THE SEQUENTIAL FUNCTIONALS OF TYPE $(\iota \to \iota)^n \to \iota$ FORM A *DCPO* FOR ALL $n \in \mathbb{N}$

DAG NORMANN

Department of Mathematics, The University of Oslo, P.O. Box 1053, Blindern N-0316 Oslo, Norway
e-mail address: dnormann@math.uio.no

ABSTRACT. We prove that the sequential functionals of some fixed types at type level 2, taking finite sequences of unary functions as arguments, form a directed complete partial ordering. This gives a full characterisation of the types for which the partially ordered set of sequential functionals has this property.

As a tool, we prove a normal form theorem for the finite sequential functionals of the types in question.

1. INTRODUCTION

When Scott [10] introduced his calculus LCF and the model of partial continuous functionals for it, it was clear that this model is not fully abstract. There will be finite elements in this model, e.g. parallel or, that are not LCF-definable, and consequently there will be observationally equivalent terms in LCF that will be interpreted as different objects in the Scott model.

Milner [5] constructed an alternative model for LCF based on Scott domains, a model that is fully abstract, and proved that up to isomorphism, there is only one such model. Since Milner's model has the cardinality of the continuum, there will be a lot of elements in this model that are not LCF-definable. However, if we let LCF^Ω denote the calculus where we extend LCF with one constant $\hat{f}$ for each $f : \mathbb{N} \to \mathbb{N}$, together with the rewrite rules $\hat{f}a \to b$ whenever $f(a) = b$, the question whether all objects in Milner's model are LCF^Ω-definable remained open for many years.

We let the *sequential functionals* be the hereditarily minimal extensional model for LCF^Ω. In this paper, we will investigate the closure properties of the sequential functionals of types $(\iota \to \iota)^n \to \iota$, where ι denotes the base type interpreted as the flat domain $\mathbb{N}_\perp$. We will use one of the known characterisations of the sequential functionals, representing them as the hereditarily extensional collapse of the sequential procedures (see Section 2). The main result is that the sequential functionals of the types $(\iota \to \iota)^n \to \iota$ will form a directed complete partial ordering. As a tool, an extended *normal form theorem* will be obtained for these types.

The first systematic analysis of the LCF-definable functionals, seen as a subclass of the Scott model, was due to Sazonov [9]. To this end, he introduced the concept of a *strategy*, a

concept that is close to sequential procedures as we define them. Later characterisations are due to Nickau [6], to Abramsky, Jagadeesan and Malacaria [1] and to Hyland and Ong [2]. These characterisations appeared during the 1990's. The question whether they characterised Milner's model as well was, however, left open through that decade. This question was answered in Normann [7]. There it is proved that the sequential functionals of pure type 3 do not form a *dcpo*, a directed complete partial ordering, and thus do not coincide with Milner's interpretation of this type. Normann and Sazonov [8] studied some anomalies inspired by [7], and among other things, they proved that even the sequential functionals of type $(\iota, \iota \to \iota) \to \iota$ do not form a *dcpo*. This type is of level 2 in the crude stratification of the finite types into levels. However, in [8] the authors were unable to offer a full characterisation of when the sequential functionals of a given type form a *dcpo*, and it is this gap of knowledge that will be closed in this note.

The sequential functionals F of type $(\iota, \iota \to \iota) \to \iota$ will take sequential functions $f : \mathbb{N}_\perp \times \mathbb{N}_\perp \to \mathbb{N}_\perp$ as arguments and give elements of $\mathbb{N}_\perp$ as values. It is essential for the construction in [8] that f ranges over the *binary* functions. We do not need the definition of sequential functionals of this type in this note.

2. Sequential procedures and functionals

Throughout this note, we will let $n > 0$ be a fixed integer. We let the set of monotone functions $f : \mathbb{N}_\perp \to \mathbb{N}_\perp$ be our interpretation of the type $\iota \to \iota$, and we let $\sqsubseteq$ be the standard pointwise ordering on this set. We will use f, g etc. for such functions. We will use $\vec{f}$ for n-sequences $(f_1, \ldots, f_n)$ of functions.

We will interpret $\sigma = (\iota \to \iota)^n \to \iota$ as the set of *sequential functionals* of the type. They can be given as the denotational, hereditarily extensional, interpretations of the *sequential procedures* as defined below. The sequential procedures will be infinite syntax trees, infinitary terms of type ι, where we essentially follow the notation from [4]. All of our procedures will be terms with free variables among $x_1, \ldots, x_n$, a list of variables of type $(\iota \to \iota)$.

