Recursion on the partial continuous functionals

Helmut Schwichtenberg

1 Introduction

We describe a constructive theory of computable functionals, based on the partial continuous functionals as their intendend domain. Such a task had long ago been started by Dana Scott [28], under the well-known abbreviation LCF. However, the prime example of such a theory, Per Martin-Löf's type theory [24] in its present form deals with total (structural recursive) functionals only. An early attempt of Martin-Löf [23] to give a domain theoretic interpretation of his type theory has not even been published, probably because it was felt that a more general approach – such as formal topology [13] – would be more appropriate.

Here we try to make a fresh start, and do full justice to the fundamental notion of computability in finite types, with the partial continuous functionals as underlying domains. The total ones then appear as a dense subset [20, 15, 7, 31, 27, 21], and seem to be best treated in this way.

Computable functionals and logic Types are built from base types by the formation of function types, $\rho \Rightarrow \sigma$. As domains for the base types we choose non-flat (cf. Figure 2) 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.

The naive model of such a finitely typed theory is the full set theoretic hierarchy of functionals of finite types. However, this immediately leads to higher cardinalities, and does not lend itself well for a theory of computability. A more appropriate semantics for typed languages has its roots in work of Kreisel [20] (who used formal neighborhoods) and Kleene [19]. This line of research was taken up and developed in a mathematically more satisfactory way by Scott and Ershov [29, 16]. Today this theory is usually presented in the context of abstract domain theory [31, 3]; it is based on classical logic.

The present work can be seen as an attempt to develop a constructive theory of formal neighborhoods for continuous functionals, in a direct and intuitive style. The task is to replace abstract domain theory by a more concrete and (in case of finitary free algebras) finitary theory of representations. As a framework we use Scott's information systems [30, 22, 31]. It turns out

that we only need to deal with "atomic" and "coherent" information systems (abbreviated acis), which simplifies matters considerably. In this setup the basic notion is that of a "token", or unit of information. The elements of the domain appear as abstract or "ideal" entitites: possibly infinite sets of tokens, which are "consistent" and "deductively closed".

Total functionals One reason to be interested in total functionals is that for base types, that is free algebras, we can prove properties of total objects by structural induction. This is also true for the more general class of *structure-total* objects, where the arguments at parameter positions in constructor terms need not be total. An example is a list whose length is determined, but whose elements need not be total.

We show that the standard way to single out the total functionals from the partial ones works with non-flat base domains as well, and that Berger's proof [7] of Kreisel's [20] density theorem can be adapted.

Terms and their denotational and operational semantics Since we have introduced domains via concrete representations, it is easy to define the computable functionals, simply as recursively enumerable ideals (= sets of tokens). However, this way to deal with computability is too general for concrete applications. In practice, one wants to define computable functionals by recursion equations. We show that and how computation rules [11, 8] can be used to achieve this task. The meaning $[\![\lambda\vec{x}\,M]\!]$ of a term M (with free variables in $\vec{x}$) involving constants D defined by computation rules will be an inductively defined set of tokens $(\vec{U}, b)$, of the type of $\lambda\vec{x}\,M$.

So we extend the term language of Plotkin's PCF [25], by constants defined via "computation rules". One instance of such rules is the definition of the fixed point operators $\mathcal{Y}_\rho$ of type $(\rho \Rightarrow \rho) \Rightarrow \rho$, by $\mathcal{Y}_\rho f = f(\mathcal{Y}_\rho f)$. Another instance is the structural recursion operator $\mathcal{R}^\tau_{\mathbf{N}}$, defined by

$$\mathcal{R}^\tau_{\mathbf{N}}(f, g, 0) = f, \qquad \mathcal{R}^\tau_{\mathbf{N}}(f, g, Sn) = g(n, \mathcal{R}^\tau_{\mathbf{N}}(f, g, n)).$$

Operationally, the term language provides some natural conversion rules to "simplify" terms: β, η, and – for every defined constant D – the defining equations $D\vec{P} \mapsto M$ with non-overlapping constructor patterns $\vec{P}$; the equivalence generated by these conversions is called operational semantics. We show that the (denotational) values are preserved under conversions, including computation rules.

Computational adequacy Clearly we want to know that the conversions mentioned above give rise to a "computationally adequate" operational semantics: If $[\![M]\!] = k$, then the conversion rules suffice to actually reduce M to the numeral k. We show that this holds true in our somewhat extended setting as well, with computation rules and non-flat base domains.

Structural recursion An important example of computation rules are those of the (Gödel) structural recursion operators. We prove their totality, by showing that the rules are strongly normalizing. A predicative proof of this fact has been given in [1], based on Aczel's notion of a set-based relation. Our proof is predicative as well, but – being based on an extension of Tait's method of strong computability predicates – more along the standard line of such proofs. Moreover, it extends the result to the present setting.

Related work The development of constructive theories of computable functionals of finite type began with Gödel's [18]. There the emphasis was on particular computable functionals, the structural (or primitive) recursive ones. In contrast to what was done later by Kreisel, Kleene, Scott and Ershov, the domains for these functionals were not constructed explicitly, but rather considered as described axiomatically by the theory.

Denotational semantics for PCF-like languages is well-developed, and usually (as in Plotkin's [25]) done in a domain-theoretic setting. The study of the semantics of non-overlapping higher type recursion equations - called here computation rules - has been initiated by Berger [11], again in a domain-theoretic setting. Recently [8] he has introduced a "strict" variant of this domain-theoretic semantics, and used it to prove strong normalization of extensions of Gödel's T by different versions of bar recursion. Information systems have been conceived by Scott [30], as an intuitive approach to domains for denotational semantics. The idea to consider atomic information systems is due to Ulrich Berger (unpublished work); coherent information systems have been introduced by Plotkin [26, p.210]. Taking up Kreisel's [20] idea of neighborhood systems, Martin-Löf developed in unpublished (but somewhat distributed) notes [23] a domain theoretic interpretation of his type theory. The intersection type discipline of Coppo and Dezani [5] can be seen as a different style of presenting the idea of a neighborhood system. The desire to have a more general framework for these ideas has lead Martin-Löf, Sambin and others to develop a formal topology [13].

It seems likely that the method in [21, Section 3.5] (which is based on an idea of Ulrich Berger) can be used to prove density in the present case, but this would require some substantial rewriting.

The first proof of an adequacy theorem (not under this name) is due to Plotkin [25, Theorem 3.1]; Plotkin's proof is by induction on the types, and uses a computability predicate. A similar result in a type-theoretic setting is in Martin-Löf's notes [23, Second Theorem]. Adequacy theorems have been proved in many contexts [2, 4, 5, 23]. Coquand [14] – building on the work of Martin-Löf [23] and Berger [8] – observed that the adequacy result even holds for untyped languages, hence also for dependently typed ones.

The problem of proving strong normalization for extensions of typed λ-calculi by higher order rewrite rules has been studied extensively in the

literature [32, 17, 33, 12, 1, 8]. Most of these proofs use impredicative methods (e.g., by reducing the problem to strong normalization of second order propositional logic, called system F by Girard [17]). Our definition of the strong computability predicates and also the proof are related to Zucker's [34] proof of strong normalization of his term system for recursion on the first three number or tree classes. However, Zucker uses a combinatory term system and defines strong computability for closed terms only. Following some ideas in an unpublished note of Berger, Benl (in his diploma thesis [6]) transferred this proof to terms in simply typed λ-calculus, possibly involving free variables. Here it is adapted to the present context.

Organization of the paper In Section 2 atomic coherent information systems are defined, and used as a concrete representation of the relevant domains, based on non-flat and possibly infinitary free algebras. Section 3 deals with total and structure-total ideals; it is shown that the density theorem holds. Section 4 introduces the term language, extending Plotkin's PCF by defined constants and computation rules. The denotational and operational semantics is defined, the former by an inductive definition of a relation $(\vec{U}, b) \in [\![\lambda\vec{x}\, M]\!]$, the latter by conversions which include the computation rules. We prove preservation of values under conversions. Section 5 contains the proof of the adequacy theorem. The structural recursion operators are taken up in Section 6, as an example of computation rules defining total objects. The paper concludes in Section 7 with remarks on an implementation of some of its ideas, in the Minlog proof assistant `www.minlog-system.de` under development in Munich.

2 Partial continuous functionals

Information systems have been introduced by Scott [30], as an intuitive approach to deal constructively with ideal, infinite objects in function spaces, by means of their finite approximations. One works with atomic units of information, called *tokens*, and a notion of *consistency* for finite sets of tokens. Finally there is an *entailment* relation, between consistent finite sets of tokens and single tokens. The *ideals* (or *objects*) of an information system are defined to be the consistent and deductively closed sets of tokens; we write $|\boldsymbol{A}|$ for the set of ideals of $\boldsymbol{A}$. One shows easily that $|\boldsymbol{A}|$ is a domain w.r.t. the inclusion relation. Conversely, every domain with countable basis can be represented as the set of all ideals of an appropriate information system [22].

Here we take Scott's notion of an information system as a basis to introduce the partial continuous functionals. Call an information system *atomic* if the entailment relation $U \vdash b$ is given by $\exists_{a\in U}\{a\} \vdash b$ and hence determined by a transitive relation on A (namely $\{a\} \vdash b$, written $a \geq b$). Call it

coherent [26, p.210] when a finite set U of tokens is consistent iff each two-element subset of it is. We will show below that if $\boldsymbol{B}$ is atomic (coherent), then so is the "function space" $\boldsymbol{A} \to \boldsymbol{B}$. Since our algebras will be given by atomic coherent information systems, this is the only kind of information systems we will have to deal with.

2.1 Types

A free algebra is given by its *constructors*, for instance zero and successor for the natural numbers. We want to treat other data types as well, like lists and binary trees. When dealing with inductively defined sets, it will also be useful to explicitely refer to the generation tree. Such trees are quite often infinitely branching, and hence we allow infinitary free algebras.

The freeness of the constructors is expressed by requiring that their ranges are disjoint and that they are injective. To allow for partiality – which is mandatory when we want to deal with computable objects –, we have to embed our algebras into domains. Both requirements together imply that we need "lazy domains".

Our type system is defined by two type forming operations: arrow types $\rho \Rightarrow \sigma$ and the formation of inductively generated types $\mu\vec{\alpha}\,\vec{\kappa}$, where $\vec{\alpha} = (\alpha_j)_{j=1,\dots,N}$ is a list of distinct "type variables", and $\vec{\kappa} = (\kappa_i)_{i=1,\dots,k}$ is a list of "constructor types", whose argument types contain $\alpha_1, \dots, \alpha_N$ in strictly positive positions only.

For instance, $\mu\alpha(\alpha, \alpha \Rightarrow \alpha)$ is the type of natural numbers; here the list $(\alpha, \alpha \Rightarrow \alpha)$ stands for two generation principles: α for "there is a natural number" (the 0), and $\alpha \Rightarrow \alpha$ for "for every natural number there is a next one" (its successor).

Definition 2.1. Let $\vec{\alpha} = (\alpha_j)_{j=1,\dots,N}$ be a list of distinct type variables. *Types* $\rho, \sigma, \tau, \mu \in \mathrm{T}$ and *constructor types* $\kappa \in \mathrm{KT}(\vec{\alpha})$ are defined inductively by

$$\frac{\vec{\rho}, \vec{\sigma}_1, \dots, \vec{\sigma}_n \in \mathrm{T}}{\vec{\rho} \Rightarrow (\vec{\sigma}_1 \Rightarrow \alpha_{j_1}) \Rightarrow \dots \Rightarrow (\vec{\sigma}_n \Rightarrow \alpha_{j_n}) \Rightarrow \alpha_j \in \mathrm{KT}(\vec{\alpha})} \quad (n \geq 0)$$

$$\frac{\kappa_1, \dots, \kappa_n \in \mathrm{KT}(\vec{\alpha})}{(\mu\vec{\alpha}\,(\kappa_1, \dots, \kappa_n))_j \in \mathrm{T}} \quad (n \geq 1) \qquad \frac{\rho, \sigma \in \mathrm{T}}{\rho \Rightarrow \sigma \in \mathrm{T}}$$

Here $\vec{\rho} \Rightarrow \sigma$ means $\rho_1 \Rightarrow \dots \Rightarrow \rho_m \Rightarrow \sigma$, associated to the right. We reserve μ for types of the form $(\mu\vec{\alpha}\,(\kappa_1, \dots, \kappa_k))_j$. The *parameter types* of μ are the members of all $\vec{\rho}$ appearing in its constructor types $\kappa_1, \dots, \kappa_k$.

Examples.

