Annals of Pure and Applied Logic 163 (2012) 575–603

Contents lists available at SciVerse ScienceDirect

Annals of Pure and Applied Logic

journal homepage: www.elsevier.com/locate/apal

The extensional ordering of the sequential functionals

D. Normann [a,*], V.Yu. Sazonov [b]

[a] Department of Mathematics, The University of Oslo, P.O. Box 1053, 0316 Oslo, Norway

[b] Department of Computer Science, The University of Liverpool, Liverpool L69 3BX, UK

ARTICLE INFO

Article history:
Available online 20 July 2011

MSC:
03D65

Keywords:
Sequential functional
Fully abstract model
Finite sequential procedure
Discontinuity

ABSTRACT

We investigate the extensional ordering of the sequential functionals of finite types, with a focus on when the sequential functionals of a given type form a directed complete partial ordering, and on when a finite sequential functional will be the nontrivial least upper bound of an infinite chain of sequential functionals. We offer a full characterization for finite functionals of pure types.

1. Introduction

In this paper, we investigate the closure properties of the extensional typed hierarchy of sequential functionals. It is always difficult to decide where a concept in mathematics came from, but we will let the history of the sequential functionals start with Platek's thesis [12]. Platek constructed an alternative to the full typed structure of hereditarily total functionals introduced by Kleene [6]. He worked with general finite types while Kleene restricted himself to the products of pure types. Platek's functionals are *partial* and they are *ordered* essentially by graph extension. In addition, they are monotone with respect to these orderings. We will not need the details of Platek's constructions.

Platek also defined a concept of *computable functional* and *relativized computable functional* based on an extension of typed λ-calculus, and he showed that this concept is equivalent to what we will obtain if we interpret Kleene's definition from [6] in Platek's typed structure.

Platek never published his thesis, but an account of the main concepts and proofs can be found in Moldestad's book [9].

Scott [19] added a continuity requirement to Platek's construction, and this led to his LCF with the denotational semantics based on a typed hierarchy of what has later been termed *Scott domains*. Scott's paper, though only published after many years as a historical document, was influential in many respects on the development of theoretical computer science.

There will be objects in Scott's hierarchy that are not the interpretation of any LCF-term, even when relativized to constants for functions $f : \mathbb{N} \to \mathbb{N}$. In [14], Sazonov gave a characterization of the objects in Scott's model that would be the interpretations of such terms, devising a natural notion of *strategies*. This may be viewed as the first characterization of what we will call *the sequential functionals*. Most of the content of [14] is also covered in [15,16].

Scott's model is not *fully abstract* for LCF in the sense that there will be LCF-terms T_1 and T_2 with different interpretations in the Scott model, but such that we may always substitute one for the other as a subterm of a program (a term of base type) without making any difference on the evaluated value. They are *observationally equivalent* but have different interpretations. Milner [8] constructed another hierarchy of Scott domains that served as the domain for a fully abstract interpretation of LCF, and he showed that up to isomorphism there can only be one such model. By construction, the interpretation of each type in Milner's model is closed under least upper bounds of countable chains, but the sequential nature of these least upper bounds were left unknown in general.

* Corresponding author.
E-mail addresses: dnormann@math.uio.no (D. Normann), sazonov@liverpool.ac.uk (V.Yu. Sazonov).

0168-0072/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.apal.2011.06.013

In his seminal paper [13] Plotkin rewrote LCF to PCF, with a sequential strategy for the operational semantics that is adequate with respect to the Scott and Milner interpretations, and PCF can be seen as a prototype for functional programming languages of the Haskell style. PCF_Ω is PCF extended with constants and rewriting rules for each $f : \mathbb{N} \to \mathbb{N}$, and one possible definition of the sequential functionals could be the typed extensional hierarchy of functionals hereditarily definable in PCF_Ω. One natural problem will then be if this hierarchy coincides with Milner's model.

Normann [11] answered this in the negative, for pure type 3 these structures do not coincide, the sequential functionals do not contain upper bounds of each countable chain.

Over the last 15 years there has been a number of independent approaches to the sequential functionals, see Nickau [10], Abramsky et al. [1], Hyland and Ong [5], Escardó and Ho [4] and Sazonov [17,18].

In this paper we will investigate the order-theoretical closure properties of the sequential functionals in some depth. Sazonov [17,18] formed five "hypotheses" on these closure properties and we verify four of them. The result from [11] has been improved in the sense that there will also be types at level 2 where the set of sequential functionals contains unbounded chains. We show that even for finite sequential functionals, interpreted as compacts in Milner's model, it is quite frequent that the object is the nontrivial least upper bound of an infinite chain of finitary functionals. We only offer a full characterization for the pure types.

We will also show that sequential functionals, and even finitary ones, need not be continuous in the sense that they commute with least upper bounds within the ordered sets of sequential functionals itself. This does not come as a surprise, and just illustrates that since we are not dealing with directed complete partial orderings, the least upper bound operator, which will be partial on the set of chains, is not a natural addition to the structure. Our result here, Corollary 5.3, actually solves a problem asked in Escardó and Ho [4], see Remark 5.4.

These results have values of two kinds. First they shed a new light on the order-theoretical closure properties of the sequential functionals, actually anomalous properties from the point of view of the usual domain theory, studied here in a systematic way, and thereby present new proof techniques giving a better understanding of the nature of sequential computability. Second, they witness that some other approaches are required to a non-traditional and more satisfactory domain theory of sequentiality such as (i) in terms of natural (pointwise) limits considered by Sazonov [17,18] which shows that sequential functionals are continuous and comprise topological f-spaces of Ershov, and (ii) in terms of rational ascending chains leading to operational and synthetic domain theory by Escardó and Ho [4] (where the reader can find more on the related work).

We have attempted to make the paper reasonably self-contained. This means that we give an introduction to the sequential functionals that can be read without detailed knowledge of the literature. We use a known characterization of the finite sequential functionals, see Hyland and Ong [5], as our starting point and see the infinite ones as natural extensions of the finite ones. Section 2 is devoted to the introduction of the sequential functionals.

In a sense, our sequential procedures may be viewed as infinitary PCF-terms on normal form. Two important motivations for introducing the sequential functionals this way, and not via some of the established characterizations, e.g. via games, are

1. We want to stay as close to the original formulations from early workers on computing with functionals, like Platek, Kleene, Scott, Plotkin and others, as possible, since we think that this provides the best intuition.
2. We need a mathematically precise definition of *an evaluation path* where the instances of applications in subcomputations are presented explicitly.

In Section 3 we prove a folklore normal form theorem for sequential functionals of a type at level 1 or of pure type 2, and show via two examples that such results cannot be extended beyond these types. In Section 4 we first give a detailed proof of the "folklore" result that the sequential functionals of pure type 2 form a directed complete partial ordering. Then we show that this cannot be extended to all types of level 2. In Sections 5 and 6 we develop our methods for producing infinite chains of sequential functionals with no upper bound or with pre-specified finite least upper bounds. In Section 7 we will discuss some loose ends and directions for further research.

Even if the paper to a large extent is self-contained, a knowledge of LCF or PCF with standard operational and denotational semantics for these calculi will be an advantage. We recommend Amadio and Curien [2] or Streicher [20] for background reading, together with the original papers we have referred to.

2. The sequential functionals of finite type

In this section we will introduce *the sequential functionals*. We do this by first defining what we will call *finite sequential procedures* with an operational semantics, then consider the extensional version, the *finite sequential functionals* and finally extend the constructions to *sequential procedures* and to *sequential functionals*. For the sake of completeness, we give detailed constructions and proofs, but even if the approach is semi-novel, there are no new results in this section.

2.1. Finite sequential procedures

We will deal with objects of simple, finite types. The types are syntactical entities, given by the grammar

Type σ
$\sigma ::= \iota \mid (\sigma \to \sigma)$

There are many ways to give an interpretation of these types, either as sets of terms or as sets of more set theoretical objects. What is common for all decent interpretations is that ι is interpreted either as $\mathbb{N} = \{0, 1, 2, \ldots\}$, as $\mathbb{N}_\perp = \mathbb{N} \cup \{\perp\}$ where $\perp$ is the "undefined" integer or as a set of terms representing elements in one of these structures. Normally we also have that $\sigma = (\tau \to \delta)$ either is interpreted as a set of functions or as a set of terms representing functions.

We will follow the standard convention, and drop the outer set of parentheses. Further, we will write $\tau, \delta \to \pi$ for $\tau \to (\delta \to \pi)$ and iterate this convention. This reflects the standard isomorphism between sets $X \times Y \to Z$ and $X \to (Y \to Z)$ known as "Currying".

Thus any type can be written on the normal form

$$\sigma = \tau_1, \ldots, \tau_t \to \iota$$

where $t \geq 0$. When $t = 0$, this simply means ι.

If $\sigma = \tau_1, \ldots, \tau_t, \delta_1, \ldots, \delta_d \to \iota$ and $\pi = \delta_1, \ldots, \delta_d \to \iota$ we will also have that

$$\sigma = \tau_1, \ldots, \tau_t \to \pi.$$

We will use the standard definition of the *level* of a type:

The level of ι is 0.

If $\sigma = \tau_1, \ldots, \tau_t \to \iota$, the level of σ will be $1 +$ the maximal level of τ_i for $i = 1, \ldots, t$.

The level of an object in a typed structure will be the level of the corresponding type.

We will now define one of the key concepts of this paper, the typed *finite sequential procedures*, abbreviated as FSP's.

We will give an inductive definition of this class. Formally, an FSP is a piece of syntax. We will discuss possible interpretations later.

Definition 2.1. Let

$$\sigma = \tau_1, \ldots, \tau_t \to \iota$$

be a type. Simultaneously for all σ we will define the set of FSP's, *the finite sequential procedures*, of type σ as follows:

1. If $a = \perp$ or $a \in \mathbb{N}$, then C_a^σ is an FSP of type σ. If $\sigma = \iota$, we identify C_a^ι with a.
2. If
   - $1 \leq i \leq t$
   - $\tau_i = \delta_1, \ldots, \delta_{d_i} \to \iota$
   - S_j is an FSP of type $\tau_1, \ldots, \tau_t \to \delta_j$ for each j from 1 to d_i
   - $K \subseteq \mathbb{N}$ is finite
   - T_k is an FSP of type σ for each $k \in K$

   then

   $$T = (i; S_1, \ldots, S_{d_i}; K; \{T_k\}_{k \in K})$$

   is an FSP of type σ. We will also write this by using variables $\vec{x}$ of the types $\vec{\tau}$ in the form of the conditional equation (see more on this notation below)

   $$T(\vec{x}) = T_k(\vec{x}) \textbf{ if } x_i(\vec{S}(\vec{x})) = k \in K.$$

We let FSP_σ denote the set of FSP's of type σ.

In case 2, if $\tau_i = \iota$, we have the formal notation

$$T = (i; ; K; \{T_k\}_{k \in K}),$$

or with variables,

$$T(\vec{x}) = T_k(\vec{x}) \textbf{ if } x_i = k \in K.$$

We may drop the upper index from C_a^σ whenever it is clear from the context or does not matter.

When we later refer to a "constant" we will by default mean an FSP of the form C_a^σ, where $a \in \mathbb{N}$.

The intuitive understanding of these procedures is best obtained by interpreting them over an arbitrary *applicative structure*:

Definition 2.2. An *applicative structure* consists of a set $[\![\sigma]\!]$ for each type σ and *application operators* $App_{\tau,\delta} : ([\![\tau \to \delta]\!]) \times [\![\tau]\!] \to [\![\delta]\!]$ for each pair τ and δ of types, such that $[\![\iota]\!] = \mathbb{N}_\perp$.

In addition we will require

(i) For every type $\sigma = \tau_1, \ldots, \tau_t \to \iota$ and $a \in \mathbb{N}_\perp$ there is an element in $[\![\sigma]\!]$ representing the constant function a defined on $[\![\tau_1]\!] \times \cdots \times [\![\tau_t]\!]$.

(ii) The structure is closed under *case*-constructions as in 2. of Definition 2.3 below.

If we have an applicative structure, $F \in [\![\tau \to \delta]\!]$ and $x \in [\![\tau]\!]$, we will write $F(x)$ or Fx instead of $App_{\tau,\delta}(F, x)$, even in the cases where F is not really a function. Examples of relevant applicative structures are the set of terms in PCF, the Scott model of algebraic domains and Milner's model. We are not going to work with applicative structures in general, only with some particular examples.

We will now give an interpretation of the FSP's over an applicative structure. In order to simplify the reading, we introduce variables for elements of the interpretations of the types, and later we will use variables in this way for this purpose. The variables are, however, not a part of our formal syntax.

Definition 2.3. Given an applicative structure $[\![\cdot]\!]$ with application operators, we give the following interpretation of each FSP of type $\sigma = \tau_1, \ldots, \tau_t \to \iota$. We use variables $x_1, \ldots, x_t$ for elements of types $\tau_1, \ldots, \tau_t$ respectively.

1. $C^\sigma_a(x_1, \ldots, x_t) = a$.
2. If $T = (i; S_1, \ldots, S_{d_i}; K; \{T_k\}_{k\in K})$ for $i > 0$ we let

$$T(x_1, \ldots, x_t) = \begin{cases} T_k(x_1, \ldots, x_t) \textbf{ if} \\ x_i(S_1(x_1, \ldots, x_t), \ldots, S_{d_i}(x_1, \ldots, x_t)) = k \in K, \\ \bot \quad \text{otherwise.} \end{cases}$$

Remark 2.4. In the sequel, we will use the terminology of this definition, and not the formally introduced syntax. This is easier to read, and we will be at liberty to choose variables in such a way that our arguments are simpler to follow. We will also let the "otherwise"-case be implicit in 2. above. We actually used this notation in order to explain the intended reading of the syntactic entities in Definition 2.1.

We will feel free to use vector notation for sequences. When the meaning is clear from the context, we e.g. will write $\vec{x}$ for sequences $x_1, \ldots, x_t$ of variables, and sometimes even expressions like $\vec{S}(\vec{x})$ for

$$(S_1(x_1, \ldots, x_t), \ldots, S_{d_i}(x_1, \ldots, x_t))$$

when the meaning is clear from the context.

It may improve the readability if we distinguish "if" in our notation with variables for FSP's from regular uses of "if", and for that reason we will use boldface, **if**. Thus our format is

$$T(\vec{x}) = T_k(\vec{x}) \textbf{ if } x_i(\vec{S}(\vec{x})) = k \in K.$$

If $\tau_i = \iota$ this reduces to

$$T(\vec{x}) = T_k(\vec{x}) \textbf{ if } x_i = k \in K.$$

There will be many cases of pairs of different finite sequential procedures that define the same function over any applicative structure. In particular, all $C^\sigma_\bot$ are definable using an empty set K.

One way to interpret an FSP is as a term in a minor extension of *simply typed* λ*-calculus*. We will need a term for each natural number, a constant for the *undefined* and terms for each

$$\textit{if } t = a \textit{ then } \ldots \textit{ else } \ldots,$$

with $a \in \mathbb{N}$, e.g. what we need is a small fragment of PCF without the fixed point constants and without much of the arithmetics. Then, of course, each set theoretical model for this fragment of typed λ-calculus can be used to give an interpretation of each FSP. In the sequel, we will be interested in the extensional model of *sequential functionals* to be defined later, but for now, we will wish to restrict ourselves to using the finite sequential procedures of type σ as the preliminary interpretation of σ. This requires that we define an *application operator* on the typed family of finite sequential procedures, or, as we might formulate it, show that the class of finite sequential procedures is closed under application.

We will view application as a special case of composition. If T is an FSP of type $\alpha \to \delta$ and if S is an FSP of type $\vec{\tau} \to \alpha$, we will consider the ordinary composition

$$H(\vec{x}) = T(S(\vec{x}))$$

where both sides are of type δ, or equivalently,

$$H(\vec{x}, \vec{y}) = T(S(\vec{x}), \vec{y})$$

where both sides are of type ι, using appropriate lists of typed variables. Then, if $\vec{\tau}$ is the empty sequence, this gives us plain application, which we will denote $[TS]$ in this context.

For the proof of the Theorem 2.8 below to go smoothly, we have to define a more general version of the composition operator $\circ$.

Example 2.5. If T is of type $(\iota \to \iota), (\iota \to \iota), \iota \to \iota$ and S is of type $\iota \to (\iota \to \iota)$, the composition $T \circ S$ is not uniquely defined unless we specify in which context the composition is made. Two possible interpretations will be

$(T \circ S)(f, n, k) = T(f, S(n), k)$.
$(T \circ S)(g, n, m) = T(S(n), g, m)$.

For the sake of simplicity, we will assume that α appears first in the list, that T is of type $\alpha, \vec{\sigma}, \vec{\tau} \to \iota$ and that S is of type $\vec{\delta}, \vec{\sigma} \to \alpha$. We assume further that z is a variable of type α, $\vec{x}$ is a list of variables of types $\vec{\delta}$, $\vec{y}$ is a list of variables of types $\vec{\sigma}$ and $\vec{u}$ is a list of variables of types $\vec{\tau}$. Then we aim at defining an FSP $(T \circ S)$ such that the following equation is valid in all applicative structures:

$$(T \circ S)(\vec{x}, \vec{y}, \vec{u}) = T(S(\vec{x}, \vec{y}), \vec{y}, \vec{u}).$$

When we consider T of type $\alpha, \vec{\sigma}, \vec{\tau} \to \iota$ and S of type $\vec{\delta}, \vec{\sigma} \to \alpha$, it is for notational convenience, and our construction will also cover the case where the list of argument types of T is any permutation of the type-vector $\alpha, \vec{\sigma}, \vec{\tau}$, and where the list of type arguments of S is any permutation of the type-vector $\vec{\delta}, \vec{\sigma}$. If we define composition where we deviate from the standard notation, we will explain carefully which type serves the rôle of α and which types serve the other rôles.

