again satisfy (3.12). We define $y :\equiv ((a_n), \lambda_p p)$. Note that we have for all n, that

$$\begin{aligned} |x - a_n| &\le \frac{1}{2^{n+1}} \\ |a_n - a_{n+1}| &\le \frac{1}{2^n} \end{aligned}$$

It remains to prove that y is a real number and that $x = y$, which are both easy exercises in constructive analysis using the inequalities above. □

Remark 16. (i) Note the decoration in the formula in the theorem above. We require the variable x to be non-computational, so intuitively we do not know anything about its sequence of rational numbers. y on the other hand is computational, so in the proof we actually need to provide such a cauchy-sequence.
(ii) Let us again formulate the informal algorithm given by the proof. Let v be a stream representing a real number x, i.e. a realizer of ${}^{co}Ix$. We define the n-th digit of v by

$$\begin{aligned} v_0 &= (\mathsf{D}v)_0 \\ v_{n+1} &= ((\mathsf{D}v)_1)_n \end{aligned}$$

with $\mathsf{D} : \mathbf{Str} \to \mathbf{SD} \times \mathbf{Str}$ the destructor. Then according to the inductive construction the rational sequence is given by

$$\begin{aligned} \mathsf{StrToCs}\ 0\ x &= \frac{v_0}{2} \\ \mathsf{StrToCs}\ (n+1)\ x &= \frac{v_{n+1}}{2^{n+2}} + \mathsf{StrToCs}\ n\ x \end{aligned}$$

3.2.3 Implementation

We now discuss the implementation of the theorems 7 and 8 in Minlog. We initially load the file `~/git/minlog/examples/analysis/sdmult.scm` which also contains some contributions by the author. The relevant files from the disc are

`StrToCs.scm` `StrToCsLemma.scm` `CsToStr.scm`

We begin with theorem 7, located in the file `CsToStr.scm`. The first step is to prove the auxiliary statement `"CsToStrAux"` which corresponds to the $X \subset Y$ part of the proof.

```
;;CsToStrAux
(set-goal "all x(Real x --> abs x<<=1 -->
           exd sd exl y(Real y andnc abs y<<=1 andnc x===(1#2)*(y+(SdToInt sd))))")
(assume "x" "Rx" "abs x<=1")

                                          ⋮


(save "StrToCsAux")
```

Note that the constructors of **SD** are given by `"SdL, SdM, SdR"` in Minlog and `"SdToInt"` is the canonical injection $\mathbf{SD} \hookrightarrow \mathbf{Z}$. When we take a look at the extracted term

```
> (define eterm (proof-to-extracted-term))
> (define neterm (rename-variables (nt eterm)))
> (pp neterm)
 [x]
  [if (cApproxSplit(IntN 1#2)0 x 1)
    (SdL pair 2*x+1)
    [if (cApproxSplit 0(1#2)x 1) (SdM pair 2*x) (SdR pair 2*x+IntN 1)]]
```

we see that it is exactly the auxiliary function that was given in remark 15. Now theorem 7, called "CsToStr", takes the following form in Minlog.

```
;;CsToStr
(set-goal "all x(Real x andnc abs x<<=1 -> CoI x)")
(assume "x" "Conj")
(assert "exd sd exl y(Real y andnc abs y<<=1 andnc x===(1#2)*(y+SdToInt sd))")
 (use "RealToCoIAux" (pt "x"))
 (use "Conj")
 (use "Conj")
(assume "Hyp")
(coind "Hyp")

                                          ⋮


(save "CsToStr")
```

As in the paper proof it is proven by coinduction on Y, i.e. by the "(coind)" command. Let us take a look at the extracted term.

```
> (define eterm (proof-to-extracted-term))
> (define neterm (rename-variables (nt eterm)))
> (pp neterm)
 [x]
  (CoRec sd yprod rea=>str)(cCsToStrAux x)
  ([(sd yprod rea)]
    [if (sd yprod rea)
      ([s,x0]
       [if (cCsToStrAux x0)
         ([s0,x1]s pair(InR (sd yprod rea) str)(s0 pair x1))])])
```

This again exactly corresponds to the intuitive algorithm already given.
We proceed with the implementation of theorem 8. We load the file StrToCsLemma.scm from the disk which contains some basic statements, and for technical reasons adds the following axioms

$$\text{ExLTotal2Intro}: \ \exists^l_{\hat{x}}\,(T^{nc}(\hat{x}) \wedge^{nc} A(\hat{x})) \to \exists^l_x A(x)$$
$$\text{ExDTotal2Intro}: \ \exists^d_{\hat{x}}\,(T^{nc} \wedge^{r} A(\hat{x})) \to \exists^d_x A(x)$$