Remark 2.1. Since the interpretation $\mathbb{N}_\perp$ of ι is a sequential retract of the interpretation $\mathbb{N}_\perp \to \mathbb{N}_\perp$ of $(\iota \to \iota)$, we will indirectly also have covered the cases $\sigma_1, \ldots, \sigma_n \to \iota$ where each σ_i is either ι or $(\iota \to \iota)$. When the meaning is clear, we will drop the brackets, and write $\iota \to \iota$.

Definition 2.2. For $a \in \mathbb{N}_\perp$, we let a also be a constant term denoting itself. The *sequential procedures* in the variables $x_1, \ldots, x_n$ of type $\iota \to \iota$ will be the largest class of infinitary terms P of type ι such that either $P = c$ for some $c \in \mathbb{N}_\perp$ or P is of the form

$$P = \text{case } x_i(Q) \text{ of } \{a \Rightarrow P_a\}_{a \in \mathbb{N}}$$

where Q and each P_a are sequential procedures. We will drop the index $a \in \mathbb{N}$ in the notation in this paper.

Remark 2.3. This definition is a special, and slightly simplified, case of the more general definition of the sequential procedures of type σ for any type σ, as given in [4]. We have actually defined the relevant class of *expressions* as defined in [4], but since we do not need terms defined by λ-abstractions here, we name them *procedures*.

The sequential procedures were used as the basis for investigating the sequential functionals both in [8] and in [4]. In both, there are proofs of the fact that the sequential

procedures form an applicative structure. The proofs differ in some essential ways, and in both cases they are non-trivial.

The class of sequential procedures is given by a coinductive definition. We may of course view the clauses as clauses in an inductive definition. If we do so, and in addition restrict a to finite sets $A \subseteq \mathbb{N}$ instead of $\mathbb{N}$ (meaning, in effect, that $P_a = \bot$ if $a \notin A$), we get the *finite sequential procedures*.

Definition 2.4. Each procedure P with variables among $x_1, \ldots, x_n$ will define a *sequential functional* $[\![P]\!] : (\mathbb{N}_\bot \to \mathbb{N}_\bot)^n \to \mathbb{N}_\bot$ by the *inductive* definition of $[\![P]\!](\vec{f})$:

(1) $[\![P]\!](\vec{f}) = a$ when $P = a$.
(2) If
$$P = \text{case } x_i(Q) \text{ of } \{a \Rightarrow P_a\}_{a \in \mathbb{N}},$$
we let $[\![P]\!](\vec{f}) = b$ if for some $a \in \mathbb{N}$ and for some $c \in \mathbb{N}_\bot$ we have that $[\![Q]\!](\vec{f}) = c$, $f_i(c) = a$, and $[\![P_a]\!](\vec{f}) = b$.
(3) $[\![P]\!](\vec{f}) = \bot$ if no value is given through (1) or (2).

We write $[\![P]\!](\vec{f})\!\downarrow$ if there is some $b \in \mathbb{N}$ such that $[\![P]\!](\vec{f}) = b$. There may be three reasons for why $[\![P]\!](\vec{f}) = \bot$. One is that the syntax tree of P is not well founded, and that we follow an infinite branch in this syntax tree when we evaluate $[\![P]\!](\vec{f})$. Another is that we, during the evaluation, are led to some $a \Rightarrow \bot$. The third reason is that we need to evaluate some $f_i([\![Q]\!](\vec{f}))$ where the value of $f_i([\![Q]\!](\vec{f})) = \bot$. There is a qualitative difference here: in the first case the process does not terminate, while in the other cases the process terminates with the value $\bot$. In our interpretation $[\![P]\!]$ we do not distinguish between these cases.

Definition 2.5. Let $k \in \mathbb{N}$, and let $\mathbf{F}_k$ be the set of sequential functionals of our fixed type over the interpretation $\{\bot, 0, \ldots, k\}$ of ι. We let $\mathbf{F}$ denote the full set of sequential functionals of the type in question. For each $k \in \mathbb{N}$, let (ν_k, π_k) be the standard sequential embedding-projection pair between $\mathbf{F}_k$ and $\mathbf{F}$ induced by the inclusion map and the restriction map at base type.

The crucial properties are:
$$\pi_k \circ \nu_k = id_{\mathbf{F}_k} \text{ and } \nu_k \circ \pi_k \sqsubseteq id_{\mathbf{F}}$$
Since we will consider $\mathbf{F}_k$ to be a subset of $\mathbf{F}$ we will also consider ν_k to be the inclusion map, and not mention it again. We will need π_k in the sequel, viewing it as a restriction map. We will use the obvious fact that each $\mathbf{F}_k$ is a finite set.

We say that two procedures are *equivalent* if they define the same functional. In [8] a procedure was defined to be *normal* if we will only use Qs of the form c with $c \in \mathbb{N}$ in the coinductive definition. It is not the case that all procedures will be equivalent to procedures in normal form, so this term is slightly misleading.