Note that the more general case, where T is of type $\alpha, \vec{\sigma}, \vec{\tau} \to \beta$, is covered by our treatment, since if $\beta = \vec{\eta} \to \iota$, we may replace $\vec{\tau}$ by $\vec{\tau}, \vec{\eta}$, and we will be in the situation we consider above.

In the proof of Theorem 2.8 we will be considering iterated compositions

$$T \circ (S_1, \ldots, S_n)$$

which will mean

$$(\cdots((T \circ S_1) \circ S_2) \cdots).$$

Here we of course have to make clear for all variables in T and $S_1, \ldots, S_n$ which rôle they play, e.g. for which variable in T do we substitute S_i.

We will only need the situation where the order of iterated composition does not really matter in the following sense: T will be of type $\alpha_1, \ldots, \alpha_n, \vec{\sigma}, \vec{\tau} \to \iota$ and each S_i will be of type $\vec{\delta}, \vec{\sigma} \to \alpha_i$, and iterated composition will be of type $\vec{\delta}, \vec{\sigma}, \vec{\tau} \to \iota$ and satisfy the equation

$$(T \circ (S_1, \ldots, S_n))(\vec{x}, \vec{y}, \vec{u}) = T(S_1(\vec{x}, \vec{y}), \ldots, S_n(\vec{x}, \vec{y}), \vec{y}, \vec{u}).$$

In our application, we will use other letters for the FSP's, other variables, and they will not come in the same order as in this explanation.

Finally, we will remark that our composition operator is *syntactical* (defined by manipulations with the syntax), and that it will be used to define the syntactical application operator $[TS]$ on the typed hierarchy of FSP's.

First we observe that all FSP's have dummy extensions:

Lemma 2.6. *Let $\vec{\tau} = (\tau_1, \ldots, \tau_n)$ and $\vec{\delta} = (\delta_1, \ldots, \delta_m)$ and assume that there is an injective map $\rho : \{1, \ldots, m\} \to \{1, \ldots, n\}$ such that for all i with $1 \le i \le m$ we have that $\tau_{\rho(i)} = \delta_i$.*

Let T be an FSP *of type $\vec{\delta} \to \iota$.*

Then there is a "dummy extension" T' of T to an FSP *of type $\vec{\tau} \to \iota$ such that for all applicative structures, if $\vec{x}$ are of types $\vec{\tau}$, $\vec{y}$ are of types $\vec{\delta}$ and $x_{\rho(i)} = y_i$ for $i \in \{1, \ldots, m\}$ we have that*

$$T(\vec{y}) = T'(\vec{x}).$$

The proof is trivial by induction on the rank of T.

We will need the following lemma in our proof of the composition theorem:

Lemma 2.7. *Let $\vec{x}$ be a sequence of variables of types $\vec{\tau}$.*

Let $S(\vec{x})$ be an FSP*of type $\vec{\tau} \to \iota$ and let $T_k(\vec{x})$ be an* FSP *of type $\vec{\tau} \to \iota$ for each k in a finite set K.*

Then there is an FSP *T of type $\vec{\tau} \to \iota$ that satisfies*

$$T(\vec{x}) = T_k(\vec{x}) \textbf{ if } S(\vec{x}) = k \in K.$$

Proof. We use induction on the rank of S.

If $S = C_k^{\vec{\tau} \to \iota}$ for some $k \in K$, we let $T = T_k$.

If S is another constant, we let $T = C_\perp^{\vec{\tau} \to \iota}$.

If

$$S(\vec{x}) = S_l(\vec{x}) \textbf{ if } x_i(\vec{R}(\vec{x})) = l \in L$$

we let

$$T(\vec{x}) = H_l(\vec{x}) \textbf{ if } x_i(\vec{R}(\vec{x})) = l \in L$$

where H_l is defined, using the induction hypothesis, as

$$H_l(\vec{x}) = T_k(\vec{x}) \textbf{ if } S_l(\vec{x}) = k \in K.$$

This ends the proof. □

When we use the expression

$$T(\vec{x}) = T_k(\vec{x}) \text{ if } S(\vec{x}) = k \in K,$$

we mean the FSP that is constructed in this proof.

We will now be ready to formulate and prove the *Composition Theorem*.

Theorem 2.8. *Let T be a finite sequential procedure of type $\alpha, \vec{\sigma}, \vec{\tau} \to \iota$ and let S be a finite sequential procedure of type $\vec{\delta}, \vec{\sigma} \to \alpha$.*

Then there is a finite sequential procedure

$$T \circ S \in \mathrm{FSP}^{\vec{\delta}, \vec{\sigma}, \vec{\tau} \to \iota}$$

such that we in any applicative structure, and for all $\vec{x}$ of types $\vec{\delta}$, $\vec{y}$ of types $\vec{\sigma}$ and $\vec{u}$ of types $\vec{\tau}$ in the given applicative structure, have that

$$(*) \quad (T \circ S)(\vec{x}, \vec{y}, \vec{u}) = T(S(\vec{x}, \vec{y}), \vec{y}, \vec{u}).$$

Proof. We first give a self-referential definition of the syntactical operator $\circ$, and then prove that it is everywhere well defined. It will then be clear from the construction that $(*)$ will hold. Recall that the way we order the types and the variables are just for convenience, and that our composition is meant to make sense independently of how we organize the types and variables.

If $T = C_a^{\alpha, \vec{\sigma}, \vec{\tau} \to \iota}$, we let

$$T \circ S = C_a^{\vec{\delta}, \vec{\sigma}, \vec{\tau} \to \iota}.$$

If

$$T(z, \vec{y}, \vec{u}) = T_k(z, \vec{y}, \vec{u}) \text{ if } y_i(R_1(z, \vec{y}, \vec{u}), \dots, R_r(z, \vec{y}, \vec{u})) = k \in K$$

we let

$$(T \circ S)(\vec{x}, \vec{y}, \vec{u}) = (T_k \circ S)(\vec{x}, \vec{y}, \vec{u})$$
$$\text{if } y_i((R_1 \circ S)(\vec{x}, \vec{y}, \vec{u}), \dots, (R_r \circ S)(\vec{x}, \vec{y}, \vec{u})) = k \in K.$$

The case

$$T(z, \vec{y}, \vec{u}) = T_k(z, \vec{y}, \vec{u}) \text{ if } u_j(R_1(z, \vec{y}, \vec{u}), \dots, R_r(z, \vec{y}, \vec{u})) = k \in K$$

is handled in the analogue way.

There will be hidden variables in $R_s(z, \vec{y}, \vec{u})$ unless its type is ι, but the same variables will be hidden in $(R_s \circ S)(\vec{x}, \vec{y}, \vec{u})$, so the typing is correct.

If

$$T(z, \vec{y}, \vec{u}) = T_k(z, \vec{y}, \vec{u}) \text{ if } z(R_1(z, \vec{y}, \vec{u}), \dots, R_s(z, \vec{y}, \vec{u})) = k \in K$$

we must split the construction into several cases.

1.1 $\alpha = \iota$ and $\vec{x}, \vec{y}, \vec{u}$ is the empty sequence.

Then S is simply an element of $\mathbb{N}_\perp$, and we want $T \circ S$ to be an element of $\mathbb{N}_\perp$ as well. Let $T \circ S = T_k \circ S$ if $S = k \in K$, and $T \circ S = \perp$ otherwise.

1.2 $\alpha = \iota$, but at least one of $\vec{x}, \vec{y}$ or $\vec{u}$ is nonempty.

Since $\alpha = \iota$, $s = 0$, i.e. there are no R_j's.

We let

$$(T \circ S)(\vec{x}, \vec{y}, \vec{u}) = (T_k \circ S)(\vec{x}, \vec{y}, \vec{u}) \text{ if } S(\vec{x}, \vec{y}) = k \in K,$$

see Lemmas 2.6 and 2.7.

2.1 $\alpha \neq \iota$ and $\vec{x}, \vec{y}, \vec{u}$ is the empty sequence.

Let $\alpha = \vec{\pi} \to \iota$, where $\vec{\pi}$ has length s, and let $\pi_i = \vec{\rho}_i \to \iota$. The variables for R_i will be z and the hidden variables for $\vec{\rho}_i$. S will be an element of type α, so, if well defined, $R_i \circ S$ is an element of type π_i.

S may also be viewed as a term of type ι in variables $\vec{v}$ of types $\vec{\pi}$, and then, when we substitute $R_i \circ S$ of type π_i for v_i using iterated composition, we obtain $S \circ ((R_1 \circ S), \dots, (R_s \circ S))$ of type ι. If this composition is well defined, it will be an element of $\mathbb{N}_\perp$.

If $S \circ ((R_1 \circ S), \dots, (R_s \circ S)) = k \in K$, we let $T \circ S = T_k \circ S$ while otherwise, we let $T \circ S = \perp$.

2.2 $\alpha \neq \iota$ and $\vec{x}, \vec{y}, \vec{u}$ is nonempty.

We then let

$$(T \circ S)(\vec{x}, \vec{y}, \vec{u}) = (T_k \circ S)(\vec{x}, \vec{y}, \vec{u}) \text{ if } (S \circ ((R_1 \circ S), \dots, (R_s \circ S)))(\vec{x}, \vec{y}, \vec{u}) = k \in K,$$

see Lemma 2.7.

When we substitute S for z in R_i we get an object of type π_i in variables $\vec{x}, \vec{y}, \vec{u}$, and when we then view S as an object of type ι in typed variables $\vec{x}, \vec{y}, \vec{v}$ and substitute $R_i \circ S$ for v_i in this term for S, we get a term of type ι in variables $\vec{x}, \vec{y}, \vec{u}$. Thus the typing of $T \circ S$ is correct.

This ends our construction. It remains to prove that $T \circ S$ is well defined, i.e. actually an FSP.

We prove this by induction on the level of the type α and subinduction on the rank of T in the inductive definition of FSP's.

The induction base will be $\alpha = \iota$, and then we will use subinduction on the rank of T.

If T is a constant, $T \circ S$ is a constant, so $T \circ S$ is well defined.

If T is not a constant, we will be in one of the cases 1.1 or 1.2. In case 1.1, S will be an element of $\mathbb{N}_\perp$, and by the subinduction hypothesis each $T_k \circ S$ is defined, so $T \circ S$ is defined. In case 1.2, we rely on the subinduction hypothesis and Lemma 2.7. The induction step will be when $\alpha \neq \iota$. If T is a constant, the result is again trivial. This is the subinduction base. For the subinduction step we will be in case 2.1 or 2.2.

In both cases, $R_i \circ S$ is well defined for $i = 1, \ldots, s$, by the subinduction hypothesis, and each $T_k \circ S$ is well defined by the same assumption.

In both cases, we consider $R_i \circ S$ as a term of type π_i in variables $\vec{x}, \vec{y}, \vec{u}$ (which is the empty sequence in case 2.1). Since the level of π_i is lower than that of α, we use the induction hypothesis and see that $S \circ ((R_1 \circ S), \ldots, (R_s \circ S))$ is well defined.

This is all we need in order to obtain that $T \circ S$ is well defined. This ends the proof of Theorem 2.8. □

Corollary 2.9. *The finite sequential procedures may be organized to an applicative structure.*

Proof. Let T be of type $\tau \to \delta$ and S of type τ.

Let $App_{\tau,\delta}(T, S) = T \circ S$ where we consider S as a closed term of type τ and compose it with the first variable in T. □

If T is an FSP of type $\vec{\tau} \to \delta$ and $\vec{S}$ are FSP's of types $\vec{\tau}$, we write $[T\vec{S}]$ for what is obtained by iterating application of T on the elements of $\vec{S}$ using the syntactical composition operator. When we write $T\vec{S}$ in plain juxtaposition terminology, we have extended our formal language to accept applicative terms like in λ-calculus.

In the sequel, we will argue by induction on the length of an evaluation, and then it will be useful to represent this evaluation in a way that is closer to the intuition. Our composition operator $\circ$ may be considered as an operational semantics on applicative terms involving FSP's:

Definition 2.10. We consider the term language $\mathcal{L}$ consisting of

1. Typed constants representing each FSP of type σ for each type σ. Constants are terms.
2. If M is a term of type $\sigma \to \tau$ and N is a term of type σ, then (MN) is a term of type τ.

We call the terms in $\mathcal{L}$ *finite applicative terms*, or just *applicative terms* when no confusion may arise.

We will follow the standard conventions for leaving out (and). Each term in $\mathcal{L}$ of type δ will correspond to an FSP of type δ via our rewriting rules for $\circ$. The rewriting process can be made deterministic, and in particular, if M is a term of type ι, the rewriting process can be seen as an evaluation of the term. We will make the concept of an evaluation path precise in Definition 2.11, and the construction of an evaluation path is what we will technically consider to be *an evaluation*.

Each term M in $\mathcal{L}$ is of the form

$$M = TN_1 \cdots N_t$$

where T is an FSP and $N_1, \ldots, N_t$ are in $\mathcal{L}$. If M is of type ι, then the evaluation of M can be viewed as a list of terms, where we at some steps need to evaluate another term in order to know which step to take.

Definition 2.11. Let M be a closed term in $\mathcal{L}$ of type ι.

Let

$$M = TN_1 \cdots N_t$$

where T is an FSP.

The *evaluation path* of M is a nested structure consisting of a sequence of terms in $\mathcal{L}$ and, recursively, of subevaluation paths surrounded by brackets [and] constructed as follows:

1. If $T = C^\sigma_a$ we let the evaluation path be the sequence M, a.
2. If

$$T(\vec{x}) = T_k(\vec{x}) \textbf{ if } x_i(R_1(\vec{x}), \ldots, R_s(\vec{x})) = k \in K$$

we let the evaluation path of M start with M and then we write the evaluation path of

$$N_i(R_1\vec{N}) \cdots (R_s\vec{N})$$

within brackets [and].

If this path ends with $\perp$ or some $k \notin K$, we end the evaluation path of M by writing $\perp$ at the end. If, on the other side, this path ends with a k in K, we continue the evaluation path for M by writing down the evaluation path of $T_k\vec{N}$.

Our construction of the evaluation path follows the rewriting procedure from the proof of Theorem 2.8 in a deterministic way, and will terminate as a consequence of Theorem 2.8 and its proof. This means that all evaluation paths are finite ending either in an integer or in $\perp$ as the recognized outcome of the evaluation.

The full argument for this is too space-consuming, but the idea behind it is as follows:

First note that the proof of the termination of the rewriting process gives us that the order of rewriting different subexpressions does not matter for termination.

If we view the term $TN_1 \dots N_t$ as an iterated composition $T \circ (N_1, \dots, N_t)$, we split the argument into two cases:

If T is a constant, then the rewriting of $TN_1 \cdots N_t$ takes t steps, while we in the evaluation path only use one step. The result will be the same.

If

$$T(\vec{x}) = T_k(\vec{x}) \textbf{ if } x_i(R_1(\vec{x}), \dots, R_s(\vec{x}))$$

the first $i-1$ steps of the rewriting of $TN_1 \cdots N_t$ just gives us as the first intermediate step

$$S_i(x_i, \dots, x_t) = TN_1 \cdots N_{i-1}(x_i, \dots, x_t) = T_k N_1 \cdots N_{i-1}(x_i, \dots, x_t)$$
$$\textbf{if } x_i(R_1 N_1 \cdots N_{i-1}(x_i, \dots, x_t), \dots, R_s N_1 \cdots N_{i-1}(x_i, \dots, x_t)) = k \in K$$

and the task to rewrite $S_i N_i \cdots N_t$. This is because we are in the simple case before step no. i in the rewriting process.

At step i we must employ the part of the construction that splits into cases 1.1, 1.2, 2.1 and 2.2, and this is where the real evaluation takes place. In our construction of an evaluation path, what we put within the brackets is the evaluation path for

$$N_i(R_1\vec{N}) \cdots (R_s\vec{N})$$

and in the rewriting process for the iterated application we will get

$$N_i \circ ((R_1 N_1 \cdots N_{i-1}), \dots, (R_s N_1 \cdots N_{i-1}))(x_i, \dots x_t).$$

In the further application of our main expression on $N_i, \dots, N_t$, we have to evaluate

$$N_i(\vec{R}\vec{N}),$$

and it is this we capture by the subpath in brackets [and] of the evaluation path.

Thus, the evaluation, considered as producing an evaluation path, is taking shortcuts in comparison with the rewriting process, but every step in the former can be found as the result of a number of steps in the latter, so termination of the evaluation process is granted.

2.2. Finite sequential functionals

We have seen that the finite sequential procedures form an applicative typed structure by themselves.

For the sake of notational simplicity, we will not distinguish between the applicative term TS and its translation $[TS]$ to an FSP, meaning that we will drop the use of $[\cdots]$. Thus all applicative terms are understood to be finite sequential procedures themselves.

We will now define *an extensional ordering* reflecting how we understand these procedures as set theoretical functions. We then prove that all FSP's are monotone with respect to this ordering and use this to extract the extensional interpretation of each FSP.

Definition 2.12. We define the pre-ordering $\sqsubseteq_\sigma$ on FSP_σ by recursion on the type σ as follows:

1. If $\sigma = \iota$, let $\sqsubseteq_\iota$ be the flat ordering on $\mathbb{N}_\perp$.
2. If $\sigma = \tau_1, \dots, \tau_t \to \iota$ where $t \geq 1$, we let $T_1 \sqsubseteq_\sigma T_2$ if for all sequences $\vec{S}$ of FSP's of types $\vec{\tau}$ we have that
$$T_1\vec{S} \sqsubseteq_\iota T_2\vec{S}.$$

We may drop the index σ from $\sqsubseteq_\sigma$ when it is not needed.