But they will actually only be used for x of type $\mathbf{Q}$ and it is easy (but somewhat long) to prove that they are equivalent to the versions "ExLTotalIntro" and "ExDTotalIntro" for type $\mathbf{Q}$ (Just note that $\forall_a(T^{nc}_{\mathbf{Q}}a \to T_{\mathbf{Q}}a)$ is derivable). The remaining implementation is contained in StrToCs.scm.
Here, as in the proof of LUB, we used an inductive construction in the paper proof and this leads to several possible implementations. Here we will make use of a suitable version of dependent choice, namely

$$\exists^d_{\hat{x}}A(0,\hat{x}) \to \forall_n\left(\exists^d_{\hat{x}}A(n,\hat{x}) \to \exists^d_{\hat{x}}A(n+1,\hat{x})\right) \to \exists^d_{\hat{f}}\forall_n A(n,\hat{f}n)$$

If $x : \alpha$ and $\tau(A) = \beta$ a realizer is given by

$$\lambda_{P^{\alpha\times\beta}}\lambda_{F^{\mathbf{N}\to\alpha\times\beta\to\alpha\times\beta}}\left\langle \lambda_n \left\langle \mathcal{R}_{\mathbf{N}}^{\alpha\times\beta} n\ P\ F\right\rangle_0, \lambda_n \left\langle \mathcal{R}_{\mathbf{N}}^{\alpha\times\beta} n\ P\ F\right\rangle_1\right\rangle$$

It is proven by invariance, directly supplying the realizer (as in the proof of theorem 3):

```
(set-goal "exd alpha^((Pvar nat alpha)Zero alpha^) ->
           all n(exd alpha^(Pvar nat alpha)n alpha^ -> exd alpha^ (Pvar nat alpha)
    (Succ n) alpha^) ->
           exd (nat=>alpha)^ all n(Pvar nat alpha) n ((nat=>alpha)^ n)")
(use-with "InvarAll"

                                          ⋮

(save "ExDDC")
```

The construction of the sequences $(a_n), (y_n)$ is again separated into two theorems `"StrToCsInit"` and `"StrToCsStep"`. Their formulas are such that they exactly fit the theorem `"ExDDC"`. The initial one is given by:

```
;;Init
(set-goal "allnc x(CoI x -> exd a^ (TotalNc a^ andr exr y(CoI y andl x===(1#2**Succ
    Zero)*y+a^)))")

                                          ⋮

(save "StrToCsInit")
```

If we animate some related theorem we can take a look at the extracted term:

```
> (animate "CoIClosure")
> (animate "SdToSdMR")
> (define neterm (rename-variables (nt (proof-to-extracted-term))))
> (pp neterm)
[u][if (DesYprod u) ([s,u0](SdToInt s#2)pair(SdToInt s#2)pair u0)]
```

We see that the left term of the pair is exactly the start of the inductive definition of StrToCs in remark 16. `"Step"` looks like the following.

```
;;Step
(set-goal "allnc x all n(exd a^ (TotalNc a^ andr exr y(CoI y andl x===(1#2**(Succ
    n))*y+a^)) ->
                exd b^ (TotalNc b^ andr exr y(CoI y andl x===(1#2**(Succ(Succ
    n)))*y+b^)))")

                                          ⋮

(save "StrToCsStep")
```

The extracted term is given by

```
> (define neterm (rename-variables (nt (proof-to-extracted-term))))
> (ppc neterm)
[n,(rat yprod str)]
 [case (rat yprod str)
   (a pair u ->
   [case (DesYprod u)
     (s pair u0 -> a+(SdToInt s#SZero(SZero(2**n)))pair crht DesYprod u)])]
```

And again the `"a+(SdToInt s#SZero(SZero(2**n)))"` term directly corresponds to the term given in remark 16. Now we can prove theorem 8. The formulation in Minlog is as follows.

```
;;StrToCs
(set-goal "allnc x(CoI x -> exl y(Real y andnc x===y))")
(assume "x" "CoIx")
(assert "exd as^ all n (TotalNc (as^ n) andr exr y(CoI y andl x===(1#2**Succ n)*y+as^
    n))")
 (use "ExDDC")
 (use-with "StrToCsInit" (pt "x") "CoIx")
 (use-with "StrToCsStep" (pt "x"))
(assume "RatSeqExHypPartial")
(by-assume "RatSeqExHypPartial" "as^" "asPropPartial")
(assert "exd as all n exr y(CoI y andl x===(1#2**Succ n)*y+as n)")
 (use "ExDTotal2Intro")

                                          ⋮

(save "StrToCs")
```

The existence of the rational sequence is proven using dependent choice and the theorems `"StrToCsInit/Step"`. Note that as given by `"ExDDC"` it is partial at first, but we can make it total by using `"ExDTotal2Intro"`. The rest of the proof proceeds along the same lines as the paper proof. Again we can display the extracted term.

```
> (define neterm (rename-variables (nt (proof-to-extracted-term))))
> (ppc neterm)
[u]
 RealConstr
 ([n]
   lft((Rec nat=>rat yprod str)n
       [case (DesYprod u) (s pair u -> (SdToInt s#2)pair u)]
       (cStrToCsStep u)))
 PosToNat
```

3.2.4 Exact real arithmetic

Using the above translations we can easily prove that some operations are closed under ${}^{co}I$, e.g. we can show

$$ {}^{co}Ix \to {}^{co}Iy \to {}^{co}I\ \frac{x+y}{2} $$

An important missing component is the following lemma.