As usual, we will say that a function f is *strict* if $f(\bot) = \bot$. It is the presence of non-strict functions, and the fact that sometimes $[\![P]\!](\vec{f})$ may terminate *because* some f_i is not strict, that makes it impossible to prove a normalisation theorem. Plotkin proved that the strict functions between *concrete domains* will form *dcpo*s, see Kahn and Plotkin [3] for definitions. From now on, S will always be a subset of $\{1, \ldots, n\}$.

Definition 2.6. If F is a sequential functional, we let F_S be defined by $F_S(\vec{f}) = a$ if there is a sequence $\vec{g} \sqsubseteq \vec{f}$ such that

- g_i is strict when $i \in S$,
- $F(\vec{g}) = a$.

F_S will be sequential when F is, since F_S is the composition of F and sequential "strictification"-functionals. This argument requires that we know that the sequential functionals form an applicative structure, see [4] or [8]. Definition 2.7 will contain a direct construction of a procedure for F_S from a procedure for F in the finite case, and then it will follow that F_S is sequential from our main theorem.

A finite sequential procedure may be nested in two directions, in depth (or to the left) and in length (or to the right). A *left bound* of a procedure will be $\geq$ the maximal number of nestings to the left in any branch of the syntax tree. We will prove that if F is a finite sequential procedure, there is a special procedure for F_S with a left bound depending only on S, and this will be our alternative to the normal form theorem for sequential procedures restricted to strict arguments.

Definition 2.7. Let P be a finite, non-empty sequential procedure. We define the procedure $P[S]$ by cases as follows:

1. If $P = c$, we let $P[S] = P$.
2. Let $P = \text{case } x_i(Q) \text{ of } \{a \Rightarrow P_a\}$:
   2.1. If $i \notin S$, let
   $$P[S] = \text{case } x_i(Q[S \cup \{i\}]) \text{ of } \{a \Rightarrow P_a[S]\}$$
   2.2. If $i \in S$, and $Q = c$ we let
   $$P[S] = \text{case } x_i(c) \text{ of } \{a \Rightarrow P_a[S]\}.$$
   2.3. If $i \in S$ and $Q = \text{case } x_j(R) \text{ of } \{b \Rightarrow Q_b\}$, we first let
   $$\hat{P} = \text{case } x_j(R) \text{ of } \{b \Rightarrow \text{case } x_i(Q_b) \text{ of } \{a \Rightarrow P_a\}\}$$
   and then let $P[S] = \hat{P}[S]$.

Lemma 2.8. Let P be a finite sequential procedure, $S \subset \{1, \ldots, n\}$.

 a) $P[S]$ is well defined, i.e. the rewriting terminates.
 b) If $F = [\![P]\!]$, then $F_S = [\![P[S]]\!]$.

Proof. a) is proved by finding a suitable complexity measure as follows: When P is a finite sequential procedure, the evaluation of $[\![P]\!](\vec{f})$ will have a bounded finite length, even if we insist on evaluating $[\![Q]\!](\vec{f})$ to a value in $\mathbb{N}_\perp$ when f_i is not strict and $f_i([\![Q]\!](\vec{f}))$ is asked for in the process. For input $\vec{f}$ we will let the length of the evaluation of $[\![P]\!](\vec{f})$ be the number of evaluations $[\![Q]\!](\vec{f})$ we thus have to perform in the process. Then we use induction on the maximal possible length of an evaluation. In Case 2.3 $\hat{P}$ will have lower complexity than P since we omit the need to evaluate $Q(\vec{f})$, while both in the evaluation of $[\![P]\!](\vec{f})$ and $[\![\hat{P}]\!](\vec{f})$ we have to evaluate $[\![R]\!](\vec{f})$, some $[\![P_a]\!](\vec{f})$ and some $Q_b(\vec{f})$ (with all their sub-evaluations), though in different orders in the two cases. For the rest of the cases, it is easy to see how the induction hypothesis is used.

In order to prove b) we will use induction on the number of rewritings following the clauses of Definition 2.7 we need in order to find $P[S]$, and prove that $[\![P[S]]\!](\vec{f}) = [\![P]\!]_S(\vec{f})$ for each input $\vec{f}$. We go through each of the steps:

1.   This case is trivial
2.1. There will be two sub-cases:
    Case 1 - f_i is strict: Then $[\![Q]\!]_S(\vec{f}) = [\![Q]\!]_{S \cup \{i\}}(\vec{f})$ by construction, and the case follows from the induction hypothesis.
    Case 2 - f_i is not strict: Then it does not matter whatever we rewrite Q to, and by the induction hypothesis $[\![P_a[S]]\!](\vec{f}) = [\![P_a]\!]_S(\vec{f})$, so the conclusion holds.
2.2. This case is trivial.
2.3. In this case we use that when f_i is strict, then $[\![P]\!](\vec{f}) = [\![\hat{P}]\!](\vec{f})$, and the rest follows by the induction hypothesis. □

Definition 2.9. Let P be a sequential procedure. We say that P is in *S-normal form* if $P_S = P$.