Lemma 2.13. *Each* FSP T *is monotone with respect to* $\sqsubseteq$.

Proof. We use induction on the type and subinduction on the rank of T.

If T is a constant, the claim is trivial, so let

$$T(\vec{x}) = T_k(\vec{x}) \textbf{ if } x_i(R_1(\vec{x}), \dots, R_r(\vec{x})) = k \in K.$$

Let $\vec{S} \sqsubseteq \vec{S}'$

Then, by the subinduction hypothesis, $R_j\vec{S} \sqsubseteq R_j\vec{S}'$ for $j = 1, \dots, r$.

By the induction hypothesis, S_i is monotone, so $S_i(\vec{R}\vec{S}) \sqsubseteq S_i(\vec{R}\vec{S}')$, and since $S_i \sqsubseteq S_i'$ we have that

$$S_i(\vec{R}\vec{S}') \sqsubseteq S_i'(\vec{R}\vec{S}').$$

It follows that if $S_i(\vec{R}\vec{S}) = k \in K$, then $S_i'(\vec{R}\vec{S}') = k$.

We may then use the subinduction hypothesis on T_k.

This ends the proof. $\square$

The partial pre-ordering $\sqsubseteq_\sigma$ induces an equivalence relation $\equiv_\sigma$ on the finite sequential procedures, and as a consequence of Lemma 2.13 all FSP's will respect this relation. Thus we have a quotient structure that will be an applicative structure, and which we will use to give the *extensional interpretation* of the procedures. The ordered set of extensional interpretations of finite sequential procedures will be isomorphic to the ordered set of compacts in Milner's model [8], and the ideal completion will in fact, up to isomorphism, be Milner's model.

In general, the pre-ordering $\sqsubseteq_\sigma$ on FSP_σ is not computable. This can be deduced from the main result in [7]. We do not give the detailed argument. Being extensional by construction, this applicative structure is isomorphic to a typed structure of set theoretical functions.

Definition 2.14. By recursion on the type σ we define *the finite sequential functional* $\langle T\rangle$ for each finite sequential procedure T as follows:

If $a \in \mathbb{N}_\perp$, we let $\langle a\rangle = a$.
If $T \in \mathrm{FSP}_{\tau\to\delta}$ and $G = \langle S\rangle$ where G is in FSP_τ, we let $\langle T\rangle(G) = \langle TS\rangle$.
The *finite sequential functionals* of type $\tau \to \delta$ are, by definition, exactly the functions of the form $\langle T\rangle$ where $T \in \mathrm{FSP}_{\tau\to\delta}$.

Overloading our notation, we will use $\sqsubseteq$ and $\sqsubseteq_\sigma$ also for the induced ordering on the finite sequential functionals.

The key structures under investigation in this paper will be the typed structure of finite sequential functionals, seen as partial orderings, and the extension to the sequential functionals to be defined below.

Now we will establish some similarities between the finite sequential functionals and the compacts in algebraic domains representing spaces of continuous functionals.

Lemma 2.15. *The finite sequential functionals of a given type form a lower semi-lattice.*

Proof. Let T and S be elements of FSP_σ. Let

$H(\vec{x}) = a$ if $T(\vec{x}) = S(\vec{x}) = a$ for some a,
$H(\vec{x}) = \perp$ otherwise.

It is a simple exercise to construct an FSP from T and S acting as H, and we clearly have that $\langle H\rangle$ is the greatest lower bound of $\langle T\rangle$ and $\langle S\rangle$. This ends the proof. □

As usual, we will write $\sqcap$ for a greatest lower bound, and $\sqcup$ for a least upper bound whenever it exists.

For each type σ and each $c \in \mathbb{N}$ we will define the c-projection P^σ_c. We will use this to show that any finite, bounded set of finite sequential functionals will have a least upper bound.

Definition 2.16. We define the c-projection P^σ_c as a finitary sequential procedure:

$P^\iota_c(x) = b$ **if** $x = b \in \{0,\dots,c\}$.
If $\sigma = \tau_1,\dots,\tau_t \to \iota$ we let

$$P^\sigma_c(x, y_1,\dots,y_t) = b \text{ if } x(P^{\tau_1}_c(y_1),\dots,P^{\tau_t}_c(y_t)) = b \in \{0,\dots,c\}.$$

We may use the monotonicity of each finite sequential functional to show, by induction on σ, that $P^\sigma_c(T) \sqsubseteq T$ for all T of type σ. Moreover, $P^\sigma_c(T)$ will always be a fixed point for P^σ_c in the sense that

$$P^\sigma_c(P^\sigma_c(T)) \equiv_\sigma P^\sigma_c(T),$$

and the image of P^σ_c is a finite set of finite sequential functionals.

Definition 2.17. We let FSP^c_σ be the set of finite sequential procedures T such that whenever some C^τ_a is a subprocedure of T then $a \le c$, and whenever a set K is used in a subprocedure of T, then K is bounded by c.

We let FSP^c be the union of all FSP^c_σ.

Lemma 2.18. *For each finite sequential procedure $T \in \mathrm{FSP}_\sigma$ there is a $T' \in \mathrm{FSP}^c_\sigma$ such that $[P^\sigma_c T] = P^\sigma_c \circ T \equiv_\sigma T'$. Moreover, for $T \in \mathrm{FSP}^c_\sigma$ we have $[P^\sigma_c T] \equiv_\sigma T$. In particular, the range of $\langle P^\sigma_c\rangle$ and the set of fixed points of $\langle P^\sigma_c\rangle$ coincide with the set of finite functionals representable by elements of FSP^c_σ.*

Proof. This follows by induction on the rank of T. □

In the sequel, T and S (with indices) will denote finite sequential procedures, while F and G (with indices) will denote finite sequential functionals. If other letters are needed, the nature of the objects they denote will be clear from the context. We will also write P^σ_c instead of $\langle P^\sigma_c\rangle$ when it is clear from the context that we operate on functionals and not procedures.

Lemma 2.19. *Let σ be a type, and let $\{F_1,\dots,F_n\}$ be a finite, bounded set of sequential functionals of type σ. Then there is a least upper bound*

$$\bigsqcup\{F_1,\dots,F_n\}$$

in the set of finite sequential functionals of type σ.

Proof. Let c be so large that each F_i is a fixed point of P^σ_c. If F is an upper bound for $\{F_1, \dots, F_n\}$ we also have that $P^\sigma_c(F)$ is an upper bound. It follows that if we take the greatest lower bound of the nonempty, but finite, set of upper bounds for $\{F_1, \dots, F_n\}$ in the image of P^σ_c, we get the least upper bound of the given set. This ends the proof. □

Theorem 2.20. *Let $t \geq 1$ and let F be a finite sequential functional of type $\sigma = \vec{\tau} \to \iota$.*

Then there is a pairwise $\sqsubseteq_{\vec{\tau}}$-incomparable set $\{\vec{G}_1, \dots, \vec{G}_n\}$ of $\vec{\tau}$-sequences of finite sequential functionals such that for any $\vec{\tau}$-sequence $\vec{G}$ we have that

$$F(\vec{G}) \in \mathbb{N} \Leftrightarrow \exists i(1 \leq i \leq n)(\vec{G}_i \sqsubseteq \vec{G}).$$

It is easy to see that there will be at most one set satisfying the required properties of $\{\vec{G}_1, \dots, \vec{G}_n\}$. We call this set the *basis* for F.

Proof. By Lemma 2.18, choose c such that F is a fixed point for P^σ_c. Then, for any $\vec{G}$ of type $\vec{\tau}$ we have that

$$F(\vec{G}) = P^\sigma_c(F)(\vec{G}) = F(P^{\vec{\tau}}_c(\vec{G})).$$

Since $P^{\vec{\tau}}_c(\vec{G}) \sqsubseteq \vec{G}$ for all $\vec{G}$, we get the basis by selecting all $\sqsubseteq$-minimal elements in

$$\{P^{\vec{\tau}}_c(\vec{G}) \mid F(\vec{G}) \in \mathbb{N}\}.$$

Since this set is finite, there will be minimal elements below any element.

This ends the proof. □

Corollary 2.21. *Let F be a finite sequential functional of type σ and let $\{F_n\}_{n\in\mathbb{N}}$ be a $\sqsubseteq_\sigma$-increasing sequence of sequential functionals bounded by F and with F as its pointwise least upper bound.*

Then there is an $n_0 \in \mathbb{N}$ such that $F = F_n$ for all $n \geq n_0$.

Proof. Let $\{\vec{G}_1, \dots, \vec{G}_k\}$ be the basis for F. Since F is the pointwise limit of the F_n's, there is a number n_0 such that $F(\vec{G}_i) = F_n(\vec{G}_i)$ for all $i = 1, \dots, k$ and $n \geq n_0$.

Then $F = F_n$ for all $n \geq n_0$. This ends the proof. □

Definition 2.22. Let $\sigma = \tau_1, \dots, \tau_t \to \iota$ and let $\vec{G}$ be a sequence of finite sequential functionals of type $\vec{\tau}$. Let $a \in \mathbb{N}$.

We define the *step function*

$$\Sigma = \vec{G} \mapsto a$$

of type σ by

$\Sigma(\vec{F}) = a$ if $\vec{G} \sqsubseteq \vec{F}$.

$\Sigma(\vec{F}) = \bot$ otherwise.

Corollary 2.23. (a) *Each step function is sequential.*

(b) *Each finite sequential functional will be the least upper bound of a finite set of step functions.*

Proof. (a) Let $\{\vec{f}_{i,1}, \dots, \vec{f}_{i,k_i}\}$ be the base for G_i. Then

$$\Sigma(\vec{F}) = a \textbf{ if } \bigwedge_{i=1}^{t} \bigwedge_{j=1}^{k_i} (F_i(\vec{f}_{i,j}) = G_i(\vec{f}_{i,j})),$$

and this can clearly be represented as a sequential functional.

(b) Let F be given, let $\{\vec{G}_1, \dots, \vec{G}_n\}$ be a basis for F and let $a_i = F(\vec{G}_i)$ for $i = 1, \dots, n$. Then

$$F = \bigsqcup_{i=1}^{n} (\vec{G}_i \mapsto a_i). \quad \square$$

Remark 2.24. We will pay special attention to step functions in this paper, see Section 5.

2.3. The sequential functionals

As already mentioned, the ideal completion of the finite sequential functionals will be Milner's model from 1977, see [8]. For many years it was open whether all functionals in Milner's model are sequential in the sense that they are PCF_Ω-definable. Normann [11] showed that this is not the case, by constructing a functional of pure type 3 in Milner's model that is not PCF_Ω-definable. In this section we will give an alternative definition of the sequential functionals. It is known, e.g. from [5] that our definition coincides with the one referring to PCF.

The finite sequential procedures are defined via an inductive definition. The *sequential procedures* will be defined in a very similar way, except that we use a co-inductive definition, and we permit infinite branching:

Definition 2.25. The sequential procedures SP_σ of type $\sigma = \tau_1, \ldots, \tau_t \to \iota$ is, uniformly for all types σ, the largest class satisfying that if $T \in \mathrm{SP}_\sigma$ then

either $T = C^\sigma_a$ for some $a \in \mathbb{N}_\perp$
or $t > 0$ and T is of the form $(i; R_1, \ldots, R_s; K; \{T_k\}_{k\in K})$, where $K \subseteq \mathbb{N}$ may now also be infinite, and
$1 \le i \le t$ and $\tau_i = \delta_1, \ldots, \delta_s \to \iota$
$R_j \in \mathrm{SP}_{\vec\tau\to\delta_j}$ for $j = 1, \ldots, s$
$T_k \in \mathrm{SP}_\sigma$ for all $k \in K$.

That is, unlike FSP's, we allow SP's to be possibly infinitely nesting formal expressions having "locally" the same style as FSP's and with possible infinite K's.

The following few lines are for the readers who would like a more formal treatment of the co-inductive definition:

Let $\mathbb{N}^*$ be the set of finite strings of natural numbers with Λ the empty string and $x \cdot y$ denoting concatenation for x, y strings or individual numbers. Let $\mathbb{N}'$ be the set of copies $1', 2', \ldots$ of positive natural numbers, also called *queries*. Note that t and s above depend on σ and so can be written as $t(\sigma)$ and $s(i, \sigma)$ for $1 \le i \le t(\sigma)$. We also denote $\vec\tau \to \delta_j$ as $\sigma(i, j)$ taking into account the actual dependence of this type on σ and i.

We can formally represent any SP T of type σ as a pair $(\sigma, \bar T)$ where $\bar T$ is a partial function (labeled tree)

$$\bar T : \mathbb{N}^* \to \mathbb{N} \cup \mathbb{N}'$$

such that the following two conditions hold:

1. If $\bar T(\Lambda) = a \in \mathbb{N}^*$ or undefined then $\bar T(x)$ is undefined for any nonempty string $x \in \mathbb{N}$, and we denote $T = C^\sigma_a$ or $= C^\sigma_\perp$, respectively.
2. If $\bar T(\Lambda) = i' \in \mathbb{N}'$ then $1 \le i \le t(\sigma)$ must hold, and we let
   - $\bar R_j = \lambda x \in \mathbb{N}^*.\bar T(j \cdot x)$ for any $1 \le j \le s(i, \sigma)$ (where $\bar T(j)$ for any of these j could be possibly undefined),
   - $\bar T_k = \lambda x \in \mathbb{N}^*.\bar T((s(i, \sigma) + 1 + k) \cdot x)$ for any $k \in \mathbb{N}$ for which $\bar T(s(i, \sigma) + 1 + k)$ is defined (this defines a $K \subseteq \mathbb{N}$),
   - and assume (recursively) that $S_j = (\sigma(i, j), \bar S_j)$ and $T_k = (\sigma, \bar T_k)$ satisfy the same conditions 1 and 2 as $T = (\sigma, \bar T)$.

Evidently, if $\bar T(x\cdot y)$ is defined then so is $\bar T(x)$, for any $x, y \in \mathbb{N}^*$. This means that $\bar T$ may be considered as a possibly infinite tree with the nodes labeled by elements of $\mathbb{N} \cup \mathbb{N}'$. The allowed shape of such trees (with labels ignored) is absolutely arbitrary. In fact, we also allow the empty tree (if $\bar T(\Lambda)$ is undefined) having even no root. All the non-leaf nodes must have queries $i' \in \mathbb{N}'$ as labels, and leafs can have any label $a \in \mathbb{N}$ or $i' \in \mathbb{N}'$.

Note that the type σ in $T = (\sigma, \bar T)$ is used for imposing a restriction on labels $i' \in \mathbb{N}'$ in the tree $\bar T$ and for treating the tree as an SP as described above.

If $\bar T$ in $T = (\sigma, \bar T)$ is *finite* then T can be identified with an FSP.

In order to see that the sequential procedures form an applicative structure, the best approach will be to use the constructions of composition and application for the finite procedures, and then extend them to the non-finite case. To this end, we need the intensional ordering of the sequential procedures as defined below.

Definition 2.26. For each type σ we will define the ordering $\prec_\sigma$ on the set of finite sequential procedures (dropping the index when appropriate) as follows:

If $\sigma = \iota$, we let $\prec_\sigma = \sqsubseteq_\sigma$.
If $\sigma = \vec\tau \to \iota$, we let $C^\sigma_\perp \prec T$ for any $T \in \mathrm{FSP}_\sigma$.
If $\sigma = \vec\tau \to \iota$,

$$T(\vec x) = T_k(\vec x) \textbf{ if } x_i(R_1(\vec x), \ldots, R_r(\vec x)) = k \in K$$

and

$$S(\vec x) = S_l(\vec x) \textbf{ if } x_i(Q_1(\vec x), \ldots, Q_r(\vec x)) = l \in L$$

we let $T \prec S$ if $K \subseteq L$, $T_k \prec S_k$ for $k \in K$ and $R_j \prec Q_j$ for all $j = 1, \ldots, r$.

We extend the $\prec$-relation to $\mathcal{L}$ adding the extra rule

$M_1N_1 \prec M_2N_2$ when $M_1 \prec M_2$ and $N_1 \prec N_2$.

Using the co-inductive analogue to the definition of $\prec$, we extend $\prec$ to the intensional ordering $\prec^*$ on the general sequential procedures. When we consider a sequential procedure T as a typed tree $(\sigma, \bar{T})$, the relation $T \prec^* S$ means just the set theoretical inclusion $\bar{T} \subseteq \bar{S}$ for trees understood as partial maps $\mathbb{N}^* \to \mathbb{N} \cup \mathbb{N}'$.

We make the following observations for FSP's:

Lemma 2.27. (a) *If $T \prec T'$ then $T \sqsubseteq T'$.*
(b) *If $M_1 \prec M_2$ are terms in $\mathcal{L}$ of type ι and M_1 evaluates to a number, then M_2 evaluates to the same number via a path of the same length, where each term N_1 in the evaluation path of M_1 is $\prec$-below the term N_2 at the corresponding location in the evaluation path for M_2, and where there is a complete matching between occurrences of [and] in the two paths.*
(c) *If $M_1 \prec M_2$ are two terms in $\mathcal{L}$ that are rewritten to the* FSP*'s T_1 and T_2 by the process in the proof of Theorem 2.8, then $T_1 \prec T_2$. More generally, the composition operator $T \circ S$ is (trivially) monotonic in T and S with respect to $\prec$.*

Proof. (a) is proved by a simple induction on the rank of T and (b) is proved by a simple induction on the location in the evaluation path of M_1. Every step in the evaluation of M_1 will be copied as a step in the evaluation of M_2 preserving $\prec$.

In order to prove (c) we must extent the definition of $\prec$ further to the set of intermediate expressions in the rewriting process involving the composition operator, and then use induction on the number of applications of the rewriting rules that are used.

