# Selected topics in proof theory

Helmut Schwichtenberg

Mathematisches Institut der Universität München
Sommersemester 2013

# Contents

CHAPTER 1

# Proof theory of arithmetic

The goal of this chapter is to present some in a sense "most complex" proofs that can be done in first-order arithmetic.

The main tool for proving theorems in arithmetic is clearly the induction schema

$$A(0) \to \forall_x(A(x) \to A(Sx)) \to \forall_x A(x).$$

Here $A(x)$ is an arbitrary formula. An equivalent form of this schema is "course-of-values" or cumulative induction

$$\forall_x(\forall_{y<x} A(y) \to A(x)) \to \forall_x A(x).$$

Both schemes refer to the standard ordering of the natural numbers. Now it is tempting to try to strengthen arithmetic by allowing more general induction schemas, e.g., with respect to the lexicographical ordering of $\mathbb{N}\times\mathbb{N}$. More generally, we might pick an arbitrary well-ordering $\prec$ over $\mathbb{N}$ and use the schema of *transfinite induction*:

$$\forall_x(\forall_{y\prec x} A(y) \to A(x)) \to \forall_x A(x).$$

This can be read as follows. Suppose the property $A(x)$ is "progressive", i.e., from the validity of $A(y)$ for all $y \prec x$ we can always conclude that $A(x)$ holds. Then $A(x)$ holds for all $x$.

One might wonder for which well-orderings this schema of transfinite induction is actually derivable in arithmetic. We will prove here a classic result of Gentzen (1943) which in a sense answers this question completely. However, in order to state the result we have to be more explicit about the well-orderings used. This is done in the next section.

## 1.1. Ordinals below $\varepsilon_0$

We want to discuss the derivability of initial cases of transfinite induction in arithmetical systems. In order to do that we shall need some knowledge and notations for ordinals. We do not want to assume set theory here; hence we introduce a certain initial segment of the ordinals (the ordinals $< \varepsilon_0$) in a formal, combinatorial way, i.e., via ordinal notations. Our treatment is

based on the Cantor normal form for ordinals; cf. Bachmann (1955). We also introduce some elementary relations and operations for such ordinal notations, which will be used later. For brevity we from now on use the word "ordinal" instead of "ordinal notation".

**1.1.1. Basic definitions.** We define the two notions

- $\alpha$ is an ordinal
- $\alpha < \beta$ for ordinals $\alpha, \beta$

simultaneously by induction:

(1) If $\alpha_m, \ldots, \alpha_0$ are ordinals, $m \geq -1$ and $\alpha_m \geq \cdots \geq \alpha_0$ (where $\alpha \geq \beta$ means $\alpha > \beta$ or $\alpha = \beta$), then

$$\omega^{\alpha_m} + \cdots + \omega^{\alpha_0}$$

is an ordinal. Note that the empty sum denoted by 0 is allowed.

(2) If $\omega^{\alpha_m} + \cdots + \omega^{\alpha_0}$ and $\omega^{\beta_n} + \cdots + \omega^{\beta_0}$ are ordinals, then

$$\omega^{\alpha_m} + \cdots + \omega^{\alpha_0} < \omega^{\beta_n} + \cdots + \omega^{\beta_0}$$

iff there is an $i \geq 0$ such that $\alpha_{m-i} < \beta_{n-i}$, $\alpha_{m-i+1} = \beta_{n-i+1}$, $\ldots$, $\alpha_m = \beta_n$, or else $m < n$ and $\alpha_m = \beta_n$, $\ldots$, $\alpha_0 = \beta_{n-m}$.

For proofs by induction on ordinals it is convenient to introduce the notion of *level* of an ordinal $\alpha$ by the stipulations (a) if $\alpha$ is the empty sum 0, $\mathrm{lev}(\alpha) = 0$, and (b) if $\alpha = \omega^{\alpha_m} + \ldots + \omega^{\alpha_0}$ with $\alpha_m \geq \ldots \geq \alpha_0$, then $\mathrm{lev}(\alpha) = \mathrm{lev}(\alpha_m) + 1$.

For ordinals of level $k+1$ we have $\omega_k \leq \alpha < \omega_{k+1}$, where $\omega_0 = 0$, $\omega_1 = \omega$, $\omega_{k+1} = \omega^{\omega_k}$.

We shall use the notation 1 for $\omega^0$, $k$ for $\omega^0 + \cdots + \omega^0$ with $k$ copies of $\omega^0$ and $\omega^\alpha k$ for $\omega^\alpha + \cdots + \omega^\alpha$ again with $k$ copies of $\omega^\alpha$.

It is easy to see (by induction on the levels) that $<$ is a linear order with 0 being the smallest element.

We define addition for ordinals by

$$\omega^{\alpha_m} + \cdots + \omega^{\alpha_0} + \omega^{\beta_n} + \cdots + \omega^{\beta_0} := \omega^{\alpha_m} + \cdots + \omega^{\alpha_i} + \omega^{\beta_n} + \cdots + \omega^{\beta_0}$$

where $i$ is minimal such that $\alpha_i \geq \beta_n$.

It is easy to see that $+$ is an associative operation which is strictly monotonic in the second argument and weakly monotonic in the first argument. Note that $+$ is not commutative: $1 + \omega = \omega \neq \omega + 1$.

There is also a commutative version on addition. The *natural* (or Hessenberg) sum of two ordinals is defined by

$$(\omega^{\alpha_m} + \cdots + \omega^{\alpha_0}) \# (\omega^{\beta_n} + \cdots + \omega^{\beta_0}) := \omega^{\gamma_{m+n}} + \cdots + \omega^{\gamma_0},$$

where $\gamma_{m+n}, \dots, \gamma_0$ is a decreasing permutation of $\alpha_m, \dots, \alpha_0, \beta_n, \dots, \beta_0$. It is easy to see that $\#$ is associative, commutative and strictly monotonic in both arguments.

We will also need to know how ordinals of the form $\beta + \omega^\alpha$ can be approximated from below. First note that

$$\delta < \alpha \to \beta + \omega^\delta k < \beta + \omega^\alpha.$$

Furthermore, for any $\gamma < \beta + \omega^\alpha$ we can find a $\delta < \alpha$ and a $k$ such that

$$\gamma < \beta + \omega^\delta k.$$

**1.1.2. Enumerating ordinals.** In order to work with ordinals in a purely arithmetical system we set up some effective bijection between our ordinals $< \varepsilon_0$ and non-negative integers (i.e., a Gödel numbering). For its definition it is useful to refer to ordinals in the form

$$\omega^{\alpha_m} k_m + \dots + \omega^{\alpha_0} k_0 \quad \text{with } \alpha_m > \dots > \alpha_0 \text{ and } k_i \neq 0 \ (m \geq -1).$$

(By convention, $m = -1$ corresponds to the empty sum.)

For every ordinal $\alpha$ we define its Gödel number $\ulcorner \alpha \urcorner$ inductively by

$$\ulcorner \omega^{\alpha_m} k_m + \dots + \omega^{\alpha_0} k_0 \urcorner := \Big( \prod_{i \leq m} p_{\ulcorner \alpha_i \urcorner}^{k_i} \Big) - 1,$$

where $p_n$ is the $n$-th prime number starting with $p_0 := 2$. For every non-negative integer $x$ we define its corresponding ordinal notation $\mathrm{o}(x)$ inductively by

$$\mathrm{o}\Big( (\prod_{i \leq l} p_i^{q_i}) - 1 \Big) := \sum_{i \leq l} \omega^{\mathrm{o}(i)} q_i,$$

where the sum is to be understood as the natural sum.

LEMMA. (a) $\mathrm{o}(\ulcorner \alpha \urcorner) = \alpha$,
(b) $\ulcorner \mathrm{o}(x) \urcorner = x$.

PROOF. This can be proved easily by induction. □

Hence we have a simple bijection between ordinals and non-negative integers. Using this bijection we can transfer our relations and operations on ordinals to computable relations and operations on non-negative integers. We use the following abbreviations.

$$\begin{aligned} x \prec y &:= \mathrm{o}(x) < \mathrm{o}(y), \\ \omega^x &:= \ulcorner \omega^{\mathrm{o}(x)} \urcorner, \\ x \oplus y &:= \ulcorner \mathrm{o}(x) + \mathrm{o}(y) \urcorner, \end{aligned}$$

$$\begin{aligned} xk &:= \ulcorner \mathrm{o}(x)k \urcorner, \\ \omega_k &:= \ulcorner \omega_k \urcorner. \end{aligned}$$

We leave it to the reader to verify that $\prec$, $\lambda_x \omega^x$, $\lambda_{x,y}(x \oplus y)$, $\lambda_{x,k}(xk)$ and $\lambda_k \ulcorner \omega_k \urcorner$ are all elementary.

## 1.2. Provability of initial cases of transfinite induction

We now derive initial cases of the principle of transfinite induction in arithmetic, i.e., of

$$\forall_x(\forall_{y \prec x} Py \to Px) \to \forall_{x \prec a} Px$$

for some number $a$ and a predicate symbol $P$, where $\prec$ is the standard order of order type $\varepsilon_0$ defined in the preceding section. One can show that our results here are optimal in the sense that for the full system of ordinals $< \varepsilon_0$ the principle

$$\forall_x(\forall_{y \prec x} Py \to Px) \to \forall_x Px$$

of transfinite induction is underivable. All these results are due to Gentzen (1943).

**1.2.1. Arithmetical systems.** By an *arithmetical system* **Z** we mean a theory based on minimal logic in the $\forall\to$-language (including equality axioms), with the following properties. The language of **Z** consists of a fixed (possibly countably infinite) supply of function and relation constants which are assumed to denote fixed functions and relations on the non-negative integers for which a computation procedure is known. Among the function constants there must be a constant S for the successor function and 0 for (the 0-place function) zero. Among the relation constants there must be a constant $=$ for equality and $\prec$ for the ordering of type $\varepsilon_0$ of the natural numbers, as introduced in section 1.1. In order to formulate the general principle of transfinite induction we also assume that a unary relation symbol $P$ is present, which acts like a free set variable.

*Terms* are built up from object variables $x, y, z$ by means of $f(t_1, \ldots, t_m)$, where $f$ is a function constant. We identify closed terms which have the same value; this is a convenient way to express in our formal systems the assumption that for each function constant a computation procedure is known. Terms of the form $\mathrm{S}(\mathrm{S}(\ldots \mathrm{S0} \ldots))$ are called *numerals*. We use the notation $\mathrm{S}^n 0$ or $\overline{n}$ or (only in this chapter) even $n$ for them. *Formulas* are built up from atomic formulas $R(t_1, \ldots, t_m)$, with $R$ a relation constant or a relation symbol, by means of $A \to B$ and $\forall_x A$.

The *axioms* of **Z** include compatibility of equality

$$x = y \to A(x) \to A(y),$$

the *Peano axioms*, i.e., the universal closures of

$$\mathrm{S}x = \mathrm{S}y \to x = y, \tag{1.1}$$

$$\mathrm{S}x = 0 \to A, \tag{1.2}$$

$$A(0) \to \forall_x(A(x) \to A(\mathrm{S}x)) \to \forall_x A(x), \tag{1.3}$$

with $A(x)$ an arbitrary formula. We express our assumption that for every relation constant $R$ a decision procedure is known by adding the axiom $R\vec{n}$ whenever $R\vec{n}$ is true. Concerning $\prec$ we require as axioms irreflexivity and transitivity for $\prec$

$$x \prec x \to A,$$

$$x \prec y \to y \prec z \to x \prec z$$

and also – following Schütte – the universal closures of

$$x \prec 0 \to A, \tag{1.4}$$

$$z \prec y \oplus \omega^0 \to (z \prec y \to A) \to (z = y \to A) \to A, \tag{1.5}$$

$$x \oplus 0 = x, \tag{1.6}$$

$$x \oplus (y \oplus z) = (x \oplus y) \oplus z, \tag{1.7}$$

$$0 \oplus x = x, \tag{1.8}$$

$$\omega^x 0 = 0, \tag{1.9}$$

$$\omega^x(\mathrm{S}y) = \omega^x y \oplus \omega^x, \tag{1.10}$$

$$z \prec y \oplus \omega^{\mathrm{S}x} \to z \prec y \oplus \omega^{e(x,y,z)} m(x,y,z), \tag{1.11}$$

$$z \prec y \oplus \omega^{\mathrm{S}x} \to e(x,y,z) \prec \mathrm{S}x, \tag{1.12}$$

where $\oplus$, $\lambda_{x,y}(\omega^x y)$, $e$ and $m$ denote the appropriate function constants and $A$ is any formula. (The reader should check that $e$, $m$ can be taken to be elementary.) These axioms are formal counterparts to the properties of the ordinal notations observed in the preceding section.

### 1.2.2. Gentzen's proof.

THEOREM (Provable initial cases of transfinite induction in **Z**). *Transfinite induction up to $\omega_n$, i.e., for arbitrary $A(x)$ the formula*

$$\forall_x(\forall_{y \prec x} A(y) \to A(x)) \to \forall_{x \prec \omega_n} A(x),$$

*is derivable in* **Z**.

PROOF. To every formula $A(x)$ we assign a formula $A^+(x)$ (with respect to a fixed variable $x$) by

$$A^+(x) := \forall_y(\forall_{z\prec y} A(z) \to \forall_{z\prec y\oplus\omega^x} A(z)).$$

We first show

$$\text{If } A(x) \text{ is progressive, then } A^+(x) \text{ is progressive,}$$

where "$B(x)$ is *progressive*" means $\forall_x(\forall_{y\prec x}B(y) \to B(x))$. So assume that $A(x)$ is progressive and

$$\forall_{y\prec x}A^+(y). \tag{1.13}$$

We have to show $A^+(x)$. So assume further

$$\forall_{z\prec y}A(z) \tag{1.14}$$

and $z \prec y \oplus \omega^x$. We have to show $A(z)$.

*Case* $x = 0$. Then $z \prec y \oplus \omega^0$. By (1.5) it suffices to derive $A(z)$ from $z \prec y$ as well as from $z = y$. If $z \prec y$, then $A(z)$ follows from (1.14), and if $z = y$, then $A(z)$ follows from (1.14) and the progressiveness of $A(x)$.

*Case* $\mathrm{S}x$. From $z \prec y \oplus \omega^{\mathrm{S}x}$ we obtain $z \prec y \oplus \omega^{e(x,y,z)}m(x,y,z)$ by (1.11) and $e(x,y,z) \prec \mathrm{S}x$ by (1.12). From (1.13) we obtain $A^+(e(x,y,z))$. By the definition of $A^+(x)$ we get

$$\forall_{u\prec y\oplus\omega^{e(x,y,z)}v}A(u) \to \forall_{u\prec(y\oplus\omega^{e(x,y,z)}v)\oplus\omega^{e(x,y,z)}}A(u)$$

and hence, using (1.7) and (1.10)

$$\forall_{u\prec y\oplus\omega^{e(x,y,z)}v}A(u) \to \forall_{u\prec y\oplus\omega^{e(x,y,z)}(\mathrm{S}v)}A(u).$$

Also from (1.14) and (1.9), (1.6) we obtain

$$\forall_{u\prec y\oplus\omega^{e(x,y,z)}0}A(u).$$

Using an appropriate instance of the induction schema we can conclude

$$\forall_{u\prec y\oplus\omega^{e(x,y,z)}m(x,y,z)}A(u)$$

and hence $A(z)$.

We now show, by induction on $n$, how for an arbitrary formula $A(x)$ we can obtain a derivation of

$$\forall_x(\forall_{y\prec x}A(y) \to A(x)) \to \forall_{x\prec\omega_n}A(x).$$

So assume the left hand side, i.e., assume that $A(x)$ is progressive.

*Case* 0. Then $x \prec \omega^0$ and hence $x \prec 0 \oplus \omega^0$ by (1.8). By (1.5) it suffices to derive $A(x)$ from $x \prec 0$ as well as from $x = 0$. Now $x \prec 0 \to A(x)$ holds by (1.4), and $A(0)$ then follows from the progressiveness of $A(x)$.

*Case* $n+1$. Since $A(x)$ is progressive, by what we have shown above $A^+(x)$ is also progressive. Applying the induction hypothesis to $A^+(x)$ yields $\forall_{x\prec\omega_n} A^+(x)$, and hence $A^+(\omega_n)$ by the progressiveness of $A^+(x)$. Now the definition of $A^+(x)$ (together with (1.4) and (1.8)) yields $\forall_{z\prec\omega^{\omega_n}} A(z)$. □

Note that in the induction step of this proof we have derived transfinite induction up to $\omega_{n+1}$ for $A(x)$ from transfinite induction up to $\omega_n$ for a formula of level higher than the level of $A(x)$. The *level* of a formula $A$ is defined by

$$\begin{aligned}
\mathrm{lev}(R\vec{t}\,) &:= 0,\\
\mathrm{lev}(A\to B) &:= \max(\mathrm{lev}(A)+1, \mathrm{lev}(B)),\\
\mathrm{lev}(\forall_x A) &:= \max(1, \mathrm{lev}(A)).
\end{aligned}$$

CHAPTER 2

# Computability in higher types

In this chapter we will develop a somewhat general view of computability theory, where not only numbers and functions appear as arguments, but also functionals of any finite type.

## 2.1. Abstract computability via information systems

There are two principles on which our notion of computability will be based: finite support and monotonicity.

It is a fundamental property of computation that evaluation must be finite. So in any evaluation of $\Phi(\varphi)$ the argument $\varphi$ can be called upon only finitely many times, and hence the value – if defined – must be determined by some finite subfunction of $\varphi$. This is the principle of finite support.

Let us carry this discussion somewhat further and look at the situation one type higher up. Let $\mathcal{H}$ be a partial functional of type-3, mapping type-2 functionals $\Phi$ to natural numbers. Suppose $\Phi$ is given and $\mathcal{H}(\Phi)$ evaluates to a defined value. Again, evaluation must be finite. Hence the argument $\Phi$ can only be called on finitely many functions $\varphi$. Furthermore each such $\varphi$ must be presented to $\Phi$ in a finite form (explicitly say, as a set of ordered pairs). In other words, $\mathcal{H}$ and also any type-2 argument $\Phi$ supplied to it must satisfy the finite support principle, and this must continue to apply as we move up through the types.

To describe this principle more precisely, we need to introduce the notion of a "finite approximation" $\Phi_0$ of a functional $\Phi$. By this we mean a finite set $X$ of pairs $(\varphi_0, n)$ such that (i) $\varphi_0$ is a finite function, (ii) $\Phi(\varphi_0)$ is defined with value $n$, and (iii) if $(\varphi_0, n)$ and $(\varphi_0', n')$ belong to $X$ where $\varphi_0$ and $\varphi_0'$ are "consistent", then $n = n'$. The essential idea here is that $\Phi$ should be viewed as the union of all its finite approximations. Using this notion of a finite approximation we can now formulate the

> *Principle of finite support.* If $\mathcal{H}(\Phi)$ is defined with value $n$, then there is a finite approximation $\Phi_0$ of $\Phi$ such that $\mathcal{H}(\Phi_0)$ is defined with value $n$.

The monotonicity principle formalizes the simple idea that once $\mathcal{H}(\Phi)$ is evaluated, then the same value will be obtained no matter how the argument $\Phi$ is extended. This requires the notion of "extension". $\Phi'$ extends $\Phi$ if for any piece of data $(\varphi_0, n)$ in $\Phi$ there exists another $(\varphi_0', n)$ in $\Phi'$ such that $\varphi_0$ extends $\varphi_0'$ (note the contravariance!). The second basic principle is then

> *Monotonicity principle*. If $\mathcal{H}(\Phi)$ is defined with value $n$ and $\Phi'$ extends $\Phi$, then also $\mathcal{H}(\Phi')$ is defined with value $n$.

An immediate consequence of finite support and monotonicity is that the behaviour of any functional is indeed determined by its set of finite approximations. For if $\Phi$, $\Phi'$ have the same finite approximations and $\mathcal{H}(\Phi)$ is defined with value $n$, then by finite support, $\mathcal{H}(\Phi_0)$ is defined with value $n$ for some finite approximation $\Phi_0$, and then by monotonicity $\mathcal{H}(\Phi')$ is defined with value $n$. Thus $\mathcal{H}(\Phi) = \mathcal{H}(\Phi')$, for all $\mathcal{H}$.

This observation now allows us to formulate a notion of abstract computability:

> *Effectivity principle*. An object is computable just in case its set of finite approximations is (primitive) recursively enumerable (or equivalently, $\Sigma_1^0$-definable).

This is an "externally induced" notion of computability, and it is of definite interest to ask whether one can find an "internal" notion of computability coinciding with it. This can be done by means of a fixed point operator introduced into this framework by Platek; and the result mentioned is due to Plotkin (1978).

The general theory of computability concerns partial functions and partial operations on them. However, we are primarily interested in total objects, so once the theory of partial objects is developed, we can look for ways to extract the total ones. Then one can prove Kreisel's density theorem, wich says that the total functionals are dense in the space of all partial "continuous" functionals.

