Computability 00 (20xx) 1–36
DOI 10.3233/COM-200321
IOS Press

Computability and non-monotone induction

Dag Normann
Department of Mathematics, The University of Oslo, Norway
dnormann@math.uio.no

Abstract. Non-monotone inductive definitions were studied in the late 1960's and early 1970's with the aim of understanding connections between the complexity of the formulas defining the induction steps and the ordinals measuring the rank of the inductive process. In general, any type 2 functional will generate an inductive process, and in this paper we will view non-monotone induction as a functional of type 3. We investigate the associated computation theory of this functional inherited from the Kleene schemes and we investigate the associated companion of sets with codes computable in the functional of non-monotone induction. The interest in this functional is motivated from observing that constructions via non-monotone induction appear to be natural in classical analysis.

There are two groups of results: We establish strong closure properties of the least ordinal without a code computable in the functional of non-monotone induction, and we provide a characterisation of the class of functionals of type 3 computable from this functional, a characterisation in terms of sequential operators working in transfinite time. We will also see that the full computational power of non-monotone induction is required when this principle is used to construct functionals witnessing the compactness of the Cantor space and of closed, bounded intervals.

Keywords: Non-monotone induction, Kleene computability, sequential procedures, reflection properties

1. Introduction

1.1. Motivation and history

With the introduction of set theory in the second half of the 19th century, mathematicians had more tools in their toolbox than before, they had a richer language in which to express mathematical properties, but they also had tools like transfinite recursion and the use of the axiom of choice. One of these tools, inspired from the new ordinal numbers introduced by Cantor, is non-monotone induction over the set of integers, seen as an operator of order four, or of type 3 in the terminology of type theory.

It is worth noticing that the set-theoretical language mostly used at the time is of third order, while coding is needed to capture the same concepts in second order arithmetic (SOA). In a series of papers [16–27], Sam Sanders and the author have investigated the logical and computability strength of some of the results using such tools, when expressed in a language close to how it was originally done. To be more precise, we worked in the framework of Kohlenbach [11] of higher order reverse mathematics combined with Kleene's [10] model for higher order computability.

Non-monotone inductive definitions were studied in the late 1960's and early 1970's, but the general interest has been low since then. Examples of papers on the subject are [1,2,4,29]. The inductive definitions were classified according to the complexity of the formulas defining them, and the key property of interest was the complexity of the corresponding closure ordinals. This could be expressed in terms of reflection properties as in [29] or by comparing classes of closure ordinals as in [1].

In this paper we will study non-monotone inductive definability over $\mathbb{N}$ via a functional $\mathcal{I}$ of type 3 (Definition 2.1), and investigate the computational strength of $\mathcal{I}$ in terms of Kleene's model for higher order computations. As there is no justifiable Church–Turing thesis for the computability theory of higher order functionals, Kleene computability is just one possible model, but since this model has proved to be fruitful for the analysis of discontinuous functionals of type 2, and for computability relative to the Superjump as defined by Gandy [6], see Harrington [8], Kleene computability is a natural model for the investigation of the computational strength of non-monotone induction. We also aim to characterise the class of functionals of type 3 computable in $\mathcal{I}$ in a way that is less model-dependent.

The motivation for bringing up non-monotone induction once again is the observation that this functional represents a natural upper complexity-bound for a number of functionals appearing as realisers for classical theorems, such as the Heine–Borel theorem and the Baire Category theorem, when these theorems are expressed in a higher order language and not within the restricted language of second order arithmetic.

The first application of non-monotone inductive definitions known to the author is due to E. Borel [3]. The motivation of Borel was to give a *direct* proof of the theorem now known as the Heine–Borel theorem. The assumption was that we are given a way to associate an open neighbourhood O_x to each x in a closed interval $[a, b]$ and the claim was that we can then explicitly find a finite sub-covering of this covering. In the terminology of today, Borel constructed a functional taking the map $x \mapsto O_x$ as the argument and yielding a finite subcovering as the value. The definition/construction of this functional is by transfinite recursion, building up finite subcoverings of larger and larger closed subintervals, a construction that can be viewed as a simultaneous non-monotone inductive definition of Dedekind cuts for numbers $c \leqslant b$ and finite subcoverings of each closed interval $[a, d]$ for $d < c$. In [16], a realiser Θ_0 of the uncountable Heine–Borel theorem (HBU) is defined. This realiser selects a finite set $x_1, \dots, x_n$ such that the corresponding open neighbourhoods form a subcovering. Θ_0 is essentially the operator constructed by Borel. It is proved in [16] that Θ_0, in conjunction with ${}^2E$, computes the Suslin functional (see below), and in Normann [15, Theorem 1(c)] it is shown that any realiser Θ of HBU as above, in conjunction with the Suslin operator, computes the functional $\mathcal{I}$ to be defined below. We will slightly improve this theorem, see Section 6.2.