A finite sequential procedure in S-normal form will have the cardinality of

$$\{1, \dots, n\} \setminus S$$

as a left bound.

3. The main Theorem

In this section, we will make use of some basic facts about the finite sequential functionals proved in e.g. [4, 8]. One fact is that any sequential functional will be the least upper bound of a sequence of finite ones, in the sense that the graph is the union of the graphs of the approximations. Another fact is that if two finite sequential functionals F and G are bounded by a sequential functional H, then there is a sequential least upper bound $F \sqcup G$ of the two. This is not necessarily the set theoretical union of F and G:

Example 3.1. Let F and G of type $(\iota \to \iota) \to \iota$ be defined by
- $F(f) = 0$ if $f(0) = 0$
- $G(f) = 0$ if $f(1) = 0$

Then the least sequential upper bound will be the constant zero, while the least set theoretical upper bound will just be

$$H(f) = 0 \text{ if } f(0) = 0 \text{ or } f(1) = 0.$$

The point is that $H(f)$ is defined when $f(0) = 0$ and $f(1) = \bot$ and when $f(0) = \bot$ and $f(1) = 0$ and a sequential procedure cannot handle this.

3.1. **Theorem and Key Lemma.** This section will be devoted to the proof of the following

Theorem 3.2. Let $\{F_k\}_{k \in \mathbb{N}}$ be an increasing sequence of sequential functionals of type $(\iota \to \iota)^n \to \iota$. Then the least upper bound

$$\bigsqcup_{k \in \mathbb{N}} F_k,$$

seen as a function, will be sequential.

If we do not allow for non-strict arguments at all, this theorem is part of the folklore, and is indeed covered by the more general result due to Plotkin mentioned in the introduction. The theorem will be a special case of the Key Lemma, see Lemma 3.5. The Key Lemma will be proved by induction, and the mentioned folklore result can be seen as the base case.

Even in the simplest case, for $n = 1$, there will be an increasing sequence $\{F_k\}_{k \in \mathbb{N}}$ of finite sequential functionals of type $(\iota \to \iota) \to \iota$ that can be enumerated in a primitive recursive way such that the least upper bound has the complexity of the Turing jump:

Example 3.3. Let $A = \bigcup_{k \in \mathbb{N}} A_k$ be a subset of $\mathbb{N}$ where A is Turing equivalent to the Turing jump, each A_k is finite with $A_k \subseteq A_{k+1}$ for each k and such that the relation $n \in A_k$ is primitive recursive.

Let $F_k(f) = 0$ if $f(0) \in A_k$ or if both $f(0) \le k$ and $f(1) \le k$. The least upper bound F will be a total functional of pure type 2. Even if we allow for repetitions of queries, there will be no computable sequential procedure for F. For any such procedure we will have that $f(0) \notin A$ if and only if $f(1)$ is eventually called for. This means that the complement of A is computable in any procedure for F.

Thus it is impossible to give a constructive proof of our main result. Arithmetical comprehension will suffice, but we will not put stress on pointing out where non-constructive arguments are used.

Throughout Section 3 we will let $\{F_k\}_{k \in \mathbb{N}}$ be a $\sqsubseteq$ - increasing sequence of finite sequential functionals and let $F = \bigsqcup_{k \in \mathbb{N}} F_k$ be the pointwise least upper bound. As noted in the start of the section, this suffices. We will use the terminology of the previous section.

Lemma 3.4. Without loss of generality, we may assume that $F_k \in \mathbf{F}_k$ for each $k \in \mathbb{N}$ and that $F_k = \pi_k(F_l)$ when $k \le l$.

Proof. Let

$$G_k = \bigsqcup_{k \le l} \pi_k(F_l).$$

Since $\mathbf{F}_k$ is finite, these least upper bounds exist as sequential functionals. Clearly the sequences $\{G_k\}_{k \in \mathbb{N}}$ and $\{F_k\}_{k \in \mathbb{N}}$ will have the same least upper bounds. □

From now on we will assume that the sequence we consider satisfies the conclusion of this lemma. We will prove the following strengthening of Theorem 3.2 by reversed induction on the size of S:

Lemma 3.5 (Key Lemma)**.**

$$\bigsqcup_{k \in \mathbb{N}} (F_k)_S$$

is sequential.