**2.1.1. Information systems.** The basic idea of information systems is to provide an axiomatic setting to describe approximations of abstract objects (like functions or functionals) by concrete, finite ones. We do not attempt to analyze the notion of "concreteness" or finiteness here, but rather take an arbitrary countable set $A$ of "bits of data" or "tokens" as a basic notion to be explained axiomatically. In order to use such data to build approximations of abstract objects, we need a notion of "consistency", which determines when the elements of a finite set of tokens are consistent with

each other. We also need an "entailment relation" between consistent sets $U$ of data and single tokens $a$, which intuitively expresses the fact that the information contained in $U$ is sufficient to compute the bit of information $a$. The axioms below are a minor modification of Scott's (1982), due to Larsen and Winskel (1991).

DEFINITION. An *information system* is a structure $(A, \mathrm{Con}, \vdash)$ where $A$ is a countable set (the *tokens*), Con is a non-empty set of finite subsets of $A$ (the *consistent* sets) and $\vdash$ is a subset of $\mathrm{Con} \times A$ (the *entailment* relation), which satisfy

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

The elements of Con are called *formal neighborhoods*. We use $U, V, W$ to denote *finite* sets, and write

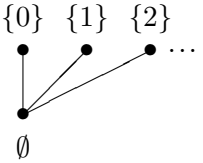
$$\begin{aligned}
&U \vdash V \quad \text{for} \quad U \in \mathrm{Con} \wedge \forall_{a \in V}(U \vdash a),\\
&a \uparrow b \quad \text{for} \quad \{a, b\} \in \mathrm{Con} \qquad (a, b \text{ are } \textit{consistent}),\\
&U \uparrow V \quad \text{for} \quad \forall_{a \in U, b \in V}(a \uparrow b).
\end{aligned}$$

DEFINITION. The *ideals* (also called *objects*) of an information system $\boldsymbol{A} = (A, \mathrm{Con}, \vdash)$ are defined to be those subsets $x$ of $A$ which satisfy

$$\begin{aligned}
&U \subseteq x \to U \in \mathrm{Con} \quad (x \text{ is } \textit{consistent}),\\
&x \supseteq U \vdash a \to a \in x \quad (x \text{ is } \textit{deductively closed}).
\end{aligned}$$

For example the *deductive closure* $\overline{U} := \{\, a \in A \mid U \vdash a \,\}$ of $U \in \mathrm{Con}$ is an ideal. The set of all ideals of $\boldsymbol{A}$ is denoted by $|\boldsymbol{A}|$.

EXAMPLES. Every countable set $A$ can be turned into a *flat* information system by letting the set of tokens be $A$, $\mathrm{Con} := \{\emptyset\} \cup \{\, \{a\} \mid a \in A \,\}$ and $U \vdash a$ mean $a \in U$. In this case the ideals are just the elements of Con. For $A = \mathbb{N}$ we have the following picture of the Con-sets.



A rather important example is the following, which concerns approximations of functions from a countable set $A$ into a countable set $B$. The tokens are the pairs $(a, b)$ with $a \in A$ and $b \in B$, and

$$\mathrm{Con} := \{\, \{\, (a_i, b_i) \mid i < k \,\} \mid \forall_{i,j<k}(a_i = a_j \to b_i = b_j) \,\},$$
$$U \vdash (a, b) := (a, b) \in U.$$

It is not difficult to verify that this defines an information system whose ideals are (the graphs of) all partial functions from $A$ to $B$.

**2.1.2. Function spaces.** We now define the "function space" $\boldsymbol{A} \to \boldsymbol{B}$ between two information systems $\boldsymbol{A}$ and $\boldsymbol{B}$.

Definition. Let $\boldsymbol{A} = (A, \mathrm{Con}_A, \vdash_A)$ and $\boldsymbol{B} = (B, \mathrm{Con}_B, \vdash_B)$ be information systems. Define $\boldsymbol{A} \to \boldsymbol{B} = (C, \mathrm{Con}, \vdash)$ by

$$C := \mathrm{Con}_A \times B,$$
$$\{\, (U_i, b_i) \mid i \in I \,\} \in \mathrm{Con} := \forall_{J \subseteq I}(\bigcup_{j \in J} U_j \in \mathrm{Con}_A \to \{\, b_j \mid j \in J \,\} \in \mathrm{Con}_B).$$

For the definition of the entailment relation $\vdash$ it is helpful to first define the notion of an *application* of $W := \{\, (U_i, b_i) \mid i \in I \,\} \in \mathrm{Con}$ to $U \in \mathrm{Con}_A$:

$$\{\, (U_i, b_i) \mid i \in I \,\}U := \{\, b_i \mid U \vdash_A U_i \,\}.$$

From the definition of Con we know that this set is in $\mathrm{Con}_B$. Now define $W \vdash (U, b)$ by $WU \vdash_B b$.

Clearly application is *monotone in the second argument*, in the sense that $U \vdash_A U'$ implies ($WU' \subseteq WU$, hence also) $WU \vdash_B WU'$. In fact, application is also *monotone in the first argument*, i.e.,

$$W \vdash W' \quad \text{implies} \quad WU \vdash_B W'U.$$

To see this let $W = \{\, (U_i, b_i) \mid i \in I \,\}$ and $W' = \{\, (U'_j, b'_j) \mid j \in J \,\}$. By definition $W'U = \{\, b'_j \mid U \vdash_A U'_j \,\}$. Now fix $j$ such that $U \vdash_A U'_j$; we must show $WU \vdash_B b'_j$. By assumption $W \vdash (U'_j, b'_j)$, hence $WU'_j \vdash_B b'_j$. Because of $WU \supseteq WU'_j$ the claim follows.

Lemma. *If $\boldsymbol{A}$ and $\boldsymbol{B}$ are information systems, then so is $\boldsymbol{A} \to \boldsymbol{B}$ defined as above.*

Proof. Let $\boldsymbol{A} = (A, \mathrm{Con}_A, \vdash_A)$ and $\boldsymbol{B} = (B, \mathrm{Con}_B, \vdash_B)$. The first, second and fourth property of the definition are clearly satisfied. For the third, suppose

$$\{(U_1, b_1), \ldots, (U_n, b_n)\} \vdash (U, b), \quad \text{i.e.,} \quad \{\, b_j \mid U \vdash_A U_j \,\} \vdash_B b.$$

We have to show that $\{(U_1, b_1), \ldots, (U_n, b_n), (U, b)\} \in \mathrm{Con}$. So let $I \subseteq \{1, \ldots, n\}$ and suppose

$$U \cup \bigcup_{i \in I} U_i \in \mathrm{Con}_A.$$

We must show that $\{b\} \cup \{\, b_i \mid i \in I \,\} \in \mathrm{Con}_B$. Let $J \subseteq \{1, \ldots, n\}$ consist of those $j$ with $U \vdash_A U_j$. Then also

$$U \cup \bigcup_{i \in I} U_i \cup \bigcup_{j \in J} U_j \in \mathrm{Con}_A.$$

Since

$$\bigcup_{i \in I} U_i \cup \bigcup_{j \in J} U_j \in \mathrm{Con}_A,$$

from the consistency of $\{(U_1, b_1), \ldots, (U_n, b_n)\}$ we can conclude that

$$\{\, b_i \mid i \in I \,\} \cup \{\, b_j \mid j \in J \,\} \in \mathrm{Con}_B.$$

But $\{\, b_j \mid j \in J \,\} \vdash_B b$ by assumption. Hence

$$\{\, b_i \mid i \in I \,\} \cup \{\, b_j \mid j \in J \,\} \cup \{b\} \in \mathrm{Con}_B.$$

For the final property, suppose

$$W \vdash W' \quad \text{and} \quad W' \vdash (U, b).$$

We have to show $W \vdash (U, b)$, i.e., $WU \vdash_B b$. We obtain $WU \vdash_B W'U$ by monotonicity in the first argument, and $W'U \vdash b$ by definition. □

We shall now give two alternative characterizations of the function space: firstly as "approximable maps", and secondly as continuous maps w.r.t. the so-called Scott topology.

The basic idea for approximable maps is the desire to study "information respecting" maps from $\boldsymbol{A}$ into $\boldsymbol{B}$. Such a map is given by a relation $r$ between $\mathrm{Con}_A$ and $B$, where $r(U, b)$ intuitively means that whenever we are given the information $U \in \mathrm{Con}_A$, then we know that at least the token $b$ appears in the value.

Definition. Let $\boldsymbol{A} = (A, \mathrm{Con}_A, \vdash_A)$ and $\boldsymbol{B} = (B, \mathrm{Con}_B, \vdash_B)$ be information systems. A relation $r \subseteq \mathrm{Con}_A \times B$ is an *approximable map* if it satisfies the following:

(a) if $r(U, b_1), \ldots, r(U, b_n)$, then $\{b_1, \ldots, b_n\} \in \mathrm{Con}_B$;
(b) if $r(U, b_1), \ldots, r(U, b_n)$ and $\{b_1, \ldots, b_n\} \vdash_B b$, then $r(U, b)$;
(c) if $r(U', b)$ and $U \vdash_A U'$, then $r(U, b)$.

We write $r \colon \boldsymbol{A} \to \boldsymbol{B}$ to mean that $r$ is an approximable map from $\boldsymbol{A}$ to $\boldsymbol{B}$.

THEOREM. *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be information systems. Then the ideals of $\boldsymbol{A} \to \boldsymbol{B}$ are exactly the approximable maps from $\boldsymbol{A}$ to $\boldsymbol{B}$.*

PROOF. Let $\boldsymbol{A} = (A, \mathrm{Con}_A, \vdash_A)$ and $\boldsymbol{B} = (B, \mathrm{Con}_B, \vdash_B)$. If $r \in |\boldsymbol{A} \to \boldsymbol{B}|$ then $r \subseteq \mathrm{Con}_A \times B$ is consistent and deductively closed. We have to show that $r$ satisfies the axioms for approximable maps.

(a) Let $r(U, b_1), \ldots, r(U, b_n)$. We must show that $\{b_1, \ldots, b_n\} \in \mathrm{Con}_B$. But this clearly follows from the consistency of $r$.

(b) Let $r(U, b_1), \ldots, r(U, b_n)$ and $\{b_1, \ldots, b_n\} \vdash_B b$. We must show that $r(U, b)$. But

$$\{(U, b_1), \ldots, (U, b_n)\} \vdash (U, b)$$

by the definition of the entailment relation $\vdash$ in $\boldsymbol{A} \to \boldsymbol{B}$. Hence $r(U, b)$ since $r$ is deductively closed.

(c) Let $U \vdash_A U'$ and $r(U', b)$. We must show that $r(U, b)$. But

$$\{(U', b)\} \vdash (U, b)$$

since $\{(U', b)\}U = \{b\}$ (which follows from $U \vdash_A U'$). Hence $r(U, b)$, again since $r$ is deductively closed.

For the other direction suppose that $r \colon \boldsymbol{A} \to \boldsymbol{B}$ is an approximable map. We must show that $r \in |\boldsymbol{A} \to \boldsymbol{B}|$.

Consistency of $r$. Let $r(U_1, b_1), \ldots, r(U_n, b_n)$ and $U = \bigcup\{ U_i \mid i \in I \} \in \mathrm{Con}_A$ for some $I \subseteq \{1, \ldots, n\}$. We must show $\{ b_i \mid i \in I \} \in \mathrm{Con}_B$. From $r(U_i, b_i)$ and $U \vdash_A U_i$ we obtain $r(U, b_i)$ by axiom (c) for all $i \in I$, and hence $\{ b_i \mid i \in I \} \in \mathrm{Con}_B$ by axiom (a).

Deductive closure of $r$. Let $r(U_1, b_1), \ldots, r(U_n, b_n)$ and

$$W := \{(U_1, b_1), \ldots, (U_n, b_n)\} \vdash (U, b).$$

We must show $r(U, b)$. By definition of $\vdash$ for $\boldsymbol{A} \to \boldsymbol{B}$ we have $WU \vdash_B b$, which is $\{ b_i \mid U \vdash_A U_i \} \vdash_B b$. Further by our assumption $r(U_i, b_i)$ we know $r(U, b_i)$ by axiom (c) for all $i$ with $U \vdash_A U_i$. Hence $r(U, b)$ by axiom (b). □

DEFINITION. Suppose $\boldsymbol{A} = (A, \mathrm{Con}, \vdash)$ is an information system and $U \in \mathrm{Con}$. Define $\mathcal{O}_U \subseteq |\boldsymbol{A}|$ by

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

Note that, since the ideals $x \in |\boldsymbol{A}|$ are deductively closed, $x \in \mathcal{O}_U$ implies $\overline{U} \subseteq x$.

LEMMA. *The system of all $\mathcal{O}_U$ with $U \in \mathrm{Con}$ forms the basis of a topology on $|\boldsymbol{A}|$, called the Scott topology.*

PROOF. Suppose $U, V \in \mathrm{Con}$ and $x \in \mathcal{O}_U \cap \mathcal{O}_V$. We have to find $W \in \mathrm{Con}$ such that $x \in \mathcal{O}_W \subseteq \mathcal{O}_U \cap \mathcal{O}_V$. Choose $W = U \cup V$. □

LEMMA. *Let $\boldsymbol{A}$ be an information system and $\mathcal{O} \subseteq |\boldsymbol{A}|$. Then the following are equivalent.*

(a) *$\mathcal{O}$ is open in the Scott topology.*
(b) *$\mathcal{O}$ satisfies*
  (i) *If $x \in \mathcal{O}$ and $x \subseteq y$, then $y \in \mathcal{O}$ (Alexandrov condition).*
  (ii) *If $x \in \mathcal{O}$, then $\overline{U} \in \mathcal{O}$ for some $U \subseteq x$ (Scott condition).*
(c) *$\mathcal{O} = \bigcup_{\overline{U} \in \mathcal{O}} \mathcal{O}_U$.*

Hence open sets $\mathcal{O}$ may be seen as those determined by a (possibly infinite) system of finitely observable properties, namely all $U$ such that $\overline{U} \in \mathcal{O}$.

PROOF. (a) → (b). If $\mathcal{O}$ is open, then $\mathcal{O}$ is the union of some $\mathcal{O}_U$'s, $U \in \mathrm{Con}$. Since each $\mathcal{O}_U$ is upwards closed, also $\mathcal{O}$ is; this proves the Alexandrov condition. For the Scott condition assume $x \in \mathcal{O}$. Then $x \in \mathcal{O}_U \subseteq \mathcal{O}$ for some $U \in \mathrm{Con}$. Note that $\overline{U} \in \mathcal{O}_U$, hence $\overline{U} \in \mathcal{O}$, and $U \subseteq x$ since $x \in \mathcal{O}_U$.

(b) → (c). Assume that $\mathcal{O} \subseteq |\boldsymbol{A}|$ satisfies the Alexandrov and Scott conditions. Let $x \in \mathcal{O}$. By the Scott condition, $\overline{U} \in \mathcal{O}$ for some $U \subseteq x$, so $x \in \mathcal{O}_U$ for this $U$. Conversely, let $x \in \mathcal{O}_U$ for some $\overline{U} \in \mathcal{O}$. Then $\overline{U} \subseteq x$. Now $x \in \mathcal{O}$ follows from $\overline{U} \in \mathcal{O}$ by the Alexandrov condition.

(c) → (a). The $\mathcal{O}_U$'s are the basic open sets of the Scott topology. □

We now give some simple characterizations of the continuous functions $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$. Call $f$ *monotone* if $x \subseteq y$ implies $f(x) \subseteq f(y)$.

LEMMA. *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be information systems and $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$. Then the following are equivalent.*

(a) *$f$ is continuous w.r.t. the Scott topology.*
(b) *$f$ is monotone and satisfies the "principle of finite support" PFS: If $b \in f(x)$, then $b \in f(\overline{U})$ for some $U \subseteq x$.*
(c) *$f$ is monotone and commutes with directed unions: for every directed $D \subseteq |\boldsymbol{A}|$ (i.e., for any $x, y \in D$ there is a $z \in D$ such that $x, y \subseteq z$)*

$$f(\bigcup_{x \in D} x) = \bigcup_{x \in D} f(x).$$

Note that in (c) the set $\{\, f(x) \mid x \in D \,\}$ is directed by monotonicity of $f$; hence its union is indeed an ideal in $|\boldsymbol{A}|$. Note also that from PFS and monotonicity of $f$ it follows immediately that if $V \subseteq f(x)$, then $V \subseteq f(\overline{U})$ for some $U \subseteq x$.

Hence continuous maps $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ are those that can be completely described from the point of view of finite approximations of the abstract objects $x \in |\boldsymbol{A}|$ and $f(x) \in |\boldsymbol{B}|$: Whenever we are given a finite approximation $V$ to the value $f(x)$, then there is a finite approximation $U$ to the argument $x$ such that already $f(\overline{U})$ contains the information in $V$; note that by monotonicity $f(\overline{U}) \subseteq f(x)$.

PROOF. (a) $\to$ (b). Let $f$ be continuous. Then for any basic open set $\mathcal{O}_V \subseteq |\boldsymbol{B}|$ (so $V \in \mathrm{Con}_B$) the set $f^{-1}[\mathcal{O}_V] = \{\, x \mid V \subseteq f(x) \,\}$ is open in $|\boldsymbol{A}|$. To prove monotonicity assume $x \subseteq y$; we must show $f(x) \subseteq f(y)$. So let $b \in f(x)$, i.e., $\{b\} \subseteq f(x)$. The open set $f^{-1}[\mathcal{O}_{\{b\}}] = \{\, z \mid \{b\} \subseteq f(z) \,\}$ satisfies the Alexandrov condition, so from $x \subseteq y$ we can infer $\{b\} \subseteq f(y)$, i.e., $b \in f(y)$. To prove PFS assume $b \in f(x)$. The open set $\{\, z \mid \{b\} \subseteq f(z) \,\}$ satisfies the Scott condition, so for some $U \subseteq x$ we have $\{b\} \subseteq f(\overline{U})$.

(b) $\to$ (a). Assume that $f$ satisfies monotonicity and PFS. We must show that $f$ is continuous, i.e., that for any fixed $V \in \mathrm{Con}_B$ the set $f^{-1}[\mathcal{O}_V] = \{\, x \mid V \subseteq f(x) \,\}$ is open. We prove

$$\{\, x \mid V \subseteq f(x) \,\} = \bigcup \{\, \mathcal{O}_U \mid U \in \mathrm{Con}_A \text{ and } V \subseteq f(\overline{U}) \,\}.$$

Let $V \subseteq f(x)$. Then by PFS $V \subseteq f(\overline{U})$ for some $U \in \mathrm{Con}_A$ such that $U \subseteq x$, and $U \subseteq x$ implies $x \in \mathcal{O}_U$. Conversely, let $x \in \mathcal{O}_U$ for some $U \in \mathrm{Con}_A$ such that $V \subseteq f(\overline{U})$. Then $\overline{U} \subseteq x$, hence $V \subseteq f(x)$ by monotonicity.

For (b) $\leftrightarrow$ (c) assume that $f$ is monotone. Let $f$ satisfy PFS, and $D \subseteq |\boldsymbol{A}|$ be directed. $f(\bigcup_{x\in D} x) \supseteq \bigcup_{x\in D} f(x)$ follows from monotonicity. For the reverse inclusion let $b \in f(\bigcup_{x\in D} x)$. Then by PFS $b \in f(\overline{U})$ for some $U \subseteq \bigcup_{x\in D} x$. From the directedness and the fact that $U$ is finite we obtain $U \subseteq z$ for some $z \in D$. From $b \in f(\overline{U})$ and monotonicity infer $b \in f(z)$. Conversely, let $f$ commute with directed unions, and assume $b \in f(x)$. Then

$$b \in f(x) = f(\bigcup_{U\subseteq x} \overline{U}) = \bigcup_{U\subseteq x} f(\overline{U}),$$

hence $b \in f(\overline{U})$ for some $U \subseteq x$. □

Clearly the identity and constant functions are continuous, and also the *composition* $g \circ f$ of continuous functions $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ and $g\colon |\boldsymbol{B}| \to |\boldsymbol{C}|$.

THEOREM. *Let $\boldsymbol{A}$ and $\boldsymbol{B} = (B, \mathrm{Con}_B, \vdash_B)$ be information systems. Then the ideals of $\boldsymbol{A} \to \boldsymbol{B}$ are in a natural bijective correspondence with the continuous functions from $|\boldsymbol{A}|$ to $|\boldsymbol{B}|$, as follows.*

(a) *With any approximable map* $r\colon \boldsymbol{A} \to \boldsymbol{B}$ *we can associate a continuous function* $|r|\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ *by*

$$|r|(z) := \{\, b \in B \mid r(U,b) \text{ for some } U \subseteq z \,\}.$$

*We call* $|r|(z)$ *the* application *of* $r$ *to* $z$.