$\mathbf{U}$	$:= \mu\alpha\,\alpha,$	Unit
$\mathbf{B}$	$:= \mu\alpha\,(\alpha, \alpha),$	Booleans

$\mathbf{N}$	$:= \mu\alpha\,(\alpha, \alpha \Rightarrow \alpha)$,	Natural numbers
$\mathbf{L}(\rho)$	$:= \mu\alpha\,(\alpha, \rho \Rightarrow \alpha \Rightarrow \alpha)$,	Lists
$\rho \otimes \sigma$	$:= \mu\alpha\,\rho \Rightarrow \sigma \Rightarrow \alpha$,	(Tensor) product
$\rho + \sigma$	$:= \mu\alpha\,(\rho \Rightarrow \alpha, \sigma \Rightarrow \alpha)$,	Sum
$(\mathrm{tree}, \mathrm{tlist})$	$:= \mu(\alpha, \beta)\,(\mathbf{N} \Rightarrow \alpha, \beta \Rightarrow \alpha, \beta, \alpha \Rightarrow \beta \Rightarrow \beta)$,	
$\mathbf{Bin}$	$:= \mu\alpha\,(\alpha, \alpha \Rightarrow \alpha \Rightarrow \alpha)$,	Binary trees
$\mathcal{O}$	$:= \mu\alpha\,(\alpha, \alpha \Rightarrow \alpha, (\mathbf{N} \Rightarrow \alpha) \Rightarrow \alpha)$,	Ordinals
$\mathcal{T}_0$	$:= \mathbf{N}$,	
$\mathcal{T}_{n+1}$	$:= \mu\alpha\,(\alpha, (\mathcal{T}_n \Rightarrow \alpha) \Rightarrow \alpha)$.	Trees

Notice that there are many equivalent ways to define these types. For instance, we could take $\mathbf{U} + \mathbf{U}$ to be the type of booleans, and $\mathbf{L}(\mathbf{U})$ to be the type of natural numbers.

A type is called *finitary* if it is a μ-type with all its parameter types $\vec{\rho}$ finitary, and in all its constructor types

$$\vec{\rho} \Rightarrow (\vec{\sigma}_1 \Rightarrow \alpha_{j_1}) \Rightarrow \ldots \Rightarrow (\vec{\sigma}_n \Rightarrow \alpha_{j_n}) \Rightarrow \alpha_j \tag{1}$$

the $\vec{\sigma}_1, \ldots, \vec{\sigma}_n$ are all empty. In the examples above $\mathbf{U}$, $\mathbf{B}$, $\mathbf{N}$, tree, tlist and **Bin** are all finitary, whereas $\mathcal{O}$ and $\mathcal{T}_{n+1}$ are not. $\mathbf{L}(\rho)$, $\rho \otimes \sigma$ and $\rho + \sigma$ are finitary provided their parameter types are. An argument position in a type is called *finitary* if it is occupied by a finitary type.

2.2 Function spaces via atomic coherent information systems

Definition 2.2. An *atomic coherent information system* (abbreviated *acis*) is a triple $(A, \mathrm{Con}, \geq)$ with A a countable set (the *tokens*, denoted $a, b, \ldots$), Con a nonempty set of finite subsets of A (the *consistent* sets or *formal neighborhoods*, denoted $U, V, \ldots$), and $\geq$ a transitive and reflexive relation on A (the *entailment relation*) which satisfy

(a) $\emptyset \in \mathrm{Con}$, and $\{a\} \in \mathrm{Con}$ for every $a \in A$,

(b) $U \in \mathrm{Con}$ iff every two-element subset of U is in Con, and

(c) if $\{a, b\} \in \mathrm{Con}$ and $b \geq c$, then $\{a, c\} \in \mathrm{Con}$.

We write $U \geq a$ for $\exists_{b \in U} b \geq a$, and $U \geq V$ for $\forall_{a \in V} U \geq a$. – Any acis is an information system in the sense of [30]; this follows from

Lemma 2.3. *Let* $\boldsymbol{A} = (A, \mathrm{Con}, \geq)$ *be an acis.* $U \geq V_1, V_2$ *implies* $V_1 \cup V_2 \in \mathrm{Con}$.

Proof. Let $b_1 \in V_1$, $b_2 \in V_2$. Then we have $a_1, a_2 \in U$ such that $a_i \geq b_i$. From $\{a_1, a_2\} \in \mathrm{Con}$ we obtain $\{a_1, b_2\} \in \mathrm{Con}$ by (c), hence $\{b_1, b_2\} \in \mathrm{Con}$ again by (c). □

Definition 2.4. Let $\boldsymbol{A} = (A, \mathrm{Con}_A, \geq_A)$ and $\boldsymbol{B} = (B, \mathrm{Con}_B, \geq_B)$ be acis's. Define $\boldsymbol{A} \to \boldsymbol{B} = (C, \mathrm{Con}, \geq)$ by

$$C := \mathrm{Con}_A \times B,$$
$$\{(U_1, b_1), \ldots (U_n, b_n)\} \in \mathrm{Con} :\leftrightarrow \forall_{i,j}\big(U_i \cup U_j \in \mathrm{Con}_A \to \{b_i, b_j\} \in \mathrm{Con}_B\big),$$
$$(U, b) \geq (V, c) :\leftrightarrow V \geq_A U \wedge b \geq_B c.$$

Lemma 2.5. *Let $\boldsymbol{A} = (A, \mathrm{Con}_A, \geq_A)$ and $\boldsymbol{B} = (B, \mathrm{Con}_B, \geq_B)$ be acis's. Then $\boldsymbol{A} \to \boldsymbol{B}$ is an acis again.*

Proof. Clearly $\geq$ is transitive and reflexive, and the conditions (a) and (b) of an acis hold; it remains to check (c). So let $\{(U_1, b_1), (U_2, b_2)\} \in \mathrm{Con}$ and $(U_2, b_2) \geq (V, c)$, hence $V \geq U_2$ and $b_2 \geq c$. We must show $\{(U_1, b_1), (V, c)\} \in \mathrm{Con}$. So assume $U_1 \cup V \in \mathrm{Con}$; we must show $\{b_1, c\} \in \mathrm{Con}$. Now $U_1 \cup V \in \mathrm{Con}$ and $V \geq U_2$ by the previous lemma imply $U_1 \cup U_2 \in \mathrm{Con}$. But then $\{b_1, b_2\} \in \mathrm{Con}$, hence $\{b_1, c\} \in \mathrm{Con}$ by (c). □

Scott [30] introduced the notion of an *approximable map* from $\boldsymbol{A}$ to $\boldsymbol{B}$. Such a map is given by a relation r between Con_A and B, where $r(U, b)$ intuitively means that whenever we are given the information $U \in \mathrm{Con}_A$ on the argument, then we know that at least the token b appears in the value.

Definition 2.6. Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be acis's. A relation $r \subseteq \mathrm{Con}_A \times B$ is an approximable map from $\boldsymbol{A}$ to $\boldsymbol{B}$ (written $r \colon \boldsymbol{A} \to \boldsymbol{B}$) iff

(a) if $r(U, b_1)$ and $r(U, b_2)$, then $\{b_1, b_2\} \in \mathrm{Con}_B$, and

(b) if $r(U, b)$, $V \geq_A U$ and $b \geq_B c$, then $r(V, c)$.

Theorem 2.7. *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be acis's. The ideals of $\boldsymbol{A} \to \boldsymbol{B}$ are exactly the approximable maps from $\boldsymbol{A}$ to $\boldsymbol{B}$.*

Proof. We show that $r \in |\boldsymbol{A} \to \boldsymbol{B}|$ satisfies the axioms for approximable maps. (a). Let $r(U, b_1)$ and $r(U, b_2)$. Then $\{b_1, b_2\} \in \mathrm{Con}_B$ by the consistency of r. (b). Let $r(U, b)$, $V \geq_A U$ and $b \geq_B c$. Then $(U, b) \geq (V, c)$ by definition, hence $r(V, c)$ by the deductive closure of r.

For the other direction suppose $r \colon \boldsymbol{A} \to \boldsymbol{B}$ is an approximable map. We must show that $r \in |\boldsymbol{A} \to \boldsymbol{B}|$. Consistency of r: Suppose $r(U_1, b_1)$, $r(U_2, b_2)$ and $U = U_1 \cup U_2 \in \mathrm{Con}_A$. We must show that $\{b_1, b_2\} \in \mathrm{Con}_B$. Now by definition of approximable maps, from $r(U_i, b_i)$ and $U \vdash_A U_i$ we obtain $r(U, b_i)$, and hence $\{b_1, b_2\} \in \mathrm{Con}_B$. Deductive closure of r: Suppose $r(U, b)$ and $(U, b) \geq (V, c)$, i.e., $V \geq_A U \wedge b \geq_B c$. Then $r(V, c)$ by definition of approximable maps. □

The set $|\boldsymbol{A}|$ of ideals for $\boldsymbol{A}$ carries a natural topology (the Scott topology), which has the deductive closures $\overline{U}$ of formal neighborhoods U as basis. The continuous maps $f \colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ and the ideals $r \in |\boldsymbol{A} \to \boldsymbol{B}|$ are

in a bijective correspondence. With any $r \in |\boldsymbol{A} \to \boldsymbol{B}|$ we can associate a continuous $|r|\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$:

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

and with any continuous $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ we can associate $\hat{f} \in |\boldsymbol{A} \to \boldsymbol{B}|$:

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

These assignments are inverse to each other, i.e., $f = |\hat{f}|$ and $r = \widehat{|r|}$. – We will usually write $r(z)$ for $|r|(z)$, and similarly $f(U, b)$ for $\hat{f}(U, b)$. It will be clear from the context where the mods and hats should be inserted.

2.3 Algebras with approximations

We can now define the acis of an algebra μ_j, given by constructors C_i. The *tokens* are

- a special one – written $*$ –, which carries no information, and
- all type correct constructor expressions with an outermost C_i, where at any finitary argument position we have a token, and at any other argument position we have a formal neighborhood of the appropriate type.

Two tokens are in the *entailment* relation $\geq$ if either the right hand one is $*$, or they start with the same constructor, and for each finitary argument position either the argument tokens a, b located there satisfy $a \geq b$, or the formal neighborhoods U, V located there satisfy $U \geq V$, as defined above (notice that this is an inductive definition). A finite set of tokens is *consistent* if each two-element subset is; two tokens are consistent if one of them is $*$, or both start with the same constructor and have consistent tokens resp. formal neighborhoods at corresponding argument positions.

For example, the tokens for the algebra $\mathbf{N}$ are as shown in Figure 1. A token a entails another one b iff there is a path from a (up) to b (down). In this case (and similarly for every finitary algebra) a finite set U of tokens is consistent iff it has an upper bound.

Every constructor C generates

$$r_{\mathrm{C}} := \{\, (\vec{U}, b) \mid b = *, \text{ or } b = \mathrm{C}\vec{b} \text{ with } U_i \geq b_i \,\},$$

with b_i tokens or formal neighborhoods. The continuous map $|r_{\mathrm{C}}|$ is defined by

$$|r_{\mathrm{C}}|(\vec{z}) := \{\, b \mid (\vec{U}, b) \in r_{\mathrm{C}} \text{ for some } \vec{U} \subseteq \vec{z} \,\}.$$

Hence the (continuous maps corresponding to) constructors are injective and their ranges are disjoint, which is what we wanted to achieve.

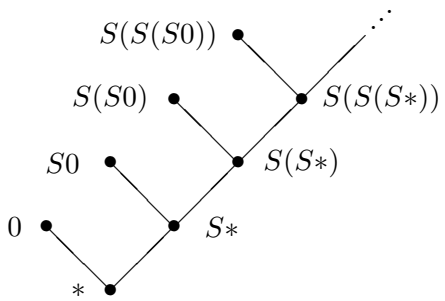

Figure 1: Tokens and entailment for **N**

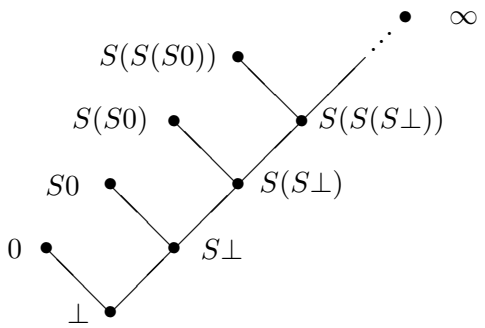

Figure 2: Ideals and inclusion for **N**, i.e., its domain

The ideals x for μ are – as for any information system – the consistent and deductively closed sets of tokens. Clearly all non-$*$ tokens in x begin with the same constructor. For instance, $\{S(S0), S(S*), S*, *\}$, $\{S(S*), S*, *\}$, $\{0, *\}$, $\{*\}$ are ideals for **N**, but also the infinite set $\{\, S^n* \mid n \geq 0 \,\}$. The ideals for **N** and their inclusion relation are pictured in Figure 2. Here we have denoted the ideals $\{*\}$, $\{0, *\}$, $\{\, S^n* \mid n \geq 0 \,\}$ by $\bot$, 0, ∞, respectively, and any other ideal by applications of (the continuous map corresponding to) the constructor S to 0 or $\bot$. The ambiguous notation – S denotes a symbol in constructor expressions and also the continuous map $|r_S|$ – should not lead to confusion.