This ends the proof. □

Lemma 2.27 shows that both evaluation and composition $\circ$ are monotonic with respect to $\prec$. Since the sequential procedures can be viewed as the limits of $\prec$-chains of finite sequential procedures, our definitions of evaluation path and composition $\circ$ are directly transferred to SP.

In particular, this means that $\{\mathrm{SP}_\sigma\}_{\sigma \text{ type}}$ can be organized to an applicative structure (that is not extensional). Note that the rewriting procedure for composite terms with general sequential procedures do not terminate in the sense of taking only finitely many steps. Composition of sequential procedures is technically defined as a limit of compositions of the finite approximations. Since we will consider mainly evaluation paths, we do not need to go into detail about how to extend $\circ$.

Given the co-inductive set of sequential procedures of type σ we extend this to a set of *general applicative terms* by

Definition 2.28. 1. A sequential procedure of type σ is a general applicative term of type σ.
2. If M is a general applicative term of type $\sigma \to \tau$ and N is a general applicative term of type σ, then (MN) is a general applicative term of type τ.

Clearly every general applicative term is of the form

$$M = TN_1 \cdots N_t$$

where T is a sequential procedure and each N_i is a general applicative term.

Definition 2.29. We define the relation $M \vdash a$ between general applicative terms M of type ι and natural numbers a by induction as follows:

1. If $M = TN_1 \cdots N_t$ and $T = C^\sigma_a$ then
$$M \vdash a.$$

1. If
- $M = TN_1 \cdots N_t$
- $T = (i; R_1, \ldots, R_s; K; \{T_k\}_{k \in K})$
- $N_i(\vec{R_1N}, \ldots, \vec{R_sN}) \vdash k$ for some $k \in K$
- $T_k\vec{N} \vdash a$

then
$$M \vdash a.$$

By recursion on this inductive definition, we define the evaluation path when $M \vdash a$ in accordance with Definition 2.11.

It follows by an easy proof by induction that if $M \vdash a$ for some a, then a is unique.

Remark 2.30. The evaluation paths from Definition 2.11 might conclude with the value $\perp$, while this will not be the case here. The inductive definition of the relation

$$M \vdash a$$

is for $a \in \mathbb{N}$ only. It can be proved by a simple induction on the rank of $M \vdash a$ that the evaluation path exists, is unique and finite.

There is a canonical interpretation of sequential procedures and applicative terms over the typed structure of Scott domains. Our operational semantics is adequate for this interpretation, in the sense that if an applicative term of type ι is interpreted as an integer, there will be an evaluation path demonstrating this. We will neither need nor prove this here.

Remark 2.31. Our approach here is very close to the original approach from [14] (see also [15,16]) using strategies.

Definition 2.32. Overloading our use of notation, we extend the definition of the extensional interpretation $\langle T\rangle$ to the set of sequential procedures such that $\langle T\rangle(\langle S\rangle) = \langle TS\rangle$ whenever T is a sequential procedure of type $\tau \to \delta$ and S is a sequential procedure of type τ.

F is a *sequential functional* of type σ if there is a sequential procedure T with $F = \langle T\rangle$.

We identify the finite sequential functionals with sequential functionals by identifying the two interpretations of $\langle\cdot\rangle$.

We let $(\mathbb{Q}_\sigma, \sqsubseteq_\sigma)$ be the partially ordered set of sequential functionals of type σ, and we consider the set of finite sequential functionals of type σ to be a subset of $\mathbb{Q}_\sigma$.

Digression 2.33. (For readers familiar with basic non-standard analysis.)

An alternative approach to the sequential procedures is to consider them as standard parts of *hyperfinite* sequential procedures. The *transfer principle* may then be used to see that phenomena like rewriting and evaluation are easily extended from the finite to the general case.

We may consider the projections P^σ_c (see Definition 2.16) as elements of $\mathbb{Q}_{\sigma\to\sigma}$, and then P^σ_c will map $\mathbb{Q}_\sigma$ onto the finite set of extensional interpretations $\langle T\rangle$ of $T \in \mathrm{FSP}^c_\sigma$. See [17] for details, where FSP's are called *finitary strategies*. Actually, *finite strategies* is a more general concept.

We use this to improve Lemma 2.19 to

Observation 2.34. *Let $X = \{F_1, \dots, F_n\}$ be a finite, bounded set of finite sequential functionals of type σ.*

Then X has a least upper bound in $\mathbb{Q}_\sigma$, and this is finite and coincides with the least upper bound in the set of finite sequential functionals of type σ.

3. Normal form theorems

In Section 2 we introduced the sequential functionals as an ordered set, and we defined them as what is known as the *extensional collapse* of the typed structure of sequential procedures. In the sequel we will be interested in when the pointwise limit of an increasing sequence of sequential functionals will be sequential. For some positive results in this direction we will use a "Normal Form Theorem". This terminology may be misleading, because it is not correct in general that an equivalence class of finite sequential procedures will contain elements on what we will call "normal form".

Definition 3.1. A finite sequential procedure T is on *normal form* if it is either a constant, or if it is of the form

$$T(\vec{x}) = T_k(\vec{x}) \textbf{ if } x_i(R_1, \dots, R_r) = k \in K$$

where $R_1, \dots, R_r$ are not depending on $\vec{x}$, and each T_k is on normal form.

Example 3.2. We define the finite sequential functional H of type

$$(\iota, \iota \to \iota) \to \iota$$

as follows:

$$H_L(f) = 0 \textbf{ if } f(0, \bot) = 0$$
$$H_R(f) = 0 \textbf{ if } f(\bot, 0) = 0$$
$$H(f) = 0 \textbf{ if } f(H_L(f), H_R(f)) = 0.$$

There can be no T on normal form that is equivalent to H, since then for any f, the evaluation path of Tf would have to consist of a sequence of evaluation paths for fa_ib_i where a_i and b_i only depend on the values of fa_jb_j for $j < i$. But this cannot work for both $f_L = (0, \bot \mapsto 0)$ and $f_R = (\bot, 0 \mapsto 0)$, since termination for both these cases forces us to have $a_i = b_i = 0$ for all i, while H does not terminate on input $0, 0 \mapsto 0$.

Example 3.3. Let $\sigma = (\iota \to \iota)^2 \to \iota$ and let G and F be the finite sequential procedures of type σ defined by

$$G(f, g) = 0 \textbf{ if } g(\bot) = 0,$$
$$F(f, g) = 0 \textbf{ if } f(G(f, g)) = 0.$$

Let f_0 be minimal such that $f_0(0) = 0$.

Then $\{(C_0, C_\bot), (f_0, C_0)\}$ is the base for F. There is no way F can be brought to a normal form.

Lemma 3.4. *If f is a sequential functional of type σ where σ has level* 1, then any sequential procedure defining f is on normal form.

Proof. Let $\sigma = \iota^n \to \iota$, and let T be a finite sequential procedure of type σ.

By induction on the rank of T, we prove that T is on normal form.

If T is a constant, T is on normal form by definition.

If T is not a constant, it is of the form

$$T(\vec{x}) = T_k(\vec{x}) \textbf{ if } x_i = k \in K.$$

Since the sequence $R_1, \ldots, R_r$ in this case is the empty sequence, it is in particular independent of the input.

Our next result is well known by workers in the field, and must be considered as a kind of "folklore"-result.

For an FSP

$$T(\vec{x}) = T_k(\vec{x}) \textbf{ if } x_i(\vec{R}(\vec{x})) = k \in K$$

we call $x_i(\vec{R}(\vec{x})) = ?$ the *first query* asked by T. Then all *queries (potentially) asked by* T are the first query asked by T together with (recursively) all queries asked by all T_k. □

Lemma 3.5. *Every* FSP *of type* $(\iota \to \iota) \to \iota$ *is equivalent to an* FSP *on normal form.*

Proof. We prove this by transforming T to an equivalent T^* that is on normal form.

If T is a constant, then it is on normal form, and we let $T^* = T$.

Let

$$T(x) = T_k(x) \textbf{ if } x(R(x)) = k \in K.$$

Now we consider two cases, depending on R:

1. If R is a constant C_a, we let
$$T^*(x) = T^*_k(x) \textbf{ if } x(a) = k \in K.$$
2. If $R(x) = R_l(x)$ **if** $x(S(x)) = l \in L$, we may rewrite the definition of T to a definition where the first query is simpler as follows: Let
$$T'(x) = \begin{cases} (T_k(x) \textbf{ if } x(R_l(x)) = k \in K) & \textbf{if } x(S(x)) = l \in L \\ (T_k(x) \textbf{ if } x(\bot) = k \in K) & \textbf{if } x(S(x)) = m \in K \setminus L \end{cases}$$
We leave the definition of T' as an FSP in the standard syntax for the reader. If x is not a constant, we can derive that $T(x) = T'(x)$, and if $x = C_a$ we have that $T(x) = T'(x) = T_a(x)$ when $a \in K$ and $T(x) = T'(x) = \bot$ otherwise. It follows that T and T' are equivalent.

   Note that the first query $x(R(x)) = ?$ asked by T was replaced by simpler initial queries $x(S(x)) = ?$, $x(R_l(x)) = ?$ and $x(\bot) = ?$ asked by the equivalent FSP T'. They are simpler e.g. by comparing the size of the tree representation of R vs. S, R_l and $C_\bot$.

Continuing with similar simplifications of the first queries asked by T_k, $(T_k)_n$, etc., we will simplify all non-constant queries asked by T in the resulting equivalent version which we call $\dot{T}$. Iterating this procedure $T \mapsto \dot{T}$ to get $\ddot{T}$, etc., we will eventually halt on some T^* equivalent to T and asking only the simplest queries of the required form $x(C_a(x)) = ?$ or $x(a) = ?$. (That all the above iterations are finite is trivial.)

This ends the proof. □

Examples 3.2 and 3.3 show that this is as far as we will be able go in proving a general normal form theorem.

The construction in the proof of Lemma 3.5 can easily be extended to finite sequential procedures of type $\sigma = (\iota \to \iota)^s \to \iota$ for all $s \in \mathbb{N}$, and the proof that the rewriting process terminates is valid, but if $s > 1$ we cannot prove that the rewriting preserves the extensional interpretations. We do however have

Corollary 3.6 (*Of Proof*)**.** *Let* T *be a finite sequential procedure of type*

$$(\iota \to \iota)^s \to \iota$$

where $s \in \mathbb{N}$.

Then there is a finite sequential procedure $H \sqsubseteq T$ *on normal form such that for every* s*-sequence* $\vec{f}$ *of unary functions such that* f_i *is not a constant for any* i *we have that*

$$T(\vec{f}) = H(\vec{f}).$$

The details are left for the reader.

4. Sequential functionals and directed complete partial orderings

A partial ordering $(X, \leq)$ is a *dcpo*, a *directed complete partial ordering*, if every directed subset of X has a least upper bound. For many years, it was an open problem if the sequential functionals of any type form a *dcpo*. This was solved negatively by Normann [11], who proved that the sequential functionals of pure type 3 is not a *dcpo*.

In this section we will prove that there even is a type σ at level 2 where $(\mathbb{Q}_\sigma, \sqsubseteq_\sigma)$ (see Definition 2.32) is not a *dcpo*. Avoiding too much notation, we will simply say that $\mathbb{Q}_\sigma$ is a *dcpo* when $(\mathbb{Q}_\sigma, \sqsubseteq_\sigma)$ is a dcpo.

Our proof will contain the key method used when we address some other problems related to the orderings $\sqsubseteq_\sigma$, so this section is also a preparation for the sections to come.

By construction, each sequential functional will be the least upper bound of a directed set of finite sequential functionals, and this least upper bound will actually be the pointwise limit in the style of domain theory. Since any finite bounded set of finite sequential functionals will have a least upper bound, and there are only countably many finite sequential functionals of a fixed type, any directed set of finite sequential functionals will contain a cofinal sequence. Thus, in order to prove that some $\mathbb{Q}_\sigma$ is a *dcpo*, it is sufficient to show that whenever we have an increasing sequence $\{F_n\}_{n\in\mathbb{N}}$ of finite sequential functionals in $\mathbb{Q}_\sigma$, the pointwise limit is sequential. However, in general the closure of $\mathbb{Q}_\sigma$ under pointwise limits is impossible, and this is a crucial point in developing an appropriate domain theory for sequentiality.

Remark 4.1. Sazonov [17,18] gave a direct construction of the sequential functionals, using strategies, and he defined the concept of *natural least upper bound* as being a least upper bound in $(\mathbb{Q}_\sigma, \sqsubseteq_\sigma)$ that is also the pointwise least upper bound. We can define (alternatively) that an ascending chain (x_n) in σ has natural (pointwise) supremum y iff

$$\forall f : \sigma \to \iota.f(y) \neq \bot \iff (\exists n : \mathbb{N}.f(x_n) \neq \bot).$$

This was pointed out by the referee. By *relativizing* the usual notions of domain theory to this kind of limit, it was shown in [17,18] that $(\mathbb{Q}_\sigma, \sqsubseteq_\sigma)$ are "natural Scott domains" with "natural compacts" that are exactly the finite sequential functionals defined above by FSP's, and that the corresponding version of "natural Scott topology" makes them also f-spaces in the sense of Ershov [3] with all functionals continuous in this topology or, equivalently relative to the above natural limits.

On the other hand, Escardó and Ho [4] use the primitive type $\bar{\omega}$ order-isomorphic to the ordinal $\omega + 1$ as an inessential extension of PCF_Ω and define *rational* ascending chains of the type σ as sequential functionals $c : \bar{\omega} \to \sigma$. All such rational chains have the pointwise lub $c(\infty)$ where ∞ is the maximum (limit) element of $\bar{\omega}$. All sequential functionals prove to be continuous by preserving these rational lubs. Then domains $U_f = \{x \in \mathbb{Q}_\sigma \mid f(x) \neq \bot\}$ of any functionals $f : \sigma \to \iota$ are called *rational open* sets of the type σ. They constitute a *rational topology* in the sense that they are closed under finite intersections and "rational" unions. Open sets in this sense are actually "rationally Scott open". *Rationally finite* elements are defined as usual, but in terms of lubs of rational chains. Then every element of any type is the lub of a rational chain of rationally finite elements. Thereby, we have some analogy with T_0-spaces (and also with the ordinary Scott domains), but the "rational topology" is not a topology in the standard sense. For example, due to absence parallel features in $PCF_\Omega + \bar{\omega}$, it follows from [4] that we cannot guarantee that open sets are closed under finite union (although closed under unions of rational ascending chains of "open" sets presented by their characteristic sequential functionals).

Anyway, there are some natural possibilities to consider fully abstract models for PCF (or versions of PCF) in a domain-theoretic and topological style, although only in somewhat non-classical versions of Scott domains either as (incomplete) topological f-spaces or in terms of rational topologies where the latter are not topological spaces in a standard sense.

Nevertheless, since these are non-dcpos, there are some interesting anomalies which we intend to study here in a regular way to get a more complete understanding on the order-theoretical nature of $(\mathbb{Q}_\sigma, \sqsubseteq_\sigma)$. It was conjectured in [17,18] that for some σ, there will be least upper bounds in $\mathbb{Q}_\sigma$ that are not natural and that we even may find finite sequential functionals that are nontrivial least upper bounds of directed sets of sequential functionals. Sazonov also conjectured that there will be sequential functionals of type $\sigma \to \iota$ that will not commute with least upper bounds of directed sets in $\mathbb{Q}_\sigma$ and that we even might find finitary sequential functionals that are not continuous in this sense.

Observe that $\mathbb{Q}_\iota = \mathbb{N}_\bot$ is a *dcpo*. It is also well-known and sufficiently straightforward fact that $\mathbb{Q}_{\iota^s\to\iota}$ are *dcpo*'s for all $s \geq 0$.

Our first result in this section is a theorem that is known from the literature, but seems to belong to the "folklore" based on a proof idea that does not fully work. We prove that the sequential functionals of pure type 2 form a *dcpo*.

$\mathbb{Q}_{(\iota\to\iota)\to\iota}$ is a dcpo

The argument is based on normal representation, which we know does not exist in full generality for sequential functionals of non-pure type 2. The "folklore" proof only works when we assume that we only deal with strict functions, or, if the limit function F terminates on a constant function, then it terminates on a strict, finite approximation to that constant. We give the proof in pedantic details, details that may be useful in future generalizations of the theorem. We have no reference to a published version of a proof for the full theorem, and this is also a reason for including one here. First, we will consider a concept of critical elements. This is given in a more general form than we will really need in this paper. We believe that it is interesting in itself and that it will probably be used for similar applications in future proofs.

Critical arguments of sequential functionals of level 2

Throughout this subsection till Theorem 4.9, we will let

$$\sigma = \tau_1, \dots, \tau_t \to \iota = \vec{\tau} \to \iota$$

where $\tau_i = \iota^{s_i} \to \iota$ for some $s_i \geq 0$.

Throughout the note, we will let $\vec{f}, \vec{g} \in \mathrm{FSP}_{\vec{\tau}}$ and we will let $F, G, H \in \mathrm{FSP}_\sigma$, also when indices are used. More generally, we can consider that F, G, H are any monotonic maps $\mathbb{Q}_{\vec{\tau}} \to \mathbb{Q}_\iota$. We will always let $1 \leq i \leq t$, $n \in \mathbb{N}$ and

$$\vec{f} = (\dots, f_i, \dots).$$

F is a *constant* if $F(C_\perp^{\iota^{s_1} \to \iota}, \dots, C_\perp^{\iota^{s_t} \to \iota}) \neq \perp$.

Definition 4.2. Let $\vec{a}$ be of type ι^{i_s}.