(b) *Conversely, with any continuous function* $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ *we can associate an approximable map* $\hat{f}\colon \boldsymbol{A} \to \boldsymbol{B}$ *by*

$$\hat{f}(U,b) := (b \in f(\overline{U})).$$

*These assignments are inverse to each other, i.e.,* $f = |\hat{f}|$ *and* $r = \widehat{|r|}$.

Proof. Let $r$ be an ideal of $\boldsymbol{A} \to \boldsymbol{B}$; then by the theorem just proved $r$ is an approximable map. We first show that $|r|$ is well-defined. So let $z \in |\boldsymbol{A}|$.

$|r|(z)$ is consistent: let $b_1, \dots, b_n \in |r|(z)$. Then there are $U_1, \dots, U_n \subseteq z$ such that $r(U_i, b_i)$. Hence $U := U_1 \cup \dots \cup U_n \subseteq z$ and $r(U, b_i)$ by axiom (c) of approximable maps. Now from axiom (a) we can conclude that $\{b_1, \dots, b_n\} \in \mathrm{Con}_B$.

$|r|(z)$ is deductively closed: let $b_1, \dots, b_n \in |r|(z)$ and $\{b_1, \dots, b_n\} \vdash_B b$. We must show $b \in |r|(z)$. As before we find $U \subseteq z$ such that $r(U, b_i)$. Now from axiom (b) we can conclude $r(U, b)$ and hence $b \in |r|(z)$.

Continuity of $|r|$ follows immediately from part (b) of the lemma above, since by definition $|r|$ is monotone and satisfies PFS.

Now let $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ be continuous. It is easy to verify that $\hat{f}$ is indeed an approximable map. Furthermore

$$\begin{aligned} b \in |\hat{f}|(z) &\leftrightarrow \hat{f}(U,b) && \text{for some } U \subseteq z \\ &\leftrightarrow b \in f(\overline{U}) && \text{for some } U \subseteq z \\ &\leftrightarrow b \in f(z) && \text{by monotonicity and PFS.} \end{aligned}$$

Finally, for any approximable map $r\colon \boldsymbol{A} \to \boldsymbol{B}$ we have

$$\begin{aligned} r(U,b) &\leftrightarrow \exists_{V \subseteq \overline{U}} r(V,b) \quad \text{by axiom (c) for approximable maps} \\ &\leftrightarrow b \in |r|(\overline{U}) \\ &\leftrightarrow \widehat{|r|}(U,b), \end{aligned}$$

so $r = \widehat{|r|}$. □

Moreover, one can easily check that

$$r \circ s := \{\, (U,c) \mid \exists_V ((U,V) \subseteq s \wedge (V,c) \in r) \,\}$$

is an approximable map (where $(U,V) := \{\, (U,b) \mid b \in V \,\}$), and

$$|r \circ s| = |r| \circ |s|, \quad \widehat{f \circ g} = \hat{f} \circ \hat{g}.$$

We usually write $r(z)$ for $|r|(z)$, and similarly $f(U,b)$ for $\hat{f}(U,b)$. It should always be clear from the context where the mods and hats should be inserted.

**2.1.3. Algebras and types.** We now consider concrete information systems, our basis for continuous functionals.

Types will be built from base types by the formation of function types, $\rho \to \sigma$. As domains for the base types we choose non-flat and possibly infinitary free algebras, given by their constructors. The main reason for taking non-flat base domains is that we want the constructors to be injective and with disjoint ranges. This generally is not the case for flat domains.

We inductively define *type forms*

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

with $\alpha, \xi$ type variables and $k \geq 1$ (since we want our algebras to be inhabited). Note that $(\rho_\nu)_{\nu<n} \to \sigma$ means $\rho_0 \to \ldots \to \rho_{n-1} \to \sigma$, associated to the right.

Let $\mathrm{FV}(\rho)$ denote the set of type variables free in $\rho$. We define $\mathrm{SP}(\alpha,\rho)$ "$\alpha$ occurs at most *strictly positive* in $\rho$" by induction on $\rho$.

$$\mathrm{SP}(\alpha,\beta) \qquad \frac{\alpha \notin \mathrm{FV}(\rho) \quad \mathrm{SP}(\alpha,\sigma)}{\mathrm{SP}(\alpha,\rho\to\sigma)} \qquad \frac{\mathrm{SP}(\alpha,\rho_{i\nu}) \text{ for all } i<k,\, \nu<n_i}{\mathrm{SP}(\alpha,\mu_\xi((\rho_{i\nu})_{\nu<n_i}\to\xi)_{i<k})}$$

Now we can define $\mathrm{Ty}(\rho)$ "$\rho$ is a *type*", again by induction on $\rho$.

$$\mathrm{Ty}(\alpha) \qquad \frac{\mathrm{Ty}(\rho) \quad \mathrm{Ty}(\sigma)}{\mathrm{Ty}(\rho\to\sigma)}$$

$$\frac{\mathrm{Ty}(\rho_{i\nu}) \text{ and } \mathrm{SP}(\xi,\rho_{i\nu}) \text{ for all } i<k,\, \nu<n_i \qquad \xi \notin \mathrm{FV}(\rho_{0\nu}) \text{ for all } \nu<n_0}{\mathrm{Ty}(\mu_\xi((\rho_{i\nu})_{\nu<n_i}\to\xi)_{i<k})}$$

We call

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

an *algebra*. Sometimes it is helpful to display the type parameters and write $\iota(\vec{\alpha},\vec{\beta}\,)$, where $\vec{\alpha},\vec{\beta}$ are all type variables except $\xi$ free in some $\rho_{i\nu}$, and $\vec{\alpha}$ are the ones occuring only strictly positive. If we write the $i$-th component of $\iota$ in the form $(\rho_\nu(\xi))_{\nu<n} \to \xi$, then we call

$$(\rho_\nu(\iota))_{\nu<n} \to \iota$$

the $i$-th *constructor type* of $\iota$.

In $(\rho_\nu(\xi))_{\nu<n} \to \xi$ we call $\rho_\nu(\xi)$ a *parameter* argument type if $\xi$ does not occur in it, and a *recursive* argument type otherwise. A recursive argument type $\rho_\nu(\xi)$ is *nested* if it has an occurrence of $\xi$ in a strictly positive parameter position of another (previously defined) algebra, and unnested otherwise. An algebra $\iota$ is called *nested* if it has a constructor with at least one nested recursive argument type, and *unnested* otherwise.

Every type $\rho$ should have a *total inhabitant*, i.e., a closed term of this type built solely from constructors, variables and assumed total inhabitants of some of its (type) variables. To ensure this we have required that for every algebra $\mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k}$ the initial $(\rho_{0\nu})_{\nu<n_0} \to \xi$ has no recursive argument types. Note that it might not be necessary to actually use assumed total inhabitants for all variables of a type. An example is the list type $\mathbf{L}(\alpha)$, which has the Nil constructor as a total inhabitant. However, for the type $\mathbf{L}(\alpha)^+$ $(:= \mu_\xi(\alpha \to \xi, \alpha \to \xi \to \xi))$ we need to assume a total inhabitant of $\alpha$.

Here are some examples of algebras.

$$\begin{aligned}
\mathbf{U} &:= \mu_\xi \xi \qquad \text{(unit)},\\
\mathbf{B} &:= \mu_\xi(\xi, \xi) \qquad \text{(booleans)},\\
\mathbf{N} &:= \mu_\xi(\xi, \xi \to \xi) \qquad \text{(natural numbers, unary)},\\
\mathbf{P} &:= \mu_\xi(\xi, \xi \to \xi, \xi \to \xi) \qquad \text{(positive numbers, binary)},\\
\mathbf{D} &:= \mu_\xi(\xi, \xi \to \xi \to \xi) \qquad \text{(binary trees, or derivations)},\\
\mathbf{O} &:= \mu_\xi(\xi, \xi \to \xi, (\mathbf{N} \to \xi) \to \xi) \qquad \text{(ordinals)},\\
\mathbf{T}_0 &:= \mathbf{N}, \quad \mathbf{T}_{n+1} := \mu_\xi(\xi, (\mathbf{T}_n \to \xi) \to \xi) \qquad \text{(trees)}.
\end{aligned}$$

Examples of algebras strictly positive in their type parameters are

$$\begin{aligned}
\mathbf{L}(\alpha) &:= \mu_\xi(\xi, \alpha \to \xi \to \xi) \qquad \text{(lists)},\\
\alpha \times \beta &:= \mu_\xi(\alpha \to \beta \to \xi) \qquad \text{(product)},\\
\alpha + \beta &:= \mu_\xi(\alpha \to \xi, \beta \to \xi) \qquad \text{(sum)}.
\end{aligned}$$

An example of a nested algebra is

$$\mathbf{T} := \mu_\xi(\mathbf{L}(\xi) \to \xi) \quad \text{(finitely branching trees)}.$$

Note that $\mathbf{T}$ has a total inhabitant since $\mathbf{L}(\alpha)$ has one (given by the Nil constructor).

Let $\rho$ be a type; we write $\rho(\vec{\alpha}\,)$ for $\rho$ to indicate its dependence on the type parameters $\vec{\alpha}$. We can substitute types $\vec{\sigma}$ for $\vec{\alpha}$, to obtain $\rho(\vec{\sigma}\,)$. Examples are $\mathbf{L}(\mathbf{B})$, the type of lists of booleans, and $\mathbf{N} \times \mathbf{N}$, the type of pairs of natural numbers.

Note that often there are many equivalent ways to define a particular type. For instance, we could take $\mathbf{U}+\mathbf{U}$ to be the type of booleans, $\mathbf{L}(\mathbf{U})$ to be the type of natural numbers, and $\mathbf{L}(\mathbf{B})$ to be the type of positive binary numbers.

For every constructor type of an algebra we provide a (typed) *constructor symbol* $\mathrm{C}_i$. In some cases they have standard names, for instance

$\mathrm{tt}^{\mathbf{B}}, \mathrm{ff}^{\mathbf{B}}$ for the two constructors of the type $\mathbf{B}$ of booleans,

$0^{\mathbf{N}}, \mathrm{S}^{\mathbf{N}\to\mathbf{N}}$ for the type $\mathbf{N}$ of (unary) natural numbers,

$1^{\mathbf{P}}, \mathrm{S}_0^{\mathbf{P}\to\mathbf{P}}, \mathrm{S}_1^{\mathbf{P}\to\mathbf{P}}$ for the type $\mathbf{P}$ of (binary) positive numbers,

$\mathrm{Nil}^{\mathbf{L}(\rho)}, \mathrm{Cons}^{\rho\to\mathbf{L}(\rho)\to\mathbf{L}(\rho)}$ for the type $\mathbf{L}(\rho)$ of lists,

$(\mathrm{Inl}_{\rho\sigma})^{\rho\to\rho+\sigma}, (\mathrm{Inr}_{\rho\sigma})^{\sigma\to\rho+\sigma}$ for the sum type $\rho+\sigma$,

$\mathrm{Branch}\colon \mathbf{L}(\mathbf{T})\to\mathbf{T}$ for the type $\mathbf{T}$ of finitely branching trees.

An algebra form $\iota$ is *structure-finitary* if all its argument types $\rho_{i\nu}$ are not of arrow form. It is *finitary* if in addition it has no type variables. In the examples above $\mathbf{U}$, $\mathbf{B}$, $\mathbf{N}$, $\mathbf{P}$ and $\mathbf{D}$ are all finitary, but $\mathbf{O}$ and $\mathbf{T}_{n+1}$ are not. $\mathbf{L}(\rho)$, $\rho\times\sigma$ and $\rho+\sigma$ are structure-finitary, and finitary if their parameter types are. The nested algebra $\mathbf{T}$ above is finitary.

An algebra is *explicit* if all its constructor types have parameter argument types only (i.e., no recursive argument types). In the examples above $\mathbf{U}$, $\mathbf{B}$, $\rho\times\sigma$ and $\rho+\sigma$ are explicit, but $\mathbf{N}$, $\mathbf{P}$, $\mathbf{L}(\rho)$, $\mathbf{D}$, $\mathbf{O}$, $\mathbf{T}_{n+1}$ and $\mathbf{T}$ are not.

We will also need the notion of the *level* of a type, which is defined by

$$\mathrm{lev}(\iota) := 0, \qquad \mathrm{lev}(\rho\to\sigma) := \max\{\mathrm{lev}(\sigma), 1+\mathrm{lev}(\rho)\}.$$

*Base* types are types of level 0, and a *higher* type has level at least 1.

**2.1.4. Partial continuous functionals.** For every type $\rho$ we define the information system $\boldsymbol{C}_\rho = (C_\rho, \mathrm{Con}_\rho, \vdash_\rho)$. The ideals $x\in|\boldsymbol{C}_\rho|$ are the *partial continuous functionals* of type $\rho$. Since we will have $\boldsymbol{C}_{\rho\to\sigma} = \boldsymbol{C}_\rho\to\boldsymbol{C}_\sigma$, the partial continuous functionals of type $\rho\to\sigma$ will correspond to the continuous functions from $|\boldsymbol{C}_\rho|$ to $|\boldsymbol{C}_\sigma|$ w.r.t. the Scott topology. It will not be possible to define $\boldsymbol{C}_\rho$ by recursion on the type $\rho$, since we allow algebras with constructors having function arguments (like $\mathbf{O}$ and Sup). Instead, we shall use recursion on the "height" of the notions involved, defined below.

Definition (Information system of type $\rho$). We simultaneously define $C_\iota$, $C_{\rho\to\sigma}$, $\mathrm{Con}_\iota$ and $\mathrm{Con}_{\rho\to\sigma}$.

(a) The *tokens* $a \in C_\iota$ are the type correct constructor expressions $\mathrm{C}a_1^* \dots a_n^*$ where $a_i^*$ is an *extended token*, i.e., a token or the special symbol $*$ which carries no information.
(b) The tokens in $C_{\rho\to\sigma}$ are the pairs $(U, b)$ with $U \in \mathrm{Con}_\rho$ and $b \in C_\sigma$.
(c) A finite set $U$ of tokens in $C_\iota$ is *consistent* (i.e., $\in \mathrm{Con}_\iota$) if all its elements start with the same constructor C, say of arity $\tau_1 \to \dots \to \tau_n \to \iota$, and all $U_i \in \mathrm{Con}_{\tau_i}$ for $i = 1, \dots, n$, where $U_i$ consists of all (proper) tokens at the $i$-th argument position of some token in $U = \{\mathrm{C}\vec{a_1^*}, \dots, \mathrm{C}\vec{a_m^*}\}$.
(d) $\{\, (U_i, b_i) \mid i \in I \,\} \in \mathrm{Con}_{\rho\to\sigma}$ is defined to mean $\forall_{J\subseteq I}(\bigcup_{j\in J} U_j \in \mathrm{Con}_\rho \to \{\, b_j \mid j \in J \,\} \in \mathrm{Con}_\sigma)$.

Building on this definition, we define $U \vdash_\rho a$ for $U \in \mathrm{Con}_\rho$ and $a \in C_\rho$.

(e) $\{\mathrm{C}\vec{a_1^*}, \dots, \mathrm{C}\vec{a_m^*}\} \vdash_\iota \mathrm{C}'\vec{a^*}$ is defined to mean $\mathrm{C} = \mathrm{C}'$, $m \geq 1$ and $U_i \vdash a_i^*$, with $U_i$ as in (c) above (and $U \vdash *$ taken to be true).
(f) $W \vdash_{\rho\to\sigma} (U, b)$ is defined to mean $WU \vdash_\sigma b$, where application $WU$ of $W = \{\, (U_i, b_i) \mid i \in I \,\} \in \mathrm{Con}_{\rho\to\sigma}$ to $U \in \mathrm{Con}_\rho$ is defined to be $\{\, b_i \mid U \vdash_\rho U_i \,\}$; recall that $U \vdash V$ abbreviates $\forall_{a\in V}(U \vdash a)$.

If we define the *height* of the syntactic expressions involved by

$$|\mathrm{C}a_1^* \dots a_n^*| := 1 + \max\{\, |a_i^*| \mid i = 1, \dots, n \,\}, \qquad |*| := 0,$$
$$|(U, b)| := \max\{1 + |U|, 1 + |b|\},$$
$$|\{\, a_i \mid i \in I \,\}| := \max\{\, 1 + |a_i| \mid i \in I \,\},$$
$$|U \vdash a| := \max\{1 + |U|, 1 + |a|\},$$

these are definitions by recursion on the height.

It is easy to see that $(C_\rho, \mathrm{Con}_\rho, \vdash_\rho)$ is an information system. Observe that all the notions involved are computable: $a \in C_\rho$, $U \in \mathrm{Con}_\rho$ and $U \vdash_\rho a$.

Definition (Partial continuous functionals). For every type $\rho$ let $\boldsymbol{C}_\rho$ be the information system $(C_\rho, \mathrm{Con}_\rho, \vdash_\rho)$. The set $|\boldsymbol{C}_\rho|$ of ideals in $\boldsymbol{C}_\rho$ is the set of *partial continuous functionals* of type $\rho$. A partial continuous functional $x \in |\boldsymbol{C}_\rho|$ is *computable* if it is recursively enumerable when viewed as a set of tokens.

Notice that $\boldsymbol{C}_{\rho\to\sigma} = \boldsymbol{C}_\rho \to \boldsymbol{C}_\sigma$ as defined generally for information systems.

For example, the tokens for the algebra $\mathbf{N}$ are shown in Figure 1. For tokens $a, b$ we have $\{a\} \vdash b$ if and only if there is a path from $a$ (up) to $b$ (down). As another (more typical) example, consider the algebra $\mathbf{D}$ of derivations with a nullary constructor 0 and a binary C. Then $\{\mathrm{C}0*, \mathrm{C}*0\}$ is consistent, and $\{\mathrm{C}0*, \mathrm{C}*0\} \vdash \mathrm{C}00$.

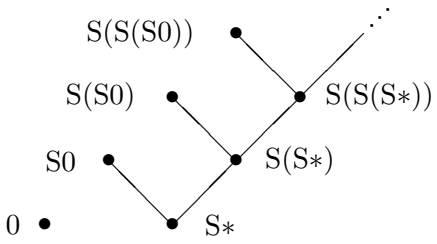



FIGURE 1. Tokens and entailment for **N**

**2.1.5. Constructors as continuous functions.** Let $\iota$ be an algebra. Every constructor C generates the following ideal in the function space:

$$r_{\mathrm{C}} := \{\, (\vec{U}, \mathrm{C}\vec{a^*}\,) \mid \vec{U} \vdash \vec{a^*}\,\}.$$

Here $(\vec{U}, a)$ abbreviates $(U_1, (U_2, \dots (U_n, a) \dots))$.

According to the general definition of a continuous function associated to an ideal in a function space the continuous map $|r_{\mathrm{C}}|$ satisfies

$$|r_{\mathrm{C}}|(\vec{x}\,) = \{\, \mathrm{C}\vec{a^*} \mid \exists_{\vec{U} \subseteq \vec{x}}(\vec{U} \vdash \vec{a^*})\,\}.$$

An immediate consequence is that the (continuous maps corresponding to) constructors are injective and their ranges are disjoint, which is what we wanted to achieve by associating non-flat rather than flat information systems with algebras.

LEMMA (Constructors are injective and have disjoint ranges). *Let $\iota$ be an algebra and* C *be a constructor of $\iota$. Then*

$$|r_{\mathrm{C}}|(\vec{x}\,) \subseteq |r_{\mathrm{C}}|(\vec{y}\,) \leftrightarrow \vec{x} \subseteq \vec{y}.$$

*If* $\mathrm{C}_1, \mathrm{C}_2$ *are distinct constructors of $\iota$, then* $|r_{\mathrm{C}_1}|(\vec{x}\,) \neq |r_{\mathrm{C}_2}|(\vec{y}\,)$*, since the two ideals are non-empty and disjoint.*

PROOF. Immediate from the definitions. □

REMARK. Notice that neither property holds for flat information systems, since for them, by monotonicity, constructors need to be *strict* (i.e., if one argument is the empty ideal, then the value is as well). But then we have

$$|r_{\mathrm{C}}|(\emptyset, y) = \emptyset = |r_{\mathrm{C}}|(x, \emptyset),$$
$$|r_{\mathrm{C}_1}|(\emptyset) = \emptyset = |r_{\mathrm{C}_2}|(\emptyset)$$

where in the first case we have one binary and, in the second, two unary constructors.

**2.1.6. Total and cototal ideals in a finitary algebra.** In the information system $\boldsymbol{C}_\iota$ associated with an algebra $\iota$, the "total" and "cototal" ideals are of special interest. Here we give an explicit definition for finitary algebras. For general algebras totality can be defined inductively and cototality coinductively (cf. 3.2.4).