The proof of the Key Lemma will take what remains of this section.

3.2. **Matching paths.** We have to develop some notation and prove some partial results before we are able to give the proof of Lemma 3.5. The argument will be the same for all S,

except that for $S = \{1, \ldots, n\}$ we will never be in a situation where we have to appeal to the induction hypothesis.

Definition 3.6.

a) A sequential functional F is an *S-functional* if $F_S = F$.
b) An *S-query* is a pair (i, G) where $1 \leq i \leq n$ and
   - G is a constant $a \in \mathbb{N}$ if $i \in S$
   - G is an $S \cup \{i\}$-functional if $i \notin S$
c) An *S-pre-path* is a finite sequence
$$(i_1, G_1, b_1), \ldots, (i_m, G_m, b_m)$$
   where each (i_j, G_j) is an S-query and each $b_j \in \mathbb{N}$.
d) Let $\vec{f} = (f_1, \ldots, f_n)$ be a sequence of functions in $\mathbb{N}_\perp \to \mathbb{N}_\perp$ and let
$$\theta = (i_1, G_1, b_1), \ldots, (i_m, G_m, b_m)$$
   be an S-pre-path. We say that $\vec{f}$ and θ *match* if $f_{i_j}(G_j(\vec{f})) = b_j$ for each $j \leq m$.
e) An *S-path* is an S-pre-path that is matching at least one sequence $\vec{f}$.

An S-path $(i_1, G_1, b_1), \ldots, (i_m, G_m, b_m)$ will satisfy the following consistency condition: If $i_j = i_l$ and G_j and G_l are consistent, then $b_j = b_l$. This will not necessarily hold for pre-paths. If P is a finite sequential procedure in S-normal form and
$$\theta = (i_1, G_1, b_1), \ldots, (i_m, G_m, b_m)$$
is an S-path, we may *evaluate P on θ* as follows:

- If P is a constant b we output b directly.
- Let $P = \text{case } x_i(Q) \text{ of } \{b \Rightarrow P_b\}$. If there is a $j \leq m$ such that $i_j = i$ and $G_j \sqsubseteq [\![Q]\!]$ we let b_j be the first intermediate value of the evaluation, and continue with evaluating P_{b_j} on θ. If there is no such $j \leq m$, the evaluation stops with output $\perp$. We write $P(\theta)$ for the output of this process.

The paths will be forms of approximations to input sequences in the sense that they will carry the information about an input sequence we may have obtained after a partial evaluation of a sequential procedure.

Remark 3.7. In the definition of how to evaluate a finite procedure on a path, we almost implicitly assume that the functionals G_j in the path are themselves finite, or at least bounded by finite objects, since we require that $G_j \sqsubseteq [\![Q]\!]$. We are going to make use of paths where the G's are infinite sequential functionals obtained by the induction hypothesis. We will have to work carefully with approximations in order still to be able to evaluate a procedure on a path in a reasonable sense.

It is clear that if $\vec{f}$ matches $\theta = (i_1, G_1, b_1), \ldots, (i_m, G_m, b_m)$ and P is a sequential procedure in S-normal form that evaluated on θ yields a value c, then $[\![P]\!](\vec{f}) = c$. As a consequence, we see that if we evaluate two equivalent procedures in S-normal forms on the same S-path, the results will be consistent, meaning that if they both terminate, the output will be the same. We do not think that the outputs have to be equal, i.e. one may terminate and the other not, but have not produced an example.

Definition 3.8. Let F be an S-functional, $\theta = (i_1, G_1, b_1), \ldots, (i_m, G_m, b_m)$ an S-path. An S-query (i, G) is *critical* for F at θ if for every sequence $\vec{f}$ matching θ for which $F(\vec{f})\!\downarrow$ we have that $f_i(G(\vec{f})) \in \mathbb{N}$.

If $\vec{f}$ is matching θ and $f_i(G(\vec{f})) \in \mathbb{N}$, then $\vec{f}$ will match θ extended with $(i, G, f_i(G(\vec{f})))$. Our strategy will be to use S-queries critical for F to build up paths matching $\vec{f}$ in a deterministic way until, when $F(\vec{f}) \in \mathbb{N}$, we have sufficient information about $\vec{f}$ to know the value of $F(\vec{f})$.

Lemma 3.9. Let $F = [\![P]\!]$ be a finite S-functional, where P is a procedure in S-normal form. Let θ be a path such that P evaluated on θ does not yield a value in $\mathbb{N}$. Assume that there is some $\vec{f}$ matching θ such that $F(\vec{f}) \in \mathbb{N}$. Then there is an S-query (i, G) that is critical for F at θ, and moreover, such that there is no $j \le m$ with $i_j = i$ and $G_j \sqsubseteq G$.