We say that $(i, \vec{a})$ is *critical* for F relative to $\vec{f}$ if $\vec{a}$ is minimal such that for all $\vec{g}$ with $\vec{f} \sqsubseteq \vec{g}$ where $F(\vec{g}) \in \mathbb{N}$ we have that $g_i(\vec{a}) \in \mathbb{N}$ while $f_i(\vec{a}) = \perp$.

We let

$$Cr_{\vec{f}}(F)$$

denote the set of $(i, \vec{a})$ that are critical for F relative to $\vec{f}$.

We observe that if $F(\vec{f}) \in \mathbb{N}$, then $Cr_{\vec{f}}(F) = \emptyset$.

Lemma 4.3. *If there is no $\vec{g} \sqsupseteq \vec{f}$ such that $F(\vec{g}) \in \mathbb{N}$, then only objects of the form $(i, \vec{\perp})$ can serve as elements of $Cr_{\vec{f}}(F)$, if there are any. This holds exactly for those i for which $f_i(\vec{\perp}) = \perp$.*

Proof. Assume that $(i, \vec{a}) \in Cr_{\vec{f}}(F)$. Then, by definition, $f_i(\vec{a}) = f_i(\vec{\perp}) = \perp$ and, by definition, the premise and the minimality of $\vec{a}$ we must have that $(i, \vec{\perp}) \in Cr_{\vec{f}}(F)$ and $\vec{a} = \vec{\perp}$. □

Lemma 4.4 (*Finiteness*)**.** *For finite F (and $\vec{f}$), $Cr_{\vec{f}}(F)$ is a finite set.*

Proof. If $F(\vec{f}) \in \mathbb{N}$ or if there is no extension $\vec{g}$ of $\vec{f}$ such that $F(\vec{g}) \in \mathbb{N}$, we have already observed this. □

Now let $(i, \vec{a}) \in Cr_{\vec{f}}(F)$ and assume that there is at least one $\vec{g} \sqsupseteq \vec{f}$ such that $F(\vec{g}) \in \mathbb{N}$.

Let $c \in \mathbb{N}$ be so large that F is a fixed point for P_c^σ and such that each f_i is a fixed point for $P_c^{\tau_i}$.

Then

$$F(\vec{g}) = P_c^\sigma(F)(\vec{g}) = F(P_c^{\vec{\tau}}(\vec{g})).$$

Moreover

$$\vec{f} = P_c^{\vec{\tau}}(\vec{f}) \sqsubseteq P_c^{\vec{\tau}}(\vec{g}).$$

It follows, by the definition of critical elements, that

$$P_c^{\tau_i}(g_i)(\vec{a}) \in \mathbb{N}.$$

Then there is a $\vec{b}_{g_i}$ in the base for $P_c^{\tau_i}(g_i)$ with $\vec{b}_{g_i} \sqsubseteq \vec{a}$.

The minimality requirement on $\vec{a}$ implies that

$$\vec{a} = \bigsqcup\{\vec{b}_{g_i} \mid \vec{f} \sqsubseteq \vec{g} \wedge F(\vec{g}) \in \mathbb{N}\}.$$

Since there are only finitely many possible least upper bounds of sets of base elements for functions in $\mathrm{FSP}_{\tau_i}^c$, the lemma follows.

Lemma 4.5 (*Monotonicity*)**.** (a) *If $F \sqsubseteq G$ and $(i, \vec{a}) \in Cr_{\vec{f}}(G)$, then there is a $\vec{b} \sqsubseteq \vec{a}$ such that $(i, \vec{b}) \in Cr_{\vec{f}}(F)$.*
(b) *If $F \sqsubseteq G \sqsubseteq H$ and $(i, \vec{a}) \in Cr_{\vec{f}}(F)$, but $(i, \vec{a}) \notin Cr_{\vec{f}}(G)$ then $(i, \vec{a}) \notin Cr_{\vec{f}}(H)$.*

Proof. (a) Let $\vec{f} \sqsubseteq \vec{g}$ and assume that $F(\vec{g}) \in \mathbb{N}$.

Then $G(\vec{g}) \in \mathbb{N}$ since $F \sqsubseteq G$, so $g_i(\vec{a}) \in \mathbb{N}$.

Then the minimality requirement on $Cr_{\vec{f}}(F)$ gives us that there is a $\vec{b} \sqsubseteq \vec{a}$ in $Cr_{\vec{f}}(F)$.

(b) If $(i, \vec{a}) \in Cr_{\vec{f}}(H)$ there is, by (a), a $\vec{b} \sqsubseteq \vec{a}$ with $(i, \vec{b}) \in Cr_{\vec{f}}(G)$, and by the assumption, $\vec{b}$ is strictly below $\vec{a}$.

By (a) again, there is a $\vec{c} \sqsubseteq \vec{b}$ such that $(i, \vec{c}) \in Cr_{\vec{f}}(F)$.

But then $\vec{c}$ is strictly below $\vec{a}$, contradicting the minimality assumption on $\vec{a}$. Thus the assumption that $(i, \vec{a}) \in Cr_{\vec{f}}(H)$ leads to a contradiction. □

Lemma 4.6 (*Stabilization*). *Let $F_0 \sqsubseteq F_1 \sqsubseteq \cdots$ be an increasing sequence of finite sequential functionals of the type $\sigma = \vec{\tau} \to \iota$, $F : \mathbb{Q}_{\vec{\tau}} \to \mathbb{Q}_\iota$ be its pointwise least upper bound, and let $\vec{f}$ be given.*

Then there is an $n_0 \in \mathbb{N}$ such that for all $m \geq n_0$ we have that

$$Cr_{\vec{f}}(F_m) = Cr_{\vec{f}}(F).$$

Proof. Let

$$X = \{(i, \vec{a}) \mid \exists n (i, \vec{a}) \in Cr_{\vec{f}}(F_n)\}.$$

We will show that X is finite by obtaining a contradiction from

Assumption. X is infinite.

Let i be such that

$$X_i = \{\vec{a} \mid (i, \vec{a}) \in X\}$$

is infinite.

For each $\vec{a} \in X_i$, let $\#(\vec{a})$ be the number of coordinates j such that $a_j \neq \bot$, and for each k, let

$$X_{i,k} = \{\vec{a} \in X_i \mid \#(\vec{a}) \leq k\}.$$

Let k be minimal such that $X_{i,k}$ is infinite.

For each $\vec{a} \in X_{i,k}$, let $n(\vec{a})$ be the least n such that $(i, \vec{a}) \in Cr_{\vec{f}}(F_n)$.

By Lemma 4.4 we have that $\{n(\vec{a}) \mid \vec{a} \in X_{i,k}\}$ is unbounded.

For each $\vec{a} \in X_{i,k}$ with $n(\vec{a}) > 0$, there is, by Lemma 4.5(a), a $\vec{a}' \sqsubseteq \vec{a}$ such that $(i, \vec{a}') \in Cr_{\vec{f}}(F_{n(\vec{a})-1})$, and by definition of $n(\vec{a})$, $\vec{a}'$ is strictly below $\vec{a}$.

It follows from Lemma 4.5(b) that if $0 < n(\vec{a}) < n(\vec{b})$ where both $\vec{a}$ and $\vec{b}$ are in $X_{i,k}$, then $\vec{a}' \neq \vec{b}'$.

Since $\vec{a}'$ is strictly below $\vec{a}$, we have that $\#(\vec{a}') < \#(\vec{a})$.

Altogether this shows that there are infinitely many $\vec{a}' \in X_i$ with $\#(\vec{a}') < k$, contradicting the choice of k.

This shows that our assumption leads to a contradiction, and that X is finite.

If we now let C_n be the upper cone of $Cr_{\vec{f}}(F_n)$ in X, it follows from Lemma 4.5(a), that $\{C_n\}_{n\in\mathbb{N}}$ is decreasing with respect to inclusion as n increases, and must stabilize since X is finite.

Since $Cr_{\vec{f}}(F_n)$ is the set of minimal elements in C_n, stabilization of the sequence $Cr_{\vec{f}}(F_n)$ from some n_0 follows.

To show $Cr_{\vec{f}}(F_{n_0}) \subseteq Cr_{\vec{f}}(F)$, first note that if $(i, \vec{c}) \in Cr_{\vec{f}}(F_{n_0})$ then $f_i(\vec{c}) = \bot$ and, by stabilization, for all $\vec{g} \sqsupseteq \vec{f}$ and $m \geq n_0$, $F_m(\vec{g}) \in \mathbb{N}$ implies $g_i(\vec{c}) \in \mathbb{N}$, that is $F(\vec{g}) \in \mathbb{N}$ implies $g_i(\vec{c}) \in \mathbb{N}$. This means that we almost have $(i, \vec{c}) \in Cr_{\vec{f}}(F)$, except we only need minimality of $\vec{c}$. Assume $\vec{b} \sqsubseteq \vec{c}$ also satisfies this condition: $f_i(\vec{b}) = \bot$ and for all $\vec{g} \sqsupseteq \vec{f}$, $F(\vec{g}) \in \mathbb{N}$ implies $g_i(\vec{b}) \in \mathbb{N}$. Then also $F_{n_0}(\vec{g}) \in \mathbb{N}$ implies $g_i(\vec{b}) \in \mathbb{N}$, and we have $\vec{b} \in Cr_{\vec{f}}(F_{n_0})$ which implies $\vec{c} = \vec{b}$.

For the converse inclusion, assume $(i, \vec{b}) \in Cr_{\vec{f}}(F)$. Then by monotonicity Lemma 4.5(a), $(i, \vec{b})$ extends some $(i, \vec{c}) \in Cr_{\vec{f}}(F_{n_0})$, and by already shown inclusion we have $(i, \vec{c}) \in Cr_{\vec{f}}(F)$. By minimality, $\vec{c} = \vec{b}$. All of this proves $Cr_{\vec{f}}(F_{n_0}) = Cr_{\vec{f}}(F)$. This ends the proof. □

Lemma 4.7 (*Existence of Critical*). *If a finite sequential F has a representation on normal form, $F(\vec{f}) = \bot$ and there is at least one i such that $f_i(\vec{\bot}) = \bot$, then $Cr_{\vec{f}}(F) \neq \emptyset$.*

Proof. If there is no $\vec{g} \sqsupseteq \vec{f}$ such that $F(\vec{g}) \in \mathbb{N}$, this follows from Lemma 4.3. □

If there is an extension $\vec{g}$ of $\vec{f}$ with $F(\vec{g}) \in \mathbb{N}$ there will be a first query $g_j(\vec{a}) = ?$ in the evaluation of $F(\vec{g})$ following the normal strategy that is answered by g_j, but unanswered by f_j in the case of computing $F(\vec{f})$, and this will be independent of $\vec{g} \sqsupseteq \vec{f}$. Then there is a $\vec{b} \sqsubseteq \vec{a}$ such that $(j, \vec{b}) \in Cr_{\vec{f}}(F)$.

Corollary 4.8. *For the pointwise lub F as in Lemma 4.6 and any list of finite arguments $\vec{f}$, $Cr_{\vec{f}}(F)$ is finite. If $F(\vec{f}) = \bot$ and not all $\vec{f}$ are constants then $Cr_{\vec{f}}(F) \neq \emptyset$.*

Proof. Use Lemmas 4.4, 4.6 and 4.7. □

Now we have to restrict the list of level 1 arguments $\vec{f}$ just to one unary function f. Thus the above considerations on critical elements will not be used in this paper in their full generality. To simplify denotations, we will consider that $Cr_f(F)$ consists of some elements $a \in \mathbb{Q}_\iota$ rather than pairs $(1, a)$ according to the above agreement.

Theorem 4.9. *Let $\sigma = (\iota \to \iota) \to \iota$.*

Let $\{F_n\}_{n\in\mathbb{N}}$ be an $\sqsubseteq_\sigma$ increasing sequence of finite sequential functionals in $\mathbb{Q}_\sigma$.

Then the pointwise least upper bound

$$F(f) = \bigsqcup_{n\in\mathbb{N}} F_n(f)$$

is sequential.

Proof. Before describing the strategy for F itself, we will do so for some approximation $G \sqsubseteq F$.

Normal strategy for G. We will now use the above Lemmas to find a normal sequential strategy for some

$$G(f) \sqsubseteq F(f) = \bigsqcup_{n\in\mathbb{N}} F_n(f)$$

where we assume that F_n of the type $(\iota \to \iota) \to \iota$ is any increasing sequence of finite functionals having normal representations. Thereby, *for non-constant* f $(f(\bot) = \bot)$ *the inclusion will be actually equality*.

We may also assume that

$(*_\bot)$ $\quad F(\bot) = \bot$, and $\bot \notin Cr_\bot(F)$.

Indeed, otherwise either we can immediately let $G(f) = F(\bot) \neq \bot$ so that G is in fact a constant functional, or the sequential procedure for computing of $F(f)$, and hence of $G(f)$, consists in asking only one query $f(\bot) = ?$. If $a \in \mathbb{N}$ is the answer, then $f = C_a$ and the strategy outputs the correct result $G(C_a) = F(C_a)$ (if defined).

Thus, under the above assumption, let any f be given, and let $f_0 = \bot \sqsubseteq f$.

By Corollary 4.8 the set $Cr_{f_0}(F)$ of elements that are critical for F relative to f_0 is finite and nonempty. Let $a_0 \in \mathbb{N}$ be critical for F relative to f_0.

If $f(a_0) = \bot$ we conclude that $G(f) = \bot$.

Otherwise, let f_1 be f restricted to $\{a_0\}$.

Assume, by induction, that we have constructed non-constant approximations

$$\bot = f_0 \sqsubseteq f_1 \sqsubseteq \cdots \sqsubseteq f_k \sqsubseteq f, \quad f_i(\bot) = \bot.$$

Further the computation will go similarly to the previous part. Thereby we will need $f_i(\bot) = \bot$ for Corollary 4.8 to be applicable to all f_i.

As for $f_0 = \bot$, we may again assume that

$(*_{f_k})$ $\quad F(f_k) = \bot$ and $\bot \notin Cr_{f_k}(F)$.

Indeed, otherwise either we can immediately let $G(f) = F(f_k) \neq \bot$, and the computation is finished, or the rest of the sequential procedure for computing $G(f)$ consists in asking only one query $f(\bot) = ?$. If $a \in \mathbb{N}$ is the answer, then ($f_k \sqsubseteq f = C_a$ and) the strategy outputs the correct result $G(C_a) = F(C_a)$ (if defined).

As in the base case, the assumption $(*_{f_k})$ and Corollary 4.8 imply that the sets $Cr_{f_k}(F)$ of elements that are critical for F relative to f_k are finite, nonempty and there should exist some $a_k \neq \bot$ in $Cr_{f_k}(F)$.

The strategy continues the process as follows.

If $f(a_k) = \bot$, we know that $F(f) = \bot$, so the strategy has no continuation.

If not, we let f_{k+1} be the extension of f_k such that $f_{k+1}(a_k) = f(a_k)$.

It follows from the fact that all $a_i \neq \bot$ that $f_{k+1}(\bot) = \bot$, as required for our inductive argument, and this finishes the induction step of our construction of the sequence f_i.

Note that a sequence f_k generated this way is infinite if for each constructed f_k, $(*_{f_k})$ holds (so that a_k exists) and $f(a_k) \neq \bot$, thus allowing to define f_{k+1}. Otherwise (if either $(*_{f_k})$ fails, or it holds, but $f(a_k) \neq \bot$), it is finite.

Anyway, the process we have described clearly is an infinitary, sequential process defining a functional $G \sqsubseteq F$. In fact, some steps of this procedure are not effective, e.g., such as checking the first part of $(*_{f_k})$, but effectivity is not required. Also in the description of this procedure we referred to a particular f. Formally, the description could (and in fact must) be done in terms of questions asked by this procedure and reaction to any possible answers, independently of whether these are generated by any f given in advance or not. For an input f, the process may terminate with a positive (in $\mathbb{N}$) or a negative ($\bot$) conclusion, or it may go on for ever. It remains to prove that for any f such that $f(\bot) = \bot$ and $F(f) \in \mathbb{N}$, this process will terminate.

We can evidently assume that f is finitary. This process involves constructing a strongly increasing sequence f_k of sequential approximations to f which therefore must be finite. Therefore for some k either $(*_{f_k})$ fails, or it holds, but $f(a_k) = \bot$. We know from the above considerations that in any of these cases the strategy computes the correct result, if defined at all.

Thus, the correct normal strategy for $G(f)$ is constructed.

Non-normal strategy for F. Now, let us construct a strategy for $F(f)$ on the base of the strategies for $G(f)$ and a_f to be defined below. The strategy for $F(f)$ should also work correctly on constant functions $f = C_k$.

It starts as $G(f)$, and if $(*_\bot)$ fails then we know that it outputs the required value $F(f)$.

If $(*_\bot)$ holds, it asks $f(a_0) = ?$, and if the answer is k, there will be two cases.

1. *If* $F(h) \neq \bot$ *for some (finite) partial constant* $h \sqsubseteq C_k$ then the strategy for $G(f)$ continues its work. Even if $f = C_k$, $G(f) = G(h) = F(h) \neq \bot$ will be correctly computed by this strategy.

2. *If $F(h) = \bot$ for all partial constants $h \sqsubseteq C_k$* then another auxiliary strategy based on G will decide if f is the constant C_k or if for some $b, f(b) \neq k$. This decision should be guaranteed to work in the case $F(f) \neq \bot$. Otherwise the decision procedure should diverge. In detail, the auxiliary strategy works as follows:

   We proceed computing the values $f(a_0), f(a_1), \ldots$, again by using the strategy for G generating the sequence $a_0, a_1, \ldots$, either forever or until the sequence a_i breaks as was described above, or until we find some value $f(a_n) \neq \bot$ which is different from k. In the last case, we let this a_n be a_f. We let $a_f = \bot$ if the last case never holds. This is actually another sequential procedure satisfying the property:

   (*) If the limit functional $F(f)$ terminates and $f \not\sqsubseteq C_k$, then $G(f)$ and hence a_f will terminate, and thereby a_f, witnessing that $f \not\sqsubseteq C_k$, will be detected from f.