Recall that a token in $\iota$ is a constructor tree $P$ possibly containing the special symbol $*$. Because of the possibility of parameter arguments we need to distinguish between "structure-" and "fully" total and cototal ideals. For the definition it is easiest to refer to a constructor tree $P(*)$ with a distinguished occurrence of $*$. This occurrence is called *non-parametric* if the path from it to the root does not pass through a parameter argument of a constructor. For a constructor tree $P(*)$, an arbitrary $P(\mathrm{C}\vec{a^*})$ is called *one-step extension* of $P(*)$, written $P(\mathrm{C}\vec{a^*}) \succ_1 P(*)$.

Definition. Let $\iota$ be an algebra, and $\boldsymbol{C}_\iota$ its associated information system. An ideal $x \in |\boldsymbol{C}_\iota|$ is *cototal* if every constructor tree $P(*) \in x$ has a $\succ_1$-predecessor $P(\mathrm{C}\vec{*}) \in x$; it is called *total* if it is cototal and the relation $\succ_1$ on $x$ is well-founded. It is called *structure-cototal* (*structure-total*) if the same holds with $\succ_1$ defined w.r.t. $P(*)$ with a non-parametric distinguished occurrence of $*$.

If there are no parameter arguments, we shall simply speak of total and cototal ideals. For example, for the algebra $\mathbf{N}$ every total ideal is the deductive closure of a token $\mathrm{S}(\mathrm{S}\ldots(\mathrm{S0})\ldots)$, and the set of all tokens $\mathrm{S}(\mathrm{S}\ldots(\mathrm{S}*)\ldots)$ is a cototal ideal. For the algebra $\mathbf{L}(\mathbf{N})$ of lists of natural numbers the total ideals are the finite lists and the cototal ones the finite or infinite lists. For the algebra $\mathbf{D}$ of derivations the total ideals can be viewed as the finite derivations, and the cototal ones as the finite or infinite "locally correct" derivations of Mints (1978); arbitrary ideals can be viewed as "partial" or "incomplete" derivations, with "holes".

Remark. From a categorical perspective (as in Hagino (1987); Rutten (2000)) finite lists of natural numbers can be seen as making up the initial algebra of the functor $TX = 1 + (\mathbf{N} \times X)$, and infinite lists (or streams) of natural numbers as making up the terminal coalgebra of the functor $TX = \mathbf{N} \times X$. In the present setting both finite and infinite lists of natural numbers appear as cototal ideals in the algebra $\mathbf{L}(\mathbf{N})$, with the finite ones the total ideals. However, to properly deal with computability we need to accommodate partiality, and hence there are more ideals in the algebra $\mathbf{L}(\mathbf{N})$.

## 2.2. Denotational and operational semantics

For every type $\rho$, we have defined what a partial continuous functional of type $\rho$ is: an ideal consisting of tokens at this type. These tokens or rather the formal neighborhoods formed from them are syntactic in nature; they are reminiscent to Kreisel's "formal neighborhoods" (Kreisel, 1959; Martin-Löf, 1983; Coquand and Spiwack, 2006). However – in contrast to Martin-Löf (1983) – we do not have to deal separately with a notion of consistency for formal neighborhoods: this concept is built into information systems.

Let us now turn our attention to a formal (functional programming) language, in the style of Plotkin's PCF (1977), and see how we can provide a denotational semantics (that is, a "meaning") for the terms of this language. A closed term $M$ of type $\rho$ will denote a partial continuous functional of this type, that is, a consistent and deductively closed set of tokens of type $\rho$. We will define this set inductively.

It will turn out that these sets are recursively enumerable. In this sense every closed term $M$ of type $\rho$ denotes a computable partial continuous functional of type $\rho$. However, it is not a good idea to *define* a computable functional in this way, by providing a recursive enumeration of its tokens. We rather want to be able to use recursion equations for such definitions. Therefore we extend the term language by constants $D$ defined by certain "computation rules", as in (Berger et al., 2003; Berger, 2005). Our semantics will cover these as well. The resulting term system can be seen as a common extension of Gödel's T (1958) and Plotkin's PCF; we call it $\mathrm{T}^+$.

**2.2.1. Structural recursion operators and Gödel's T.** We begin with a discussion of particularly important examples of such constants $D$, the (structural) higher type *recursion operators* $\mathcal{R}^\tau_\iota$ introduced by Hilbert (1925) and Gödel (1958). They are used to construct maps from the algebra $\iota$ to $\tau$, by recursion on the structure of $\iota$. For instance, $\mathcal{R}^\tau_{\mathbf{N}}$ has type $\mathbf{N} \to \tau \to (\mathbf{N} \to \tau \to \tau) \to \tau$. The first argument is the recursion argument, the second one gives the base value, and the third one gives the step function, mapping the recursion argument and the previous value to the next value. For example, $\mathcal{R}^{\mathbf{N}}_{\mathbf{N}} nm\lambda_{n,p}(\mathrm{S}p)$ defines addition $m + n$ by recursion on $n$. For $\lambda_{n,p}(\mathrm{S}p)$ we often write $\lambda_{\_,p}(\mathrm{S}p)$ since the bound variable $n$ is not used.

Generally, we define the type of the recursion operator $\mathcal{R}^\tau_\iota$ for the algebra $\iota = \mu_\xi((\rho_{i\nu}(\xi))_{\nu<n_i} \to \xi)_{i<k}$ and result type $\tau$ to be

$$\iota \to ((\rho_{i\nu}(\iota \times \tau))_{\nu<n_i} \to \tau)_{i<k} \to \tau.$$

Here $\iota$ is the type of the recursion argument, and each $(\rho_{i\nu}(\iota\times\tau))_{\nu<n_i}\to\tau$ is called a *step type*. Usage of $\iota\times\tau$ rather than $\tau$ in the step types can be seen as a "strengthening", since then one has more data available to construct the value of type $\tau$. Moreover, for unnested recursive argument types $\vec{\sigma}\to\tau$ we avoid the product type in $\vec{\sigma}\to\iota\times\tau$ and take the two argument types $\vec{\sigma}\to\iota$ and $\vec{\sigma}\to\tau$ instead ("duplication").

For some algebras we spell out the type of their recursion operators:

$$\mathcal{R}^\tau_{\mathbf{B}}\colon \mathbf{B}\to\tau\to\tau\to\tau,$$
$$\mathcal{R}^\tau_{\mathbf{N}}\colon \mathbf{N}\to\tau\to(\mathbf{N}\to\tau\to\tau)\to\tau,$$
$$\mathcal{R}^\tau_{\mathbf{P}}\colon \mathbf{P}\to\tau\to(\mathbf{P}\to\tau\to\tau)\to(\mathbf{P}\to\tau\to\tau)\to\tau,$$
$$\mathcal{R}^\tau_{\mathbf{D}}\colon \mathbf{D}\to\tau\to(\mathbf{D}\to\tau\to\mathbf{D}\to\tau\to\tau)\to\tau,$$
$$\mathcal{R}^\tau_{\mathbf{O}}\colon \mathbf{O}\to\tau\to(\mathbf{O}\to\tau\to\tau)\to((\mathbf{N}\to\mathbf{O})\to(\mathbf{N}\to\tau)\to\tau)\to\tau,$$
$$\mathcal{R}^\tau_{\mathbf{L}(\rho)}\colon \mathbf{L}(\rho)\to\tau\to(\rho\to\mathbf{L}(\rho)\to\tau\to\tau)\to\tau,$$
$$\mathcal{R}^\tau_{\rho+\sigma}\colon \rho+\sigma\to(\rho\to\tau)\to(\sigma\to\tau)\to\tau,$$
$$\mathcal{R}^\tau_{\rho\times\sigma}\colon \rho\times\sigma\to(\rho\to\sigma\to\tau)\to\tau,$$
$$\mathcal{R}^\tau_{\mathbf{T}}\colon \mathbf{T}\to(\mathbf{L}(\mathbf{T}\times\tau)\to\tau)\to\tau.$$

There is an important variant of recursion, where no recursive calls occur. This variant is called the *cases operator*; it distinguishes cases according to the outer constructor form. For the algebra $\iota=\mu_\xi((\rho_{i\nu}(\xi))_{\nu<n_i}\to\xi)_{i<k}$ and result type $\tau$ the type of the cases operator $\mathcal{C}^\tau_\iota$ is

$$\iota\to((\rho_{i\nu}(\iota))_{\nu<n_i}\to\tau)_{i<k}\to\tau.$$

The simplest example (for type $\mathbf{B}$) is *if-then-else*. Another example is

$$\mathcal{C}^\tau_{\mathbf{N}}\colon \mathbf{N}\to\tau\to(\mathbf{N}\to\tau)\to\tau.$$

It can be used to define the *predecessor* function on $\mathbf{N}$, i.e., $\mathrm{P}0:=0$ and $\mathrm{P}(\mathrm{S}n):=n$, by the term

$$\mathrm{P}m:=\mathcal{C}^{\mathbf{N}}_{\mathbf{N}}m0(\lambda_n n).$$

Remark. When computing the value of a cases term, we do not want to (eagerly) evaluate all arguments, but rather compute the test argument first and depending on the result (lazily) evaluate at most one of the other arguments. This phenomenon is well known in functional languages; for instance, in Scheme the `if`-construct is called a *special form* (as opposed to an operator). Therefore instead of taking the cases operator applied to a full list of arguments, one rather uses a `case`-construct to build this term; it differs from the former only in that it employs lazy evaluation. Hence the

predecessor function is written in the form $[\mathbf{case}\ m\ \mathbf{of}\ 0 \mid \lambda_n n]$. If there are exactly two cases, we also write $\lambda_m[\mathbf{if}\ m\ \mathbf{then}\ 0\ \mathbf{else}\ \lambda_n n]$ instead.

We shall also need *map operators*. Let $\rho(\vec{\alpha}\,)$ be a type and $\vec{\alpha}$ strictly positive type parameters. We define

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

(where $(\vec{\sigma} \to \vec{\tau}) \to \rho(\vec{\tau}\,)$ means $(\sigma_1 \to \tau_1) \to \ldots \to (\sigma_n \to \tau_n) \to \rho(\vec{\tau}\,)$). If none of $\vec{\alpha}$ appears free in $\rho(\vec{\alpha}\,)$ let

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

Otherwise we use an outer recursion on $\rho(\vec{\alpha}\,)$ and if $\rho(\vec{\alpha}\,)$ is $\iota(\vec{\alpha}\,)$ an inner one on $x$. In case $\rho(\vec{\alpha}\,)$ is $\iota(\vec{\alpha}\,)$ we abbreviate $\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\iota(\vec{\alpha}\,)}$ by $\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\iota}$ or $\mathcal{M}^{\vec{\tau}}_{\iota(\vec{\sigma}\,)}$.

The immediate cases for the outer recursion are

$$\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\alpha_i} x \vec{f} := f_i x, \qquad \mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}(\sigma\to\rho)} h \vec{f} x := \mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\rho}(hx)\vec{f}.$$

It remains to consider $\iota(\vec{\pi}(\vec{\alpha}\,))$. In case $\vec{\pi}(\vec{\alpha}\,)$ is not $\vec{\alpha}$ let

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

with $\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\pi_i(\vec{\alpha}\,)} \cdot \vec{f} := \lambda_x \mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\pi_i(\vec{\alpha}\,)} x \vec{f}$. In case $\vec{\pi}(\vec{\alpha}\,)$ is $\vec{\alpha}$ we use recursion on $x$ and define for a constructor $\mathrm{C}_i \colon (\rho_\nu(\vec{\sigma}, \iota(\vec{\sigma}\,)))_{\nu<n} \to \iota(\vec{\sigma}\,)$

$$\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\iota}(\mathrm{C}_i \vec{x}\,)\vec{f}$$

to be the result of applying $\mathrm{C}'_i$ of type $(\rho_\nu(\vec{\tau}, \iota(\vec{\tau}\,)))_{\nu<n} \to \iota(\vec{\tau}\,)$ (the same constructor as $\mathrm{C}_i$ with only the type changed) to, for each $\nu < n$,

$$\mathcal{M}^{\vec{\sigma},\iota(\vec{\sigma}\,)\to\vec{\tau},\iota(\vec{\tau}\,)}_{\lambda_{\vec{\alpha},\beta}\rho_\nu(\vec{\alpha},\beta)} x_\nu \vec{f}(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\iota} \cdot \vec{f}\,).$$

Note that the final function argument provides the recursive call w.r.t. the recursion on $x$.

Example.

$$\mathcal{M}^{\tau}_{\mathbf{L}(\sigma)} \mathrm{Nil} f^{\sigma\to\tau} := \mathrm{Nil},$$
$$\mathcal{M}^{\tau}_{\mathbf{L}(\sigma)}(x^\sigma :: l^{\mathbf{L}(\sigma)}) f^{\sigma\to\tau} := (fx) :: (\mathcal{M}\, l\, f).$$

Definition. *Terms of Gödel's* T for nested algebras are inductively defined from typed variables $x^\rho$ and constants for constructors $\mathrm{C}^\iota_i$, recursion operators $\mathcal{R}^\tau_\iota$ and map operators $\mathcal{M}^{\vec{\rho}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\pi}$ by abstraction $\lambda_{x^\rho} M^\sigma$ and application $M^{\rho\to\sigma} N^\rho$.

**2.2.2. Conversion.** We define a *conversion relation* $\mapsto_\rho$ between terms of type $\rho$ by

$$(\lambda_x M(x))N \mapsto M(N), \tag{2.1}$$

$$\lambda_x(Mx) \mapsto M \quad \text{if } x \notin \mathrm{FV}(M) \ (M \text{ not an abstraction}), \tag{2.2}$$

$$\mathcal{R}^\tau_\iota(\mathrm{C}^\iota_i\vec{N})\vec{M} \mapsto M_i(\mathcal{M}^{\iota\to\iota\times\tau}_{\lambda_\alpha\rho_\nu(\alpha)}N_\nu\lambda_x\langle x^\iota,\mathcal{R}^\tau_\iota x\vec{M}\rangle)_{\nu<n} \tag{2.3}$$

where $(\rho_\nu(\iota))_{\nu<n}\to\iota$ is the type of the $i$-th constructor $\mathrm{C}_i$.

In the special case $\rho_\nu(\alpha)=\alpha$ we can avoid the product type and instead of the pair

$$\mathcal{M}^{\iota\to\iota\times\tau}_{\lambda_\alpha\alpha}N_\nu\lambda_x\langle x^\iota,\mathcal{R}^\tau_\iota x\vec{M}\rangle \quad \text{i.e.,} \quad \langle N^\iota_\nu,\mathcal{R}^\tau_\iota N_\nu\vec{M}\rangle$$

take its two components $N^\iota_\nu$ and $\mathcal{R}^\tau_\iota N_\nu\vec{M}$ as separate arguments of $M_i$.

The rule (2.1) is called $\beta$-conversion, and (2.2) $\eta$-conversion; their left hand sides are called *$\beta$-redexes* or *$\eta$-redexes*, respectively. The left hand side of (2.3) is called *$\mathcal{R}$-redex*; it is a special case of a redex associated with a constant $D$ defined by "computation rules" (cf. 2.2.3), and hence also called a *$D$-redex*.

**2.2.3. A common extension $\mathrm{T}^+$ of Gödel's T and Plotkin's PCF.** *Terms* of $\mathrm{T}^+$ are built from (typed) variables and (typed) constants (constructors C or defined constants $D$, see below) by (type-correct) application and abstraction:

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

Definition (Computation rule). Every defined constant $D$ comes with a system of *computation rules*, consisting of finitely many equations

$$D\vec{P}_i(\vec{y}_i) = M_i \qquad (i=1,\dots,n) \tag{2.4}$$

with free variables of $\vec{P}_i(\vec{y}_i)$ and $M_i$ among $\vec{y}_i$, where the arguments on the left hand side must be "constructor patterns", i.e., lists of applicative terms built from constructors and distinct variables. To ensure consistency of the defining equations, we require that for $i\neq j$ $\vec{P}_i$ and $\vec{P}_j$ have disjoint free variables, and either $\vec{P}_i$ and $\vec{P}_j$ are non-unifiable (i.e., there is no substitution which identifies them), or else for the most general unifier $\vartheta$ of $\vec{P}_i$ and $\vec{P}_j$ we have $M_i\vartheta = M_j\vartheta$. Notice that the substitution $\vartheta$ assigns to the variables $\vec{y}_i$ in $M_i$ constructor patterns $\vec{R}_k(\vec{z})$ $(k=i,j)$. A further requirement on a system of computation rules $D\vec{P}_i(\vec{y}_i)=M_i$ is that the lengths of all $\vec{P}_i(\vec{y}_i)$ are the same; this number is called the *arity* of $D$, denoted by $\mathrm{ar}(D)$. A substitution instance of a left hand side of (2.4) is called a *$D$-redex*.

More formally, constructor patterns are defined inductively by (we write $\vec{P}(\vec{x}\,)$ to indicate all variables in $\vec{P}$):

(a) $x$ is a constructor pattern.
(b) The empty list $\langle\rangle$ is a constructor pattern.
(c) If $\vec{P}(\vec{x}\,)$ and $Q(\vec{y}\,)$ are constructor patterns whose variables $\vec{x}$ and $\vec{y}$ are disjoint, then $(\vec{P}, Q)(\vec{x}, \vec{y}\,)$ is a constructor pattern.
(d) If C is a constructor and $\vec{P}$ a constructor pattern, then so is $\mathrm{C}\vec{P}$, provided it is of ground type.

REMARK. The requirement of disjoint variables in constructor patterns $\vec{P}_i$ and $\vec{P}_j$ used in computation rules of a defined constant $D$ is needed to ensure that applying the most general unifier produces constructor patterns again. However, for readability we take this as an implicit convention, and write computation rules with possibly non-disjoint variables.

Examples of constants $D$ defined by computation rules are abundant. In particular, the map and (structural) recursion operators can be viewed as defined by computation rules, which in this case are called *conversion* rules; cf. 2.2.2.

The boolean connectives andb, impb and orb are defined by

$$\begin{array}{lll} \mathtt{tt}\ \mathrm{andb}\ y = y, & \mathtt{ff}\ \mathrm{impb}\ y = \mathtt{tt}, & \mathtt{tt}\ \mathrm{orb}\ y = \mathtt{tt}, \\ x\ \mathrm{andb}\ \mathtt{tt} = x, & \mathtt{tt}\ \mathrm{impb}\ y = y, & x\ \mathrm{orb}\ \mathtt{tt} = \mathtt{tt}, \\ \mathtt{ff}\ \mathrm{andb}\ y = \mathtt{ff}, & x\ \mathrm{impb}\ \mathtt{tt} = \mathtt{tt}, & \mathtt{ff}\ \mathrm{orb}\ y = y, \\ x\ \mathrm{andb}\ \mathtt{ff} = \mathtt{ff}, & & x\ \mathrm{orb}\ \mathtt{ff} = x. \end{array}$$

Notice that when two such rules overlap, their right hand sides are equal under any unifier of the left hand sides.

Decidable *equality* $=_\iota \colon \iota \to \iota \to \mathbf{B}$ for a finitary algebra $\iota$ can be defined easily by computation rules. For example,

$$\begin{array}{ll} (0 =_{\mathbf{N}} 0) = \mathtt{tt}, & (\mathrm{S}m =_{\mathbf{N}} 0) = \mathtt{ff}, \\ (0 =_{\mathbf{N}} \mathrm{S}n) = \mathtt{ff}, & (\mathrm{S}m =_{\mathbf{N}} \mathrm{S}n) = (m =_{\mathbf{N}} n). \end{array}$$

For the algebra $\mathbf{D}$ of binary trees with constructors $L$ (leaf) and C (construct a new tree from two given ones) we have

$$\begin{array}{ll} (L =_{\mathbf{D}} L) = \mathtt{tt}, & (\mathrm{C}m =_{\mathbf{D}} L) = \mathtt{ff}, \\ (L =_{\mathbf{D}} \mathrm{C}n) = \mathtt{ff}, & (\mathrm{C}a_1a_2 =_{\mathbf{D}} \mathrm{C}b_1b_2) = (a_1 =_{\mathbf{D}} b_1\ \mathrm{andb}\ a_2 =_{\mathbf{D}} b_2). \end{array}$$

**2.2.4. Ideals as denotation of terms.** How can we use computation rules to define an ideal $z$ in a function space? The general idea is to inductively define the set of tokens $(U, b)$ that make up $z$. It is convenient to define the value $[\![\lambda_{\vec{x}} M]\!]$, where $M$ is a term with free variables among $\vec{x}$. Since this value is a token set, we can define inductively the relation $(\vec{U}, b) \in [\![\lambda_{\vec{x}} M]\!]$.