3 Total functionals

Total ideals are important because one can prove their properties by (structural) induction. We also introduce the concept of *structure-total* ideals, first for a free algebra μ and then for arbitrary types. They are more general, because ideals at parameter positions need not be total, but still allow to argue by induction. An example of the latter notion are lists whose structure (number of Cons's) is known, but whose elements may be partial. This is

of interest, because for such "structure-total" objects an obvious induction principle holds.

In [20] Kreisel states the important density theorem, which says that any finite functional can be extended to a total one. Full proofs of various versions of the density theorem are in [15, 7, 31, 27, 21]. Here we give a proof for the practically important case where the base domains are not just the flat domain of natural numbers, but non-flat and possibly parametrized free algebras.

3.1 Total and structure-total ideals

It is well-known how one can single out the total functionals from the partial ones. One good reason to be interested in total functionals is that for base types, that is free algebras, we can prove properties of total objects by structural induction. This is also true for the slightly more general class of *structure-total* objects, where the arguments at parameter positions in constructor terms need not be total. An example is a list whose length is determined, but whose elements may be partial.

Definition 3.1. The *total* ideals of type ρ are defined inductively.

- **Case** μ. For an algebra μ, the total ideals x are those of the form $\mathrm{C}\vec{z}$ with C a constructor of μ and $\vec{z}$ total (C denotes the continuous function $|r_{\mathrm{C}}|$).
- **Case** $\rho \Rightarrow \sigma$. An ideal r of type $\rho \Rightarrow \sigma$ is total iff for all total z of type ρ, the result $|r|(z)$ of applying r to z is total.

The *structure-total* ideals are defined similarly; the difference is that in case μ the ideals at parameter positions of C need not be total. – We write $x \in G_\rho$ to mean that x is a total ideal of type ρ.

For instance, for $\mathbf{N}$ the ideals 0, $S0$, $S(S0)$ etc. in Figure 2 are total, but $\bot$, $S\bot$, $S(S\bot)$, $\ldots$, ∞ are not. For $\mathbf{L}(\rho)$, precisely all ideals of the form $\mathrm{Cons}(x_1, \ldots \mathrm{Cons}(x_n, \mathrm{Nil}) \ldots)$ are structure-total. The total ones are those where in addition all list elements $x_1, \ldots, x_n$ are total.

For non-flat base domains it is easy to see that there are maximal but not total ideals: ∞ is an example for $\mathbf{N}$. This is less easy for flat base domains; a counterexample has been given by Yuri Ershov in [16]; a more perspicious one (at type $(\mathbf{N} \Rightarrow \mathbf{N}) \Rightarrow \mathbf{N}$) is in [31].

Conversely, the total continuous functionals need not be maximal ideals in $\boldsymbol{C}_\rho$: A counterexample is $\{\, (S^n 0, 0) \mid n \in \mathbf{N} \,\}$, which clearly is a total object of type $\mathbf{N} \Rightarrow \mathbf{N}$ representing the constant function with value 0. However, addition of the pair $(\{*\}, 0)$ yields a different total object of type $\mathbf{N} \Rightarrow \mathbf{N}$. However, it is easy to show both functionals are "equivalent" in the sense that they have the same behaviour on total arguments.

3.2 Induction

The principle of *induction* over (the total ideals of) simultaneous free algebras $\vec{\mu} = \mu\vec{\alpha}\,\vec{\kappa}$ can now be formulated as follows; it clearly holds in our domains. For readability let the variables x_i, y_j range over total ideals only. For the constructor type

$$\kappa_i = \vec{\rho} \Rightarrow (\vec{\sigma}_1 \Rightarrow \alpha_{j_1}) \Rightarrow \ldots \Rightarrow (\vec{\sigma}_n \Rightarrow \alpha_{j_n}) \Rightarrow \alpha_j \in \mathrm{KT}(\vec{\alpha})$$

we have the *step formula*

$$\begin{aligned} D_i := \forall_{y_1^{\rho_1},\ldots,y_m^{\rho_m},y_{m+1}^{\vec{\sigma}_1\Rightarrow\mu_{j_1}},\ldots,y_{m+n}^{\vec{\sigma}_n\Rightarrow\mu_{j_n}}}.\forall_{\vec{x}^{\vec{\sigma}_1}}\,\hat{P}_{j_1}(y_{m+1}\vec{x}) \to \cdots \to \\ \forall_{\vec{x}^{\vec{\sigma}_n}}\,\hat{P}_{j_n}(y_{m+n}\vec{x}) \to \\ \hat{P}_j(\mathrm{C}_i^{\vec{\mu}}(\vec{y})). \end{aligned}$$

$\vec{y} = y_1^{\rho_1}, \ldots, y_m^{\rho_m}, y_{m+1}^{\vec{\sigma}_1\Rightarrow\mu_{j_1}}, \ldots, y_{m+n}^{\vec{\sigma}_n\Rightarrow\mu_{j_n}}$ are the *components* of the ideal $\mathrm{C}_i^{\vec{\mu}}(\vec{y})$ of type μ_j under consideration, and

$$\forall_{\vec{x}^{\vec{\sigma}_1}}\,\hat{P}_{j_1}(y_{m+1}\vec{x}), \ldots, \forall_{\vec{x}^{\vec{\sigma}_n}}\,\hat{P}_{j_n}(y_{m+n}\vec{x})$$

are the hypotheses available in the induction step. The induction axiom $\mathrm{Ind}_{\mu_j}^{\vec{x},\vec{A}}$ with $\vec{x} = (x_j^{\mu_j})_{j=1,\ldots,N}$ and $\vec{A} = (A_j)_{j=1,\ldots,N} = (\hat{P}_j(x_j^{\mu_j}))_{j=1,\ldots,N}$ then proves the formula

$$D_1 \to \cdots \to D_k \to \forall_{x_j^{\mu_j}}\,\hat{P}_j(x_j).$$

We will often write $\mathrm{Ind}_j^{\vec{x},\vec{A}}$ for $\mathrm{Ind}_{\mu_j}^{\vec{x},\vec{A}}$, and omit the upper indices $\vec{x}, \vec{A}$ when they are clear from the context. In case of a non-simultaneous free algebra, i.e. of type $\mu\alpha\,\kappa$, for $\mathrm{Ind}_{\mu}^{x,A}$ we normally write $\mathrm{Ind}_{x,A}$.

Examples. Again all variables are supposed to range over total ideals.

$$\begin{aligned} &\mathrm{Ind}_{p,A}\colon A[p := \mathrm{tt}] \to A[p := \mathrm{ff}] \to \forall_{p^{\mathbf{B}}}\,A, \\ &\mathrm{Ind}_{n,A}\colon A[n := 0] \to \forall_n(A \to A[n := \mathrm{S}n]) \to \ \forall_{n^{\mathbf{N}}}\,A, \\ &\mathrm{Ind}_{l,A}\colon A[l := \mathrm{Nil}] \to \forall_{x,l}(A \to A[l := \mathrm{Cons}(x,l)]) \to \forall_{l^{\mathbf{L}(\alpha)}}\,A \\ &\mathrm{Ind}_{x,A}\colon \forall_{y_1}\,A[x := \mathrm{Inl}(y_1)] \to \forall_{y_2}\,A[x := \mathrm{Inr}(y_2)] \to \forall_{x^{\rho_1+\rho_2}}\,A. \end{aligned}$$

Induction over the structure-total ideals is defined similarly. For instance, in the formula above expressing list induction $\mathrm{Ind}_{l,A}$ we can let x range over arbitrary ideals, and l over the structure-total ones.

3.3 Dense and separating sets

We now prove the density theorem, which says that any finitely generated functional (i.e., any $\overline{U}$ with $U \in \mathrm{Con}_\rho$) can be extended to a total functional.

However, we need some assumptions on the base types for this theorem to hold. Otherwise, density might fail for the trivial reason that there are no total ideals at all (e.g., in $\mu\alpha\,(\alpha \to \alpha)$). A type $\mu\alpha_1, \ldots, \alpha_N(\kappa_1, \ldots, \kappa_n)$ is said to *have total ideals* if for every j $(1 \le j \le N)$ there is a constructor type κ_{i_j} of form (1) with $j_1, \ldots, j_n < j$. Then clearly for every j we have a total ideal of type α_j; call it z_j. Moreover, we assume that all base types are finitary. Then their total ideals are finite and maximal, which will be used in the proof.

Theorem 3.2 (Density). *Assume that all base types are finitary and have total ideals. Then for any $U \in \mathrm{Con}_\rho$ we can find an $x \in G_\rho$ such that $U \subseteq x$.*

Proof. Call a type ρ *dense* if $\forall_{U \in \mathrm{Con}_\rho} \exists_{x \in G_\rho} U \subseteq x$, and *separating* if

$$\forall_{U_1, U_2 \in \mathrm{Con}_\rho} \big(U_1 \cup U_2 \notin \mathrm{Con}_\rho \to \exists_{\vec{z} \in G} \mathrm{InCon}(\overline{U_1}(\vec{z}) \cup \overline{U_2}(\vec{z})) \big).$$

Here $\vec{z} \in G$ means that $\vec{z}$ is a sequence of total z_i such that $U_j \vec{z}$ is of a base type μ. We prove by simultaneous induction on ρ that any type ρ is dense and separating. This extended claim is needed for the inductive argument.

For base types μ both claims are easy: the fact that μ is separating is obvious, and density for μ can be inferred from the induction hypothesis, as follows. For simplicity of notation assume that μ is non-simultaneously defined. Let $U \in \mathrm{Con}_\mu$. Then (since μ is finitary) $\exists_b \forall_{a \in U} b \ge a$. In the token b, replace every constructor symbol by its corresponding continuous function, every token at a parameter argument position by a total ideal of its type (which exists by induction hypothesis), and every $*$-token at a type-μ-position by the total ideal z of type μ (which exists by assumption). The result is the required total ideal.

$\rho \Rightarrow \sigma$ is separating: This will follow from the inductive hypotheses that ρ is dense and σ is separating. So let $W, W' \in \mathrm{Con}_{\rho \Rightarrow \sigma}$ such that $W \cup W' \notin \mathrm{Con}_{\rho \Rightarrow \sigma}$. Then there are $(U, a) \in W$ and $(U', a') \in W'$ such that $U \cup U' \in \mathrm{Con}_\rho$ but $\{a, a'\} \notin \mathrm{Con}_\sigma$. Since ρ is dense, we have a $z \in G_\rho$ such that $U \cup U' \subseteq z$. Hence $a \in \overline{W}(z)$ and $a' \in \overline{W'}(z)$. Now since σ is separating there are $\vec{z} \in G$ such that

$$\mathrm{InCon}(\overline{\{a\}}(\vec{z}) \cup \overline{\{a'\}}(\vec{z})),$$

hence also

$$\mathrm{InCon}(\overline{W}(z, \vec{z}) \cup \overline{W'}(z, \vec{z})).$$

This concludes the proof that $\rho \Rightarrow \sigma$ is separating.

$\rho \Rightarrow \sigma$ is dense: This will follow from the inductive hypotheses that ρ is separating and σ is dense. So fix $W = \{\, (U_i, a_i) \mid i \in I \,\} \in \mathrm{Con}_{\rho \Rightarrow \sigma}$.

Consider i, j such that $\{a_i, a_j\} \notin \mathrm{Con}_\sigma$. Then $U_i \cup U_j \notin \mathrm{Con}_\rho$. Since ρ is separating, there are $\vec{z}_{ij} \in G$ and $k_{ij}, l_{ij} \in G_\mu$ such that with $k_{ij} := \overline{U_i}(\vec{z}_{ij})$ and $l_{ij} := \overline{U_j}(\vec{z}_{ij})$

$$\mathrm{InCon}(k_{ij} \cup l_{ij}).$$

We clearly may assume that $\vec{z}_{ij} = \vec{z}_{ji}$ and $(k_{ij}, l_{ij}) = (l_{ji}, k_{ji})$.

Now define for any $U \in \mathrm{Con}_\rho$ a set I_U of indices $i \in I$ such that "U behaves as U_i with respect to the $\vec{z}_{ij}$". More precisely, let

$$I_U := \{\, i \in I \mid \forall_j(\{a_i, a_j\} \notin \mathrm{Con}_\sigma \to \overline{U}(\vec{z}_{ij}) = k_{ij}) \,\}.$$

We first show that

$$\{\, a_i \mid i \in I_U \,\} \in \mathrm{Con}_\sigma. \tag{2}$$

It suffices to show that $\{a_i, a_j\} \in \mathrm{Con}_\sigma$ for all $i, j \in I_U$. So let $i, j \in I_U$ and assume $\{a_i, a_j\} \notin \mathrm{Con}_\sigma$. Then $\overline{U}(\vec{z}_{ij}) = k_{ij}$ as $i \in I_U$ and $\overline{U}(\vec{z}_{ji}) = k_{ji}$ as $j \in I_U$, and because of $\vec{z}_{ij} = \vec{z}_{ji}$ and $\mathrm{InCon}(k_{ij} \cup k_{ji})$ (recall $l_{ij} = k_{ji}$) we could conclude that $\overline{U}(\vec{z}_{ij})$ would be inconsistent. This contradiction proves $\{a_i, a_j\} \in \mathrm{Con}_\sigma$ and hence (2).