Proof. For any $\vec{f}$ that matches θ, the evaluation of P on $\vec{f}$ will follow the same path through the procedure tree as the evaluation of P on θ, until the latter comes to a halt. This must be because the evaluation reached a subprocedure

$$P' = \text{case } x_i(Q) \text{ of } \{a \Rightarrow P'_a\}$$

while there is no $j \le m$ such that $i = i_j$ and $G_j \sqsubseteq [\![Q]\!]$. For the evaluation on $\vec{f}$ to proceed, we must have that $f([\![Q]\!](\vec{f})) \in \mathbb{N}$. Let $G = [\![Q]\!]$. Then (i, G) will be critical for F at θ and satisfy the extra requirement. $\qquad\square$

3.3. **Projections and preservation.** One problem with the concepts we have defined so far is that if P is a procedure that we may evaluate on a path θ, and we extend the G_j's in the path to some G'_j's, then the evaluation may not work anymore. This is not so strange as it may seem. If we extend the items of a path, the path will match more function sequences $\vec{f}$, also some for which the original procedure does not terminate. However, it is in order to handle the technical problem induced by this phenomenon that we assume that our increasing sequence $\{F_k\}_{k \in \mathbb{N}}$ has the properties of Lemma 3.4. We will take the liberty to extend the use of the term π_k for the projection to $\mathbf{F}_k$ for all types in question, in particular to $\mathbf{I}_k$ and $\mathbf{I}_k^n$ defined by:

Definition 3.10.
 a) Let $\mathbf{I}$ be the functions of type $\iota \to \iota$, and let $\mathbf{I}_k$ be the corresponding set of functions in $\{\bot, 0, \ldots, k\} \to \{\bot, 0, \ldots, k\}$.
 b) If (i, G) is an S-query, we let $\pi_k(i, G) = (i, \pi_k(G))$.
 c) If $\theta = (i_1, G_1, b_1), \ldots, (i_m, G_m, b_m)$ is a path, we let

$$\pi_k(\theta) = (i_1, \pi_k(G_1), b_1), \ldots, (i_m, \pi_k(G_m), b_m).$$

Notice that $\pi_k(\theta)$ is not necessarily a path, since it does not have to match any $\vec{f}$ if too much information is removed.

Lemma 3.11.
 a) *If $F \in \mathbf{F}$ and $\vec{f} \in \mathbf{I}_k^n$, then $\pi_k(F(\vec{f})) = \pi_k(F)(\vec{f})$.*
 b) *If $F \in \mathbf{F}_k$ and $\vec{f} \in \mathbf{I}^n$, then $F(\vec{f}) = F(\pi_k(\vec{f}))$.*
 c) *If $\vec{f} \in \mathbf{I}_k^n$ and θ is a path matching $\vec{f}$, then $\pi_k(\theta)$ is a path matching $\vec{f}$.*

Proof. a) and b) follows from the iterative construction of the projection π_k over the types. The definition of matching requires that for any (i, G, b) in θ we must have that $f_i(G(\vec{f})) = b$. For this to hold, we must either have that f_i is constant $b \le k$ or that $G(\vec{f}) \le k$. In the first case, the claim is trivial, and in the second case we may use $a)$. This establishes c). $\qquad\square$

We will extend the π_k also to procedures:

Definition 3.12. If P is a procedure, we let $\pi_k(P)$ be the procedure we obtain by the following top-down recursive transformation:
- $\pi_k(a) = a$ if $a \leq k$ and $\perp$ if $k < a$
- If $P = \text{case } x_i(Q)$ of $\{a \Rightarrow P_a\}_{a \in \mathbb{N}}$, we let $\pi_k(P) = \text{case } x_i(\pi_k(Q))$ of $\{a \Rightarrow \pi_k(P_a)\}_{a \leq k}$

It is easy to see that $\pi_k(\llbracket P \rrbracket) = \llbracket \pi_k(P) \rrbracket$, because any evaluation of P with an input from $\mathbf{I}_k^n$ can only utilise values bounded by k in the evaluation.

Lemma 3.13. Let θ be a path, F a functional and (i, G) be critical for F at θ. Then $(i, \pi_k(G))$ is critical for $\pi_k(F)$ at $\pi_k(\theta)$.