For a constructor pattern $\vec{P}(\vec{x}\,)$ and a list $\vec{V}$ of the same length and types as $\vec{x}$ we define a list $\vec{P}(\vec{V})$ of formal neighborhoods of the same length and types as $\vec{P}(\vec{x}\,)$, by induction on $\vec{P}(\vec{x}\,)$. $x(V)$ is the singleton list $V$, and for $\langle\rangle$ we take the empty list. $(\vec{P}, Q)(\vec{V}, \vec{W})$ is covered by the induction hypothesis. Finally

$$(\mathrm{C}\vec{P})(\vec{V}) := \{\, \mathrm{C}\vec{b^*} \mid b_i^* \in P_i(\vec{V_i}) \text{ if } P_i(\vec{V_i}) \neq \emptyset, \text{ and } b_i^* = * \text{ otherwise} \,\}.$$

We use the following notation. $(\vec{U}, b)$ means $(U_1, \dots (U_n, b) \dots)$, and $(\vec{U}, V) \subseteq [\![\lambda_{\vec{x}} M]\!]$ means $(\vec{U}, b) \in [\![\lambda_{\vec{x}} M]\!]$ for all (finitely many) $b \in V$.

Definition (Inductive, of $a \in [\![\lambda_{\vec{x}} M]\!]$). *Case* $\lambda_{\vec{x},y,\vec{z}} M$ with $\vec{x}$ free in $M$, but not $y$.

$$\frac{(\vec{U}, \vec{W}, a) \in [\![\lambda_{\vec{x},\vec{z}} M]\!]}{(\vec{U}, V, \vec{W}, a) \in [\![\lambda_{\vec{x},y,\vec{z}} M]\!]}(K).$$

*Case* $\lambda_{\vec{x}} M$ with $\vec{x}$ the free variables in $M$.

$$\frac{U \vdash a}{(U, a) \in [\![\lambda_x x]\!]}(V), \quad \frac{(\vec{U}, V, a) \in [\![\lambda_{\vec{x}} M]\!] \quad (\vec{U}, V) \subseteq [\![\lambda_{\vec{x}} N]\!]}{(\vec{U}, a) \in [\![\lambda_{\vec{x}} (MN)]\!]}(A).$$

For every constructor C and defined constant $D$:

$$\frac{\vec{U} \vdash \vec{a^*}}{(\vec{U}, \mathrm{C}\vec{a^*}) \in [\![\mathrm{C}]\!]}(\mathrm{C}), \quad \frac{(\vec{V}, a) \in [\![\lambda_{\vec{x}} M]\!] \quad \vec{U} \vdash \vec{P}(\vec{V})}{(\vec{U}, a) \in [\![D]\!]}(D),$$

with one rule $(D)$ for every defining equation $D\vec{P}(\vec{x}\,) = M$.

This "denotational semantics" has good properties; however, we do not carry out the proofs here (cf. Schwichtenberg and Wainer (2012)). First of all, one can prove that $[\![\lambda_{\vec{x}} M]\!]$ is an ideal . Moreover, our definition above of the denotation of a term is reasonable in the sense that it is not changed by an application of the standard ($\beta$- and $\eta$-) conversions or a computation rule. For the $\beta$-conversion part of this proof it is helpful to first introduce a more standard notation, which involves variable environments.

Definition. $[\![M]\!]^{\vec{U}}_{\vec{x}} := \{\, b \mid (\vec{U}, b) \in [\![\lambda_{\vec{x}} M]\!] \,\}$, $[\![M]\!]^{\vec{u},\vec{V}}_{\vec{x},\vec{y}} := \bigcup_{\vec{U} \subseteq \vec{u}} [\![M]\!]^{\vec{U},\vec{V}}_{\vec{x},\vec{y}}$.

We have a useful monotonicity property, which follows from the deductive closure of $[\![\lambda_{\vec{x}} M]\!]$.

LEMMA. (a) *If* $\vec{V} \vdash \vec{U}$, $b \vdash c$ *and* $b \in [\![M]\!]^{\vec{U}}_{\vec{x}}$, *then* $c \in [\![M]\!]^{\vec{V}}_{\vec{x}}$.
(b) *If* $\vec{v} \supseteq \vec{u}$, $b \vdash c$ *and* $b \in [\![M]\!]^{\vec{u}}_{\vec{x}}$, *then* $c \in [\![M]\!]^{\vec{v}}_{\vec{x}}$.

LEMMA. (a) $[\![x_i]\!]^{\vec{U}}_{\vec{x}} = \overline{U}_i$ *and* $[\![x_i]\!]^{\vec{u}}_{\vec{x}} = u_i$.
(b) $[\![\lambda_y M]\!]^{\vec{U}}_{\vec{x}} = \{\, (V, b) \mid b \in [\![M]\!]^{\vec{U},V}_{\vec{x},y} \,\}$ *and* $[\![\lambda_y M]\!]^{\vec{u}}_{\vec{x}} = \{\, (V, b) \mid b \in [\![M]\!]^{\vec{u},V}_{\vec{x},y} \,\}$.
(c) $[\![MN]\!]^{\vec{U}}_{\vec{x}} = [\![M]\!]^{\vec{U}}_{\vec{x}} [\![N]\!]^{\vec{U}}_{\vec{x}}$ *and* $[\![MN]\!]^{\vec{u}}_{\vec{x}} = [\![M]\!]^{\vec{u}}_{\vec{x}} [\![N]\!]^{\vec{u}}_{\vec{x}}$.

COROLLARY. $[\![\lambda_y M]\!]^{\vec{u}}_{\vec{x}} v = [\![M]\!]^{\vec{u},v}_{\vec{x},y}$.

LEMMA (Substitution). $[\![M(z)]\!]^{\vec{u},[\![N]\!]^{\vec{u}}_{\vec{x}}}_{\vec{x},z} = [\![M(N)]\!]^{\vec{u}}_{\vec{x}}$.

LEMMA (Preservation of values, $\beta$). $[\![(\lambda_y M(y))N]\!]^{\vec{u}}_{\vec{x}} = [\![M(N)]\!]^{\vec{u}}_{\vec{x}}$.

LEMMA (Preservation of values, $\eta$). $[\![\lambda_y (My)]\!]^{\vec{u}}_{\vec{x}} = [\![M]\!]^{\vec{u}}_{\vec{x}}$ *if* $y \notin \mathrm{FV}(M)$.

Then it follows that values are preserved under computation rules:

LEMMA. *For every computation rule* $D\vec{P}(\vec{y}) = M$ *of a defined constant* $D$, $[\![\lambda_{\vec{y}} (D\vec{P}(\vec{y}))]\!]^{\vec{u}}_{\vec{x}} = [\![\lambda_{\vec{y}} M]\!]^{\vec{u}}_{\vec{x}}$.

CHAPTER 3

# Extracting computational content from proofs

## 3.1. A theory of computable functionals

**3.1.1. Brouwer-Heyting-Kolmogorov and Gödel.** The Brouwer-Heyting-Kolmogorov interpretation (BHK-interpretation for short) of intuitionistic (and minimal) logic explains what it means to prove a logically compound statement in terms of what it means to prove its components; the explanations use the notions of *construction* and *constructive proof* as unexplained primitive notions. For prime formulas the notion of proof is supposed to be given. The clauses of the BHK-interpretation are:

(i) $p$ proves $A \wedge B$ if and only if $p$ is a pair $\langle p_0, p_1 \rangle$ and $p_0$ proves $A$, $p_1$ proves $B$;
(ii) $p$ proves $A \to B$ if and only if $p$ is a construction transforming any proof $q$ of $A$ into a proof $p(q)$ of $B$;
(iii) $\bot$ is a proposition without proof;
(iv) $p$ proves $\forall_{x \in D} A(x)$ if and only if $p$ is a construction such that for all $d \in D$, $p(d)$ proves $A(d)$;
(v) $p$ proves $\exists_{x \in D} A(x)$ if and only if $p$ is of the form $\langle d, q \rangle$ with $d$ an element of $D$, and $q$ a proof of $A(d)$.

The problem with the BHK-interpretation clearly is its reliance on the unexplained notions of construction and constructive proof. Gödel was concerned with this problem for more than 30 years. In 1941 he gave a lecture at Yale university with the title "In what sense is intuitionistic logic constructive?". According to Kreisel, Gödel "wanted to establish that intuitionistic proofs of existential theorems provide explicit realizers" (Feferman et al., 1986, 1990, 1995, 2002, 2002, Vol II, p.219). Gödel published his "Dialectica interpretation" in (1958), and revised this work over and over again; its state in 1972 has been published in the same volume. Troelstra, in his introductory note to the latter two papers writes (loc. cit., pp.220/221):

> Gödel argues that, since the finististic methods considered are not sufficient to carry out Hilbert's program, one has

> to admit at least some abstract notions in a consistency proof; ... However, Gödel did not want to go as far as admitting Heyting's abstract notion of constructive proof; hence he tried to replace the notion of constructive proof by something more definite, less abstract (that is, more nearly finitistic), his principal candidate being a notion of "computable functional of finite type" which is to be accepted as sufficiently well understood to justify the axioms and rules of his system T, an essentially logic-free theory of functionals of finite type.

We intend to utilize the notion of a computable functional of finite type as an ideal in an information system, as explained in the previous chapter. However, Gödel noted that his proof interpretation is largely independent of a precise definition of computable functional; one only needs to know that certain basic functionals are computable (including primitive recursion operators in finite types), and that they are closed under composition. Building on Gödel (1958), we assign to every formula $A$ a new one $\exists_x A_1(x)$ with $A_1(x)$ $\exists$-free. Then from a derivation of $A$ we want to extract a "realizing term" $r$ such that $A_1(r)$. Of course its meaning should in some sense be related to the meaning of the original formula $A$. However, Gödel explicitly states in (1958, p.286) that his Dialectica interpretation is *not* the one intended by the BHK-interpretation.

**3.1.2. Formulas and predicates.** When we want to make propositions about computable functionals and their domains of partial continuous functionals, it is perfectly natural to take, as initial propositions, ones formed inductively or coinductively. However, for simplicity we omit the treatment of coinductive definitions and deal with inductive definitions only. For example, in the algebra $\mathbf{N}$ we can inductively define *totality* by the clauses

$$T_{\mathbf{N}}0, \qquad \forall_n(T_{\mathbf{N}}n \to T_{\mathbf{N}}(\mathrm{S}n)).$$

Its least-fixed-point scheme will now be taken in the form

$$\forall_n(T_{\mathbf{N}}n \to A(0) \to \forall_n(T_{\mathbf{N}}n \to A(n) \to A(\mathrm{S}n)) \to A(n)).$$

The reason for writing it in this way is that it fits more conveniently with the logical elimination rules, which will be useful in the proof of the soundness theorem. It expresses that every "competitor" $\{\, n \mid A(n) \,\}$ satisfying the same clauses contains $T_{\mathbf{N}}$. This is the usual induction schema for natural numbers, which clearly only holds for "total" numbers (i.e., total ideals

in the information system for $\mathbf{N}$). Notice that we have used a "strengthened" form of the "step formula", namely $\forall_n(T_{\mathbf{N}}n \to A(n) \to A(\mathrm{S}n))$ rather than $\forall_n(A(n) \to A(\mathrm{S}n))$. In applications of the least-fixed-point axiom this simplifies the proof of the "induction step", since we have the additional hypothesis $T(n)$ available. Totality for an arbitrary algebra can be defined similarly. Consider for example the non-finitary algebra $\mathbf{O}$ (cf. 2.1.3), with constructors 0, successor S of type $\mathbf{O} \to \mathbf{O}$ and supremum Sup of type $(\mathbf{N} \to \mathbf{O}) \to \mathbf{O}$. Its clauses are

$$T_{\mathbf{O}}0, \qquad \forall_x(T_{\mathbf{O}}x \to T_{\mathbf{O}}(\mathrm{S}x)), \qquad \forall_f(\forall_{n\in T_{\mathbf{N}}}T_{\mathbf{O}}(fn) \to T_{\mathbf{O}}(\mathrm{Sup}(f))),$$

and its least-fixed-point scheme is

$$\begin{aligned}\forall_x(T_{\mathbf{O}}x \to A(0) \to\ & \\ & \forall_x(T_{\mathbf{O}}x \to A(x) \to A(\mathrm{S}x)) \to \\ & \forall_f(\forall_{n\in T}T_{\mathbf{O}}(fn) \to \forall_{n\in T}A(fn) \to A(\mathrm{Sup}(f))) \to \\ & A(x)).\end{aligned}$$

Generally, an inductively defined predicate $I$ is given by $k$ clauses, which are of the form

$$K_i := \forall_{\vec{x}}((A_\nu(I))_{\nu<n} \to I\vec{r}) \quad (i < k).$$

Our formulas will be defined by the operations of implication $A \to B$ and universal quantification $\forall_{x^\rho}A$ from inductively defined predicates $\mu_X\vec{K}$, where $X$ is a "predicate variable", and the $K_i$ are "clauses". Every predicate has an *arity*, which is a possibly empty list of types.

Definition (Formulas and predicates). By simultaneous induction we define formula forms

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

and predicate forms

$$P, Q ::= X \mid \{\, \vec{x} \mid A \,\} \mid \mu_X(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to X\vec{r}_i))_{i<k}$$

with $X$ a predicate variable, $k \geq 1$ and $\vec{x}_i$ all free variables in $(A_{i\nu})_{\nu<n_i} \to X\vec{r}_i$ (it is not necessary to allow object parameters in inductively defined predicates, since they can be taken as extra arguments). Let $C$ denote both formula and predicate forms. Let $\mathrm{FPV}(C)$ denote the set of free predicate variables in $C$. We define $\mathrm{SP}(Y, C)$ "$Y$ occurs at most strictly positive in $C$" by induction on $C$.

$$\frac{\mathrm{SP}(Y,P)}{\mathrm{SP}(Y,P\vec{r})} \qquad \frac{Y \notin \mathrm{FPV}(A) \quad \mathrm{SP}(Y,B)}{\mathrm{SP}(Y, A \to B)} \qquad \frac{\mathrm{SP}(Y,A)}{\mathrm{SP}(Y,\forall_x A)}$$

$$\mathrm{SP}(Y,X) \qquad \frac{\mathrm{SP}(Y,A)}{\mathrm{SP}(Y,\{\,\vec{x}\mid A\,\})} \qquad \frac{\mathrm{SP}(Y,A_{i\nu}) \text{ for all } i{<}k,\ \nu{<}n_i}{\mathrm{SP}(Y,\mu_X(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i}\to X\vec{r}_i))_{i<k})}$$

Now we can define $\mathrm{F}(A)$ "$A$ is a formula" and $\mathrm{Pred}(P)$ "$P$ is a predicate", again by simultaneous induction.

$$\frac{\mathrm{Pred}(P)}{\mathrm{F}(P\vec{r})} \qquad \frac{\mathrm{F}(A)\quad \mathrm{F}(B)}{\mathrm{F}(A\to B)} \qquad \frac{\mathrm{F}(A)}{\mathrm{F}(\forall_x A)}$$

$$\mathrm{Pred}(X) \qquad \frac{\mathrm{F}(A)}{\mathrm{Pred}(\{\,\vec{x}\mid A\,\})}$$

$$\frac{\mathrm{F}(A_{i\nu}) \text{ and } \mathrm{SP}(X,A_{i\nu}) \text{ for all } i{<}k,\ \nu{<}n_i \qquad X\notin \mathrm{FPV}(A_{0\nu}) \text{ for all } \nu{<}n_0}{\mathrm{Pred}(\mu_X(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i}\to X\vec{r}_i))_{i<k})}$$

We call

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

an inductive (or inductively defined) predicate. Sometimes it is helpful to display the predicate parameters and write $I(\vec{Y},\vec{Z}\,)$, where $\vec{Y},\vec{Z}$ are all predicate variables free in some $A_{i\nu}$ except $X$, and $\vec{Y}$ are the ones occuring only strictly positive. If we write the $i$-th component of $I$ in the form $\forall_{\vec{x}}((A_\nu(X))_{\nu<n}\to X\vec{r}\,)$, then we call

$$K_i := \forall_{\vec{x}}((A_\nu(I))_{\nu<n}\to I\vec{r}\,) \tag{3.1}$$

the $i$-th *clause* (or *introduction axiom*) of $I$, denoted $I_i^+$.

Here $\vec{A}\to B$ means $A_0\to\cdots\to A_{n-1}\to B$, associated to the right. The terms $\vec{r}$ are those introduced in section 2.2, i.e., typed terms built from variables and constants by abstraction and application, and (importantly) those with a common reduct are identified. In $\forall_{\vec{x}}((A_\nu(X))_{\nu<n}\to X\vec{r}\,)$ we call $A_\nu(X)$ a *parameter* premise if $X$ does not occur in it, and a *recursive* premise otherwise. A recursive premise $A_\nu(X)$ is *nested* if it has an occurrence of $X$ in a strictly positive parameter position of another (previously defined) inductive predicate, and unnested otherwise. An inductive predicate $I$ is called *nested* if it has a clause with at least one nested recursive premise, and *unnested* otherwise.

A predicate of the form $\{\,\vec{x}\mid C\,\}$ is called a *comprehension term*. We identify $\{\,\vec{x}\mid C(\vec{x}\,)\,\}\vec{r}$ with $C(\vec{r}\,)$. The letter $I$ will be used for predicates of the form $\mu_X(K_0,\dots,K_{k-1})$; they are called *inductively defined predicates*. An inductively defined predicate is *finitary* if its clauses have recursive premises of the form $X\vec{s}$ only.

Definition (Theory of computable functionals, TCF). TCF is the system in minimal logic for $\to$ and $\forall$, whose formulas are those in F above, and

whose axioms are the following. For each inductively defined predicate, there are "closure" or introduction axioms, together with a "least-fixed-point" or elimination axiom. In more detail, consider an inductively defined predicate $I := \mu_X(K_0, \dots, K_{k-1})$. For each of the $k$ clauses we have the introduction axiom (3.1). Moreover, we have an *elimination axiom* $I^-$:

$$\forall_{\vec{x}}(I\vec{x} \to (\forall_{\vec{x}_i}((A_{i\nu}(I \wedge X))_{\nu<n_i} \to X\vec{r}_i))_{i<k} \to X\vec{x}\,) \tag{3.2}$$

where $I \wedge X$ abbreviates $\{\,\vec{x} \mid I\vec{x} \wedge X\vec{x}\,\}$ with $\wedge$ defined (inductively) below. Here $X$ can be thought of as a "competitor" predicate.

**3.1.3. Examples of inductive predicates.** We first deal with the concept of an equality. A word of warning is in order here: we need to distinguish four separate but closely related equalities.

(i) Firstly, defined function constants $D$ are introduced by computation rules, written $l = r$, but intended as left-to-right rewrites.
(ii) Secondly, we have Leibniz equality Eq inductively defined below.
(iii) Thirdly, pointwise equality between partial continuous functionals will be defined inductively as well.
(iv) Fourthly, if $l$ and $r$ have a finitary algebra as their type, $l = r$ can be read as a boolean term, where $=$ is the decidable equality defined in 2.2.3 as a boolean-valued binary function.

*Leibniz equality.* We define Leibniz equality by

$$\mathrm{Eq}(\rho) := \mu_X(\forall_x X(x^\rho, x^\rho)).$$

The introduction axiom is

$$\forall_x \mathrm{Eq}(x^\rho, x^\rho)$$

and the elimination axiom

$$\forall_{x,y}(\mathrm{Eq}(x, y) \to \forall_x Xxx \to Xxy),$$

where $\mathrm{Eq}(x, y)$ abbreviates $\mathrm{Eq}(\rho)(x^\rho, y^\rho)$.

Lemma (Compatibility of Eq). $\forall_{x,y}(\mathrm{Eq}(x, y) \to A(x) \to A(y))$.

Proof. Exercise. □

Using compatibility of Eq one easily proves symmetry and transitivity. Define *falsity* by $\mathbf{F} := \mathrm{Eq}(\mathtt{ff}, \mathtt{tt})$. Then we have

Theorem (Ex-falso-quodlibet). *For every formula $A$ without predicate parameters we can derive $\mathbf{F} \to A$.*

PROOF. We first show that $\mathbf{F} \to \mathrm{Eq}(x^\rho, y^\rho)$. To see this, we first obtain $\mathrm{Eq}(\mathcal{R}^\rho_{\mathbf{B}}\mathtt{ff}xy, \mathcal{R}^\rho_{\mathbf{B}}\mathtt{ff}xy)$ from the introduction axiom. Then from $\mathrm{Eq}(\mathtt{ff}, \mathtt{tt})$ we get $\mathrm{Eq}(\mathcal{R}^\rho_{\mathbf{B}}\mathtt{tt}xy, \mathcal{R}^\rho_{\mathbf{B}}\mathtt{ff}xy)$ by compatibility. Now $\mathcal{R}^\rho_{\mathbf{B}}\mathtt{tt}xy$ converts to $x$ and $\mathcal{R}^\rho_{\mathbf{B}}\mathtt{ff}xy$ converts to $y$. Hence $\mathrm{Eq}(x^\rho, y^\rho)$, since we identify terms with a common reduct.