Since (2) holds and σ is dense by induction hypothesis, we can find $y_{I_U} \in G_\sigma$ such that $a_i \in y_{I_U}$ for all $i \in I_U$. Define $r \subseteq \mathrm{Con}_\rho \times C_\sigma$ by

$$r(U, a) \iff \begin{cases} a \in y_{I_U}, & \text{if } \overline{U}(\vec{z}_{ij}) \text{ is finite and maximal for all } \vec{z}_{ij}; \\ \exists_{i \in I_U}\, a_i \geq_\sigma a, & \text{otherwise.} \end{cases}$$

We will show that $r \in G_{\rho \Rightarrow \sigma}$ and $W \subseteq r$.

For $W \subseteq r$ we have to show $r(U_i, a_i)$ for all $i \in I$. But this holds, since clearly $i \in I_{U_i}$ and also $a_i \in y_{I_{U_i}}$.

We now show that r is an approximable map, i.e., that $r \in |\boldsymbol{C}_{\rho \Rightarrow \sigma}|$. To prove this we have to verify the defining properties of approximable maps.

(a). $r(U, b_1)$ and $r(U, b_2)$ implies $\{b_1, b_2\} \in \mathrm{Con}_\sigma$. If $\overline{U}(\vec{z}_{ij})$ is finite and maximal for all $\vec{z}_{ij}$, the claim follows from the consistency of y_{I_U}. If not, the claim follows from Lemma 2.3.

(b). $r(U, b)$, $V \geq_A U$ and $b \geq_B c$ implies $r(V, c)$. First assume that $\overline{U}(\vec{z}_{ij})$ is finite and maximal for all $\vec{z}_{ij}$. Then also $\overline{V}(\vec{z}_{ij})$ is maximal for all $\vec{z}_{ij}$. From $r(U, b)$ we get $b \in y_{I_U}$. We have to show that $c \in y_{I_V}$. But since $\overline{U}(\vec{z}_{ij})$ and $\overline{V}(\vec{z}_{ij})$ are maximal for all $\vec{z}_{ij}$ and $V \geq_\rho U$, they must have the same values on the $\vec{z}_{ij}$, hence $I_U = I_V$, so $y_{I_U} = y_{I_V}$ and therefore $c \in y_{I_V}$ by deductive closure. Now assume the contrary. From $r(U, b)$ we get $a_i \geq_\sigma b$ for some $i \in I_U$. From $V \geq_\rho U$ we can conclude $I_U \subseteq I_V$, by the definition of I_U. Hence $i \in I_V$, and also $b \in y_{I_V}$ (since $a_i \in y_{I_U}$ for all $i \in I_U$, and y_{I_V} is deductively closed). Therefore $r(V, b)$ and hence $r(V, c)$.

This concludes the proof that r is an approximable map. It remains to prove $r \in G_{\rho \Rightarrow \sigma}$. So let $x \in G_\rho$. We must show

$$|r|(x) = \{\, a \in C_\sigma \mid \exists_{U \subseteq x}\, r(U, a) \,\} \in G_\sigma.$$

Now $x(\vec{z}_{ij})$ is total for all i, j, hence by our assumption on base types finite and maximal. So there is some $U_{ij} \subseteq x$ such that $\overline{U_{ij}}(\vec{z}_{ij}) = x(\vec{z}_{ij})$. Let $U \subseteq x$ be the union of all the U_{ij}. Then by definition $r(U, a)$ for all $a \in y_{I_U}$. Therefore $y_{I_U} \subseteq |r|(x)$ and hence $|r|(x) \in G_\sigma$. □

4 Terms; denotational and operational semantics

For every type ρ, we have defined what a partial continuous functional of type ρ 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" [20, 23, 14]. However – in contrast to [23] – 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 [25], 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 ρ will denote a partial continuous functional of this type, that is, a consistent and deductively closed set of tokens of type ρ. We will define this set inductively.

It will turn out that these sets are recursively enumerable. In this sense each closed term M of type ρ denotes a computable partial continuous functional of type ρ. 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 [11, 8]. Our semantics will cover these as well.

There are some natural questions one can have for such a term language:

1. Preservation of values under conversion (as in [23, First Theorem]). Here we need to include applications of computation rules.

2. An adequacy theorem (as in [25, Theorem 3.1] or [23, Second Theorem]), which in our setting says that whenever a closed term has a proper token in the ideal it denotes, then it evaluates to a constructor term entailing this token.

3. Strong normalization, that is, termination of arbitrary conversions, provided the only defined constants are the structural recursion operators.

Properties 1, 2 and 3 will be proved in the present section, in Section 5 and in Section 6, respectively.

Coquand [14] observed that the types play only a somewhat minor role in this setup. It suffices to know the *arity* (a natural number) of the constants (constructors and defined constants), to guide the definitions. An

interesting consequence is that one can use this approach for dependently typed languages as well, for instance, the terms of Martin-Löf's type theory.

4.1 Terms

Terms are built from (typed) variables and (typed) constants (constructors C or defined constants D, see below) by (type-correct) application and abstraction:

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

Every defined constant comes with a *system of computation rules*, consisting of finitely many equations $D\vec{P}_i = M_i$ $(i = 1, \ldots, n)$ with constructor patterns $\vec{P}_i$, such that $\vec{P}_i$ and $\vec{P}_j$ $(i \neq j)$ are non-unifiable. *Constructor patterns* are lists of applicative terms with distinct variables, defined inductively as follows (we write $\vec{P}(\vec{x}\,)$ to indicate all variables in $\vec{P}$; notice that x can be a variable for a formal neighborhood, and that all expressions must be type-correct):

- $x(x)$ is a constructor pattern.
- If C is a constructor and $\vec{P}(\vec{x}\,)$ a constructor pattern, then $(C\vec{P})(\vec{x}\,)$ is a constructor pattern.
- 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.

4.2 Ideals as meaning 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. However, since arbitrary terms are allowed on the right, we need 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 $(\vec{U}, b) \in [\![\lambda\vec{x}\, M]\!]$.

We use the following notation. $(\vec{U}, b)$ means $(U_1, \ldots (U_n, b) \ldots)$, and at argument positions of constructors we use b for tokens as well as for formal neighborhoods. $(\vec{U}, V) \subseteq [\![\lambda\vec{x}\, M]\!]$ means $(\vec{U}, b) \in [\![\lambda\vec{x}\, M]\!]$, for all (finitely many) $b \in V$.

Definition 4.1 (Inductive, of $(\vec{U}, b) \in [\![\lambda\vec{x}\, M]\!]$).

$$\frac{U_i \geq b}{(\vec{U}, b) \in [\![\lambda\vec{x}\, x_i]\!]}(V), \qquad \frac{(\vec{U}, V) \subseteq [\![\lambda\vec{x}\, N]\!] \qquad (\vec{U}, V, c) \in [\![\lambda\vec{x}\, M]\!]}{(\vec{U}, c) \in [\![\lambda\vec{x}.MN]\!]}(A).$$

For every constructor C we have

$$\frac{}{(\vec{U}, \vec{V}, *) \in [\![\lambda\vec{x}\, C]\!]}(C_*), \qquad \frac{\vec{V} \geq \vec{b}}{(\vec{U}, \vec{V}, C\vec{b}\,) \in [\![\lambda\vec{x}\, C]\!]}(C).$$

For every defined constant D we have

$$\frac{}{(\vec{U},\vec{V},*)\in[\![\lambda\vec{x}\,D]\!]}(D_*),\qquad \frac{(\vec{U},\vec{V},b)\in[\![\lambda\vec{x},\vec{y}\,M]\!]}{(\vec{U},\vec{P}(\vec{V}),b)\in[\![\lambda\vec{x}\,D]\!]}(D),$$

with one such rule (D) for every computation rule $D\vec{P}(\vec{y}\,)=M$.

Here are some simple consequences of this definition. First we show a useful property of constructors:

Lemma 4.2. *For $b\neq *$, $(\vec{U},b)\in[\![\lambda\vec{x}.C\vec{N}]\!]$ iff there are $\vec{c}\geq\vec{b}$ such that $b=C\vec{b}$ and $(\vec{U},c_i)\in[\![\lambda\vec{x}\,N_i]\!]$ $(i=1,\ldots,n)$.*

Proof. Assume $(\vec{U},c_i)\in[\![\lambda\vec{x}\,N_i]\!]$ $(c_i\geq b_i,\ i=1,\ldots,n)$. For $j=0,\ldots,n$ we show $(\vec{U},\{c_{j+1}\},\ldots,\{c_n\},C\vec{b})\in[\![\lambda\vec{x}.CN_1\ldots N_j]\!]$. In case $j=0$ use (C):

$$\frac{\{c_1\}\geq b_1\quad\ldots\quad\{c_n\}\geq b_n}{(\vec{U},\{c_1\},\ldots,\{c_n\},C\vec{b}\,)\in[\![\lambda\vec{x}\,C]\!]}.$$

In the step from $j-1$ to j use (A):

$$\frac{(\vec{U},c_j)\in[\![\lambda\vec{x}\,N_j]\!]\qquad(\vec{U},\{c_j\},\ldots,\{c_n\},C\vec{b}\,)\in[\![\lambda\vec{x}.CN_1\ldots N_{j-1}]\!]}{(\vec{U},\{c_{j+1}\},\ldots,\{c_n\},C\vec{b}\,)\in[\![\lambda\vec{x}.CN_1\ldots N_j]\!]}.$$

For $j=n$ the claim follows. – For the other direction, observe that only (A) could have been applied. Hence the argument can be read backwards. □

Using the fact that the left hand sides of computation rules are non-unifiable we can prove:

Lemma 4.3. *$[\![\lambda\vec{x}\,M]\!]$ is an ideal, i.e., consistent and deductively closed.*

Proof. Induction on $(\vec{U},b)\in[\![\lambda\vec{x}\,M]\!]$.

(1) Consistency. **Case** (V). Assume $(\vec{U}_1,b_1),(\vec{U}_2,b_2)\in[\![\lambda\vec{x}\,x_i]\!]$, and that $\vec{U}_1$ and $\vec{U}_2$ are pairwise consistent. We must show $\{b_1,b_2\}\in\mathrm{Con}$. By (V), $U_{1i}\geq b_1$ and $U_{2i}\geq b_2$. Now $\{b_1,b_2\}\in\mathrm{Con}$ follows from $U_{1i}\cup U_{2i}\in\mathrm{Con}$.

Case (C). Assume $(\vec{U}_1,\vec{V}_1,C\vec{b}_1),(\vec{U}_2,\vec{V}_2,C\vec{b}_2)\in[\![\lambda\vec{x}\,C]\!]$, and that $\vec{U}_1,\vec{V}_1$ and $\vec{U}_2,\vec{V}_2$ are pairwise consistent. We must show $\{C\vec{b}_1,C\vec{b}_2\}\in\mathrm{Con}$. By (C), $\vec{V}_i\geq\vec{b}_i$ $(i=1,2)$. From the pairwise consistency of $\vec{V}_1$ and $\vec{V}_2$ we obtain the pairwise consistency of $\vec{b}_1$ and $\vec{b}_2$. Hence $\{C\vec{b}_1,C\vec{b}_2\}\in\mathrm{Con}$.

Case (A). Let $(\vec{U}_1,c_1),(\vec{U}_2,c_2)\in[\![\lambda\vec{x}.MN]\!]$, with $\vec{U}_1$ and $\vec{U}_2$ pairwise consistent. We show $\{c_1,c_2\}\in\mathrm{Con}$. By (A), $(\vec{U}_1,V_1),(\vec{U}_2,V_2)\subseteq[\![\lambda\vec{x}\,N]\!]$, so by IH $V_1\cup V_2\in\mathrm{Con}$. Similarly, again by (A), $(\vec{U}_1,V_1,c_1),(\vec{U}_2,V_2,c_2)\in[\![\lambda\vec{x}\,M]\!]$, hence $\{c_1,c_2\}\in\mathrm{Con}$ by IH.

Case (D). Let $(\vec{U}_i,\vec{P}_i(\vec{V}_i),b_i)\in[\![\lambda\vec{x}\,D]\!]$ $(i=1,2)$, and assume that $\vec{U}_1,\vec{P}_1(\vec{V}_1)$ and $\vec{U}_2,\vec{P}_2(\vec{V}_2)$ are pairwise consistent. From the fact that the

left hand sides of computation rules are non-unifiable we can infer $\vec{P}_1 = \vec{P}_2$, and that $\vec{V}_1$ and $\vec{V}_2$ are pairwise consistent. Then $\{b_1, b_2\} \in \mathrm{Con}$ by IH.