Realisers Ξ for the Lindelöf lemma for Baire space $\mathbb{N}^{\mathbb{N}}$ (homeomorphic to the irrationals) is one class of functionals discussed in [15], where it is proved in Theorem 1 that any such realiser will compute $\mathcal{I}$ and that there is at least one such realiser computable in $\mathcal{I}$. Thus non-monotone induction reflects the complexity of witnessing the Lindelöf lemma in this special case.

In [20] the aim is to investigate real line topology with the purpose of classifying the complexity of theorems and concepts in terms of their *reverse mathematics* and *computational complexity*. Representations of open sets, such as being countable unions of rational neighbourhoods, are based on mathematical insight, and analysing the logical and computational strength of such insight is part of the aim of [20]. Given representations of open sets as in classical reverse mathematics, using second order arithmetic, the Baire Category Theorem is effective in the sense that given (a representation of) a sequence of dense open sets we can compute a fast-converging Cauchy-sequence for a point in the intersection. In [20, Theorem 6.5] it is proved that, using non-monotone induction, we can find a functional ξ taking a sequence $\{X_k\}_{k\in\mathbb{N}}$ of subsets of $\mathbb{R}$ as arguments and yielding an $x \in \mathbb{R}$ as value, such that whenever each X_k is dense and open then

$$\xi\big(\{X_k\}_{k\in\mathbb{N}}\big) \in \bigcap_{k\in\mathbb{N}} X_k.$$

In [20, Theorem 6.6] it is proved that no such functional ξ can be computable in any functional of type 2, but it remains open to decide if the full power of non-monotone induction is needed for obtaining a functional ξ like this.

1.2. Overview and results

In Section 2 we will define the functional $\mathcal{I}$ that is our main subject of investigation, and we will define the Kleene-computations via the schemes S1-S9 with $\mathcal{I}$ as the one argument of type 3. We observe two interpretations of these schemes, one where we follow Kleene and restrict the application scheme S8 to total inputs and one where we relax on this requirement. We show that the two interpretations lead to the same class of functions of type 1 computable in $\mathcal{I}$. We use this to prove what is known as *stage comparison* and *Gandy selection* for the interpretation using partial inputs.

In Section 3 we investigate the least ordinal π not computable in $\mathcal{I}$, and the associated companion L_π. We prove that the set of codes $f \in \mathbb{N}^{\mathbb{N}}$ of π is not computable in $\mathcal{I}$, and thus in particular not a Π^1_1-set (Corollary 3.8). We also establish a number of closure properties and reflection properties for L_π.

Section 4 is a preparation for Section 5. In Section 4 we introduce what we call *hyper-sequential procedures* and in Section 5 we narrow down this concept to *inductive procedures*. These procedures model nested systems of non-monotone inductions, using our new concept of *blockings* to organise the nesting. The inductive procedures can be used to characterise the class of functionals of type 3 computable in $\mathcal{I}$.

In Section 6 we look at some of the functionals serving as realisers for classical theorems in analysis, primarily theorems where the proof in some way depends on the compactness of the unit line or Cantor space. We will see that when such realisers are constructed in a natural way, they implicitly have the full power of non-monotone induction. In conjunction with the Suslin functional **S**, all realisers of the theorems in question will compute $\mathcal{I}$. We will illustrate how to use compactness for computing $\mathcal{I}$ in the proof of Lemma 6.9, a lemma that is a slight improvement of [15, Theorem 1 (c)].

In Section 7 we briefly discuss what it means to relativise these results to functionals of type 2 and in Section 8 we summarise the paper and discuss a few open problems.