Proof. Let $\vec{f} \in \mathbf{I}_k^n$ be such that $\pi_k(\theta)$ matches $\vec{f}$ and such that $\pi_k(F)(\vec{f})\downarrow$. Then θ matches $\vec{f}$ and $F(\vec{f})\downarrow$, so $f_i(G(\vec{f}))\downarrow$. By Lemma 3.11 then, $f_i(\pi_k(G)(\vec{f}))\downarrow$. $\qquad\square$

3.4. **Proof of the Key Lemma.** Recall that we will prove this lemma by reversed induction on the cardinality of S. We will now let the sequence $\{F_k\}_{k \in \mathbb{N}}$ be given, satisfying the assumption of Lemma 3.4, and we let $\vec{f} \in \mathbf{I}^n$ be given. We have that $F_S = \bigsqcup_{k \in \mathbb{N}}(F_k)_S$, and we will devise a sequential procedure for evaluating $F_S(\vec{f})$. To this end, we will construct a path matching $\vec{f}$ step by step until we have enough information about $\vec{f}$ to know the value of $F_S(\vec{f})$. We will prove that if $F_S(\vec{f})\downarrow$, then our procedure applied to $\vec{f}$ will come to a halt, and that whenever it halts, the conclusion will be the right one for the given input.

For each $k \in \mathbb{N}$, let P_k be in S-normal form such that $(F_k)_S = \llbracket P_k \rrbracket$. We will describe one step in the construction:

Let $\theta = (i_1, G_1, b_1), \ldots, (i_m, G_m, b_m)$ be the path matching $\vec{f}$ constructed after step m.
(1) If there is some k such that $P_k(\pi_k(\theta))\downarrow$ we know that $F_S(\vec{f}) = P_k(\pi_k(\theta))$ and we output this value.
(2) If there is no $\vec{g}$ such that $\vec{g}$ matches θ and $F_S(\vec{g})\downarrow$, we know that $F_S(\vec{f}) = \perp$, and our procedure applied to $\vec{f}$ comes to a halt with output $\perp$.
(3) Otherwise, let k be minimal such that for some $\vec{g} \in \mathbf{I}_k^n$ matching θ we have that $(F_k)_S(\vec{g})$ terminates. Select one such $\vec{g}$. Notice that we can split between the cases 1., 2. and 3. on the basis of θ, and that the values of k and $\vec{g}$ in this case only depend on θ, so we do not need to know $\vec{f}$ in order to know what to do.

In case of alternative (3), we will see how to extend θ to a longer path matching $\vec{f}$:

Let $l \geq k$. Then $P_l(\vec{g})\downarrow$ while $P_l(\pi_l(\theta)) = \perp$. By Lemma 3.9 there will be an S-query (i_l, H_l) that is critical for $(F_l)_S$ at $\pi_l(\theta)$ such that there is no $j \leq m$ with $i_l = j$ and $\pi_l(G_j) \sqsubseteq H_l$. Then, by Lemma 3.13, $(i_l, \pi_k(H_l))$ will be critical for $\pi_k(F_k)$ at $\pi_k(\theta)$. Since $g_{i_l}(H_l(\vec{g}))\downarrow$ and $\vec{g} \in \mathbf{I}_k^n$, we have that $g_{i_l}(\pi_k(H_l)(\vec{g}))\downarrow$. Since $\pi_k(F_l) = F_k$ we also have that $\pi_k(P_l)$ is a procedure for F_k, and the evaluation of $\pi_k(P_l)(\vec{g})$ will match the evaluation of $\pi_k(P_l)(\pi_k(\theta))$ for as long as the latter evaluation goes. It follows that $(i_l, \pi_k(H_l))$ will be the query unanswered by $\pi_k(\theta)$ that we get when proving Lemma 3.9 for $\pi_k(P_l)$, $\pi_k(\theta)$ and $\vec{g}$. These considerations contain a proof of the following claim:

Claim: *For each $l \geq k$, let X_l be the set consisting of pairs (i, H) that are critical S-queries for $(F_l)_S$ at $\pi_l(\theta)$ and such that for no j we have that $i = j$ and $\pi_k(G_j) \sqsubseteq \pi_k(H)$. Then X_l is nonempty.*

If $k \leq l_1 \leq l_2$, then π_{l_1} will map X_{l_2} into X_{l_1}, and each X_l is finite. We use König's lemma to see that there will be one $i \leq n$ and one increasing sequence $\{H_l\}_{l \geq k}$ of $S \cup \{i\}$-functionals, with each $(i, H_l) \in X_l$. We select one such sequence (depending on θ, but not on $\vec{f}$).

If $i \in S$ we actually have that each H_l's is a constant, and that they are all the same. In this case, we let H be this constant. If $i \notin S$, we use the induction hypothesis, and let $H = \bigsqcup_{l \geq k} H_k$ be sequential.