   Indeed, $G(f)$ must terminate with the same result as $F(f)$ because f is not a constant. If, however, a_f does not terminate then $G(f)$ has terminated without detecting any difference between f and C_k. That is, for some partial constant $h \sqsubseteq C_k$, $h \sqsubseteq f$ and both $G(h)$ and $F(h)$ will terminate, contrary to our initial assumption of the case 2.

   Saying that about the auxiliary procedure for a_f, our main procedure $F(f)$ (for any f) in the case 2 considered moves to the "non-normal" part and asks for $f(a_f)$. Three cases are possible:

   (a) If $f = C_k$ we get the answer $f(a_f) = k$.
   (b) If $f \not\sqsubseteq C_k$ and $F(f) \neq \bot$, then (*) implies that a_f is defined and $f(a_f) \neq k$.
   (c) If $f \sqsubseteq C_k$ and $f \neq C_k$ then $a_f = \bot$ and hence $f(a_f) = \bot$. But in this case also $F(f) = \bot$ by our initial assumption of the case 2. So, this diverging possibility should not bother us.

   Then, when we have separated between $f = C_k$ and $f \not\sqsubseteq C_k$ (knowing that $f(a_0) = k$), by asking the question $f(a_f) = ?$, we just give the right answer $F(C_k)$ in the first case, and use the normal strategy for $G(f)$ in the second case when f is definitely not a constant.

This ends the proof of the theorem. □

This theorem gives us that $\mathbb{Q}_\sigma$ is a *dcpo*, where $\sigma = (\iota \to \iota) \to \iota$. Our next result shows that the above theorem cannot be generalized to full generality.

Types σ for which $\mathbb{Q}_\sigma$ is not a dcpo

Proposition 4.10. *Let $\sigma = (\iota, \iota \to \iota) \to \iota$.*
Then $\mathbb{Q}_\sigma$ is not a dcpo.

Proof. We will give an explicit construction of an increasing sequence of finite sequential functionals of type σ that is not bounded by any sequential functional, and later we will refer to this construction.

We let i, j, k, l etc. denote integers, f, g etc. denote functions of type $\iota, \iota \to \iota$ and F, G etc. denote sequential functionals of type σ.

Construction

Let $f_0(a, b) = 0$ if $a = 0, f_0(a, b) = \bot$ otherwise.
If $m > 0$, let $f_m(a, b) = 1$ if $b = m$ or if $b < m$ and $a = 1$ and let $f_m(a, b) = \bot$ otherwise.

Claim 1. For each n there is a sequential G_n such that for all m

$$G_n(f_m) \in \mathbb{N} \Leftrightarrow m \leq n,$$

and such that $G_n(f_0) = 0$ while $G_n(f_m) = 1$ for $0 < m \leq n$.

Proof of Claim 1. Let $G_0(f) = 0$ **if** $f(0, \bot) = 0$.
For $n > 0$, let

$$G_n(f) = f(\cdots f(f(G_{n-1}(f), n), n-1), \ldots, 0)$$

provided all intermediate values are either 0 or 1.

We prove by induction on n that G_n satisfies the requirement.

Clearly $G_0(f_0) = 0$ and $G_0(f_m) = \bot$ for $m > 0$, since then $f_m(0, \bot) = \bot$.

Now let $n > 0$ and assume that G_{n-1} satisfies the claim. First we see that $G_n(f_0) = 0$. This follows from the induction hypothesis, that $G_{n-1}(f_0) = 0$ and the fact that

$$G_n(f_0) = f_0(\cdots f_0(f_0(0, n), n-1), \ldots, 0).$$

The left hand side arguments will all be evaluated to 0, while the values of the right hand side arguments are of no importance (since $\bot$ suffices there).

Now let $1 \leq m \leq n$. Then the value of $f_m(G_{n-1}(f_m), n)$ is $\bot$ if $m < n$ and 1 if $m = n$. When we then iterate application of f_m using smaller and smaller arguments on the right hand side, we get the value 1 all the way after reaching the argument m, so finally the value of $G_n(f_m) = 1$.

If $m > n$ we use that $G_{n-1}(f_m) = \bot$ and that $f_m(\bot, i) = \bot$ for all $i \leq n$ to see that $G_n(f_m) = \bot$.

This ends the proof of the claim.

Claim 2. Let F be a sequential functional (not necessarily finite) such that $F(f_n) \in \mathbb{N}$ for $n = 0$ and for infinitely many $n > 0$.

Then $F(\bot_{\iota,\iota\to\iota}) \in \mathbb{N}$.

Proof of Claim 2. We assume that F is given by a sequential strategy M, and prove the claim by induction on the length of the evaluation path for $M(f_0)$.

Either M is a constant, or M is of the form

$$M(f) = M_k(f) \textbf{ if } f(L(f), R(f)) = k \in K$$

where K is not necessarily finite and each L, R and M_k are sequential.

Since $M(f_0) \in \mathbb{N}$ we must have that $f_0(L(f_0), R(f_0)) \in \mathbb{N}$ which actually means that $L(f_0) = 0$, and then with a shorter evaluation path. If $n > 0$ and $M(f_n) \in \mathbb{N}$ we must have that $f_n(L(f_n), R(f_n)) \in \mathbb{N}$, and then in particular that $R(f_n) \in \mathbb{N}$.

If $1 \leq n, m \leq k$ we have that f_n and f_m are bounded by g_k where $g_k(a, b) = 1$ if $b \leq k$ and $\bot$ otherwise. It follows that if $R(f_n)$ and $R(f_m)$ both are in $\mathbb{N}$, then they are equal. So, for infinitely many n we have that $R(f_n) < n$, and we then must have that $L(f_n) = 1$ for these infinitely many n.

By the induction hypothesis, L must be a constant, contradicting that $L(f_0) = 0$ and $L(f_n) = 1$ for infinitely many n.

Thus M cannot be of the described form, and must be a constant.

This ends the proof of the claim.

By Claim 1 and Lemma 2.19 there will be a minimal sequential functional F_n such that $F_n(f_0) = 0$ and $F_n(f_m) = 1$ for $0 < m \leq n$, and this sequence will be strictly increasing.

By Claim 2, this sequence, that is not bounded by a constant functional, is not bounded by any sequential F.

This ends the proof of the proposition. □

Definition 4.11. Let σ and τ be types.

A *sequential embedding-projection-pair*, or *e-p-pair* for short, between $\mathbb{Q}_\sigma$ and $\mathbb{Q}_\tau$ is a pair (Φ, Ψ) of sequential functionals of types $\sigma \to \tau$ and $\tau \to \sigma$ respectively such that $\Psi(\Phi(x)) = x$ for all $x \in \mathbb{Q}_\sigma$ and $\Phi(\Psi(y)) \sqsubseteq y$ for all y in $\mathbb{Q}_\tau$.

We say that $\sigma \hookrightarrow \tau$ if there is a sequential e-p-pair between $\mathbb{Q}_\sigma$ and $\mathbb{Q}_\tau$.

We extend these definitions to products $\mathbb{Q}_{\vec{\sigma}}$ in the obvious way.

Lemma 4.12. (a) $\hookrightarrow$ *is transitive and reflexive.*

(b) *If* $\vec{\sigma} \hookrightarrow \vec{\tau}$ *then* $\vec{\sigma} \to \iota \hookrightarrow \vec{\tau} \to \iota$.

(c) $\tau_i \hookrightarrow \vec{\tau}$ *when* τ_i *is one of the coordinates in* $\vec{\tau}$.

(d) *If* $n \geq 2$, *then* $\iota^2 \hookrightarrow \iota^n$.

(e) *If the level of* τ *is* ≥ 1, *then* $\iota^2 \hookrightarrow \tau$.

All proofs are both standard and trivial, and are left for the reader. When we prove (b) we use the standard way of lifting an e-p-pair between $\mathbb{Q}_\sigma$ and $\mathbb{Q}_\tau$ to an e-p-pair between $\mathbb{Q}_{\sigma\to\iota}$ and $\mathbb{Q}_{\tau\to\iota}$.

Theorem 4.13. *Let* $\sigma = \tau_1, \ldots, \tau_t \to \iota$ *be a type.*

If at least one τ_i *is of level* ≥ 2 *or of the form* $\iota^s \to \iota$ *where* $s \geq 2$, *then* $(\mathbb{Q}_\sigma, \sqsubseteq_\sigma)$ *is not a dcpo.*

Proof. By assumption there is a τ_i that is not of the form ι or $\iota \to \iota$.

By combining the claims of Lemma 4.12 we see that $(\iota, \iota \to \iota) \to \iota \hookrightarrow \sigma$, so there are sequential functionals Φ of type $((\iota, \iota \to \iota) \to \iota) \to \sigma$ and Ψ of type $\sigma \to ((\iota, \iota \to \iota) \to \iota)$ such that $\Psi \circ \Phi$ is the identity on $\mathbb{Q}_{(\iota,\iota\to\iota)\to\iota}$.

Let $\{F_n\}_{n\in\mathbb{N}}$ be the increasing, unbounded sequence in $\mathbb{Q}_{(\iota,\iota\to\iota)\to\iota}$ constructed in the proof of Proposition 4.10.

Then $\{\Phi(F_n)\}_{n\in\mathbb{N}}$ is an increasing sequence in $\mathbb{Q}_\sigma$. If H is an upper bound for this sequence, $\Psi(H)$ will be an upper bound for $\{F_n\}_{n\in\mathbb{N}}$, so this sequence must be unbounded in $\mathbb{Q}_\sigma$, and $\mathbb{Q}_\sigma$ is not a *dcpo*.

This ends the proof. □

Remark 4.14. There is an interesting gap in our knowledge. If $s > 1$ we actually do not know if

$$\mathbb{Q}_{(\iota\to\iota)^s\to\iota}$$

is a *dcpo* or not.

5. Step functions of finite sequential procedures

In the introductory part of Section 4 we discussed the four conjectures set up by Sazonov [17,18]. In this section we will verify these four conjectures.

Definition 5.1. We say that a sequential functional F of type σ is a *limit functional* if there is a strictly increasing sequence $\{F_n\}_{n\in\mathbb{N}}$ of sequential functionals of type σ such that

$$F = \bigsqcup_{n\in\mathbb{N}} F_n$$

in $\mathbb{Q}_\sigma$.

This concept is closely related to the concept of not being compact for algebraic domains, but since the definition of a *compact* is meant to work for *dcpo*'s and we are dealing with structures that are not *dcpo*'s, we choose this terminology. However note that we do not relate this specific concept of a limit with the topological limit for any topology discussed above in Remark 4.1 because we do not require here from this limit to be either pointwise or rational one, and this is the crucial point for the results of this paper.

Clearly every sequential functional that is not a finite sequential functional will be a limit functional, so our problem will be to decide when a finite sequential functional also will be a limit functional. (But for pointwise or rational limits the answer is always "no".) The existence of an example was Conjecture 2 of Sazonov, and will also verify Conjecture 1.

Limit step functions of level 2

Proposition 5.2. *$C_0^{(\iota,\iota\to\iota)\to\iota}$ is a limit functional.*

Proof. Let $\{f_n\}_{n\in\mathbb{N}}$ be the sequence of functions in $\mathbb{Q}_{\iota,\iota\to\iota}$ constructed in the proof of Proposition 4.10, and for each n let H_n be the minimal sequential functional such that $H_n(f_m) = 0$ when $m \leq n$. These functionals exist by using Claim 1 of the proof of Proposition 4.10. This is a strictly increasing sequence, and by construction it is bounded by $C_0^{(\iota,\iota\to\iota)\to\iota}$. Claim 2 in the same proof gives us that $C_0^{(\iota,\iota\to\iota)\to\iota}$ is the least upper bound of the sequence.

This ends the proof of the proposition. □

Corollary 5.3. *The finite sequential functional Φ of type $((\iota,\iota\to\iota)\to\iota)\to\iota$ defined by*

$$\Phi(H) = 0 \text{ if } H(C_\perp^{\iota,\iota\to\iota}) = 0$$

does not commute with least upper bounds of increasing sequences in $\mathbb{Q}_{(\iota,\iota\to\iota)\to\iota}$.

Proof. Let H_n be as in the proof of Proposition 5.2. Then $\Phi(H_n) = \perp$ for each n, while

$$\Phi\left(\bigsqcup_{n\in\mathbb{N}} H_n\right) = \Phi(C_0^{(\iota,\iota\to\iota)\to\iota}) = 0.$$

This ends the proof. □

Remark 5.4. Corollary 5.3 verifies the other two conjectures of [17,18].

It also answers one part of the problem asked after Remark 8.2 in [4], there is an increasing ω-sequence with a least upper bound, and a program that does not respect this least upper bound.

We do not know the answer to the other part of the problem in [4], as we have no examples of a bounded set of sequential functionals that does not have a least upper bound.

In rest of this section and in Section 6 we will use a construction similar to the one used in the proof of Proposition 4.10, and an adaption of the proof used in [11] to produce further examples of finitary limit functionals.

We will employ the following notation:

Definition 5.5. Let $\sigma = \vec{\tau} \to \iota$ be a type, and let $\vec{f}$ be a sequence of finite sequential functionals of types $\vec{\tau}$.

We let $\Sigma_{\vec{f}}$ be the step function $\vec{f} \mapsto 0$, see Definition 2.22.

Lemma 5.6. *Let $\sigma = \tau, \vec{\tau} \to \iota$, let f be a finite sequential functional of type τ and let $\vec{f}$ be finite sequential functionals of types $\vec{\tau}$.*

If Σ_f is a limit functional, then $\Sigma_{f,\vec{f}}$ is a limit functional.

Proof. If F is of type $\tau \to \iota$ we define $\Phi_{\vec{f}}(F)$ of type $\tau, \vec{\tau} \to \iota$ by

$\Phi_{\vec{f}}(F)(g,\vec{g}) = F(g)$ if $\vec{f} \sqsubseteq \vec{g}$.
$\Phi_{\vec{f}}(F)(g,\vec{g}) = \perp$ otherwise.

If G is of type $\tau, \vec{\tau} \to \iota$, we define $\Psi_{\vec{f}}(G)$ of type $\tau \to \iota$ by

$$\Psi_{\vec{f}}(G)(g) = G(g, \vec{f}).$$

Then $(\Phi_{\vec{f}}, \Psi_{\vec{f}})$ is a sequential e-p-pair between $\mathbb{Q}_{\tau\to\iota}$ and $\mathbb{Q}_{\tau,\vec{\tau}\to\iota}$

It follows that if

$$\Sigma_f = \bigsqcup_{n\in\mathbb{N}} F_n$$

then

$$\Phi_{\vec{f}}(\Sigma_f) = \Sigma_{f,\vec{f}} = \bigsqcup_{n\in\mathbb{N}} \Phi(F_n).$$

This ends the proof of the lemma. □

This lemma shows that we can obtain much by focusing on the simpler cases $\sigma = \tau \to \iota$.

First we will prove a number of propositions of the form "Σ_f is a limit functional", and then we will summarize the consequences of Lemma 5.6 and these other propositions.

Proposition 5.7. *If $g \in \mathbb{Q}_{\iota^r\to\iota}$ is finite, where $r \geq 3$, and $g(\bot^r) = \bot$, then Σ_g is a limit functional.*

Proof. Choose c such that $g \in \mathrm{FSP}^{c-1}$, see Definition 2.17.

Since g is not a constant and the ordering in which we write the variables is not important, we may assume that g is of the form

$$g(x, y, z, \vec{t}) = g_k(y, z, \vec{t}) \textbf{ if } x = k \in K$$

(the variable x can be omitted from g_k when its value is set to k in advance).

For each m, let f_m be as in the proof of Proposition 4.10.

For each $m \geq 0$, we define $h_m \sqsupseteq g$ as follows, leaving it for the reader to bring the definition to the syntactically correct form:

$h_m(x, y, z, \vec{t}) = g_k(y, z, \vec{t})$ when $x = k \in K$
$h_m(x, y, z, \vec{t}) = f_m(y, z)$ if $x = c$
$h_m(x, y, z, \vec{t}) = \bot$ otherwise.

Claim 1. For each n there is a finitary sequential H_n such that

$$H_n(h_m) \in \mathbb{N} \Leftrightarrow m \leq n$$

and then $H_n(h_i) = 0$ for $0 \leq i \leq n$.

Proof of Claim 1. Let G_n be as in the proof of Proposition 4.10 and let

$$H_n(f) = 0 \cdot G_n(\lambda(y, z).f(c, y, z, \vec{\bot})).$$

Then $H_n(h_m) = 0 \cdot G_n(f_m)$, and the claim holds.

From now on in this proof we will let $H_n \sqsubseteq \Sigma_g$ be the least finite sequential functional satisfying Claim 1.

Claim 2. Let M be of type $(\iota^r \to \iota) \to \iota$ be sequential, but not necessarily finite.

If $M(h_n) \in \mathbb{N}$ for infinitely many n including $n = 0$, then $M(g) \in \mathbb{N}$.

Proof of Claim 2. This claim is similar to Claim 2 of the proof of Proposition 4.10, but with g in place of $\bot$, and our argument follows the same pattern.

We use induction on the length of the evaluation path of $M(h_0)$.

If M is a constant, then the conclusion obviously holds.

Let $M(f) = M_l(f)$ **if** $f(A(f), L(f), R(f), \vec{T}(f)) = l \in L$.