2. Non-monotone induction and computability

2.1. Inductive definitions

Mathematically we can identify the Cantor set $\mathsf{C} = \{0,1\}^{\mathbb{N}}$ with the powerset $\mathcal{P}(\mathbb{N})$ of the integers, where we identify a set with its characteristic function. In this paper, we will use both notations, as it sometimes is essential that we consider the set C, the compact set of infinite binary sequences, and sometimes more natural to consider the set $\mathcal{P}(\mathbb{N})$ where the inclusion ordering is essential. The latter view will be relevant when we define non-monotone induction, but mathematically we use C as the formal definition of the set under consideration, and treat it as $\mathcal{P}(\mathbb{N})$ when this is convenient. When elements of C are viewed as characteristic functions, the point-wise ordering $\leqslant$ coincides with the inclusion ordering $\subseteq$.

Definition 2.1. Let $F : \mathsf{C} \to \mathsf{C}$ be a functional of type 2.

(a) We view F as an inductive definition, defining the increasing sequence f_β in C, where β runs over the countable ordinals, by transfinite recursion as follows:

  (1) f_0 is the constant zero
  (2) $f_{\beta+1} = \max\{f_\beta, F(f_\beta)\}$
  (3) If β is a limit ordinal, $f_\beta = \sup_{\gamma<\beta} f_\gamma$.

(b) There will, for cardinality reasons, be a least countable ordinal α_F such that $f_{\alpha_F} = f_{\alpha_F+1}$. Then α_F is the least ordinal α such that $F(f_\alpha) \leqslant f_\alpha$. We let $\mathcal{I}$ be defined by $\mathcal{I}(F) = f_{\alpha_F}$, with the notation introduced above.

If we need to point to the functional F, we write f^F_β.

Remark 2.2. If $F : \mathsf{C} \to \mathsf{C}$ is monotone, that is, if $f \leqslant g \to F(f) \leqslant F(g)$, we say that the corresponding induction is *monotone*. With our terminology, the class of non-monotone inductive definitions will include the class of monotone ones.

Remark 2.3. We are not fully in the realm of Kleene-computability, since this is developed for total functionals of pure type only. However, if G is of pure type 2, we may consider G as a code for

$$F_G(f)(n) = \min\{G(n\,\hat{}\,f), 1\},$$

where $f \in \mathsf{C}$ and with the standard concatenation-understanding of $n\,\hat{}\,f \in \mathbb{N}^{\mathbb{N}}$. Using standard coding, we my also consider $\mathcal{I}$ as a functional of pure type 3. For the sake of readability, we prefer to use a customised version of Kleene's definition, as defined in Section 2.2, when we investigate the computational strength of $\mathcal{I}$.[1]

Example 1. We view C as the powerset of $\mathbb{N}$ and let $G : \mathsf{C} \to \mathbb{N}$. For pure cardinality reasons, there must be $A \neq B \subseteq \mathbb{N}$ such that $G(A) = G(B)$, and by the axiom of choice there will be a functional Φ such that for every G, $\Phi(G)$ is such a pair. Now, the axiom of choice is not needed for this, as will be seen from an easy application of $\mathcal{I}$:

[1] In [25] we have introduced an alternative, but equivalent, version of Kleene computability. This approach is based on typed λ-calculus and deals directly with mixed types.

Given $G : \mathsf{C} \to \mathbb{N}$, let F_G be defined by $F_G(A) = A \cup \{G(A)\}$. We then see that the transfinite iteration of F_G will generate a strictly increasing sequence of sets $\{A_\beta\}_{\beta \leqslant \alpha}$ exactly until we have an α, and a $\beta < \alpha$, such that $G(A_\beta) = G(A_\alpha)$.

In [22] the complexity of such functionals Φ witnessing that there is no injection from C to $\mathbb{N}$ is studied in more detail, and it is proved that no such functional can be computed from an object of type two.

2.2. Kleene computability

Kleene [10] defined a relation $\{e\}(\vec{\Phi}) = a$, in the form of a positive inductive definition with nine cases, where e is an *index*, a natural number that serves as a Gödel number for a generalised algorithm, and $\vec{\Phi}$ is a sequence of functionals of pure types in the type-structure of total functionals. The nine cases in the definition are called *schemes* and are numbered as S1–S9. For a recent introduction to Kleene computability, see Longley and Normann [12, Chapter 5]. For an alternative approach to Kleene computability, see [25].