Lemma 9. (a) $\forall^{nc}_{x,y}\,(x = y \to {}^{co}Ix \to {}^{co}Iy)$
(b) $\forall^{nc}_{x}\,({}^{co}Ix \to |x| \leq 1)$

Proof. See [1] □

Theorem 9. (a) $\forall^{nc}_{x,y}\big({}^{co}Ix \to {}^{co}Iy \to {}^{co}I\big(\frac{x+y}{2}\big)\big)$
(b) $\forall^{nc}_{x,y}\,({}^{co}Ix \to {}^{co}Iy \to {}^{co}I\,(xy))$
(c) $\forall^{nc}_{x,y}\big(|x| \leq |y| \wedge^{nc} 0 < |y| \to {}^{co}Ix \to {}^{co}Iy \to {}^{co}I\big(\frac{x}{y}\big)\big)$

Proof. The proofs of all parts will proceed along the same lines. We just consider part (a). Assume we have x, y with ${}^{co}Ix$ and ${}^{co}Iy$. By theorem 8 we have real numbers $z_1 = ((a_n), M)$ and $z_2 = ((b_n), N)$ with $x = z_1$ and $y = z_2$. By the first part of the lemma above we get that ${}^{co}Iz_i$ and by the second part that $|z_i| \leq 1$. This implies that

$$\left|\frac{z_1 + z_2}{2}\right| \leq 1$$

Now by Theorem 7 we have ${}^{co}I\,\big(\frac{z_1+z_2}{2}\big)$ and the lemma above with

$$\frac{x+y}{2} = \frac{z_1+z_2}{2}$$

yields the claim. □

Note that the proofs of part a) and b) are implemented at the end of the file `StrToCs.scm`. But the extracted terms are not very efficient if compared e.g. to the term that was extracted in [10] for the average of two streams. This is due to the general method invoked in the proof above, whereas in [10] the algorithm was supplied more or less directly in the proof.

Bibliography

[1] U. Berger, K. Miyamoto, H. Schwichtenberg, and H. Tsuiki. "Logic for Gray-code Computation". In: *Concepts of Proof in Mathematics, Philosophy, and Computer Science* (2016).

[2] E. Bishop and D. Bridges. *Constructive analysis*. Grundlehren der mathematischen Wissenschaften. Springer-Verlag, 1985. ISBN: 9783540150664.

[3] H. Curry. *Grundlagen der kombinatorischen Logik*. Johns Hopkins Press, 1930.

[4] Y. Ershov. In: *Logic Colloquium 76*. Ed. by R. Gandy and J. Hyland. Vol. 87. Studies in Logic and the Foundations of Mathematics Supplement C. Elsevier, 1977, pp. 455–467.

[5] K. Gödel. "Über Eine Bisher Noch Nicht Benützte Erweiterung Des Finiten Standpunktes". In: *Dialectica* 12.34 (1958), pp. 280–287.

[6] L. Hoffmann. "Rechnerischer Gehalt von Beweisen in der konstruktiven Analysis". Ludwig-Maximilians-Universität München.

[7] G. Kreisel. "Interpretation of Analysis by Means of Constructive Functionals of Finite Types". In: *Constructivity in Mathematics*. Ed. by A. Heyting. Amsterdam: North-Holland Pub. Co., 1959, pp. 101–128.

[8] P. Martin-Löf and G. Sambin. *Intuitionistic type theory*. Studies in proof theory. Bibliopolis, 1984.

[9] K. Miyamoto. "Program extraction from coinductive proofs and its application to exact real arithmetic". Dec. 2013.

[10] K. Miyamoto and H. Schwichtenberg. "Program extraction in exact real arithmetic". In: *Mathematical Structures in Computer Science* 25.8 (2015), pp. 1692–1704. DOI: 10.1017/S0960129513000327.

[11] H. Schwichtenberg and S. Wainer. *Proofs and Computations*. Perspectives in Logic. Cambridge University Press, 2011. ISBN: 9781139504164.

[12] H. Schwichtenberg. *Constructive Analysis with witnesses*. URL: http://www.mathematik.uni-muenchen.de/~schwicht/seminars/semws16/constr16.pdf.

[13] H. Schwichtenberg. *MINLOG REFERENCE MANUAL*. URL: http://www.mathematik.uni-muenchen.de/~logik/download/ref.pdf.

[14] D. S. Scott. *Domains for denotational semantics*. Ed. by M. Nielsen and E. M. Schmidt. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, pp. 577–610. ISBN: 978-3-540-39308-5. DOI: 10.1007/BFb0012801.