Since $M(h_0) \in \mathbb{N}$ we have that $h_0(A(h_0), L(h_0), R(h_0), \vec{T}(h_0)) = l \in L$. There are two possibilities:

1. $A(h_0) = k \in K$ and $g_k(L(h_0), R(h_0), \vec{T}(h_0)) = l \in L$.
2. $A(h_0) = c$ and $f_0(L(h_0), R(h_0)) \in \mathbb{N}$ and hence $L(h_0) \in \mathbb{N}$.

We split the argument into those cases:

1. There will be two subcases, out of which at least one must hold:
   1.1 $A(h_n) = k \in K$ for infinitely many n such that $M(h_n) \in \mathbb{N}$. Then we use the induction hypothesis, and obtain that $A(g) = k \in K$.
   Either g_k is the constant l or the evaluation of
   $g_k(L(h_0), R(h_0), \vec{T}(h_0))$
   will involve the evaluation of some $S_1(h_0), \dots, S_t(h_0)$ where $S_1, \dots, S_t$ all are among $L, R, \vec{T}$.
   In the first case, $g_k(L(g), R(g), \vec{T}(g)) = l$.
   In the other case, we prove by induction on $j \leq t$ that we will have that $S_j(h_n) \in \mathbb{N}$ for infinitely many $n \in \mathbb{N}$ and that, by the main induction hypothesis, $S_j(g) = S_j(h_0)$.
   As a consequence, we have that $g_k(L(g), R(g), \vec{T}(g)) = l$ also in this case.
   We finally use the induction hypothesis to see that $M_l(g) \in \mathbb{N}$. It follows that $M(g) = M(h_0) \in \mathbb{N}$.
   1.2 $A(h_n) = c$ for infinitely many n. This is, however, impossible, since by the induction hypothesis, $A(g) \in \mathbb{N}$, and then we cannot have different values c and k of $A(h)$ for different extensions h of g.
1. We will prove that this case is impossible. This case also splits into two subcases:
   2.1 $A(h_n) = k \in K$ for infinitely many n.
   This contradicts the induction hypothesis like the case 1.2.
   2.2 $A(h_n) = c$ for infinitely many n such that $M(h_n) \in \mathbb{N}$.
   As in the proof of Claim 2 in the proof of Proposition 4.10 we have that all $R(h_n)$ are compatible when $n > 0$, and thus will all be in a set $\{\perp, k\}$ for some fixed k.
   Then, if $n > k$ and $h_n(c, L(h_n), R(h_n), \vec{T}(h_n)) = f_n(L(h_n), R(h_n)) \in \mathbb{N}$ we must have that $L(h_n) = 1$.
   This, together with the observation that $L(h_0) = 0$ in this case, contradicts to the conclusion that $L(g) \in \mathbb{N}$ since $g \sqsubseteq h_n, n \geq 0$.

This ends the proof of Claim 2.

A consequence of Claim 2 is that if a sequential functional H is an upper bound of the strictly increasing sequence $\{H_n\}_{n\in\mathbb{N}}$, then $H(g) = 0$. Thus Σ_g is the nontrivial least upper bound of the sequence, and Σ_g is a limit functional.

This ends the proof of Proposition 5.7. □

Limit step functions of level > 2

We will now consider step functions of a type σ of level above 2. We let FSP^c_σ be as before.

Definition 5.8. We define the *c-total* sequential functionals by recursion on σ as follows:

The c-total elements of type ι are the numbers $0, \dots, c$.
$H \in \mathbb{Q}_{\tau\to\delta}$ is c-total if H maps c-total objects of type τ to c-total objects of type δ.

Lemma 5.9. *Let $\sigma = \vec{\tau} \to \iota$ be an arbitrary type. For every c-total functional H of type σ, there is a unique minimal c-total functional $H' \sqsubseteq H$. It is definable by an FSP^c_σ, gives the same values of type ι for c-total inputs of types $\vec{\tau}$ and is undefined on non-c-total inputs. In particular, the base of H' consists of the set $\{\vec{f}_1, \dots, \vec{f}_t\}$ of all minimal c-total inputs of types $\vec{\tau}$.*

Proof. By the induction hypothesis, the set $\{\vec{f}_1, \dots, \vec{f}_t\}$ of all minimal c-total inputs of types $\vec{\tau}$ is finite. We can compute $H'(\vec{f})$ sequentially, by identifying an i such that $\vec{f}_i \sqsubseteq \vec{f}$ and outputting $H(\vec{f}_i)$. If not all $\vec{f}$ are c-total then no $\vec{f}_i \sqsubseteq \vec{f}$ and so $H(\vec{f}) = \perp$. There are evidently only finitely many of so defined minimal c-total H'. □

Lemma 5.10. *For each type σ and $c \in \mathbb{N}$ there is a sequential functional Δ_σ of type $\sigma \to \iota$ that terminates exactly on extensions of c-total H's of type σ, and such that H' of the previous lemma is uniformly sequentially computable from $\Delta_\sigma(H)$.*

Proof. If $\sigma = \iota$ this is trivial.

If $\sigma = \vec{\tau} \to \iota$ we test if H of type σ is c-total by listing, up to equivalence, all sequences $\vec{f}_1, \dots, \vec{f}_t$ of minimal c-total elements of type $\vec{\tau}$ and, one by one evaluate $H(\vec{f}_i)$.

If one of the evaluations fails by not giving a value $\leq c$, H is not c-total.

If all evaluations terminate with values $\leq c$, we conclude that H is c-total, we let $\Delta_\sigma(H)$ be a natural number coding the list of values of $H(\vec{f}_i)$ for $i = 1, \dots, t$. □

From this number we can compute $H'(\vec{f})$, by first computing $\Delta_{\vec{\tau}}(\vec{f})$ and then, if it terminates, use the value of $\Delta_{\vec{\tau}}(\vec{f})$ to identify an i such that $\vec{f}_i \sqsubseteq \vec{f}$. (Here we use Lemma 5.9 on the base of H'.)

Then we go back to $\Delta_\sigma(H)$ to deduce what $H'(\vec{f})$ must be.

Definition 5.11. For each type $\sigma \neq \iota$, let g^c_σ be the least c-total object of type σ that takes the value c on all c-total inputs.
We let $g^c_\iota = c$.

We will write $\vec{g}^c$ for the more cumbersome $\vec{g}^c_{\vec{\tau}}$ when $\vec{\tau}$ is clear from the context.

Lemma 5.12. *Let $H \in \mathrm{FSP}^{c-1}_\sigma$ be such that H is not a constant. Then there is an extension $\tilde{H}$ in FSP^c_σ of H such that*

1. *$\tilde{H}$ is c-total.*
2. *$\tilde{H}(\vec{g}^c) = c$.*

Proof. If H is a constant, we let $\tilde{H} = H$. Then $\tilde{H}$ is c-total, but we do not have that $\tilde{H}(\vec{g}^c) = c$.
Now assume that H is not a constant.
Let

$$H(\vec{f}) = H_k(\vec{f}) \text{ if } f_i(\vec{T}(\vec{f})) = k \in K,$$

where s_i is the length of $\vec{T}$.
We then define $\tilde{H}(\vec{f})$ by

If $f_i(\tilde{T}_1(\vec{f}), \dots, \tilde{T}_{s_i}(\vec{f})) = k \in K$ then $\tilde{H}(\vec{f}) = \tilde{H}_k(\vec{f})$.
If $f_i(\tilde{T}_1(\vec{f}), \dots, \tilde{T}_{s_i}(\vec{f})) \in \{0, \dots, c\} \setminus K$ then $\tilde{H}(\vec{f}) = c$

where we leave it for the reader to transform this to the correct syntax. Note that $c \notin K$ since $H \in \mathrm{FSP}^\sigma_{c-1}$. By the induction hypothesis, each $\tilde{T}_j$ is c-total, and each $\tilde{H}_k$ is c-total when $k \in K$.
Thus, if $\vec{f}$ is c-total, then each $\tilde{T}_j(\vec{f})$ is c-total, and

$$f_i(\tilde{T}_1(\vec{f}), \dots, \tilde{T}_{s_i}(\vec{f})) \in \{0, \dots, c\}.$$

If the value is $k \in K$, we use the induction hypothesis on H_k to show that $\tilde{H}(\vec{f}) \in \{0, \dots, c\}$, otherwise this conclusion follows directly.
If the input is $\vec{g}^c$, it follows from the first part that $\tilde{H}(\vec{g}^c) \in \{0, \dots, c\}$.
Since g^c_i only takes the value c, part 2 follows directly.
This ends the proof of the lemma. □

Remark 5.13. If for some $\vec{f}$ (not necessarily c-total) we have that $H(\vec{f}) \in \mathbb{N}$, we clearly have that $\tilde{H}(\vec{f}) \in \mathbb{N}$ by the same evaluation, since we nowhere in the construction change anything as long as we follow one of the old evaluation paths. Moreover, in this case $H(\vec{f}) = \tilde{H}(\vec{f}) < c$.

Lemma 5.14. *Let $H \in \mathrm{FSP}^{c-1}_\sigma$ and assume that H is not a constant.*
Then there is an $\sqsubseteq$-extension $\hat{H}$ of H such that $\hat{H}(\vec{g}^c) = c$ and such that the base for $\hat{H}$ consists of the base for H together with $\vec{g}^c$.

Proof. Let $\hat{H}(\vec{f}) = H(\vec{f})$ if $\tilde{H}(\vec{f}) < c$ and let $\hat{H}(\vec{f}) = c$ if $\tilde{H}(\vec{f}) = c$ and $\vec{g}^c \sqsubseteq \vec{f}$.
By Lemma 2.7, $\hat{H}$ is in FSP_σ. This ends the proof. □

We will now prove the final proposition, Proposition 5.19 in the direction of step functions being limit functionals. The proof of this proposition is an elaboration of the proof of the main result in [11], and we split the argument into several definitions and lemmas that in reality just are parts of the proof of 5.19. For the gain of notational simplicity, we will not distinguish between the FSP H and its interpretation as a finite sequential functional.

Until the proof of Proposition 5.19 we will fix $H \in FSP^c_\sigma$ to be of level ≥ 2, and we will assume that H is not a constant. The aim will be to prove that Σ_H is a limit functional.

We let $\sigma = \vec{\tau} \to \iota$ where $\tau_j = \vec{\delta}_j \to \iota$ with s and s_j the lengths of tuple types $\vec{\tau}$ and $\vec{\delta}_j$, respectively. Without restricting the generality we can assume that $\tau_1 = \vec{\delta}_1 \to \iota$ has level ≥ 1. For each natural number d, we let $\vec{d}$ be the s_1-tuple $(C_d, \dots, C_d)$ of the tuple type $\vec{\delta}_1$ with each occurrence of C_d having its own type depending on the position according to the structure of $\vec{\delta}_1$. We will assume that $\hat{H}$ and $\vec{g}^{\,c}$ are as in Definition 5.11 and Lemma 5.14.

We will define the sequential functionals $F_n \sqsupseteq H$ for each n, and in parallel we will define the objects T_n for all $n \geq 0$ and S_n for all $n > 0$. These will be finite objects of type τ_1. These objects will depend on H.

Definition 5.15 (*Of F_n, S_n, T_n*)**.**

$n = 0$ If $\hat{H}(\vec{f}) < c$ we let $F_0(\vec{f}) = H(\vec{f})$.
If $\hat{H}(\vec{f}) = c$ and $f_1(\overline{c+1}) = 1$ we let $F_0(\vec{f}) = 0$.
In all other cases we let $F_0(\vec{f}) = \bot$.
Let $T_0(\overline{c+1}) = 1$ and let T_0 be undefined for all other inputs.

$n > 0$ If $\hat{H}(\vec{f}) < c$ we let $F_n(\vec{f}) = H(\vec{f})$.

If $\hat{H}(\vec{f}) = c$ then $F_n(\vec{f}) = 1$ in cases (i) and (ii) below, otherwise the value is $\perp$.

(i) $f_1(\overline{c+1+n}) = 0$ and $f_1(\overline{c+1}) = 0$.

(ii) $f_1(\overline{c+1+n}) = 1$ and $f_1(\overline{c+1+i}) = 0$ for $0 < i < n$.

We let $T_n(\overline{c+1+n}) = 0$ and $T_n(\overline{c+1}) = 0$ and we let T_n be undefined for all other inputs.

We let $S_n(\overline{c+1+n}) = 1$, we let $S_n(\overline{c+1+i}) = 0$ for $0 < i < n$, and we let S_n be $\perp$ for all other inputs.

We see that each F_n is in FSP_σ since we (almost) gave the algorithm for each of them. (We do not follow the convention that the letter F is restricted to sequential *functionals* here.)

Since $\overline{d}$ is both maximal and finite, we can replace the phrase **if** $\vec{x} = \overline{d}$ with a sequential query depending on d. We used this implicitly in defining S_n and T_n as finite sequential procedures.

This and the following considerations are also used in the proof of the next lemma.

The base for F_0 will consist of the base for H together with $((g_1^c \sqcup T_0), g_2^c, \ldots, g_s^c)$, where g_j^c are of the types of τ_j. When $n > 0$, the base for F_n will consist of the base for H together with $((g_1^c \sqcup T_n), g_2^c, \ldots, g_s^c)$ and $((g_1^c \sqcup S_n), g_2^c, \ldots, g_s^c)$.

Note that mentioned above lubs $T_n' = g_1^c \sqcup T_n$ and $S_n' = g_1^c \sqcup S_n$ of type τ_1 (of level ≥ 1) indeed exist. For τ_1 of level 1 this is trivial. For level > 1 appropriate $T_n'(\vec{x})$ and $S_n'(\vec{x})$ are sequentially computable by checking first whether $x_r(\vec{C}_c) \leq c$ for any chosen argument x_r of level > 0. If "yes", output $g_1^c(\vec{x})$; if $c < x_r(\vec{C}_c) \leq c+1+n$, output $T_n(\vec{x})$ or $S_n(\vec{x})$, respectively; otherwise the result is $\perp$. Then, we can show e.g. that $g_1^c \sqsubseteq T_n'$ by using Lemma 5.9: if $g_1^c(\vec{x}) \in \mathbb{N}$ then all $\vec{x}$ are c-total and so $x_r(\vec{C}_c) \leq c$ and $T_n'(\vec{x})$ computes as $g_1^c(\vec{x})$. We leave the rest of the proof on T_n' and S_n' to the reader.

Lemma 5.16. *For each $n \in \mathbb{N}$ there is an* FSP *Θ_n of type $\sigma \to \iota$ such that for all $i \in \mathbb{N}$:*

$$\Theta_n(F_i)\downarrow \leftrightarrow i \leq n.$$

Proof. We will give an explicit construction of Θ_n by recursion on n. We will in addition prove from the construction that

$$\Theta_n(F_0) = 1 \text{ and } \Theta_n(F_i) = 0 \text{ if } 1 \leq i \leq n.$$

We let

$$\Theta_0(F) = \begin{cases} 1 & \textbf{if } F((g_1^c \sqcup T_0), g_2^c, \ldots, g_s^c) = 0 \\ \perp & \text{otherwise} \end{cases}$$

Θ_0 clearly has the specified property.

Assume now that Θ_n is defined with the specified property.

We define the auxiliary R_{n+1} of type $\sigma, \vec{\delta}_1 \to \iota$ as follows:

$$R_{n+1}(F, \vec{x}) = \begin{cases} \Theta_n(F) & \textbf{if } \vec{x} = \overline{c+1} \\ 1 & \textbf{if } \vec{x} = \overline{c+1+n+1} \\ 0 & \textbf{if } \vec{x} = \overline{c+1+i} \wedge 1 \leq i \leq n \\ \perp & \text{otherwise} \end{cases}$$

in other words,

$$R_{n+1}(F) = [\overline{c+1} \mapsto \Theta_n(F)] \sqcup S_{n+1},$$

and we let

$$\Theta_{n+1}(F) = 1 \dot{-} F(g_1^c \sqcup R_{n+1}(F), g_2^c, \ldots, g_s^c).$$

Since we are interested in $\Theta_{n+1}(F_i)$, we need to know more on $R_{n+1}(F_i)$.

From the induction hypothesis we will see that $T_0 \sqsubseteq R_{n+1}(F_0)$, that $T_i \sqsubseteq R_{n+1}(F_i)$ for $1 \leq i \leq n$ and that $S_{n+1} = R_{n+1}(F_{n+1})$ and, moreover, that $S_{n+1} = R_{n+1}(F_i)$ for all $i \geq n+1$.

Then the desired property of Θ_{n+1} follows, and the proof is complete. □

We will now modify H (any $H \in \mathrm{FSP}_\sigma^{c-1}$ which is not a constant) to $\dot{H}$ with the same observational interpretation as H, but such that *evaluations* relative to $\dot{H}$ can be seen as evaluations relative to each F_n, i.e. $\dot{H} \prec F_n$.

Definition 5.17. We define $\dot{H}$ via the following procedure:

$$\dot{H}(\vec{f}) = H(\vec{f}) \textbf{ if } \hat{H}(\vec{f}) < c.$$

Note that if $\hat{H}(\vec{f}) = c$ (or, equivalently, $\vec{g}^{\,c} \sqsubseteq \vec{f}$) then both $\dot{H}(\vec{f}) = \perp$ and $H(\vec{f}) = \perp$.

If M is an applicative term with one free variable x of some type τ, and F is an FSP of type τ, we let $M[F]$ be the term where we have substituted F for x.

Lemma 5.18 (*Main*). *Let M be a typed applicative term of base type comprising of* SP*'s and one free variable of type σ. Assume that $M[F_n]\downarrow$ for infinitely many n, in particular terminating on $n = 0$. Then, for all n, all evaluations of $M[F_n]$ will follow the same path, and this will be an evaluation path for $M[\dot{H}]$ as well.*

A consequence will be that if $M[F_n]\downarrow$ for infinitely many n including $n = 0$, then it does so for all n and the value must be the same for all n because of $H = \dot{H} \sqsubseteq F_n$.