(2) Closure under $\geq$. **Case** (V). Assume $\vec{V} \geq \vec{U}$ and $b \geq c$. We must show $(\vec{V}, c) \in [\![\lambda \vec{x}\, x_i]\!]$. By (V) it suffices to show $V_i \geq c$. But this follows from $V_i \geq U_i \geq b \geq c$.

Case (C). Assume $\vec{U}_1 \geq \vec{U}$, $\vec{V}_1 \geq \vec{V}$ and $C\vec{b} \geq C\vec{c}$. We must show $(\vec{U}_1, \vec{V}_1, C\vec{c}) \in [\![\lambda \vec{x}\, C]\!]$. By (C) it suffices to show $\vec{V}_1 \geq \vec{c}$. But this follows from $\vec{V}_1 \geq \vec{V} \geq \vec{b} \geq \vec{c}$.

Case (A). The IH clearly suffices here.

Case (D). Assume $\vec{U}_1 \geq \vec{U}$, $\vec{Z} \geq \vec{P}(\vec{V})$ and $b \geq b_1$. Notice that $\vec{Z} \geq \vec{P}(\vec{V})$ implies $\vec{Z} = \vec{P}(\vec{V}_1)$ with $\vec{V}_1 \geq \vec{V}$, so we must show $(\vec{U}_1, \vec{P}(\vec{V}_1), b_1) \in [\![\lambda \vec{x}\, D]\!]$. By IH we have $(\vec{U}_1, \vec{V}_1, b_1) \in [\![\lambda \vec{x}, \vec{y}\, M]\!]$. Now use (D). □

4.3 Preservation of values

We now prove that our definition above of the meaning of a term is reasonable in the sense that an application of the standard (β- and η-) conversions and also of a computation rule does not change the meaning of a term. For the β-conversion part of this proof it is helpful to first introduce a more standard notation, which involves variable environments.

Definition 4.4. Assume that all free variables in M are among $\vec{x}$.

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

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

Lemma 4.5. (a) *If* $\vec{V} \geq \vec{U}$, $b \geq c$ *and* $b \in [\![M]\!]^{\vec{U}}_{\vec{x}}$, *then* $c \in [\![M]\!]^{\vec{V}}_{\vec{x}}$.

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

Proof. (a). By the deductive closure of $[\![\lambda \vec{x}\, M]\!]$, $\vec{V} \geq \vec{U}$, $b \geq c$ and $(\vec{U}, b) \in [\![\lambda \vec{x}\, M]\!]$ together imply $(\vec{V}, c) \in [\![\lambda \vec{x}\, M]\!]$. (b) follows from (a). □

Lemma 4.6. (a) $[\![x_i]\!]^{\vec{u}}_{\vec{x}} = u_i$.

(b) $[\![\lambda y\, M]\!]^{\vec{u}}_{\vec{x}} = \{\, (V, b) \mid b \in [\![M]\!]^{\vec{u}, V}_{\vec{x}, y} \,\}$.

(c) $[\![MN]\!]^{\vec{u}}_{\vec{x}} = [\![M]\!]^{\vec{u}}_{\vec{x}} [\![N]\!]^{\vec{u}}_{\vec{x}}$.

Proof. (b). It suffices to prove this with $\vec{U}$ for $\vec{u}$. But $(V, b) \in [\![\lambda y\, M]\!]^{\vec{U}}_{\vec{x}}$ and $b \in [\![M]\!]^{\vec{U}, V}_{\vec{x}, y}$ are both equivalent to $(\vec{U}, V, b) \in [\![\lambda \vec{x}, y\, M]\!]$.

(c).

$$
\begin{aligned}
& c \in [\![M]\!]^{\vec{u}}_{\vec{x}} [\![N]\!]^{\vec{u}}_{\vec{x}} \\
& \leftrightarrow \exists_{V \subseteq [\![N]\!]^{\vec{u}}_{\vec{x}}} (V, c) \in [\![M]\!]^{\vec{u}}_{\vec{x}} \quad \text{(application in acis's)} \\
& \leftrightarrow \exists_{V \subseteq [\![N]\!]^{\vec{u}}_{\vec{x}}} \exists_{\vec{U} \subseteq \vec{u}} (V, c) \in [\![M]\!]^{\vec{U}}_{\vec{x}} \\
& \leftrightarrow \exists_{\vec{U}_1 \subseteq \vec{u}} \exists_{V \subseteq [\![N]\!]^{\vec{U}_1}_{\vec{x}}} \exists_{\vec{U} \subseteq \vec{u}} (V, c) \in [\![M]\!]^{\vec{U}}_{\vec{x}} \\
& \leftrightarrow^{(*)} \exists_{\vec{U} \subseteq \vec{u}} \exists_{V \subseteq [\![N]\!]^{\vec{U}}_{\vec{x}}} (V, c) \in [\![M]\!]^{\vec{U}}_{\vec{x}} \\
& \leftrightarrow \exists_{\vec{U} \subseteq \vec{u}} \exists_V . (\vec{U}, V) \subseteq [\![\lambda \vec{x}\, N]\!] \wedge (\vec{U}, V, c) \in [\![\lambda \vec{x}\, M]\!] \\
& \leftrightarrow \exists_{\vec{U} \subseteq \vec{u}} (\vec{U}, c) \in [\![\lambda \vec{x}. MN]\!] \quad \text{(by } (A)\text{)} \\
& \leftrightarrow \exists_{\vec{U} \subseteq \vec{u}} c \in [\![MN]\!]^{\vec{U}}_{\vec{x}} \\
& \leftrightarrow c \in [\![MN]\!]^{\vec{u}}_{\vec{x}}.
\end{aligned}
$$

Here is the proof of the equivalence marked $(*)$. The upwards direction is obvious. For the downwards direction we use monotonicity. Assume $\vec{U}_1 \subseteq \vec{u}$, $V \subseteq [\![N]\!]^{\vec{U}_1}_{\vec{x}}$, $\vec{U} \subseteq \vec{u}$ and $(V, c) \in [\![M]\!]^{\vec{U}}_{\vec{x}}$. Let $\vec{U}_2 := \vec{U}_1 \cup \vec{U} \subseteq \vec{u}$. Then by monotonicity $V \subseteq [\![N]\!]^{\vec{U}_2}_{\vec{x}}$ and $(V, c) \in [\![M]\!]^{\vec{U}_2}_{\vec{x}}$. □

Corollary 4.7. $[\![\lambda y\, M]\!]^{\vec{u}}_{\vec{x}} v = [\![M]\!]^{\vec{u}, v}_{\vec{x}, y}$.

Proof.

$$
\begin{aligned}
b \in [\![\lambda y\, M]\!]^{\vec{u}}_{\vec{x}} v & \leftrightarrow \exists_{V \subseteq v} (V, b) \in [\![\lambda y\, M]\!]^{\vec{u}}_{\vec{x}} \quad \text{(application in acis's)} \\
& \leftrightarrow \exists_{V \subseteq v} b \in [\![M]\!]^{\vec{u}, V}_{\vec{x}, y} \quad \text{(by the lemma)} \\
& \leftrightarrow b \in [\![M]\!]^{\vec{u}, v}_{\vec{x}, y}.
\end{aligned}
$$

□

Lemma 4.8 (Substitution). $[\![M]\!]^{\vec{u}, [\![N]\!]^{\vec{u}}_{\vec{x}}}_{\vec{x}, z} = [\![M[z := N]]\!]^{\vec{u}}_{\vec{x}}$.

Proof. **Case** $\lambda y\, M$. For readability we leave out $\vec{x}$ and $\vec{u}$.

$$
\begin{aligned}
[\![\lambda y\, M]\!]^{[\![N]\!]}_{z} &= \{\, (V, b) \mid b \in [\![M]\!]^{[\![N]\!], V}_{z, y} \,\} \\
&= \{\, (V, b) \mid b \in [\![M[z := N]]\!]^{V}_{y} \,\} \quad \text{(by IH)} \\
&= [\![\lambda y. M[z := N]]\!] \quad \text{(by the lemma)} \\
&= [\![(\lambda y\, M)[z := N]]\!].
\end{aligned}
$$

The other cases are easy. □

Lemma 4.9. $[\![(\lambda y\, M) N]\!]^{\vec{u}}_{\vec{x}} = [\![M[y := N]]\!]^{\vec{u}}_{\vec{x}}$.

Proof. For readability we leave out $\vec{x}$ and $\vec{u}$. By the last two lemmas and the corollary, $[\![(\lambda y\, M)N]\!] = [\![\lambda y\, M]\!][\![N]\!] = [\![M]\!]_y^{[\![N]\!]} = [\![M[y := N]]\!]$. □

Lemma 4.10. $[\![\lambda y.My]\!]_{\vec{x}}^{\vec{u}} = [\![M]\!]_{\vec{x}}^{\vec{u}}$, *if* $y \notin \mathrm{FV}(M)$.

Proof. For readability we leave out $\vec{x}$ and $\vec{u}$.

$$\begin{aligned}(V, b) \in [\![\lambda y.My]\!] &\leftrightarrow b \in [\![My]\!]_y^V \\ &\leftrightarrow b \in [\![M]\!]\overline{V} \\ &\leftrightarrow \exists_{U \subseteq \overline{V}}\, (U, b) \in [\![M]\!] \quad \text{(application in acis's)} \\ &\leftrightarrow (V, b) \in [\![M]\!],\end{aligned}$$

where in the last step we have used monotonicity. □

To prove preservation of values under computation rules, the following observation will be needed: (it removes the need for "(generalized) predecessor functions" [11, 8]):

Lemma 4.11. $(\vec{U}, \vec{V}, b) \in [\![\lambda\vec{x}, \vec{y}.M[z := \mathrm{C}\vec{y}]]\!]$ *iff* $(\vec{U}, \mathrm{C}\vec{V}, b) \in [\![\lambda\vec{x}, z\, M]\!]$.

Proof. Induction on $(\vec{U}, \vec{V}, b) \in [\![\lambda\vec{x}, \vec{y}.M[z := \mathrm{C}\vec{y}]]\!]$, and cases on the form of M. **Case** MN.

$$\begin{aligned}&(\vec{U}, \vec{V}, c) \in [\![\lambda\vec{x}, \vec{y}.M[z := \mathrm{C}\vec{y}]N[z := \mathrm{C}\vec{y}]]\!] \\ &\leftrightarrow \exists_Z.(\vec{U}, \vec{V}, Z) \subseteq [\![\lambda\vec{x}, \vec{y}\, N[z := \mathrm{C}\vec{y}]]\!] \wedge (\vec{U}, \vec{V}, Z, c) \in [\![\lambda\vec{x}, \vec{y}\, M[z := \mathrm{C}\vec{y}]]\!] \\ &\leftrightarrow \exists_Z.(\vec{U}, \mathrm{C}\vec{V}, Z) \subseteq [\![\lambda\vec{x}, z\, N]\!] \wedge (\vec{U}, \mathrm{C}\vec{V}, Z, c) \in [\![\lambda\vec{x}, z\, M]\!] \quad \text{(by IH)} \\ &\leftrightarrow (\vec{U}, \mathrm{C}\vec{V}, c) \in [\![\lambda\vec{x}, z.MN]\!] \quad \text{(by } (A)\text{)}.\end{aligned}$$

Case z.

$$\begin{aligned}(\vec{U}, \vec{V}, c) \in [\![\lambda\vec{x}, \vec{y}.\mathrm{C}\vec{y}]\!] = [\![\lambda\vec{x}\ \mathrm{C}]\!] &\leftrightarrow \exists_{\vec{b}}.\vec{V} \geq \vec{b} \wedge \mathrm{C}\vec{b} \geq c, \\ &\leftrightarrow \mathrm{C}\vec{V} \geq c, \\ &\leftrightarrow (\vec{U}, \mathrm{C}\vec{V}, c) \in [\![\lambda\vec{x}, z\, z]\!].\end{aligned}$$

In all other cases both sides are clearly equivalent. □

We can now prove preservation of values under computation rules:

Lemma 4.12. *For every computation rule* $D\vec{P}(\vec{y}) = M$ *of a defined constant* D, $[\![\lambda\vec{y}.D\vec{P}(\vec{y})]\!] = [\![\lambda\vec{y}\, M]\!]$.

Proof. The following are equivalent:

$$\begin{aligned}&(\vec{V}, b) \in [\![\lambda\vec{y}.D\vec{P}(\vec{y})]\!] \\ &(\vec{P}(\vec{V}), b) \in [\![D]\!] = [\![\lambda\vec{z}.D\vec{z}]\!] \quad \text{(by Lemma 4.11)} \\ &(\vec{V}, b) \in [\![\lambda\vec{y}\, M]\!],\end{aligned}$$

where the last step is by definition. □

4.4 Examples

We consider the doubling function $D\colon \mathbf{N} \Rightarrow \mathbf{N}$, addition $+\colon \mathbf{N} \Rightarrow \mathbf{N} \Rightarrow \mathbf{N}$ and the fixed point operators $\mathcal{Y}_\rho$. Structural recursion could be treated as well.