The claim can now be proved by induction on $A \in \mathrm{F}$. *Case* $I\vec{s}$. Let $K_i$ be the nullary clause, with final conclusion $I\vec{t}$. By induction hypothesis from $\mathbf{F}$ we can derive all parameter premises. Hence $I\vec{t}$. From $\mathbf{F}$ we also obtain $\mathrm{Eq}(s_i, t_i)$, by the remark above. Hence $I\vec{s}$ by compatibility. The cases $A \to B$ and $\forall_x A$ are obvious. □

A crucial use of the equality predicate Eq is that it allows us to lift a boolean term $r^{\mathbf{B}}$ to a formula, using $\mathrm{atom}(r^{\mathbf{B}}) := \mathrm{Eq}(r^{\mathbf{B}}, \mathtt{tt})$. This opens up a convenient way to deal with equality on finitary algebras. The computation rules ensure that, for instance, the boolean term $\mathrm{S}r =_{\mathbf{N}} \mathrm{S}s$, or more precisely $=_{\mathbf{N}}(\mathrm{S}r, \mathrm{S}s)$, is identified with $r =_{\mathbf{N}} s$. We can now turn this boolean term into the formula $\mathrm{Eq}(\mathrm{S}r =_{\mathbf{N}} \mathrm{S}s, \mathtt{tt})$, which again is abbreviated by $\mathrm{S}r =_{\mathbf{N}} \mathrm{S}s$, but this time with the understanding that it is a formula. Then (importantly) the two formulas $\mathrm{S}r =_{\mathbf{N}} \mathrm{S}s$ and $r =_{\mathbf{N}} s$ are identified because the latter is a reduct of the first. Consequently there is no need to prove the implication $\mathrm{S}r =_{\mathbf{N}} \mathrm{S}s \to r =_{\mathbf{N}} s$ explicitly.

*Existence, conjunction and disjunction.* One of the main points of TCF is that it allows the logical connectives existence, conjunction and disjunction to be inductively defined as predicates. This was first discovered by Martin-Löf (1971).

$$\begin{aligned}
\mathrm{Ex}(Y) &:= \mu_X(\forall_x(Yx^\rho \to X)),\\
\mathrm{And}(Y_1, Y_2) &:= \mu_X(Y_1 \to Y_2 \to X),\\
\mathrm{Or}(Y_1, Y_2) &:= \mu_X(Y_1 \to X,\ Y_2 \to X).
\end{aligned}$$

We will use the abbreviations

$$\begin{aligned}
\exists_x A &:= \mathrm{Ex}(\{\, x^\rho \mid A \,\}),\\
A \wedge B &:= \mathrm{And}(\{\mid A\,\}, \{\mid B\,\}),\\
A \vee B &:= \mathrm{Or}(\{\mid A\,\}, \{\mid B\,\}),
\end{aligned}$$

The introduction axioms are

$$\begin{aligned}
&\forall_x(Yx \to \exists_x Yx),\\
&Y_1 \to Y_2 \to Y_1 \wedge Y_2,\\
&Y_1 \to Y_1 \vee Y_2, \qquad Y_2 \to Y_1 \vee Y_2.
\end{aligned}$$

The elimination axioms are

$$\exists_x Yx \to \forall_x(Yx \to X) \to X,$$
$$Y_1 \wedge Y_2 \to (Y_1 \to Y_2 \to X) \to X,$$
$$Y_1 \vee Y_2 \to (Y_1 \to X) \to (Y_2 \to X) \to X.$$

We give some more familiar examples of inductively defined predicates.

*The even numbers.* The introduction axioms are

$$\mathrm{Even}(0), \qquad \forall_n(\mathrm{Even}(n) \to \mathrm{Even}(\mathrm{S}(\mathrm{S}n)))$$

and the elimination axiom is

$$\forall_n(\mathrm{Even}(n) \to X0 \to \forall_n(\mathrm{Even}(n) \to Xn \to X(\mathrm{S}(\mathrm{S}n))) \to Xn).$$

*Transitive closure.* Let $\prec$ be a predicate variable representing a binary relation. The *transitive closure* $\mathrm{TC}_\prec$ of $\prec$ is inductively defined as follows. The introduction axioms are

$$\forall_{x,y}(x \prec y \to \mathrm{TC}_\prec(x,y)),$$
$$\forall_{x,y,z}(x \prec y \to \mathrm{TC}_\prec(y,z) \to \mathrm{TC}_\prec(x,z))$$

and the elimination axiom is

$$\forall_{x,y}(\mathrm{TC}_\prec(x,y) \to \forall_{x,y}(x \prec y \to Xxy) \to \forall_{x,y,z}(x \prec y \to \mathrm{TC}_\prec(y,z) \to Xyz \to Xxz) \to Xxy).$$

## 3.2. Realizability interpretation

At this point we come to the crucial step of identifying "computational content" in proofs, which can then be extracted. Recall that the BHK-interpretation (described in 3.1.1) left open what a proof of a prime formula is. However, in TCF we can be more definite, since a closed prime formula must be of the form $I\vec{r}$ with $I$ an inductive predicate. The obvious idea is to view a proof of $I\vec{r}$ as a "generation tree", witnessing how the arguments $\vec{r}$ were put into $I$. For example, let Even be defined by the clauses Even(0) and $\forall_n(\mathrm{Even}(n) \to \mathrm{Even}(\mathrm{S}(\mathrm{S}n)))$. A generation tree for Even(6) should consist of a single branch with nodes Even(0), Even(2), Even(4) and Even(6). More formally, such a generation tree can seen as an ideal in an algebra $\iota_I$ associated naturally with $I$.

Consider the more general situation when parameters are involved, i.e., when we have a proof (in TCF) of a closed formula $\forall_{\vec{x}}(\vec{A} \to I\vec{r})$. It is of obvious interest which of the variables $\vec{x}$ and assumptions $\vec{A}$ are actually used

in the "solution" provided by the proof (in the sense of Kolmogorov (1932)). To be able to express dependence on and independence of such parameters we split each of our (only!) logical connectives $\to, \forall$ into two variants, a "computational" one $\forall^{\mathrm{c}}, \to^{\mathrm{c}}$ and a "non-computational" one $\forall^{\mathrm{nc}}, \to^{\mathrm{nc}}$. This distinction (for the universal quantifier) is due to Berger (1993, 2005). Then a proof of $\forall^{\mathrm{nc}}_{\vec{x}} \forall^{\mathrm{c}}_{\vec{y}} (\vec{A} \to^{\mathrm{nc}} \vec{B} \to^{\mathrm{c}} I\vec{r})$ provides a construction of an ideal in $\iota_I$ independent of $\vec{x}$ and assumed proofs of $\vec{A}$. One can view this "decoration" of $\to, \forall$ as turning our (minimal) logic into a "computational logic", which is able to express dependence on and independence of parameters. The rules for $\to^{\mathrm{nc}}, \forall^{\mathrm{nc}}$ are similar to the ones for $\to^{\mathrm{c}}, \forall^{\mathrm{c}}$; they will be given in 3.2.2.

Now the clauses of inductive predicates can and should be decorated as well. Without loss of generality we can assume that they have the form

$$\forall^{\mathrm{nc}}_{\vec{x}} \forall^{\mathrm{c}}_{\vec{y}} (\vec{A} \to^{\mathrm{nc}} \vec{B} \to^{\mathrm{c}} X\vec{r}).$$

This will lead to a different (i.e., simplified) algebra $\iota_I$ associated with the inductive predicate $I$.

Of special importance is the case when we only have $\to^{\mathrm{nc}}, \forall^{\mathrm{nc}}$, and there is only one clause. Such inductive predicates are called "uniform one-clause defined", and their associated algebra is the unit algebra $\mathbf{U}$. Examples are Leibniz equality, existence and conjunction when defined with $\to^{\mathrm{nc}}, \forall^{\mathrm{nc}}$:

$$\begin{aligned}
\mathrm{Eq}(\rho) &:= \mu_X (\forall^{\mathrm{nc}}_x X(x^\rho, x^\rho)),\\
\mathrm{ExU}(Y) &:= \mu_X (\forall^{\mathrm{nc}}_x (Y x^\rho \to^{\mathrm{nc}} X)),\\
\mathrm{AndU}(Y_1, Y_2) &:= \mu_X (Y_1 \to^{\mathrm{nc}} Y_2 \to^{\mathrm{nc}} X).
\end{aligned}$$

From now on we only use this uniform one-clause definition of Leibniz equality Eq, and use the abbreviations

$$\begin{aligned}
\exists^{\mathrm{u}}_x A &:= \mathrm{ExU}(\{\, x^\rho \mid A \,\}).\\
A \wedge^{\mathrm{u}} B &:= \mathrm{AndU}(\{ \mid A \}, \{ \mid B \}).
\end{aligned}$$

Prime formulas $I\vec{r}$ with $\iota_I = \mathbf{U}$ only have a trivial generation tree, and in this sense are without computational content. Clearly this is also the case for formulas with such an $I\vec{r}$ as conclusion. These formulas are called non-computational (n.c.) or *Harrop formulas*. Moreover, a Harrop formula in a premise can be ignored when we are interested in the computational content of a proof of this formula: its only contribution would be of unit type. Therefore when defining the type of a formula in 3.2.5 we will use a "cleaned" form of such a type, not involving the unit type.

The next thing to do is to properly accomodate the BHK-interpretation and define what it means that a term $t$ "realizes" the formula $A$, written

$t$ **r** $A$. In the prime formula case $I\vec{r}$ this will involve a predicate "$t$ realizes $I\vec{r}$", which will be defined inductively as well, following the clauses of $I$. But since this is a "meta" statement already containing the term $t$ representing a generation tree, we are not interested in the generation tree for such realizing formulas and consider them as non-computational.

Finally we will define in 3.2.6 the "extracted term" $\mathrm{et}(M)$ of a proof $M$ of a formula $A$, and prove the soundness theorem $\mathrm{et}(M)$ **r** $A$.

Remark. We have encountered two situations where inductive definitions do not have computational content: uniform one-clause defined predicates, and realizability predicates. There is a third occasion when this can happen and is in fact rather useful, namely when the all clauses have "invariant" premises $A$ only; a formula $A$ is called invariant if $\exists_x(x$ **r** $A)$ is equivalent to $A$. We write $\mu^{\mathrm{nc}}_X(K_0,\dots,K_{k-1})$ whenever an inductive predicate is n.c. The soundness theorem continues to hold if we restrict usage of the least-fixed-point (or elimination) axiom for such n.c. inductive predicates to Harrop formulas.

**3.2.1. An informal explanation.** The ideas that we develop here are illustrated by the following simple situation. The computational content of an implication $Pn \to^{\mathrm{c}} P(\mathrm{S}n)$ is that demanded of an implication by the BHK interpretation, namely a function from evidence for $Pn$ to evidence for $P(\mathrm{S}n)$. The universal quantifier $\forall_n$ is non-computational if it merely supplies $n$ as an "input", whereas to say that a universal quantifier is computational means that a construction of input $n$ is also supplied. Thus a realization of

$$\forall^{\mathrm{nc}}_n(Pn \to^{\mathrm{c}} P(\mathrm{S}n))$$

will be a unary function $f$ such that if $r$ "realizes" $Pn$, then $fr$ realizes $P(\mathrm{S}n)$, for every $n$. On the other hand, a realization of

$$\forall^{\mathrm{c}}_n(Pn \to^{\mathrm{c}} P(\mathrm{S}n))$$

will be a binary function $g$ which, given a number $n$ and a realization $r$ of $Pn$, produces a realization $g(n,r)$ of $P(\mathrm{S}n)$. Therefore an induction with basis and step of the form

$$P0, \qquad \forall^{\mathrm{nc}}_n(Pn \to^{\mathrm{c}} P(\mathrm{S}n))$$

will be realized by iterates $f^{(n)}(r_0)$, whereas a computational induction

$$P0, \qquad \forall^{\mathrm{c}}_n(Pn \to^{\mathrm{c}} P(\mathrm{S}n))$$

will be realized by the primitive recusively defined $h(n,r_0)$ where $h(0,r_0) = r_0$ and $h(\mathrm{S}n,r_0) = g(n,h(n,r_0))$.

Finally, a word about the non-computational implication: a realizer of $A \to^{\mathrm{nc}} B$ will depend solely on the existence of a realizer of $A$, but will be completely independent of which one it is. An example would be an induction

$$P0, \qquad \forall^{\mathrm{c}}_n(Pn \to^{\mathrm{nc}} P(\mathrm{S}n))$$

where the realizer $h(n, r_0)$ is given by $h(0, r_0) = r_0$, $h(\mathrm{S}n, r_0) = g(n)$, without recursive calls. The point is that in this case $g$ does not depend on a realizer for $Pn$, only upon the number $n$ itself.

**3.2.2. Decorating $\to$ and $\forall$.** We adapt the definition in 3.1.2 of predicates and formulas to the newly introduced decorated connectives $\to^{\mathrm{c}}, \forall^{\mathrm{c}}$ and $\to^{\mathrm{nc}}, \forall^{\mathrm{nc}}$. Let $\to$ denote either $\to^{\mathrm{c}}$ or $\to^{\mathrm{nc}}$, and similarly $\forall$ either $\forall^{\mathrm{c}}$ or $\forall^{\mathrm{nc}}$. Then the definition in 3.1.2 can be read as it stands.

We also need to adapt our definition of TCF to the decorated connectives $\to^{\mathrm{c}}, \to^{\mathrm{nc}}$ and $\forall^{\mathrm{c}}, \forall^{\mathrm{nc}}$. The introduction and elimination rules for $\to^{\mathrm{c}}$ and $\forall^{\mathrm{c}}$ remain as before, and also the elimination rules for $\to^{\mathrm{nc}}$ and $\forall^{\mathrm{nc}}$. However, the introduction rules for $\to^{\mathrm{nc}}$ and $\forall^{\mathrm{nc}}$ must be restricted: the abstracted (assumption or object) variable must be "non-computational", in the following sense. Simultaneously with a derivation $M$ we define the sets $\mathrm{CV}(M)$ and $\mathrm{CA}(M)$ of *computational* object and assumption variables of $M$, as follows. Let $M^A$ be a derivation. If $A$ is non-computational (n.c.), i.e., the type $\tau(A)$ of $A$ (defined below in 3.2.5) is the "nulltype" symbol $\circ$, then $\mathrm{CV}(M^A) := \mathrm{CA}(M^A) := \emptyset$. Otherwise

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

and similarly

$$\begin{aligned}
&\mathrm{CA}(c^A) := \emptyset \quad (c^A \text{ an axiom}),\\
&\mathrm{CA}(u^A) := \{u\},
\end{aligned}$$

$$\mathrm{CA}((\lambda_{u^A} M^B)^{A\to^{\mathrm{c}}B}) := \mathrm{CA}((\lambda_{u^A} M^B)^{A\to^{\mathrm{nc}}B}) := \mathrm{CA}(M^A)\setminus\{u\},$$
$$\mathrm{CA}((M^{A\to^{\mathrm{c}}B}N^A)^B) := \mathrm{CA}(M)\cup\mathrm{CA}(N),$$
$$\mathrm{CA}((M^{A\to^{\mathrm{nc}}B}N^A)^B) := \mathrm{CA}(M),$$
$$\mathrm{CA}((\lambda_x M^A)^{\forall^{\mathrm{c}}_x A}) := \mathrm{CA}((\lambda_x M^A)^{\forall^{\mathrm{nc}}_x A}) := \mathrm{CA}(M),$$
$$\mathrm{CA}((M^{\forall^{\mathrm{c}}_x A(x)}r)^{A(r)}) := \mathrm{CA}((M^{\forall^{\mathrm{nc}}_x A(x)}r)^{A(r)}) := \mathrm{CA}(M).$$

The introduction rules for $\to^{\mathrm{nc}}$ and $\forall^{\mathrm{nc}}$ then are

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

(ii) If $M^A$ is a derivation, $x$ is not free in any formula of a free assumption variable of $M$ and $x \notin \mathrm{CV}(M)$, then $(\lambda_x M^A)^{\forall^{\mathrm{nc}}_x A}$ is a derivation.

An alternative way to formulate these rules is simultaneously with the notion of the "extracted term" $\mathrm{et}(M)$ of a derivation $M$. This will be done in 3.2.6.

**3.2.3. Decorating inductive definitions.** Now we can and should decorate inductive definitions. The introduction axioms are

$$K_i := \forall^{\mathrm{c/nc}}_{\vec{x}}((A_\nu(I))_{\nu<n} \to^{\mathrm{c/nc}} I\vec{r}) \tag{3.3}$$

and the elimination axiom is

$$\forall^{\mathrm{nc}}_{\vec{x}}(I\vec{x} \to^{\mathrm{c}} (\forall^{\mathrm{c/nc}}_{\vec{x}_i}((A_{i\nu}(I \wedge^{\mathrm{d}} X))_{\nu<n_i} \to^{\mathrm{c/nc}} X\vec{r}_i))_{i<k} \to^{\mathrm{c}} X\vec{x}) \tag{3.4}$$

where $I \wedge^{\mathrm{d}} X$ abbreviates $\{\,\vec{x} \mid I\vec{x} \wedge^{\mathrm{d}} X\vec{x}\,\}$ with $\wedge^{\mathrm{d}}$ defined below.

Let us decorate the inductively defined predicates in 3.1.3, that is, take computational aspects into account. For $\exists$, $\wedge$ and $\vee$ we obtain $\exists^{\mathrm{d}}, \exists^{\mathrm{l}}, \exists^{\mathrm{r}}, \exists^{\mathrm{u}}$, $\wedge^{\mathrm{d}}, \wedge^{\mathrm{l}}, \wedge^{\mathrm{r}}, \wedge^{\mathrm{u}}$ $\vee^{\mathrm{d}}, \vee^{\mathrm{l}}, \vee^{\mathrm{r}}, \vee^{\mathrm{u}}$ with d for "double", l for "left", r for "right" and u for "uniform". They are defined by their introduction axioms, which involve both $\to^{\mathrm{c}}, \forall^{\mathrm{c}}$ and $\to^{\mathrm{nc}}, \forall^{\mathrm{nc}}$.

$$\forall^{\mathrm{c}}_x(A \to^{\mathrm{c}} \exists^{\mathrm{d}}_x A), \qquad \exists^{\mathrm{d}}_x A \to^{\mathrm{c}} \forall^{\mathrm{c}}_x(A \to^{\mathrm{c}} P) \to^{\mathrm{c}} P,$$
$$\forall^{\mathrm{c}}_x(A \to^{\mathrm{nc}} \exists^{\mathrm{l}}_x A), \qquad \exists^{\mathrm{l}}_x A \to^{\mathrm{c}} \forall^{\mathrm{c}}_x(A \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P,$$
$$\forall^{\mathrm{nc}}_x(A \to^{\mathrm{c}} \exists^{\mathrm{r}}_x A), \qquad \exists^{\mathrm{r}}_x A \to^{\mathrm{c}} \forall^{\mathrm{nc}}_x(A \to^{\mathrm{c}} P) \to^{\mathrm{c}} P,$$
$$\forall^{\mathrm{nc}}_x(A \to^{\mathrm{nc}} \exists^{\mathrm{u}}_x A), \qquad \exists^{\mathrm{u}}_x A \to^{\mathrm{c}} \forall^{\mathrm{nc}}_x(A \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P,$$

and similar for $\wedge$:

$$A \to^{\mathrm{c}} B \to^{\mathrm{c}} A \wedge^{\mathrm{d}} B, \qquad A \wedge^{\mathrm{d}} B \to^{\mathrm{c}} (A \to^{\mathrm{c}} B \to^{\mathrm{c}} P) \to^{\mathrm{c}} P,$$
$$A \to^{\mathrm{c}} B \to^{\mathrm{nc}} A \wedge^{\mathrm{l}} B, \qquad A \wedge^{\mathrm{l}} B \to^{\mathrm{c}} (A \to^{\mathrm{c}} B \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P,$$
$$A \to^{\mathrm{nc}} B \to^{\mathrm{c}} A \wedge^{\mathrm{r}} B, \qquad A \wedge^{\mathrm{r}} B \to^{\mathrm{c}} (A \to^{\mathrm{nc}} B \to^{\mathrm{c}} P) \to^{\mathrm{c}} P$$
$$A \to^{\mathrm{nc}} B \to^{\mathrm{nc}} A \wedge^{\mathrm{u}} B, \qquad A \wedge^{\mathrm{u}} B \to^{\mathrm{c}} (A \to^{\mathrm{nc}} B \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P$$

and for $\vee$:

$$\begin{array}{ll} A \to^{\mathrm{c}} A \vee^{\mathrm{d}} B, & B \to^{\mathrm{c}} A \vee^{\mathrm{d}} B, \\ A \to^{\mathrm{c}} A \vee^{\mathrm{l}} B, & B \to^{\mathrm{nc}} A \vee^{\mathrm{l}} B, \\ A \to^{\mathrm{nc}} A \vee^{\mathrm{r}} B, & B \to^{\mathrm{c}} A \vee^{\mathrm{r}} B, \\ A \to^{\mathrm{nc}} A \vee^{\mathrm{u}} B, & B \to^{\mathrm{nc}} A \vee^{\mathrm{u}} B \end{array}$$

with elimination schemes

$$\begin{array}{l} A \vee^{\mathrm{d}} B \to^{\mathrm{c}} (A \to^{\mathrm{c}} P) \to^{\mathrm{c}} (B \to^{\mathrm{c}} P) \to^{\mathrm{c}} P, \\ A \vee^{\mathrm{l}} B \to^{\mathrm{c}} (A \to^{\mathrm{c}} P) \to^{\mathrm{c}} (B \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P, \\ A \vee^{\mathrm{r}} B \to^{\mathrm{c}} (A \to^{\mathrm{nc}} P) \to^{\mathrm{c}} (B \to^{\mathrm{c}} P) \to^{\mathrm{c}} P, \\ A \vee^{\mathrm{u}} B \to^{\mathrm{c}} (A \to^{\mathrm{nc}} P) \to^{\mathrm{c}} (B \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P. \end{array}$$