Proof. We prove this by induction on the length of the evaluation of $M[F_0]$. We will follow the evaluation path of $M[F_0]$ and we will use the induction hypothesis to show that this must agree with an evaluation path for $M[\dot{H}]$ and thus, for all $M[F_n]$.

The only nontrivial case is when $M = x\vec{N}$, since all evaluation paths will agree until we meet a case like this.

So we assume that $F_n(\vec{N}[F_n])$ terminates for infinitely many n including $n = 0$, and let X be the set of n for which this happens.

For each n, the procedure for computing $F_n(\vec{N}[F_n])$ starts by computing $\hat{H}(\vec{N}[F_n])$. By the induction hypothesis, we may as well evaluate $\hat{H}(\vec{N}[\dot{H}])$.

Now we must distinguish between two cases:

1. $\hat{H}(\vec{N}[\dot{H}])$ evaluates to c.
2. $\hat{H}(\vec{N}[\dot{H}])$ evaluates to a number $< c$.

We will verify that the statement of the lemma holds in case 2 and prove that case 1 is impossible. Together, this will form the induction step in this case. The rest of the proof splits into those two cases.

1. If this is the case, the evaluation paths of $F_0(\vec{N}[F_0])$ and the various $F_n(\vec{N}[F_n])$ will be different from each other. Indeed, the next step for $n = 0$ is to evaluate $N_1[F_0](c+1)$ and test if the value is 1. Termination requires that this is so.

   If $n > 0$ the next step will be to evaluate $N_1[F_n](\overline{c+1+n})$ and test if the value is 0 or 1, and termination of $M[F_n]$ requires that one of these two values will be the result in this case. Moreover,

   1.0 If the value is 0, we must in addition have $N_1[F_n](\overline{c+1}) = 0$.

   1.1 If the value is 1, we must in addition have $N_1[F_n](\overline{c+1+i}) = 0$ for all i such that $0 < i < n$.

   If $0 < n < m$ and $n, m \in X$ (the set where termination is assumed), we cannot be in case 1.1 for both n and m since F_n and F_m are consistent, and we require $N_1[F_n](\overline{c+1+n}) = 1$ and $N_1[F_m](\overline{c+1+n}) = 0$.

   Thus, with one possible exception, we must be in case 1.0 for all positive $n \in X$. But then, we have that $N_1[F_0](\overline{c+1}) = 1$ terminates via a subevaluation of the evaluation of $M[F_0]$ and for infinitely many n we have that $N_1[F_n](c+1) = 0$ terminates. By the induction hypothesis then $N_1[\dot{H}](\overline{c+1})$ terminates, which would mean that this term has two different values 1 and 0, so this case is impossible.
2. In this case, the computation path of

   $$M[F_n] = F_n(\vec{N}[F_n])$$

   contains a proper subpath evaluating $H(\vec{N}[F_n])$ as given in Definition 5.15 which terminates for all $n \in X$, including $n = 0$. Then we can use the induction hypothesis, that we actually get $H(\vec{N}[\dot{H}])\downarrow$, and the evaluation path used here is the one we used in evaluating $H(\vec{N}[F_n])$ actually for all n.

   But this also gives us the complete path for evaluating $\dot{H}(\vec{N}[\dot{H}])$, and we are through.

This ends the proof. □

It follows that

Proposition 5.19. *For any finite sequential functional H of level ≥ 2 such that $H(\vec{\perp}) = \perp$ we have that Σ_H of level > 2 is a limit functional.*

Proof. By $H \sqsubseteq F_n$, we have $\Sigma_{F_n} \sqsubseteq \Sigma_H$. Assume Θ is any upper bound of Σ_{F_n}. Then $\Theta(F_n) = 0$ for all n and hence $\Theta(H) = \Theta(\dot{H}) = 0$ as well, that is $\Sigma_H \sqsubseteq \Theta$. Thus, $\Sigma_H = \bigsqcup_n \Sigma_{F_n}$.

It remains to see that the sequence $\Theta_n = \bigsqcup_{k\leq n} \Sigma_{F_k}$ is strictly increasing.

This follows from Lemma 5.16, (the Θ_n's are not the same), and the proof is complete. □

Let us summarize what we have proved.

Theorem 5.20. *Let $\sigma = \tau_1, \ldots, \tau_s \to \iota$ and let $\vec{f}$ be a sequence of type $\vec{\tau}$.*

Assume that for some $i \leq s$ we have that τ_i is either of level ≥ 2 or of the form $\iota^r \to \iota$ where $r \geq 3$, and that f_i is not a constant. Then $\Sigma_{\vec{f}}$ is a limit functional.

Proof. Combine Lemma 5.6 and Propositions 5.7 and 5.19. □

This theorem gives also a full characterization of limit step functions of the types $\sigma = \vec{\tau} \to \iota$ with *each* τ_i either of level ≥ 2 or of the form $\iota^r \to \iota$ where $r \geq 3$ by using the following simple

Proposition 5.21. *Any finite functional whose all base tuples consist of constants has only finitely many approximations and so is not a limit one.*

6. Extensions to finite sequential functionals of type level above 2

In Section 5 we did not achieve a full characterization of when a step function is a limit functional. Interestingly enough, the remaining problems are with step functions of certain types of level 2, and we have to leave these problems open. In this section we will consider arbitrary finite sequential functionals Θ at type level ≥ 3, and we will see that if there is at least one base element $\vec{H}$ with a non-constant H_i of type level ≥ 2, we can elaborate the argument from the proof of Theorem 5.20 and obtain that the functional at hand is a limit functional. We will of course make a precise statement of what we can prove.

The proof of Proposition 5.7 does not extend from step functions to arbitrary finite sequential functionals, so we do not have a similar result for type $(\iota^r \to \iota) \to \iota$ even when $r \geq 3$.

Theorem 6.1. *Let $\alpha = \sigma, \vec{\sigma} \to \iota$ where σ is of level ≥ 2 and let Θ be a finite sequential functional of type α.*

Let $\{(H_j, H_{j,1}, \dots H_{j,s}) \mid j \leq m\}$ be finite sequential procedures representing the base elements of Θ and assume that there is at least one $j \leq m$ such that H_j is not a constant.

Then Θ is a limit functional.

Proof. Let $\vec{H}_j = (H_{j,1}, \dots, H_{j,s})$.

Our argument will be based on the proof of Theorem 5.20, and we will use notation from the last part of Section 5 .

Let c be so large that H_j and $H_{j,i}$ are in FSP^{c-1} for all $j \leq m$ and $i \leq s$.

Whenever H_j is not a constant, we let $\hat{H}_j$ be constructed from H_j as $\hat{H}$ was constructed from H in Lemma 5.14. We also construct $F_{j,n}$ extending H_j as in Section 5 when H_j is not a constant. Then we let

$$\vec{F}_{j,n} = (F_{j,n}, \vec{H}_j).$$

If H_j is a constant, we let

$$\vec{F}_{j,n} = (H_j, \vec{H}_j).$$

Claim 1. For each $n \in \mathbb{N}$ there is a finite sequential procedure Θ_n of type α such that

1. $\Theta_n(\vec{F}_{j,k}) = \Theta(H_j, \vec{H}_j)$ when $k \leq n$ and H_j is not a constant.
2. $\Theta_n(\vec{F}_{j,k}) = \bot$ when $k > n$ and H_j is not a constant.
3. $\Theta_n(\vec{F}_{j,k}) = \Theta(H_j, \vec{H}_j)$ when H_j is a constant.

Proof of Claim 1. For $n > 0$, let Θ'_n be the Θ_n from Lemma 5.16.

If $n = 0$ we adjust the definition to

$$\Theta'_0(F) = \begin{cases} 1 & \textbf{if } F((g^c_1 \sqcup T_0), g^c_2, \dots, g^c_s) \leq c \\ \bot & \text{otherwise.} \end{cases}$$

If H_j is not a constant, the key property of Θ'_n is that

$$\Theta'_n(F_{j,k})\downarrow \Leftrightarrow k \leq n,$$

and this property is not altered by our adjustment. From the construction of Θ'_n for $n > 0$, we see that $\Theta'_n(H_j)\downarrow$ if H_j is a constant. Our adjustment for $n = 0$ makes the same hold then since $H_j \in \mathrm{FSP}^{c-1}$.

We let

$$\Theta_n(F, \vec{F}) = \Theta(F, \vec{F}) \textbf{ if } \Theta'_n(F)\downarrow.$$

2. and 3. follow directly from the property of Θ'_n and the definition of $\vec{F}_{j,k}$.

1. follows because in this case

$$\Theta_n(\vec{F}_{j,k}) = \Theta(\vec{F}_{j,k}) = \Theta(F_{j,k}, \vec{H}_j) = \Theta(H_j, \vec{H}_j).$$

The final equation holds because $H_j \sqsubseteq F_{j,k}$ and $\Theta(H_j, \vec{H}_j)\downarrow$. This ends the proof of Claim 1.

We now let Θ_n be a $\sqsubseteq$-minimal FSP satisfying Claim 1. Then $\Theta_n \sqsubseteq \Theta$ and the sequence is strictly increasing.

Claim 2. If M is a sequential functional of type α and $\Theta_n \sqsubseteq M$ for each n, then $\Theta \sqsubseteq M$.

Proof of Claim 2. We have to prove that $M(H_j, \vec{H}_j) = \Theta(H_j, \vec{H}_j)$ for each base element $(H_j, \vec{H}_j)$ for Θ.

If H_j is a constant, we have for every n that

$$\Theta(H_j, \vec{H}_j) = \Theta_n(H_j, \vec{H}_j) \sqsubseteq M(H_j, \vec{H}_j).$$

If H_j is not a constant, we will rely on the Main Lemma, Lemma 5.18:

Let $M_j(F) = M(F, \vec{H}_j)$.

Then, for all k and $n \geq k$, e.g. $n = k$,

$$\Theta(H_j, \vec{H}_j) = \Theta_n(\vec{F}_{j,k}) = M(\vec{F}_{j,k}) = M_j(F_{j,k}) \in \mathbb{N}.$$

It follows from the Main Lemma that $M_j(H_j)$ terminates with the same value, and consequently that

$$M(H_j, \vec{H}_j) = \Theta(H_j, \vec{H}_j).$$

This ends the proof of the claim.

Then Θ is the least upper bound in $\mathbb{Q}_\alpha$ of the strictly increasing sequence $\{\Theta_n\}_{n\in\mathbb{N}}$, and thus Θ is a limit functional.

This ends the proof of the theorem. □

It is of course not important that it is the first type in the list of σ's that satisfies the special requirements.

For functionals of pure types and also for functionals with all arguments having levels ≥ 2 we have a full characterization (by also using Proposition 5.21 and Theorem 4.9 on the pure type 2 and similar simpler versions of the latter theorem for types of level 1):

Corollary 6.2. (a) *Let Θ be a finite sequential functional of any type $\alpha = \vec{\sigma} \to \iota$ with all $\vec{\sigma}$ of levels ≥ 2.*
Then Θ is a limit functional if and only if there at least one component of some its base tuple is not a constant.
(b) *Let Θ be a finite sequential functional of pure type α at any level k.*
Then Θ is a limit functional if and only if $k > 2$ and at least one base element of Θ is not a constant.

We also know that for types α of levels ≤ 1 there are no limit finite objects.

7. Discussion and conclusions

In this paper we have shown that if σ is a type of level ≤ 1 or pure type 2, then the sequential functionals $\mathbb{Q}_\sigma$ of type σ form a *dcpo*, and we have shown that if $\sigma = (\iota, \iota \to \iota) \to \iota$, or more complex than that one, then $\mathbb{Q}_\sigma$ is not a *dcpo*. We have neither been able to decide from the literature nor from our attempts of proving it one way or the other if $\mathbb{Q}_\sigma$ is a *dcpo* when $\sigma = (\iota \to \iota)^s \to \iota$, where $s > 1$. The observed fact that we do not have a normal form theorem for these functionals is a definite obstacle in the attempted proofs, but we dare conjecture that these types will form *dcpo*'s.

For pure types we have obtained a full characterization of when a finite sequential functional is a limit functional, and for sequential step functions, we also have a full characterization when the type is $\sigma = \vec{\tau} \to \iota$ with the list $\vec{\tau}$ containing only types of level ≥ 2 or level 1 types $\iota^s \to \iota$ with $s \geq 3$ (see a comment after Theorem 5.20). It came as a surprise that the final difficulties would be for types of the form $\tau \to \iota$ where τ is at level 1. $\mathbb{Q}_{\iota^s}$ can be embedded into

$$\{0, \ldots, s-1\}_\perp \to \mathbb{N}_\perp,$$

and may be viewed as a space of functions over a finite domain. In order to prove Propositions 4.10 and 5.2 we only need that this domain has two elements. In order to adjust the method to showing that certain step functions are limit functionals, we need an extra element in the domain, see Proposition 5.7. In order to extend this to finite sequential functionals in general, we need an infinite domain, i.e. we have to step one type level up, since the domain then cannot be represented as a cartesian product at type level 0.

As we have left some characterizations open, we intend to continue the exploration of the ordering of sequential functionals. There are also other questions to ask, like

1. Given a bounded chain F_n and an element S such that $\{F_n, S\}$ is bounded for each n, will the chain and the element have a common upper bound?
2. Will a finite, bounded set in $\mathbb{Q}_\sigma$ have a least upper bound in $\mathbb{Q}_\sigma$?
3. Will every infinite bounded set have a least upper bound?

The motivation for investigating such problems is not so much that finding the answers is so important, but that we believe that the process of finding the answers will teach us more about the limitations on the one side and the potential of sequential functional algorithms on the other side.

The first problem was solved in the negative during the revision after acceptance of this paper.

Assuming that sufficiently many interesting results can be found, we plan to write a sequel to the present paper, and the solution of the first problem will then be included.

Acknowledgements

We thank the anonymous referee for many helpful comments and suggestions.

The first author will thank John Longley, Alex Simpson and Gordon Plotkin for the hospitality and fruitful discussions during the 10 week visit to Edinburgh in the spring of 2009, and Martin Escardó and Achim Jung for inviting him to Birmingham to give a seminar and for good discussions on results from the paper.

He is also grateful to the University of Liverpool for supporting his visit there in spring 2009.

Some of the results were presented in an informal contribution to CiE 2009 in Heidelberg and in a lecture given at Mal'tsev 2009 in Novosibirsk.

The research for this paper has been partly supported by a grant from the Norwegian Research Council.

References

[1] S. Abramsky, R. Jagadeesan, P. Malacaria, Full abstraction for PCF, Information and Computation 163 (2) (2000) 409–470.
[2] R.M. Amadiou, P.-L. Curien, Domains and Lambda-Calculi, Cambridge University Press, 1998.
[3] Yu.L. Ershov, Computable functionals of finite types, Algebra and Logic 11 (4) (1972) 367–437. doi:10.1007/BF02219096. The journal is translated in English; available via http://www.springerlink.com.
[4] M.H. Escardó, W.K. Ho, Operational domain theory and topology of sequential programming languages, Information and Computation 207 (2009) 411–437.
[5] J.M.E. Hyland, C.-H.L. Ong, On full abstraction for PCF: I, II, and III, Information and Computation 163 (2000) 285–408.
[6] S.C. Kleene, Recursive functionals and quantifiers of finite types I, Trans. Amer. Math. Soc. 91 (1959) 1–59.
[7] R. Loader, Finitary PCF is not decidable, Theoretical Computer Science 266 (2001) 341–364.
[8] R. Milner, Fully abstract models of typed λ-calculi, Theoretical Computer Science 4 (1977) 1–22.
[9] J. Moldestad, Computations in Higher Types, in: Lecture Notes in Mathematics, vol. 574, Springer Verlag, 1977.
[10] H. Nickau, Hereditarily-sequential functionals: a game-theoretic approach to sequentiality, Ph.D. Thesis, Siegen, 1996.
[11] D. Normann, On sequential functionals of type 3, Mathematical Structures in Computer Science 16 (2) (2006) 279–289.
[12] R.A. Platek, Foundations of recursion theory, Ph.D. Thesis, Stanford University, 1966.
[13] G. Plotkin, LCF considered as a programming language, Theoretical Computer Science 5 (1977) 223–256.
[14] V.Yu. Sazonov, On semantics of the applicative algorithmic languages, Ph.D. Thesis, Novosibirsk, Institute of Mathematics, 1976 (in Russian) Available from the Russian State Library in Moscow.
[15] V.Yu. Sazonov, Expressibility of functionals in D. Scott's LCF language, Algebra and Logic 15 (3) (1976) 308–330. doi:10.1007/BF01876321. The journal is translated in English; available via http://www.springerlink.com.
[16] V.Yu. Sazonov, Functionals computable in series and in parallel, Siberian Mathematical Journal 17 (3) (1976) 648–672. doi:10.1007/BF00967869. The journal is translated in English; available via http://www.springerlink.com.
[17] V.Yu. Sazonov, An inductive definition and domain theoretic properties of fully abstract models of PCF and PCF+, Logical Methods in Computer Science 3 (2007) 1–50.
[18] V.Yu. Sazonov, Natural non-dcpo domains and f-spaces, Annals of Pure and Applied Logic 159 (2009) 341–355.
[19] D.S. Scott, A type-theoretical alternative to ISWIM, CUCH, OWHY, Theoretical Computer Science 121 (1 & 2) (1993) 411–440. Böhm Festschrift. Article has been widely circulated as an unpublished manuscript since 1969.
[20] T. Streicher, Domain-theoretic Foundations of Functional Programming, World Scientific, 2006.