Doubling $D\colon \mathbf{N} \Rightarrow \mathbf{N}$ is defined by the computation rules

$$D0 = 0, \qquad D(Sn) = S(S(Dn)).$$

We prove

$$(\{S^n 0\}, S^{2n}0) \in [\![D]\!] \text{ and } (\{S^n a\}, S^{2n}*) \in [\![D]\!] \text{ for all } a,$$

by induction on n: For the first claim, in the base case we have $0 \in [\![0]\!]$ by (C), hence $(\{0\}, 0) \in [\![D]\!]$ by (D). In the step case, from $(\{S^n 0\}, S^{2n}0) \in [\![D]\!]$ we obtain $(\{S^n 0\}, S^2(S^{2n}0)) \in [\![\lambda y.S^2(Dy)]\!]$ by Lemma 4.11, and hence $(\{S(S^n 0)\}, S^2(S^{2n}0)) \in [\![D]\!]$ by (D). For the second claim, in the base case use $(\{a\}, *) \in [\![D]\!]$ (which holds generally), and in the step case the argument is the same.

Conversely, assume $(U, b) \in [\![D]\!]$. Then either $b = *$ and U is arbitrary, or $b = 0$ and $0 \in U$, or else $(U, b) \in [\![D]\!]$ is a conclusion of (D), hence of the form $(SV, b) \in [\![D]\!]$ with $(V, b) \in [\![\lambda y.S^2(Dy)]\!]$. But then $b = S^2 c$ and $(V, c) \in [\![D]\!]$. Argueing by induction on the generation of $[\![D]\!]$, we conclude that either $b = S^{2n}0$ and $S^n 0 \in U$, or $b = S^{2n}*$ and $S^n a \in U$ for some a.

Addition $+\colon \mathbf{N} \Rightarrow \mathbf{N} \Rightarrow \mathbf{N}$ is defined by the computation rules

$$n + 0 = n, \qquad n + Sm = S(n + m).$$

As above one shows that $(U, b) \in [\![\lambda m.0 + m]\!]$ iff either $b = *$ and U is arbitrary, or $b = 0$ and $0 \in U$, or $b = S^n 0$ and $S^n 0 \in U$, or $b = S^n *$ and $S^n a \in U$ for some a. So we can conclude that $[\![\lambda m.0 + m]\!] = [\![\lambda m\, m]\!]$. This is of interest, because it allows us to replace $0 + M$ by M for an *arbitrary* (not necessarily total) term M without affecting the values.

Fixed points The computation rule $\mathcal{Y}_\rho f = f(\mathcal{Y}_\rho f)$ defines the fixed point operator $\mathcal{Y}_\rho$ of type $(\rho \Rightarrow \rho) \Rightarrow \rho$.

5 Adequacy

The adequacy theorem of Plotkin [25, Theorem 3.1] says that whenever the value of a closed term M is a numeral, then M head-reduces to this numeral. So in this sense the (denotational) semantics is (computationally) "adequate". Plotkin's proof is by induction on the types, and uses a computability predicate. We prove an adequacy theorem in our setting, for arbitrary computation rules.

5.1 Operational semantics

Recall that a token of a base type μ is either $*$ or a constructor expression (possibly involving $*$) whose outermost constructor is for μ. We use B to denote both, constructors C and defined constants D.

Definition 5.1 ($M \succ_1 N$, M head-reduces to N).

$$(\lambda x\, M)N \succ_1 M[x := N], \qquad \frac{M \succ_1 M'}{MN \succ_1 M'N},$$
$$D\vec{P}(\vec{N}\,) \succ_1 M[\vec{y} := \vec{N}] \quad \text{for } D\vec{P}(\vec{y}) = M \text{ a computation rule},$$
$$\frac{M \succ_1 M'}{Ba_1 \dots a_n M \succ_1 Ba_1 \dots a_n M'} \quad \text{for } n < \mathrm{ar}(B).$$

$\succeq$ denotes the reflexive transitive closure of $\succ_1$.

Clearly for every term M there is at most one M' such that $M \succ_1 M'$; call M *normal* if there is no such M'.

We define an "operational interpretation" [23] of formal neighborhoods U. To this end we define a notion $M \in [a]$, for M closed, by induction on the type of the token a, and write $M \in [U]$ for $\forall_{a \in U}\, M \in [a]$.

Definition 5.2. (a) For a of base type μ, $M \in [a]$ iff $\exists_{b \geq a}\, M \succeq b$.

(b) $M \in [(U, b)]$ iff $M \succeq \lambda x\, M'$ or $M \succeq B\vec{M}$ with length of $\vec{M}$ less than $\mathrm{ar}(B)$, and $\forall_{N \in [U]}\, MN \in [b]$.

Lemma 5.3. *If $M \succeq N$, $N \in [V]$ and $V \geq U$, then $M \in [U]$.*

Proof. Induction on $N \in [b]$. □

Theorem 5.4 (Adequacy). *If $(\vec{U}, b) \in [\![\lambda\vec{x}\, M]\!]$ with b a proper token, then $\lambda\vec{x}\, M \in [(\vec{U}', b')]$ for some $(\vec{U}', b') \geq (\vec{U}, b)$.*

Proof. By induction on the rules defining $(\vec{U}, b) \in [\![\lambda\vec{x}\, M]\!]$, and cases on the form of M.

Case x_i.

$$\frac{U_i \geq b}{(\vec{U}, b) \in [\![\lambda\vec{x}\, x_i]\!]}(V).$$

We need $(\vec{U}', b') \geq (\vec{U}, b)$ such that $\lambda\vec{x}\, x_i \in [(\vec{U}', b')]$, i.e., $\forall_{\vec{K} \in [\vec{U}']}\, K_i \in [b']$. Take $\vec{U}' = \vec{U}$, $b' = b$. Let $K_i \in [U_i]$. Then by definition $K_i \in [b']$.

Case MN.

$$\frac{(\vec{U}, V) \subseteq [\![\lambda\vec{x}\, N]\!] \qquad (\vec{U}, V, c) \in [\![\lambda\vec{x}\, M]\!]}{(\vec{U}, c) \in [\![\lambda\vec{x}.MN]\!]}(A).$$

We need to find some $(\vec{U}', c') \geq (\vec{U}, c)$ such that $\lambda\vec{x}.MN \in [(\vec{U}', c')]$, i.e., $\forall_{\vec{K} \in [\vec{U}']}\, (MN)[\vec{x} := \vec{K}] \in [c']$.

By IH, for all $b \in V$ we have some $(\vec{U}_1, b') \geq (\vec{U}, b)$ such that $\lambda\vec{x}\, N \in [(\vec{U}_1, b')]$, i.e., $\forall_{\vec{K}\in[\vec{U}_1]} N[\vec{x} := \vec{K}] \in [b']$. Recall that $(\vec{U}_1, b') \geq (\vec{U}, b)$ means $\vec{U} \geq \vec{U}_1$ and $b' \geq b$. Hence we can pick the same U_1 for all $b \in V$, and

$$\forall_{\vec{K}\in[\vec{U}_1]} N[\vec{x} := \vec{K}] \in [V].$$

Also, by IH we have $(\vec{U}_2, V', c') \geq (\vec{U}, V, c)$ such that $\lambda\vec{x}\, M \in [(\vec{U}_2, V', c')]$, i.e.,

$$\forall_{\vec{K}\in[\vec{U}_2]} M[\vec{x} := \vec{K}] \in [(V', c')].$$

Recall that $(\vec{U}_2, V', c') \geq (\vec{U}, V, c)$ means $\vec{U} \geq \vec{U}_2$, $V \geq V'$ and $c' \geq c$.

Let $\vec{U}' := \vec{U}_1 \cup \vec{U}_2$ (component wise union), and fix $\vec{K} \in \vec{U}'$. Clearly $\vec{K} \in [\vec{U}_1]$ and $\vec{K} \in [\vec{U}_2]$. From $M[\vec{x} := \vec{K}] \in (V', c')$ we know that $M[\vec{x} := \vec{K}] \succeq \lambda x\, M'$ or $M \succeq B\vec{M}$ with length of $\vec{M}$ less than $\mathrm{ar}(B)$, and also $\forall_{L\in[V']}.M[\vec{x} := \vec{K}]L \in [c']$.

Since $N[\vec{x} := \vec{K}] \in [V]$ and hence $\in [V']$ we obtain $(MN)[\vec{x} := \vec{K}] \in [c']$, as required.

Case D.

$$\frac{(\vec{U}, \vec{V}, b) \in [\![\lambda\vec{x}, \vec{y}\, M]\!]}{(\vec{U}, \vec{P}(\vec{V}), b) \in [\![\lambda\vec{x}\, D]\!]}(D),$$

with $D\vec{P}(\vec{y}) = M$ be a computation rule. We need $(\vec{U}', \vec{Z}, b') \geq (\vec{U}, \vec{P}(\vec{V}), b)$ such that $\lambda\vec{x}\, D \in [(\vec{U}', \vec{Z}, b')]$. Recall that $(\vec{U}', \vec{Z}, b') \geq (\vec{U}, \vec{P}(\vec{V}), b)$ means $\vec{U} \geq \vec{U}'$, $\vec{P}(\vec{V}) \geq \vec{Z}$ and $b' \geq b$.

By IH we have $(\vec{U}', \vec{V}', b') \geq (\vec{U}, \vec{V}, b)$ such that $\lambda\vec{x}, \vec{y}\, M \in [(\vec{U}', \vec{V}', b')]$, i.e.,

$$\forall_{\vec{K}\in[\vec{U}']}\forall_{\vec{N}\in[\vec{V}']} M[\vec{y} := \vec{N}] \in [b'].$$

Recall that $(\vec{U}', \vec{V}', b') \geq (\vec{U}, \vec{V}, b)$ means $\vec{U} \geq \vec{U}'$, $\vec{V} \geq \vec{V}'$ and $b' \geq b$.

Pick the required $\vec{U}', b'$ as the ones provided by the IH, and $\vec{Z} := \vec{P}(\vec{V}')$. We must show $\lambda\vec{x}\, D \in [(\vec{U}', \vec{P}(\vec{V}'), b')]$, i.e.,

$$\forall_{\vec{K}\in[\vec{U}']}\forall_{\vec{L}\in[\vec{P}(\vec{V}')]} D\vec{L} \in [b'].$$

Now fix $\vec{K} \in [\vec{U}']$ and $\vec{L} \in [\vec{P}(\vec{V}')]$. Then $\vec{L} = \vec{P}(\vec{N})$ with $\vec{N} \in [\vec{V}']$. From $D\vec{P}(\vec{N}) \succ_1 M[\vec{y} := \vec{N}]$ and $M[\vec{y} := \vec{N}] \in [b']$ the claim follows.

Case C.

$$\frac{\vec{V} \geq \vec{b}}{(\vec{U}, \vec{V}, C\vec{b}\,) \in [\![\lambda\vec{x}\, C]\!]}(C).$$

We need $(\vec{U}', \vec{V}', b') \geq (\vec{U}, \vec{V}, C\vec{b}\,)$ such that $\lambda\vec{x}\, C \in [(\vec{U}', \vec{V}', b')]$. Recall that $(\vec{U}', \vec{V}', b) \geq (\vec{U}, \vec{V}, C\vec{b}\,)$ means $\vec{U} \geq \vec{U}'$, $\vec{V} \geq \vec{V}'$ and $b' \geq C\vec{b}$.

Pick $\vec{U}' := \vec{U}$, $\vec{V}' := \vec{V}$ and $b' := C\vec{b}$. We must show $\lambda\vec{x}\, C \in [(\vec{U}, \vec{V}, C\vec{b}\,)]$, i.e.,

$$\forall_{\vec{K}\in[\vec{U}]}\forall_{\vec{L}\in[\vec{V}]} C\vec{L} \in [C\vec{b}\,].$$

This follows from $\vec{V} \geq \vec{b}$. □

6 Strong normalization for structural recursion

It is well-known that in a system of simultaneously defined free algebras every term (possibly involving recursion operators) is strongly normalizing. However, the standard proof reduces the problem to strong normalization of second order propositional logic (called system F by Girard [17]). This latter result requires a method not formalizable in analysis. Here we give a much simpler proof, which only uses predicative methods.