In this section we will mainly be concerned with computations of the form

$$\{e\}(\mathcal{I}, \vec{F}, \vec{f}, \vec{a})$$

where $\vec{F}$ is a sequence of functionals of type 2, $\vec{f}$ is a sequence of functions of type 1 and $\vec{a}$ is a sequence from $\mathbb{N}$. In Definition 2.4 we will restrict S1–S9 to this case. In Section 4.2 we will give a more general version of S8, accommodated to the content of that section. Our version of S8 here, when restricted to the use of $\mathcal{I}$ as the only object of type 3, will be equivalent to using the version of S8 in Section 4.2 to the functional of pure type 3 that will represent $\mathcal{I}$.

Definition 2.4. Using transfinite recursion, we define the relation $\{e\}(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = c$, where $\mathcal{I}$ is as defined, $\vec{F} = (F_1, \dots, F_m)$ is a sequence from $\mathbb{N}^{\mathbb{N}} \to \mathbb{N}$, $\vec{f} = (f_1, \dots, f_n)$ is a sequence from $\mathbb{N}^{\mathbb{N}}$, $\vec{a} = (a_1, \dots, a_k)$ is a sequence from $\mathbb{N}$ and $c \in \mathbb{N}$, as follows.

S1 If $e = \langle 1 \rangle$, then $\{e\}(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = a_1 + 1$.
S2 If $e = \langle 2, q \rangle$, then $\{e\}(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = q$.
S3 If $e = \langle 3 \rangle$, then $\{e\}(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = a_1$.
S4 If $e = \langle 4, e_1, e_2 \rangle$, $\{e_2\}(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = b$ and $\{e_1\}(\mathcal{I}, \vec{F}, \vec{f}, b, \vec{a}) = c$, then $\{e\}(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = c$.
S6 If $e = \langle 6, e_1, \tau_1, \tau_2, \tau_3 \rangle$, where τ_1, τ_2 and τ_3 are (integer codes of) permutations of (the index sets for) the input sequences $\vec{F}$, $\vec{f}$ and $\vec{a}$, then $\{e\}(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = \{e_1\}(\mathcal{I}, \vec{F}_{\tau_1}, \vec{f}_{\tau_2}, \vec{a}_{\tau_3})$.
S7 If $e = \langle 7 \rangle$, then $\{e\}(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = f_1(a_1)$.
S8 For this scheme there will be subcases, one for each type > 1. For us, there will be two subcases, where the case for type 3 is where we adjust the definition to application of $\mathcal{I}$:

2. If $e = \langle 8, 2, d \rangle$ then $\{e\}(\mathcal{I}, \vec{F}, \vec{f}, b, \vec{a}) = F_1(g)$ when $g(a) = \{d\}(\mathcal{I}, \vec{F}, \vec{f}, a, \vec{a})$ is a total function. We write

$$\{e\}(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = F_1\big(\lambda a.\{d\}(\mathcal{I}, \vec{F}, \vec{f}, a, \vec{a})\big).$$

3. If $e = \langle 8, 3, d \rangle$ we let $\{e\}(\mathcal{I}, \vec{F}, \vec{f}, b, \vec{a}) = \mathcal{I}(F_G)(b)$ where $G(f) = \{d\}(\mathcal{I}, \vec{F}, f, \vec{f}, \vec{a})$.

S9 If $e = \langle 9 \rangle$ then $\{e\}(\mathcal{I}, \vec{F}, \vec{f}, d, \vec{a}) = c$ if $\{d\}(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = c$.

Remark 2.5. We have excluded S5, the scheme of primitive recursion, from our definition. There are two reasons for this. The main reason is that one may prove the recursion theorem on the basis of the other schemes, and thus S5 will be redundant. The other reason is that, since recursion is iterated composition, all arguments involving S5 that we need will be covered by how we deal with S4.

Kleene computability inherits several of the key properties of classical computability, such as the $\mathsf{S}_{n,m}$-theorem and the recursion theorem. The existence of universal algorithms is *axiomatised* in the form of S9. In the sequel, we will assume familiarity with these basic properties.

2.3. The computability theory of $\mathcal{I}$

We first prove that the prototype of discontinuity is computable in $\mathcal{I}$.