If $f_i(H(\vec{f})) = \bot$ our procedure for $F_S(\vec{f})$ will not terminate, which is as it should be since (i, H) is critical at θ, so $F_S(\vec{f})$ will not terminate either. If $f_i(H(\vec{f})) = b \in \mathbb{N}$, we add (i, H, b) to our path and continue the process. This is the only place where we actually use $\vec{f}$, and clearly, the new path will also match $\vec{f}$.

This ends one full step in the construction of the path matching $\vec{f}$. Clearly, if this process on $\vec{f}$ leads to a conclusion after finitely many steps, the output in $\mathbb{N}_{\bot}$ will coincide with $F_S(\vec{f})$. It remains to prove that if $F_S(\vec{f})\downarrow$, then the process will terminate.

Let $F_S(\vec{f})\downarrow$ and let k_0 be minimal such that $F_S(\pi_{k_0}(\vec{f}))\downarrow$. Without loss of generality we may assume that $\vec{f} \in \mathbf{I}_{k_0}^n$, and then $(F_{k_0})_S(\vec{f})\downarrow$.

When we apply our strategy to the input $\vec{f}$, we will build up a path matching $\vec{f}$, and as long as the S-normal procedure for $(F_{k_0})_S$ does not terminate on the path built up, the k of the construction will be bounded by k_0. But as long as this is the case, we are properly extending the path even when projected to the k_0-level. This level is finite, so this cannot go on forever. The conclusion is that our process, when applied to $\vec{f}$, must terminate.

What remains is to summarise how we construct a sequential procedure from the S-paths. The tree of paths depends only on S, F, the sequence of F_k's and the choice of $\vec{g}$ in each step, and which branch, or path, we follow in that tree depends deterministically on $\vec{f}$. Each item (i, G, b) in one of the S-paths constructed will correspond to an instruction

$$\text{case } x_i(G) \text{ of } b \Rightarrow \cdot$$

in the sequential procedure for F_S that we actually construct. It is easy to see that this top-down translation of the tree of S-paths to a sequential procedure will work.   □

4. Conclusions and discussion

In this note we have not just completed the characterisation of when the sequential functionals of a type σ form a *dcpo*. As a bonus, we have proved that the sequential functionals of the types $(\iota \to \iota)^n \to \iota$ are what is called "left bounded" in [4], and with a uniform bound on the left nesting needed. One key aspect of the examples in [7, 8] is that the sequence without a sequential upper bound in both cases will have increasing left depth, and in the limit there has to be an evaluation with an infinite left nesting of sub-procedure calls. It is then natural to ask if the substructure of the sequential functionals of any type with a fixed bound on the left nesting will form a *dcpo*. This seems likely, but since these classes in general do not seem to possess other interesting closure properties, a result along these lines will be of limited interest.

In Section 7 of [8] some further open problems were listed, and we rephrase them here. It is still not known exactly when a finite sequential functional is the least upper bound of some non-trivial increasing sequence of sequential functionals. We know that a finite set of finite sequential functionals with a sequential upper bound will have a least sequential

upper bound, but this is unknown for finite classes of sequential functionals in general. We may also ask if a stronger property may hold, will every finite bounded set of sequential functionals have a sequential least upper bound? These problems may be worthwhile to look into, not because the results will be of particular interest in themselves, but because solving the problems may lead to a better understanding of the sequential functionals.

Acknowledgement

References

[1] Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF. Information and Computation **163(2)**, 409–470 (2000)
[2] Hyland, J.M.E., Ong, C.-H.L.: On full abstraction for PCF: I, II, and III. Information and Computation **163(2)** , 285–408 (2000)
[3] Kahn, G., Plotkin, G.D.: Concrete domains. Theoretical Computer Science **121**, 178–277 (1993), first appeared in French as INRIA technical report (1978)
[4] Longley, J.R., Normann, D.: Higher-Order Computability, Springer Verlag, 2015
[5] Milner, R.: Fully abstract models of typed λ-calculi. Theoretical Computer Science **4(1)**, 1–22 (1977)
[6] Nickau, H.: Hereditarily sequential functionals. In: Third International Symposium on Logical Foundations of Computer Science, St. Petersburg, 1994, pp. 253-264. Springer, Heidelberg (1995)
[7] Normann, D.: Sequential functionals of type 3. Mathematical Structures in Computer Science **16(2)**, 279–289 (2006)
[8] Normann, D., Sazonov, V.Yu.: The extensional ordering of the sequential functionals. Annals of Pure and Applied Logic **163(5)** 575–603 (2012)
[9] Sazonov, V.Yu.: Expressibility in D. Scott's LCF language. Algebra and Logic **15(3)**, 192–206 (1976)
[10] Scott, D.S.: A type-theoretical alternative to ISWIM, CUCH, OWHY. Unpublished note (1969)