6.1 Structural recursion

The inductive structure of the types $\vec{\mu} = \mu\vec{\alpha}\,\vec{\kappa}$ corresponds to two sorts of constants. With the *constructors* $\mathrm{C}_i^{\vec{\mu}} \colon \kappa_i[\vec{\mu}]$ we can construct elements of a type μ_j, and with the *recursion operators* $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$ we can construct total functionals from μ_j to τ_j by recursion on the structure of $\vec{\mu}$. In order to define the type of the recursion operators w.r.t. $\vec{\mu} = \mu\vec{\alpha}\,\vec{\kappa}$ and result types $\vec{\tau}$, we first define for

$$\kappa_i = \vec{\rho} \Rightarrow (\vec{\sigma}_1 \Rightarrow \alpha_{j_1}) \Rightarrow \ldots \Rightarrow (\vec{\sigma}_n \Rightarrow \alpha_{j_n}) \Rightarrow \alpha_j \in \mathrm{KT}(\vec{\alpha})$$

the *step type*

$$\begin{aligned}\delta_i^{\vec{\mu},\vec{\tau}} := \vec{\rho} \Rightarrow (\vec{\sigma}_1 \Rightarrow \mu_{j_1}) \Rightarrow \ldots \Rightarrow (\vec{\sigma}_n \Rightarrow \mu_{j_n}) \Rightarrow \\ (\vec{\sigma}_1 \Rightarrow \tau_{j_1}) \Rightarrow \ldots \Rightarrow (\vec{\sigma}_n \Rightarrow \tau_{j_n}) \Rightarrow \tau_j.\end{aligned}$$

Here $\vec{\rho}, (\vec{\sigma}_1 \Rightarrow \mu_{j_1}), \ldots, (\vec{\sigma}_n \Rightarrow \mu_{j_n})$ correspond to the *components* of the object of type μ_j under consideration, and $(\vec{\sigma}_1 \Rightarrow \tau_{j_1}), \ldots, (\vec{\sigma}_n \Rightarrow \tau_{j_n})$ to the previously defined values. The recursion operator $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$ has type

$$\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}} \colon \delta_1^{\vec{\mu},\vec{\tau}} \Rightarrow \ldots \Rightarrow \delta_k^{\vec{\mu},\vec{\tau}} \Rightarrow \mu_j \Rightarrow \tau_j$$

(recall that k is the total number of constructors for all types $\mu_1, \ldots, \mu_N$).

We will often write $\mathcal{R}_j^{\vec{\mu},\vec{\tau}}$ for $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$, and omit the upper indices $\vec{\mu}, \vec{\tau}$ when they are clear from the context. In case of a non-simultaneous free algebra, i.e., of type $\mu\alpha\,\kappa$, for $\mathcal{R}_\mu^{\mu,\tau}$ we write $\mathcal{R}_\mu^\tau$.

Examples.

$$\begin{aligned}&\mathrm{tt}^{\mathbf{B}} := \mathrm{C}_1^{\mathbf{B}}, \quad \mathrm{ff}^{\mathbf{B}} := \mathrm{C}_2^{\mathbf{B}},\\ &\mathcal{R}_{\mathbf{B}}^\tau \colon \tau \Rightarrow \tau \Rightarrow \mathbf{B} \Rightarrow \tau,\\ &0^{\mathbf{N}} := \mathrm{C}_1^{\mathbf{N}}, \quad \mathrm{S}^{\mathbf{N}\Rightarrow\mathbf{N}} := \mathrm{C}_2^{\mathbf{N}},\\ &\mathcal{R}_{\mathbf{N}}^\tau \colon \tau \Rightarrow (\mathbf{N} \Rightarrow \tau \Rightarrow \tau) \Rightarrow \mathbf{N} \Rightarrow \tau,\\ &\mathrm{Nil}^{\mathbf{L}(\alpha)} := \mathrm{C}_1^{\mathbf{L}(\alpha)}, \quad \mathrm{Cons}^{\alpha\Rightarrow\mathbf{L}(\alpha)\Rightarrow\mathbf{L}(\alpha)} := \mathrm{C}_2^{\mathbf{L}(\alpha)},\end{aligned}$$

$$\mathcal{R}^\tau_{\mathbf{L}(\alpha)} \colon \tau \Rightarrow (\alpha \Rightarrow \mathbf{L}(\alpha) \Rightarrow \tau \Rightarrow \tau) \Rightarrow \mathbf{L}(\alpha) \Rightarrow \tau,$$

$$\big(\mathrm{Inl}_{\rho\sigma}\big)^{\rho \Rightarrow \rho+\sigma} := \mathrm{C}_1^{\rho+\sigma},$$
$$\big(\mathrm{Inr}_{\rho\sigma}\big)^{\sigma \Rightarrow \rho+\sigma} := \mathrm{C}_2^{\rho+\sigma},$$
$$\mathcal{R}^\tau_{\rho+\sigma} \colon (\rho \Rightarrow \tau) \Rightarrow (\sigma \Rightarrow \tau) \Rightarrow \rho + \sigma \Rightarrow \tau,$$

$$\big(\otimes^+_{\rho\sigma}\big)^{\rho \Rightarrow \sigma \Rightarrow \rho\otimes\sigma} := \mathrm{C}_1^{\rho\otimes\sigma},$$
$$\mathcal{R}^\tau_{\rho\otimes\sigma} \colon (\rho \Rightarrow \sigma \Rightarrow \tau) \Rightarrow \rho\otimes\sigma \Rightarrow \tau.$$

6.2 Conversion

To define the conversion relation, it will be useful to employ the following notation. Let $\vec{\mu} = \mu\vec{\alpha}\,\vec{\kappa}$ and

$$\kappa_i = \rho_1 \Rightarrow \ldots \Rightarrow \rho_m \Rightarrow (\vec{\sigma}_1 \Rightarrow \alpha_{j_1}) \Rightarrow \ldots \Rightarrow (\vec{\sigma}_n \Rightarrow \alpha_{j_n}) \Rightarrow \alpha_j \in \mathrm{KT}(\vec{\alpha}),$$

and consider $\mathrm{C}_i^{\vec{\mu}}\vec{N}$. Then we write $\vec{N}^P = N_1^P, \ldots, N_m^P$ for the *parameter arguments* $N_1^{\rho_1}, \ldots, N_m^{\rho_m}$ and $\vec{N}^R = N_1^R, \ldots, N_n^R$ for the *recursive arguments* $N_{m+1}^{\vec{\sigma}_1 \Rightarrow \mu_{j_1}}, \ldots, N_{m+n}^{\vec{\sigma}_n \Rightarrow \mu_{j_n}}$, and n^R for the number n of recursive arguments.

We define a *conversion relation* $\mapsto_\rho$ between terms of type ρ by

$$(\lambda x M)N \mapsto M[x := N] \tag{3}$$
$$\lambda x.Mx \mapsto M \quad \text{if } x \notin \mathrm{FV}(M) \ (M \text{ not an abstraction}) \tag{4}$$
$$(\mathcal{R}_j\vec{M})^{\mu_j \Rightarrow \tau_j}(\mathrm{C}_i^{\vec{\mu}}\vec{N}) \mapsto M_i\vec{N}\big((\mathcal{R}_{j_1}\vec{M}) \circ N_1^R\big) \ldots \big((\mathcal{R}_{j_n}\vec{M}) \circ N_n^R\big) \tag{5}$$

Here we have written $\mathcal{R}_j$ for $\mathcal{R}^{\vec{\mu},\vec{\tau}}_{\mu_j}$.

The *one step reduction relation* $\to$ can now be defined as follows. $M \to N$ if N is obtained from M by replacing a subterm M' in M by N', where $M' \mapsto N'$. The reduction relations $\to^+$ and $\to^*$ are the transitive and the reflexive transitive closure of $\to$, respectively. For $\vec{M} = M_1, \ldots, M_n$ we write $\vec{M} \to \vec{M}'$ if $M_i \to M_i'$ for some $i \in \{1, \ldots, n\}$ and $M_j = M_j'$ for all $i \neq j \in \{1, \ldots, n\}$. A term M is *normal* (or in *normal form*) if there is no term N such that $M \to N$.

Clearly normal closed terms are of the form $\mathrm{C}_i^{\vec{\mu}}\vec{N}$.

6.3 Strong computability predicates

Definition 6.1. The set SN of *strongly normalizable* terms is inductively defined by

$$(\forall N.M \to N \Rightarrow N \in \mathrm{SN}) \Rightarrow M \in \mathrm{SN} \tag{6}$$

Note that with M clearly every subterm of M is strongly normalizable.

Definition 6.2. We define *strong computability predicates* SC^ρ by induction on ρ.

Case $\mu_j = (\mu\vec{\alpha}\,\vec{\kappa})_j$. Then $M \in \mathrm{SC}^{\mu_j}$ if

$$\forall N.M \to N \Rightarrow N \in \mathrm{SC}, \text{ and} \tag{7}$$

$$M = \mathrm{C}_i^{\vec{\mu}}\vec{N} \Rightarrow \vec{N}^P \in \mathrm{SC} \wedge \bigwedge\!\!\!\bigwedge_{p=1}^{n^R} \forall_{\vec{K}\in\mathrm{SC}}\, N_p^R\vec{K} \in \mathrm{SC}^{\mu_{jp}}. \tag{8}$$

Case $\rho \Rightarrow \sigma$.

$$M \in \mathrm{SC}^{\rho\Rightarrow\sigma} \;:\Longleftrightarrow\; \forall_{N\in\mathrm{SC}^\rho}\, MN \in \mathrm{SC}^\sigma.$$

Notice that the reference to $\vec{N}^P \in \mathrm{SC}$ and $\vec{K}\in\mathrm{SC}$ in (8) is legal, because the types $\vec{\rho}, \vec{\sigma}_i$ of $\vec{N}, \vec{K}$ must have been generated *before* μ_j. Note also that by (8) $\mathrm{C}_i^{\vec{\mu}}\vec{N} \in \mathrm{SC}$ implies $\vec{N} \in \mathrm{SC}$.

We now set up a sequence of lemmas leading to a proof that every term is strongly normalizing.

Lemma 6.3. *If $M \in \mathrm{SC}^\rho$ and $M \to M'$, then $M' \in \mathrm{SC}$.*

Proof. Induction on ρ. **Case** μ. By (7). **Case** $\rho \Rightarrow \sigma$. Assume $M \in \mathrm{SC}^{\rho\Rightarrow\sigma}$ and $M \to M'$; we must show $M' \in \mathrm{SC}$. So let $N \in \mathrm{SC}^\rho$; we must show $M'N \in \mathrm{SC}^\sigma$. But this follows from $MN \to M'N$ and $MN \in \mathrm{SC}^\rho$ by induction hypothesis (IH) on σ. □

Lemma 6.4. $\forall_{\vec{M}\in\mathrm{SN}}.\vec{M} \in \mathrm{SC} \Rightarrow (x\vec{M})^\mu \in \mathrm{SC}$.

Proof. Induction on $\vec{M} \in \mathrm{SN}$. Assume $\vec{M} \in \mathrm{SN}$ and $\vec{M} \in \mathrm{SC}$; we must show $(x\vec{M})^\mu \in \mathrm{SC}$. So assume $x\vec{M} \to N$; we must show $N \in \mathrm{SC}$. Now by the form of the conversion rules N must be of the form $x\vec{M'}$ with $\vec{M} \to \vec{M'}$. But $\vec{M'} \in \mathrm{SC}$ by Lemma 6.3, hence $x\vec{M'} \in \mathrm{SC}$ by IH for $\vec{M'}$. □

Lemma 6.5. *(a)* $\mathrm{SC}^\rho \subseteq \mathrm{SN}$*, (b)* $x \in \mathrm{SC}^\rho$.

Proof. By simultaneous induction on ρ. **Case** $\mu_j = (\mu\vec{\alpha}\,\vec{\kappa})_j$. (a). We show $M \in \mathrm{SC}^{\mu_j} \Rightarrow M \in \mathrm{SN}$ by (side) induction on $M \in \mathrm{SC}^{\mu_j}$. So assume $M \in \mathrm{SC}^{\mu_j}$; we must show $M \in \mathrm{SN}$. But for every N with $M \to N$ we have $N \in \mathrm{SC}$ by (7), hence $N \in \mathrm{SN}$ by the side induction hypothesis SIH. (b). $x \in \mathrm{SC}^{\mu_j}$ holds trivially.

Case $\rho \Rightarrow \sigma$. (a). Assume $M \in \mathrm{SC}^{\rho\Rightarrow\sigma}$; we must show $M \in \mathrm{SN}$. By IH(b) for ρ we have $x \in \mathrm{SC}^\rho$, hence $Mx \in \mathrm{SC}^\sigma$, hence $Mx \in \mathrm{SN}$ by IH(a) for σ. But $Mx \in \mathrm{SN}$ clearly implies $M \in \mathrm{SN}$. (b). Let $\vec{M} \in \mathrm{SC}^{\vec{\rho}}$ with $\rho_1 = \rho$; we must show $x\vec{M} \in \mathrm{SC}^\mu$. But this follows from Lemma 6.4, using IH(a) for $\vec{\rho}$. □

Corollary 6.6. $\vec{N} \in \mathrm{SC} \Rightarrow \mathrm{C}_i^{\vec{\mu}}\vec{N} \in \mathrm{SC}$, *i.e.* $\mathrm{C}_i^{\vec{\mu}} \in \mathrm{SC}$.