Let $\prec$ be a predicate variable representing a binary relation. A computational variant of the inductively defined *transitive closure* $\mathrm{TC}_{\prec}$ of $\prec$ has introduction axioms

$$\begin{array}{l} \forall^{\mathrm{c}}_{x,y}(x \prec y \to^{\mathrm{nc}} \mathrm{TC}_{\prec}(x,y)), \\ \forall^{\mathrm{c}}_{x,y}\forall^{\mathrm{nc}}_{z}(x \prec y \to^{\mathrm{nc}} \mathrm{TC}_{\prec}(y,z) \to^{\mathrm{c}} \mathrm{TC}_{\prec}(x,z)), \end{array}$$

and the elimination scheme is

$$\begin{array}{l} \forall^{\mathrm{nc}}_{x,y}(\mathrm{TC}_{\prec}(x,y) \to^{\mathrm{c}} \forall^{\mathrm{c}}_{x,y}(x \prec y \to^{\mathrm{nc}} Pxy) \to^{\mathrm{c}} \\ \qquad \forall^{\mathrm{c}}_{x,y}\forall^{\mathrm{nc}}_{z}(x \prec y \to^{\mathrm{nc}} \mathrm{TC}_{\prec}(y,z) \to^{\mathrm{c}} Pyz \to^{\mathrm{c}} Pxz) \to^{\mathrm{c}} \\ \qquad Pxy). \end{array}$$

Consider the accessible part of a binary relation $\prec$. A computational variant $\mathrm{Acc}_{\prec}$ is determined by the introduction axioms

$$\begin{array}{l} \forall^{\mathrm{c}}_{x}(\mathbf{F} \to^{\mathrm{nc}} \mathrm{Acc}_{\prec}(x)), \\ \forall^{\mathrm{nc}}_{x}(\forall^{\mathrm{c}}_{y \prec x}\mathrm{Acc}_{\prec}(y) \to^{\mathrm{c}} \mathrm{Acc}_{\prec}(x)), \end{array}$$

where $\forall^{\mathrm{c}}_{y \prec x}A$ stands for $\forall^{\mathrm{c}}_{y}(y \prec x \to^{\mathrm{nc}} A)$. The elimination scheme is

$$\begin{array}{l} \forall^{\mathrm{nc}}_{x}(\mathrm{Acc}_{\prec}(x) \to^{\mathrm{c}} \forall^{\mathrm{c}}_{x}(\mathbf{F} \to^{\mathrm{nc}} Px) \to^{\mathrm{c}} \\ \qquad \forall^{\mathrm{nc}}_{x}(\forall^{\mathrm{c}}_{y \prec x}\mathrm{Acc}_{\prec}(y) \to^{\mathrm{c}} \forall^{\mathrm{c}}_{y \prec x}Py \to^{\mathrm{c}} Px) \to^{\mathrm{c}} \\ \qquad Px). \end{array}$$

**3.2.4. Totality and induction.** In 2.1.6 we have defined what the total and structure-total ideals of a finitary algebra are. We now inductively define general totality predicates. Let us first look at some examples. The clauses defining totality for the algebra $\mathbf{N}$ are

$$T_{\mathbf{N}}0, \qquad \forall^{\mathrm{nc}}_n(T_{\mathbf{N}}n \to^{\mathrm{c}} T_{\mathbf{N}}(\mathrm{S}n)).$$

The least-fixed-point axiom is

$$\forall^{\mathrm{nc}}_n(T_{\mathbf{N}}n \to^{\mathrm{c}} X0 \to^{\mathrm{c}} \forall^{\mathrm{nc}}_n(T_{\mathbf{N}}n \to^{\mathrm{c}} Xn \to^{\mathrm{c}} X(\mathrm{S}n)) \to^{\mathrm{c}} Xn).$$

Clearly the partial continuous functionals with $T_{\mathbf{N}}$ interpreted as the total ideals for $\mathbf{N}$ provide a model of TCF extended by these axioms.

For the algebra $\mathbf{D}$ of derivations totality is inductively defined by the clauses

$$T_{\mathbf{D}}0^{\mathbf{D}}, \qquad \forall^{\mathrm{nc}}_{x,y}(T_{\mathbf{D}}x \to^{\mathrm{c}} T_{\mathbf{D}}y \to^{\mathrm{c}} T_{\mathbf{D}}(\mathrm{C}^{\mathbf{D}\to\mathbf{D}\to\mathbf{D}}xy)),$$

with least-fixed-point axiom

$$\begin{aligned}\forall^{\mathrm{nc}}_x(T_{\mathbf{D}}x \to^{\mathrm{c}} X0^{\mathbf{D}} \to^{\mathrm{c}}\ & \\ & \forall^{\mathrm{nc}}_{x,y}(T_{\mathbf{D}}x \to^{\mathrm{c}} T_{\mathbf{D}}y \to^{\mathrm{c}} Xx \to^{\mathrm{c}} Xy \to^{\mathrm{c}} X(\mathrm{C}^{\mathbf{D}\to\mathbf{D}\to\mathbf{D}}xy)) \to^{\mathrm{c}} \\ & Xx).\end{aligned}$$

Again, the partial continuous functionals with $T_{\mathbf{D}}$ interpreted as the total ideals for $\mathbf{D}$ (i.e., the finite derivations) provide a model.

Generally we define

(i) $\mathrm{RT}_\rho$ called *relative totality*, and its special cases
(ii) $T_\rho$ called (absolute) *totality* and
(iii) $\mathrm{ST}_\rho$ called *structural totality*.

The least-fixed-point axiom for $\mathrm{ST}_\iota$ will provide us with the induction axiom for the algebra $\iota$.

The definition of $\mathrm{RT}_\rho$ is relative to an assigment of predicate variables $Y$ of arity $(\alpha)$ to type variables $\alpha$.

Definition (Relative totality RT). Let $\iota = \mu_\xi(\kappa_0, \dots, \kappa_{k-1}) \in \mathrm{Alg}(\vec{\alpha}\,)$ with $\kappa_i = (\rho_\nu(\vec{\alpha}, \xi))_{\nu<n} \to \xi$. Then $\mathrm{RT}_\iota := \mu_X(K_0, \dots, K_{k-1})$, with

$$K_i := \forall^{\mathrm{nc}}_{\vec{x}}((\mathrm{RT}_{\rho_\nu}(\vec{Y}, X)x_\nu)_{\nu<n} \to^{\mathrm{c}} X(\mathrm{C}_i\vec{x}\,))$$

and

$$\begin{aligned}\mathrm{RT}_{\alpha_j}(\vec{Y}, X) &:= Y_j,\\ \mathrm{RT}_{\xi}(\vec{Y}, X) &:= X,\\ \mathrm{RT}_{\sigma\to\rho}(\vec{Y}, X) &:= \{\, f \mid \forall^{\mathrm{nc}}_{\vec{x}}(\mathrm{RT}_\sigma\vec{x} \to^{\mathrm{c}} \mathrm{RT}_\rho(\vec{Y}, X)(f\vec{x}\,))\,\}.\end{aligned}$$

For important special cases of the parameter predicates $\vec{Y}$ we introduce a separate notation. Suppose we want to argue about total ideals only. Note that this only makes sense when when no type variables occur. However, to allow a certain amount of abstract reasing (involving type variables to be substituted later by concrete closed types), we introduce special predicate variables $T_\alpha$ which under a substitution $\alpha \mapsto \rho$ with $\rho$ closed turn into the inductively defined predicate $T_\rho$. Using this convention we define totality for an arbitrary algebra by specializing $Y$ of arity $(\rho)$ to $T_\rho$.

Definition (Absolute totality $T$). Let $\iota = \mu_\xi(\kappa_0, \dots, \kappa_{k-1}) \in \mathrm{Alg}(\vec{\alpha}\,)$ with $\kappa_i = (\rho_\nu(\vec{\alpha}, \xi))_{\nu<n} \to \xi$. Then $T_\iota := \mu_X(K_0, \dots, K_{k-1})$, with

$$K_i := \forall^{\mathrm{nc}}_{\vec{x}}((T_{\rho_\nu}(X)x_\nu)_{\nu<n} \to^{\mathrm{c}} X(\mathrm{C}_i\vec{x}\,))$$

and

$$\begin{aligned} T_{\alpha_j}(X) &:= T_{\alpha_j}, \\ T_\xi(X) &:= X, \\ T_{\sigma\to\rho}(X) &:= \{\, f \mid \forall^{\mathrm{nc}}_{\vec{x}}(T_\sigma\vec{x} \to^{\mathrm{c}} T_\rho(X)(f\vec{x}\,))\,\}. \end{aligned}$$

Another important special case occurs when we substitute the predicate variables $Y$ by truth predicates. The resulting totality predicate is called structural totality.

Definition (Structural totality ST). Let $\iota = \mu_\xi(\kappa_0, \dots, \kappa_{k-1}) \in \mathrm{Alg}(\vec{\alpha}\,)$ with $\kappa_i = (\rho_\nu(\vec{\alpha}, \xi))_{\nu<n} \to \xi$. Then $\mathrm{ST}_\iota := \mu_X(K_0, \dots, K_{k-1})$, with

$$K_i := \forall^{\mathrm{nc}}_{\vec{x}}((\mathrm{ST}_{\rho_\nu}(X)x_\nu)_{\nu<n} \to^{\mathrm{c}} X(\mathrm{C}_i\vec{x}\,))$$

and

$$\begin{aligned} \mathrm{ST}_{\alpha_j}(X) &:= \{\, x \mid \top\,\} \quad \text{(omitted whenever possible)}, \\ \mathrm{ST}_\xi(X) &:= X, \\ \mathrm{ST}_{\sigma\to\rho}(X) &:= \{\, f \mid \forall^{\mathrm{nc}}_{\vec{x}}(\mathrm{ST}_\sigma\vec{x} \to^{\mathrm{c}} \mathrm{ST}_\rho(X)(f\vec{x}\,))\,\}. \end{aligned}$$

For example, the main clause for the predicate $\mathrm{ST}_{\mathbf{L}(\alpha)}$ expressing structural totality of lists of elements of type $\alpha$ is

$$\forall^{\mathrm{nc}}_{x,l}(\underbrace{\mathrm{ST}_\alpha(X)x}_{\top;\ \text{omit}} \to^{\mathrm{c}} \underbrace{\mathrm{ST}_\xi(X)}_{X}\, l \to^{\mathrm{c}} X(x :: l))$$

where $x :: l$ is shorthand for $\mathrm{Cons}(x, l)$. It leads to the introduction axiom

$$\forall^{\mathrm{nc}}_{x,l}(\mathrm{ST}_{\mathbf{L}(\alpha)}l \to^{\mathrm{c}} \mathrm{ST}_{\mathbf{L}(\alpha)}(x :: l))$$

with no assumptions on $x$.

The least-fixed-point axiom for $\mathrm{ST}_{\mathbf{L}(\alpha)}$ is according to (3.4)

$$\forall^{\mathrm{nc}}_l(\mathrm{ST}(l) \to^{\mathrm{c}} X(\mathrm{Nil}) \to^{\mathrm{c}} \forall^{\mathrm{nc}}_{x,l}((\mathrm{ST} \wedge^{\mathrm{d}} X)l \to^{\mathrm{c}} X(x :: l)) \to^{\mathrm{c}} Xl^{\mathbf{L}(\rho)}).$$

Written differently (with "duplication") we obtain the induction axiom

$$\forall^{\mathrm{nc}}_l(\mathrm{ST}(l) \to^{\mathrm{c}} X(\mathrm{Nil}) \to \forall^{\mathrm{nc}}_{x,l}(\mathrm{ST}(l) \to^{\mathrm{c}} Xl \to^{\mathrm{c}} X(x :: l)) \to^{\mathrm{c}} Xl^{\mathbf{L}(\rho)})$$

denoted $\mathrm{Ind}_{l,X}$.

Note that in all these definitions we allow usage of totality predicates for previously introduced algebras $\iota'$. An example is totality $T_{\mathbf{T}}$ for the algebra $\mathbf{T}$ of finitely branching trees. It is defined by the single clause

$$\forall^{\mathrm{nc}}_{as}(\mathrm{RT}_{\mathbf{L}(\mathbf{T})}(T_{\mathbf{T}})(as) \to^{\mathrm{c}} T_{\mathbf{T}}(\mathrm{Branch}(as))).$$

Clearly all three notions of totality coincide for algebras without type parameters. Abbreviating $\forall^{\mathrm{nc}}_x(Tx \to^{\mathrm{c}} A)$ by $\forall^{\mathrm{c}}_{x\in T} A$ we obtain from the elimination axioms *computational induction schemes*, for example

$$\mathrm{Ind}_{p,P}\colon \forall^{\mathrm{c}}_{p\in T}(P\mathtt{tt} \to^{\mathrm{c}} P\mathtt{ff} \to^{\mathrm{c}} Pp^{\mathbf{B}}),$$
$$\mathrm{Ind}_{n,P}\colon \forall^{\mathrm{c}}_{n\in T}(P0 \to^{\mathrm{c}} \forall^{\mathrm{c}}_{n\in T}(Pn \to^{\mathrm{c}} P(\mathrm{S}n)) \to^{\mathrm{c}} Pn^{\mathbf{N}}).$$

The types of these formulas (as defined in 3.2.5) will be the types of the recursion operators of the respective algebras.

**3.2.5. The type of a formula, realizability and witnesses.** For every formula or predicate $C$ we define $\tau(C)$ (a type or the "nulltype" symbol $\circ$). In case $\tau(C) = \circ$ proofs of $C$ have no computational content; such $C$ are called *non-computational* (n.c.) (or *Harrop*); the other ones are called *computationally relevant* (c.r.). The definition can be conveniently written if we extend the use of $\rho \to \sigma$ to the nulltype symbol $\circ$:

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

Definition (Type $\tau(C)$ of a formula or predicate $C$, and $\iota_I$). Assume a global injective assignment of a type variable $\xi$ to every predicate variable $X$.

$$\tau(P\vec{r}) := \tau(P),$$
$$\tau(A \to^{\mathrm{c}} B) := (\tau(A) \to \tau(B)), \quad \tau(A \to^{\mathrm{nc}} B) := \tau(B),$$
$$\tau(\forall^{\mathrm{c}}_{x^\rho} A) := (\rho \to \tau(A)), \quad \tau(\forall^{\mathrm{nc}}_{x^\rho} A) := \tau(A),$$
$$\tau(X) := \xi,$$
$$\tau(\{\, \vec{x} \mid A \,\}) := \tau(A),$$
$$\tau(\mu^{\mathrm{nc}}_X(K_0, \dots, K_{k-1})) := \circ,$$

$$\tau(\mu_X(\forall^{\mathrm{nc}}_{\vec{x}_i}\forall^{\mathrm{c}}_{\vec{y}_i}(\vec{A}_i \to^{\mathrm{nc}} \vec{B}_i \to^{\mathrm{c}} X\vec{r}_i))_{i<k}) := \mu_\xi(\tau(\vec{y}_i) \to \tau(\vec{B}_i) \to \xi)_{i<k}.$$

We call $\iota_I := \mu_\xi(\tau(\vec{y}_i) \to \tau(\vec{B}_i) \to \xi)_{i<k}$ the algebra associated with the inductive predicate $I := \mu_X(\forall^{\mathrm{nc}}_{\vec{x}_i}\forall^{\mathrm{c}}_{\vec{y}_i}(\vec{A}_i \to^{\mathrm{nc}} \vec{B}_i \to^{\mathrm{c}} X\vec{r}_i))_{i<k}$.

Hence

$$\tau(I\vec{r}) = \begin{cases} \iota_I & \text{if } I \text{ is c.r.,} \\ \circ & \text{if } I \text{ is n.c.} \end{cases}$$

We now define *realizability*. It will be convenient to introduce a special "nullterm" symbol $\varepsilon$ to be used as a "realizer" for n.c. formulas. We extend term application to the nullterm symbol by

$$\varepsilon t := \varepsilon, \quad t\varepsilon := t, \quad \varepsilon\varepsilon := \varepsilon.$$

Definition ($t$ realizes $A$). Let $A$ be a formula and $t$ either a term of type $\tau(A)$ if the latter is a type, or the nullterm symbol $\varepsilon$ for n.c. $A$. We assume an injective global assignment, giving for every predicate variable $X$ of arity $\vec{\rho}$ a predicate variable $X^{\mathbf{r}}$ of arity $(\tau(X), \vec{\rho})$.

$$\begin{aligned}
&t\ \mathbf{r}\ X\vec{r} := X^{\mathbf{r}}t\vec{r},\\
&t\ \mathbf{r}\ (A \to^{\mathrm{c}} B) := \forall^{\mathrm{nc}}_x(x\ \mathbf{r}\ A\ \to^{\mathrm{nc}}\ tx\ \mathbf{r}\ B),\\
&t\ \mathbf{r}\ (A \to^{\mathrm{nc}} B) := \forall^{\mathrm{nc}}_x(x\ \mathbf{r}\ A\ \to^{\mathrm{nc}}\ t\ \mathbf{r}\ B),\\
&t\ \mathbf{r}\ \forall^{\mathrm{c}}_x A := \forall^{\mathrm{nc}}_x(tx\ \mathbf{r}\ A),\\
&t\ \mathbf{r}\ \forall^{\mathrm{nc}}_x A := \forall^{\mathrm{nc}}_x(t\ \mathbf{r}\ A),\\
&t\ \mathbf{r}\ (\mu_X(\forall^{\mathrm{nc}}_{\vec{x}_i}\forall^{\mathrm{c}}_{\vec{y}_i}((A_{i\nu})_{\nu<n_i} \to^{\mathrm{nc}} (B_{i\nu})_{\nu<m_i} \to^{\mathrm{c}} X\vec{r}_i))_{i<k})\vec{s} := I^{\mathbf{r}}t\vec{s}
\end{aligned}$$

with

$$\begin{aligned}
I^{\mathbf{r}} := \{\, w, \vec{x} \mid (\mu^{\mathrm{nc}}_X(\forall^{\mathrm{nc}}_{\vec{x}_i,\vec{y}_i,\vec{u}_i}((\exists_{u_{i\nu}} u_{i\nu}\ \mathbf{r}\ A_{i\nu})_{\nu<n_i} \to^{\mathrm{nc}} (v_{i\nu}\ \mathbf{r}\ B_{i\nu})_{\nu<m_i} \to^{\mathrm{nc}}\\
X(\mathrm{C}_i\vec{y}_i\vec{v}_i)\vec{r}_i))_{i<k})w\vec{x}\,\}.
\end{aligned}$$

In case $A$ is n.c., $\forall^{\mathrm{nc}}_x(x\ \mathbf{r}\ A \to^{\mathrm{nc}} B(x))$ means $\varepsilon\ \mathbf{r}\ A \to^{\mathrm{nc}} B(\varepsilon)$. For a general n.c. inductively defined predicate (with restricted elimination scheme) we define $\varepsilon\ \mathbf{r}\ I\vec{s}$ to be $I\vec{s}$. For the special n.c. inductively defined predicates $I^{\mathbf{r}}$, Eq, $\exists^{\mathrm{u}}$ and $\wedge^{\mathrm{u}}$ introduced above realizability is defined by

$$\begin{aligned}
\varepsilon\ \mathbf{r}\ I^{\mathbf{r}}t\vec{s} &:= I^{\mathbf{r}}t\vec{s},\\
\varepsilon\ \mathbf{r}\ \mathrm{Eq}(t,s) &:= \mathrm{Eq}(t,s),\\
\varepsilon\ \mathbf{r}\ \exists^{\mathrm{u}}_x A &:= \exists^{\mathrm{u}}_{x,y}(y\ \mathbf{r}\ A),\\
\varepsilon\ \mathbf{r}\ (A \wedge^{\mathrm{u}} B) &:= \exists^{\mathrm{u}}_x(x\ \mathbf{r}\ A) \wedge^{\mathrm{u}} \exists^{\mathrm{u}}_y(y\ \mathbf{r}\ B).
\end{aligned}$$

NOTE. Call two formulas $A$ and $A'$ *computationally equivalent* if each of them computationally implies the other, and in addition the identity realizes each of the two derivations of $A' \to^{\mathrm{c}} A$ and of $A \to^{\mathrm{c}} A'$. It is an easy exercise to verify that for n.c. $A$, the formulas $A \to^{\mathrm{c}} B$ and $A \to^{\mathrm{nc}} B$ are computationally equivalent, and hence can be identified. In the sequel we shall simply write $A \to B$ for either of them. Similarly, for n.c. $A$ the two formulas $\forall^{\mathrm{c}}_x A$ and $\forall^{\mathrm{nc}}_x A$ are n.c., and both $\varepsilon\ \mathbf{r}\ \forall^{\mathrm{c}}_x A$ and $\varepsilon\ \mathbf{r}\ \forall^{\mathrm{nc}}_x A$ are defined to be $\forall^{\mathrm{nc}}_x(\varepsilon\ \mathbf{r}\ A)$. Hence they can be identified as well, and we shall simply write $\forall_x A$ for either of them. Since the formula $t\ \mathbf{r}\ A$ is n.c., under this convention the $\to, \forall$-cases in the definition of realizability can be written

$$\begin{aligned}
t\ \mathbf{r}\ (A \to^{\mathrm{c}} B) &:= \forall_x(x\ \mathbf{r}\ A \to t x\ \mathbf{r}\ B),\\
t\ \mathbf{r}\ (A \to^{\mathrm{nc}} B) &:= \forall_x(x\ \mathbf{r}\ A \to t\ \mathbf{r}\ B),\\
t\ \mathbf{r}\ \forall^{\mathrm{c}}_x A &:= \forall_x(t x\ \mathbf{r}\ A),\\
t\ \mathbf{r}\ \forall^{\mathrm{nc}}_x A &:= \forall_x(t\ \mathbf{r}\ A).
\end{aligned}$$

Here are some examples. Consider the totality predicate $T$ for $\mathbf{N}$ inductively defined by the clauses

$$T0, \qquad \forall^{\mathrm{nc}}_n(Tn \to^{\mathrm{c}} T(\mathrm{S}n)).$$

More precisely $T := \mu_X(K_0, K_1)$ with $K_0 := X0$, $K_1 := \forall^{\mathrm{nc}}_n(Xn \to^{\mathrm{c}} X(\mathrm{S}n))$. These clauses have types $\kappa_0 := \tau(K_0) = \tau(X0) = \xi$ and $\kappa_1 := \tau(K_1) = \tau(\forall^{\mathrm{nc}}_n(Xn \to^{\mathrm{c}} X(\mathrm{S}n))) = \xi \to \xi$. Therefore the algebra of witnesses is $\iota_T := \mu_\xi(\xi, \xi \to \xi)$, that is, $\mathbf{N}$ again. The witnessing predicate $T^{\mathbf{r}}$ is defined by the clauses

$$T^{\mathbf{r}}00, \qquad \forall_{n,m}(T^{\mathbf{r}}mn \to T^{\mathbf{r}}(\mathrm{S}m, \mathrm{S}n))$$

and it has as its elimination scheme

$$\begin{aligned}
\forall^{\mathrm{nc}}_n \forall^{\mathrm{c}}_m (T^{\mathbf{r}}mn \to Q(0,0) \to^{\mathrm{c}}\ & \\
& \forall^{\mathrm{nc}}_{n,m}(T^{\mathbf{r}}mn \to Qmn \to^{\mathrm{c}} Q(\mathrm{S}m, \mathrm{S}n)) \to^{\mathrm{c}}\\
& Qmn.
\end{aligned}$$

As an example involving parameters, consider the formula $\exists^{\mathrm{d}}_x A$ with a c.r. formula $A$, and view $\exists^{\mathrm{d}}_x A$ as inductively defined by the clause

$$\forall^{\mathrm{c}}_x(A \to^{\mathrm{c}} \exists^{\mathrm{d}}_x A).$$

More precisely, $\mathrm{Ex}^{\mathrm{d}}(Y) := \mu_X(K_0)$ with $K_0 := \forall^{\mathrm{c}}_x(Yx^\rho \to^{\mathrm{c}} X)$. Then $\exists^{\mathrm{d}}_x A$ abbreviates $\mathrm{Ex}^{\mathrm{d}}(\{\, x^\rho \mid A \,\})$. The single clause has type $\kappa_0 := \tau(K_0) = \tau(\forall^{\mathrm{c}}_x(Yx^\rho \to^{\mathrm{c}} X)) = \rho \to \alpha \to \xi$. Therefore the algebra of witnesses is $\iota := \iota_{\exists^{\mathrm{d}}_x A} := \mu_\xi(\rho \to \alpha \to \xi)$, that is, $\rho \times \alpha$. We write $\langle x, u \rangle$ for the values

of the (only) constructor of $\iota$, i.e., the pairing operator. The witnessing predicate $(\exists^{\mathrm{d}}_x A)^{\mathbf{r}}$ is defined by the clause $K^{\mathbf{r}}_0((\exists^{\mathrm{d}}_x A)^{\mathbf{r}}, \{\, x^\rho \mid A \,\}) :=$

$$\forall_{x,u}(u\ \mathbf{r}\ A \to (\exists^{\mathrm{d}}_x A)^{\mathbf{r}}\langle x, u\rangle)$$

and its elimination scheme is

$$\forall^{\mathrm{c}}_w((\exists^{\mathrm{d}}_x A)^{\mathbf{r}} w \to \forall^{\mathrm{nc}}_{x,u}(u\ \mathbf{r}\ A \to Q\langle x, u\rangle) \to^{\mathrm{c}} Qw).$$

Definition (Leibniz equality Eq and $\exists^{\mathrm{u}}$, $\wedge^{\mathrm{u}}$). The introduction axioms are

$$\forall^{\mathrm{nc}}_x \mathrm{Eq}(x, x), \qquad \forall^{\mathrm{nc}}_x(A \to^{\mathrm{nc}} \exists^{\mathrm{u}}_x A), \qquad A \to^{\mathrm{nc}} B \to^{\mathrm{nc}} A \wedge^{\mathrm{u}} B,$$

and the elimination schemes are

$$\begin{aligned}
&\forall^{\mathrm{nc}}_{x,y}(\mathrm{Eq}(x, y) \to \forall^{\mathrm{nc}}_x Pxx \to^{\mathrm{c}} Pxy),\\
&\exists^{\mathrm{u}}_x A \to \forall^{\mathrm{nc}}_x(A \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P,\\
&A \wedge^{\mathrm{u}} B \to (A \to^{\mathrm{nc}} B \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P.
\end{aligned}$$

An important property of the realizing formulas $t\ \mathbf{r}\ A$ is that they are *invariant*.

Proposition. *$\varepsilon\ \mathbf{r}\ (t\ \mathbf{r}\ A)$ is the same formula as $t\ \mathbf{r}\ A$.*

Proof. By induction on the simultaneous inductive definition of formulas and predicates in 3.1.2.

*Case $t\ \mathbf{r}\ I\vec{s}$.* By definition the formulas $\varepsilon\ \mathbf{r}\ (t\ \mathbf{r}\ I\vec{s})$, $\varepsilon\ \mathbf{r}\ I^{\mathbf{r}}t\vec{s}$, $I^{\mathbf{r}}t\vec{s}$ and $t\ \mathbf{r}\ I\vec{s}$ are identical.

*Case $I^{\mathbf{r}}t\vec{s}$.* By definition $\varepsilon\ \mathbf{r}\ (\varepsilon\ \mathbf{r}\ I^{\mathbf{r}}t\vec{s})$ and $\varepsilon\ \mathbf{r}\ I^{\mathbf{r}}t\vec{s}$ are identical.

*Case* $\mathrm{Eq}(t, s)$. By definition $\varepsilon\ \mathbf{r}\ (\varepsilon\ \mathbf{r}\ (\mathrm{Eq}(t, s)))$ and $\varepsilon\ \mathbf{r}\ (\mathrm{Eq}(t, s))$ are identical.

*Case $\exists^{\mathrm{u}}_x A$.* The following formulas are identical.

$$\begin{aligned}
&\varepsilon\ \mathbf{r}\ (\varepsilon\ \mathbf{r}\ \exists^{\mathrm{u}}_x A),\\
&\varepsilon\ \mathbf{r}\ \exists^{\mathrm{u}}_x \exists^{\mathrm{u}}_y(y\ \mathbf{r}\ A),\\
&\exists^{\mathrm{u}}_x(\varepsilon\ \mathbf{r}\ \exists^{\mathrm{u}}_y(y\ \mathbf{r}\ A)),\\
&\exists^{\mathrm{u}}_x \exists^{\mathrm{u}}_y(\varepsilon\ \mathbf{r}\ (y\ \mathbf{r}\ A)),\\
&\exists^{\mathrm{u}}_x \exists^{\mathrm{u}}_y(y\ \mathbf{r}\ A) \quad \text{by induction hypothesis},\\
&\varepsilon\ \mathbf{r}\ \exists^{\mathrm{u}}_x A.
\end{aligned}$$

*Case $A \wedge^{\mathrm{u}} B$.* The following formulas are identical.

$$\begin{aligned}
&\varepsilon\ \mathbf{r}\ (\varepsilon\ \mathbf{r}\ (A \wedge^{\mathrm{u}} B)),\\
&\varepsilon\ \mathbf{r}\ (\exists^{\mathrm{u}}_x(x\ \mathbf{r}\ A) \wedge^{\mathrm{u}} \exists^{\mathrm{u}}_y(y\ \mathbf{r}\ B)),
\end{aligned}$$

$$\varepsilon\ \mathbf{r}\ \exists^{\mathrm{u}}_x(x\ \mathbf{r}\ A) \wedge^{\mathrm{u}} \varepsilon\ \mathbf{r}\ \exists^{\mathrm{u}}_y(y\ \mathbf{r}\ B)),$$
$$\exists^{\mathrm{u}}_x(\varepsilon\ \mathbf{r}\ (x\ \mathbf{r}\ A)) \wedge^{\mathrm{u}} \exists^{\mathrm{u}}_y(\varepsilon\ \mathbf{r}\ (y\ \mathbf{r}\ B)),$$
$$\exists^{\mathrm{u}}_x(x\ \mathbf{r}\ A) \wedge^{\mathrm{u}} \exists^{\mathrm{u}}_y(y\ \mathbf{r}\ B) \quad \text{by induction hypothesis},$$
$$\varepsilon\ \mathbf{r}\ (A \wedge^{\mathrm{u}} B).$$

*Case* $A \to^{\mathrm{c}} B$. The following formulas are identical.

$$\varepsilon\ \mathbf{r}\ (t\ \mathbf{r}\ (A \to^{\mathrm{c}} B)),$$
$$\varepsilon\ \mathbf{r}\ \forall_x(x\ \mathbf{r}\ A\ \to\ tx\ \mathbf{r}\ B),$$
$$\forall_x(\varepsilon\ \mathbf{r}\ (x\ \mathbf{r}\ A)\ \to\ \varepsilon\ \mathbf{r}\ (tx\ \mathbf{r}\ B)),$$
$$\forall_x(x\ \mathbf{r}\ A\ \to\ tx\ \mathbf{r}\ B) \quad \text{by induction hypothesis},$$
$$t\ \mathbf{r}\ (A \to^{\mathrm{c}} B).$$

*Case* $A \to^{\mathrm{nc}} B$. The following formulas are identical.

$$\varepsilon\ \mathbf{r}\ (t\ \mathbf{r}\ (A \to^{\mathrm{nc}} B)),$$
$$\varepsilon\ \mathbf{r}\ \forall_x(x\ \mathbf{r}\ A\ \to\ t\ \mathbf{r}\ B),$$
$$\forall_x(\varepsilon\ \mathbf{r}\ (x\ \mathbf{r}\ A)\ \to\ \varepsilon\ \mathbf{r}\ (t\ \mathbf{r}\ B)),$$
$$\forall_x(x\ \mathbf{r}\ A\ \to\ t\ \mathbf{r}\ B) \quad \text{by induction hypothesis},$$
$$t\ \mathbf{r}\ (A \to^{\mathrm{nc}} B).$$

*Case* $\forall^{\mathrm{c}}_x A$. The following formulas are identical.

$$\varepsilon\ \mathbf{r}\ (t\ \mathbf{r}\ \forall^{\mathrm{c}}_x A),$$
$$\varepsilon\ \mathbf{r}\ \forall_x(tx\ \mathbf{r}\ A),$$
$$\forall_x(\varepsilon\ \mathbf{r}\ (tx\ \mathbf{r}\ A)),$$
$$\forall_x(tx\ \mathbf{r}\ A) \quad \text{by induction hypothesis},$$
$$t\ \mathbf{r}\ \forall^{\mathrm{c}}_x A.$$

*Case* $\forall^{\mathrm{nc}}_x A$. The following formulas are identical.

$$\varepsilon\ \mathbf{r}\ (t\ \mathbf{r}\ \forall^{\mathrm{nc}}_x A),$$
$$\varepsilon\ \mathbf{r}\ \forall_x(t\ \mathbf{r}\ A),$$
$$\forall_x(\varepsilon\ \mathbf{r}\ (t\ \mathbf{r}\ A)),$$
$$\forall_x(t\ \mathbf{r}\ A) \quad \text{by induction hypothesis},$$
$$t\ \mathbf{r}\ \forall^{\mathrm{nc}}_x A.$$

This completes the proof. □

**3.2.6. Extracted terms.** For a derivation $M$ of a formula $A$ we define its *extracted term* $\mathrm{et}(M)$, of type $\tau(A)$. This definition is relative to a fixed assignment of object variables to assumption variables: to every assumption variable $u^A$ for a formula $A$ we assign an object variable $x_u$ of type $\tau(A)$.

Definition (Extracted term $\mathrm{et}(M)$ of a derivation $M$). For derivations $M^A$ with $A$ n.c. let $\mathrm{et}(M^A) := \varepsilon$. Otherwise

$$
\begin{aligned}
\mathrm{et}(u^A) &:= x_u^{\tau(A)} \quad (x_u^{\tau(A)} \text{ uniquely associated with } u^A),\\
\mathrm{et}((\lambda_{u^A} M^B)^{A\to^{\mathrm{c}}B}) &:= \lambda_{x_u^{\tau(A)}}\mathrm{et}(M),\\
\mathrm{et}((M^{A\to^{\mathrm{c}}B}N^A)^B) &:= \mathrm{et}(M)\mathrm{et}(N),\\
\mathrm{et}((\lambda_{x^\rho} M^A)^{\forall^{\mathrm{c}}_x A}) &:= \lambda_{x^\rho}\mathrm{et}(M),\\
\mathrm{et}((M^{\forall^{\mathrm{c}}_x A(x)}r)^{A(r)}) &:= \mathrm{et}(M)r,\\
\mathrm{et}((\lambda_{u^A} M^B)^{A\to^{\mathrm{nc}}B}) &:= \mathrm{et}(M),\\
\mathrm{et}((M^{A\to^{\mathrm{nc}}B}N^A)^B) &:= \mathrm{et}(M),\\
\mathrm{et}((\lambda_{x^\rho} M^A)^{\forall^{\mathrm{nc}}_x A}) &:= \mathrm{et}(M),\\
\mathrm{et}((M^{\forall^{\mathrm{nc}}_x A(x)}r)^{A(r)}) &:= \mathrm{et}(M).
\end{aligned}
$$

Here $\lambda_{x_u^{\tau(A)}}\mathrm{et}(M)$ means just $\mathrm{et}(M)$ if $A$ is n.c.

It remains to define extracted terms for the axioms. Consider a (c.r.) inductively defined predicate $I$. For its introduction axioms (3.3) and elimination axiom (3.4) define $\mathrm{et}(I_i^+) := \mathrm{C}_i$ and $\mathrm{et}(I^-) := \mathcal{R}$, where both the constructor $\mathrm{C}_i$ and the recursion operator $\mathcal{R}$ refer to the algebra $\iota_I$ associated with $I$.

Now consider the special non-computational inductively defined predicates. Since they are n.c., we only need to define extracted terms for their elimination axioms. For the witnessing predicate $I^{\mathbf{r}}$ we define $\mathrm{et}((I^{\mathbf{r}})^-) := \mathcal{R}$ (referring to the algebra $\iota_I$ again), and for Leibniz equality Eq, the n.c. existential quantifier $\exists^{\mathrm{u}}_x A$ and conjunction $A \wedge^{\mathrm{u}} B$ we take identities of the appropriate type.

Remark. If derivations $M$ are defined simultaneously with their extracted terms $\mathrm{et}(M)$, we can formulate the introduction rules for $\to^{\mathrm{nc}}$ and $\forall^{\mathrm{nc}}$ by

(i) If $M^B$ is a derivation and $x_{u^A} \notin \mathrm{FV}(\mathrm{et}(M))$, then $(\lambda_{u^A} M^B)^{A\to^{\mathrm{nc}}B}$ is a derivation.

(ii) If $M^A$ is a derivation, $x$ is not free in any formula of a free assumption variable of $M$ and $x \notin \mathrm{FV}(\mathrm{et}(M))$, then $(\lambda_x M^A)^{\forall^{\mathrm{nc}}_x A}$ is a derivation.

**3.2.7. Soundness.** One can prove that every theorem in $\mathrm{TCF} + \mathrm{Ax}_{\mathrm{nci}}$ has a realizer: the extracted term of its proof. Here $(\mathrm{Ax}_{\mathrm{nci}})$ is an arbitrary set of non-computational invariant formulas viewed as axioms.

Theorem (Soundness). *Let $M$ be a derivation of $A$ from assumptions $u_i \colon C_i$ $(i < n)$. Then we can derive* $\mathrm{et}(M)$ $\mathbf{r}$ *$A$ from assumptions $x_{u_i}$* $\mathbf{r}$ *$C_i$ (with $x_{u_i} := \varepsilon$ in case $C_i$ is n.c.).*

For the proof is (by induction on $M$) we have to refer to the literature.

# Bibliography


H. Bachmann. *Transfinite Zahlen.* Springer Verlag, Berlin, Heidelberg, New York, 1955.

U. Berger. Program extraction from normalization proofs. In M. Bezem and J. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *LNCS*, pages 91–106. Springer Verlag, Berlin, Heidelberg, New York, 1993.

U. Berger. Uniform Heyting arithmetic. *Annals of Pure and Applied Logic*, 133:125–148, 2005.

U. Berger, M. Eberl, and H. Schwichtenberg. Term rewriting for normalization by evaluation. *Information and Computation*, 183:19–42, 2003.

T. Coquand and A. Spiwack. A proof of strong normalisation using domain theory. In *Proceedings LICS 2006*, pages 307–316, 2006.

S. Feferman, J. W. Dawson, et al., editors. *Kurt Gödel Collected Works, Volume I-V.* Oxford University Press, 1986, 1990, 1995, 2002, 2002.

G. Gentzen. Beweisbarkeit und Unbeweisbarkeit von Anfangsfällen der transfiniten Induktion in der reinen Zahlentheorie. *Mathematische Annalen*, 119:140–161, 1943.

K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunkts. *Dialectica*, 12:280–287, 1958.

T. Hagino. A typed lambda calculus with categorical type constructions. In D. Pitt, A. Poigné, and D. Rydeheard, editors, *Category Theory and Computer Science*, volume 283 of *LNCS*, pages 140–157. Springer Verlag, Berlin, Heidelberg, New York, 1987.

A. Heyting, editor. *Constructivity in Mathematics*, 1959. North-Holland, Amsterdam.

D. Hilbert. Über das Unendliche. *Mathematische Annalen*, 95:161–190, 1925.

A. N. Kolmogorov. Zur Deutung der intuitionistischen Logik. *Math. Zeitschr.*, 35:58–65, 1932.

G. Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In Heyting (1959), pages 101–128.

K. G. Larsen and G. Winskel. Using information systems to solve recursive domain equations. *Information and Computation*, 91:232–258, 1991.

P. Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 179–216. North-Holland, Amsterdam, 1971.

P. Martin-Löf. The domain interpretation of type theory. Talk at the workshop on semantics of programming languages, Chalmers University, Göteborg, August 1983.

G. Mints. Finite investigations of transfinite derivations. *Journal of Soviet Mathematics*, 10:548–596, 1978. Translated from: *Zap. Nauchn. Semin. LOMI* 49 (1975).

G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

G. D. Plotkin. $\mathbf{T}^{\omega}$ as a universal domain. *Journal of Computer and System Sciences*, 17:209–236, 1978.

J. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, 2000.

H. Schwichtenberg and S. S. Wainer. *Proofs and Computations*. Perspectives in Logic. Association for Symbolic Logic and Cambridge University Press, 2012.

D. Scott. Domains for denotational semantics. In E. Nielsen and E. Schmidt, editors, *Automata, Languages and Programming*, volume 140 of *LNCS*, pages 577–613. Springer Verlag, Berlin, Heidelberg, New York, 1982.

A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, second edition, 2000.

# Index