Proof. First show $\forall_{\vec{N}\in\mathrm{SN}}.\vec{N}\in\mathrm{SC}\Rightarrow \mathrm{C}_i^{\vec{\mu}}\vec{N}\in\mathrm{SC}$ by induction on $\vec{N}\in\mathrm{SN}$ as in Lemma 6.4, and then use Lemma 6.5(a). □

Lemma 6.7. $\forall_{M,N,\vec{N}\in\mathrm{SN}}.M[x:=N]\vec{N}\in\mathrm{SC}^{\mu}\Rightarrow(\lambda xM)N\vec{N}\in\mathrm{SC}^{\mu}$.

Proof. By induction on $M,N,\vec{N}\in\mathrm{SN}$. Let $M,N,\vec{N}\in\mathrm{SN}$ and assume $M[x:=N]\vec{N}\in\mathrm{SC}$; we must show $(\lambda xM)N\vec{N}\in\mathrm{SC}$. Assume $(\lambda xM)N\vec{N}\to K$; we must show $K\in\mathrm{SC}$. **Case** $K=(\lambda xM')N'\vec{N'}$ with $M,N,\vec{N}\to M',N',\vec{N'}$. Then $M[x:=N]\vec{N}\to^* M'[x:=N']\vec{N'}$, hence by (7) from our assumption $M[x:=N]\vec{N}\in\mathrm{SC}$ we can infer $M'[x:=N']\vec{N'}\in\mathrm{SC}$, therefore $(\lambda xM')N'\vec{N'}\in\mathrm{SC}$ by IH. **Case** $K=M[x:=N]\vec{N}$. Then $K\in\mathrm{SC}$ by assumption. □

Corollary 6.8. $\forall_{M,N,\vec{N}\in\mathrm{SN}}.M[x:=N]\vec{N}\in\mathrm{SC}^{\rho}\Rightarrow(\lambda xM)N\vec{N}\in\mathrm{SC}^{\rho}$.

Proof. By induction on ρ, using Lemma 6.5(a). □

Lemma 6.9. $\forall_{N\in\mathrm{SC}^{\mu_j}}\forall_{\vec{M},\vec{L}\in\mathrm{SN}}.\vec{M},\vec{L}\in\mathrm{SC}\Rightarrow\mathcal{R}_j\vec{M}N\vec{L}\in\mathrm{SC}^{\mu}$.

Proof. By main induction on $N\in\mathrm{SC}^{\mu_j}$, and side induction on $\vec{M},\vec{L}\in\mathrm{SN}$. Assume

$$\mathcal{R}_j\vec{M}N\vec{L}\to L.$$

We must show $L\in\mathrm{SC}$.

Case 1. $\mathcal{R}_j\vec{M'}N\vec{L'}\in\mathrm{SC}$ by the SIH.

Case 2. $\mathcal{R}_j\vec{M}N'\vec{L}\in\mathrm{SC}$ by the main induction hypothesis (IH).

Case 3. $N=\mathrm{C}_i^{\vec{\mu}}\vec{N}$ and

$$L=M_i\vec{N}\big((\mathcal{R}_j\vec{M})\circ N_1^R\big)\ldots\big((\mathcal{R}_j\vec{M})\circ N_n^R\big)\vec{L}.$$

$\vec{M},\vec{L}\in\mathrm{SC}$ by assumption. $\vec{N}\in\mathrm{SC}$ follows from $N=\mathrm{C}_i^{\vec{\mu}}\vec{N}\in\mathrm{SC}$ by (8). Note that for all recursive arguments N_p^R of N and all strongly computable $\vec{K}$ by (8) we have the IH for $N_p^R\vec{K}$ available. It remains to show $(\mathcal{R}_j\vec{M})\circ N_p^R=\lambda\vec{x}_p.\mathcal{R}_j\vec{M}(N_p^R\vec{x}_p)\in\mathrm{SC}$. So let $\vec{K},\vec{Q}\in\mathrm{SC}$ be given. We must show $(\lambda\vec{x}_p.\mathcal{R}_j\vec{M}(N_p^R\vec{x}_p))\vec{K}\vec{Q}\in\mathrm{SC}$. By IH for $N_p^R\vec{K}$ we have $\mathcal{R}_j\vec{M}(N_p^R\vec{K})\vec{Q}\in\mathrm{SC}$, since by Lemma 6.5(a) $\vec{K},\vec{Q}\in\mathrm{SN}$. Now Corollary 6.8 yields the claim. □

Corollary 6.10. $\mathcal{R}_j\in\mathrm{SC}$. □

Definition 6.11. A substitution ξ is *strongly computable*, if $\xi(x)\in\mathrm{SC}$ for all variables x. A term M is *strongly computable under substiꝡtution*, if $M\xi\in\mathrm{SC}$ for all strongly computable substitutions ξ.

Theorem 6.12. *Every term is strongly computable under substitution.*

Proof. Induction on the term M. **Case** x. $x\xi \in \mathrm{SC}$, since ξ is strongly computable. **Case** $\mathrm{C}_i^{\vec{\mu}}$. By Corollary 6.6. **Case** $\mathcal{R}_j$. By Corollary 6.10. **Case** MN. By IH $M\xi, N\xi \in \mathrm{SC}$, hence $(MN)\xi = (M\xi)(N\xi) \in \mathrm{SC}$. **Case** λxM. Let ξ be a strongly computable substitution; we must show $(\lambda xM)\xi = \lambda xM\xi_x^x \in \mathrm{SC}$. So let $N \in \mathrm{SC}$; we must show $(\lambda xM\xi_x^x)N \in \mathrm{SC}$. By IH $M\xi_x^N \in \mathrm{SC}$, hence $(\lambda xM\xi_x^x)N \in \mathrm{SC}$ by Corollary 6.8. □

Corollary 6.13. *Every term is strongly normalizable.* □

7 Implementation

This "logic for computable functionals" is the basis for the Minlog proof assistant `www.minlog-system.de`, under development in Munich. It treats partial functionals as first class citizens: variables range over all partial continuous functionals of a given type. Since these functionals are viewed as sets of tokens, we in fact quantify over sets, so we have a second order theory. However, the existence axioms – here in the form of which terms are allowed in $\forall$-elimination – are weak in the sense that these terms involve quantifiers over functionals, so our theory remains predicative.

In contrast to [24], formulas and types are kept separate. This makes it possible to avoid dependent types, which simplifies the theory considerably. More importantly, by separating the logic rules from type theory one avoids the well-known difficulty then when propositions are viewed as types and types as domains, then – as every domain is inhabited by its bottom element – every proposition would have a proof.

Types are built from base types (non-flat and possibly infinitary free algebras, with type parameters) by forming function spaces; this suffices for our intended mathematical applications. For more metamathematical subjects one may also add universe formation processes, as in [9]. Decidable predicates are viewed as boolean valued functions (and hence the rewrite mechanism described below applies to them), and inductive definitions are the common way to introduce undecidable predicates. In addition to free type variables also free predicate variables are allowed. They are viewed as placeholders for formulas (or more precisely, comprehension terms, that is formulas with some variables abstracted). However, in comprehension terms quantification over predicate variables is not allowed, since this would form a glaring impredicativity: we then would define a predicate (by the comprehension term) with reference to the totality of all predicates, to which the one to be defined belongs. A central application domain for the Minlog proof assistant is program extraction from constructive (and also classical [10]) proofs. This is done by means of a realizability interpretation, which requires – when the formula to be realized is given by an inductively defined predicate – a (possibly non-finitary) free algebra as domain of the realizers.

Computable functionals are defined by "computation rules" [11, 8]; these rules are added to the standard conversion rules of typed λ-calculus. To simplify equational reasoning, the system identifies terms with the same normal form. Then it clearly is desirable to use other equations as rewrite rules as well; for instance, we not only want to rewrite $M+0$ into M (which is an instance of a computation rule), but also $0+M$ into M. To justify this we need to prove $0+\hat{m}=\hat{m}$, where $\hat{m}$ ranges over *all* (possibly partial) objects of type $\mathbf{N}$. The standard way to prove such equations is of course induction. However, induction is only valid for total objects (or – for types with parameters – "structure-total" objects; cf. Section 3.1), hence cannot be used for equations involving partial variables. Here the approach developed in the present paper helps: one can prove the equality of the *values* of the two terms, by showing that both contain the same tokens, and then use reflection to conclude that the terms must be equal. The present paper aims at preparing the ground for such proofs.

References

[1] Andreas Abel and Thorsten Altenkirch. A predicative strong normalization proof for a λ-calculus with interleaving inductive types. In *Types for Proofs and Programs, International Workshop, TYPES '99, Lökeberg, Sweden, June 1999*, volume 1956 of *LNCS*, pages 21–40. Springer Verlag, Berlin, Heidelberg, New York, 2000.

[2] Samson Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.

[3] Samson Abramsky and Achim Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, 1994.

[4] Roberto M. Amadio and Pierre-Louis Curien. *Domains and Lambda-Calculi*. Cambridge University Press, 1998.

[5] Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *The Journal of Symbolic Logic*, 48(4):931–940, 1983.

[6] Holger Benl. Konstruktive Interpretation induktiver Definitionen. Master's thesis, Mathematisches Institut der Universität München, 1998.

[7] Ulrich Berger. Total sets and objects in domain theory. *Annals of Pure and Applied Logic*, 60:91–117, 1993.

[8] Ulrich Berger. Continuous semantics for strong normalization. In *Proc. CiE 2005*, volume 3526 of *LNCS*, pages 23–34, 2005.

[9] Ulrich Berger, Stefan Berghofer, Pierre Letouzey, and Helmut Schwichtenberg. Program extraction from normalization proofs. *Studia Logica*, 82:27–51, 2006.

[10] Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg. Refined program extraction from classical proofs. *Annals of Pure and Applied Logic*, 114:3–25, 2002.

[11] Ulrich Berger, Matthias Eberl, and Helmut Schwichtenberg. Term rewriting for normalization by evaluation. *Information and Computation*, 183:19–42, 2003.

[12] Frédéric Blanqui, Jean-Pierre Jouannaud, and Mitsuhiro Okada. The Calculus of Algebraic Constructions. In *RTA'99. LNCS 1631*, 1999.

[13] Thierry Coquand, Giovanni Sambin, Jan Smith, and Silvio Valentini. Inductively generated formal topologies. *Annals of Pure and Applied Logic*, 124:71–106, 2003.

[14] Thierry Coquand and Arnaud Spiwack. Proof of normalisation using domain theory. Slides of a talk, October 2005.

[15] Yuri L. Ershov. Everywhere defined continuous functionals. *Algebra i Logika*, 11(6):656–665, 1972.

[16] Yuri L. Ershov. Maximal and everywhere defined functionals. *Algebra i Logika*, 13(4):374–397, 1974.

[17] Jean-Yves Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J.E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 63–92. North–Holland, Amsterdam, 1971.

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

[19] Stephen C. Kleene. Countable functionals. In A. Heyting, editor, *Constructivity in Mathematics*, pages 81–100. North–Holland, Amsterdam, 1959.

[20] Georg Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In A. Heyting, editor, *Constructivity in Mathematics*, pages 101–128. North–Holland, Amsterdam, 1959.

[21] Lill Kristiansen and Dag Normann. Total objects in inductively defined types. *Archive for Mathematical Logic*, 36(6):405–436, 1997.

[22] Kim G. Larsen and Glynn Winskel. Using information systems to solve recursive domain equations. *Information and Computation*, 91:232–258, 1991.

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

[24] Per Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, 1984.

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

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

[27] Helmut Schwichtenberg. Density and choice for total continuous functionals. In P. Odifreddi, editor, *Kreiseliana. About and Around Georg Kreisel*, pages 335–362. A.K. Peters, Wellesley, Massachusetts, 1996.

[28] Dana Scott. A type theoretical alternative to ISWIM, CUCH, OWHY. Published in Theoret. Comput. Sci. 121 (1993), 411–440, 1969.

[29] Dana Scott. Outline of a mathematical theory of computation. Technical Monograph PRG–2, Oxford University Computing Laboratory, 1970.

[30] Dana Scott. Domains for denotational semantics. In E. Nielsen and E.M. Schmidt, editors, *Automata, Languages and Programming*, volume 140 of *LNCS*, pages 577–613. Springer Verlag, Berlin, Heidelberg, New York, 1982. A corrected and expanded version of a paper prepared for ICALP'82, Aarhus, Denmark.

[31] Viggo Stoltenberg-Hansen, Edward Griffor, and Ingrid Lindström. *Mathematical Theory of Domains.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1994.

[32] William W. Tait. Normal form theorem for bar recursive functions of finite type. In J.E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 353–367. North–Holland, Amsterdam, 1971.

[33] Anne S. Troelstra, editor. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics.* Springer Verlag, Berlin, Heidelberg, New York, 1973.

[34] Jeffrey Zucker. Iterated inductive definitions, trees and ordinals. In A.S. Troelstra, editor, *Mathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*, pages 392–453. Springer Verlag, Berlin, Heidelberg, New York, 1973.