Definition 2.6. We define the functional ${}^2E$ of type 2 by

$$ {}^2E(f) = \begin{cases} 0 & \text{if } \forall k(f(k) = 0) \\ 1 & \text{if } \exists k(f(k) > 0) \end{cases} $$

Lemma 2.7. *The functional ${}^2E$ is computable in $\mathcal{I}$.*

Proof. Given $f \in \mathbb{N}^{\mathbb{N}}$, we want to decide if $\exists k(f(k) > 0)$. Let

$$ F_f(A) = \{k : f(k) > 0\} \cup \{k : k + 1 \in A\}. $$

Then $\exists k(f(k) > 0)$ if and only if $0 \in \mathcal{I}(F_f)$. □

Remark 2.8. ${}^2E$ is sometimes denoted as $\exists^2$, and is equivalent, within S1–S9, to Feferman's μ.

The Suslin functional $\mathbf{S}$ is defined by

$$ \mathbf{S}(f) = \begin{cases} 0 & \text{if } \forall g\, \exists n(f(\bar{g}(n)) = 0) \\ 1 & \text{if } \exists g\, \forall n(f(\bar{g}(n)) > 0) \end{cases} $$

Lemma 2.9. *The Suslin functional* $\mathbf{S}$ *is computable in $\mathcal{I}$.*

Proof. We use that ${}^2E$ is computable in $\mathcal{I}$. Given f, we let T_f be the tree of finite sequences $s = (s_0, \ldots, s_{n-1})$ such that we for all $m \leqslant n$ have that $f(\langle s_0, \ldots, s_{m-1}\rangle) = 0$. Then $\mathbf{S}(f) = 0$ if and only if T_f is well founded. For all f, the subset of T_f consisting of all sequences that cannot be extended to an infinite branch in T_f can be defined using an arithmetical inductive definition, and we then use ${}^2E$ to decide if this subset is the whole tree T_f. □

2.4. Totality vs. partiality

In the original definition of higher order computability via Kleene's S1–S9, all objects were assumed to be total. This can be considered to be a defect of S8, where the input $\lambda\xi.\{d\}(\xi, - - --)$ has to be defined for all ξ of the type in question in order to accept the termination of $\Psi(\lambda\xi.\{d\}(\xi, - - --))$, even if Ψ is defined in such a way that it only requires some values of the input functional. In the case of $\mathcal{I}$, we only need F to be total on the set of functions f_β for $\beta \leqslant \alpha_F$ in order to identify $\mathcal{I}(F)$, and insisting that F is defined on irrelevant arguments as well may seem as an unnatural restriction on what is considered to be an algorithm relative to $\mathcal{I}$.

Remark 2.10. A similar phenomenon takes place for Gandy's Superjump $\mathbb{S}$, introduced in [6]. The superjump is defined by

$$ \mathbb{S}(F, e) = \begin{cases} 1 & \text{if } \{e\}(F, e) \downarrow \\ 0 & \text{if } \{e\}(F, e) \uparrow \end{cases} $$

where $\downarrow$ means that there is a value of the computation, while $\uparrow$ means the converse. In order to find the value of $\mathbb{S}(F, e)$ we only need to know F restricted to the set of f computable in F, the so called *1-section* of F. This was used by Harrington [8] in an essential way when he classified the computational strength of $\mathbb{S}$, and was also important in Hartley's [9] analysis of the countably based functionals (See Section 4 for a further discussion). We will show that loosening up the requirement of totality of the input functional to $\mathcal{I}$ does not add to the computational strength of $\mathcal{I}$. This is as it is for $\mathbb{S}$, but not, for instance, as for computations with continuous inputs in general. Then we add considerable strength by relaxing on S8, see e.g. [12, Sections 6.4 and 8.5] for results and further references.

Definition 2.11. We write $\{e\}_t(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = b$ if $\{e\}(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = b$ according to the original definition, while we write $\{e\}_p(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = b$ if we interpret S8 according to the following extension of $\mathcal{I}$ to partial $F : \mathsf{C} \to \mathsf{C}$. We will not accept non-total inputs to F, and for each $f \in \mathsf{C}$ we either have that $F(f) \in \mathsf{C}$ or $F(f)$ is totally undefined. We stick to the notation from Section 2.1:

(i) By recursion on the ordinal β, f^F_β is *defined* if $\beta = 0$ or $\beta > 0$ and both f^F_γ and $F(f^F_\gamma)$ are defined for all $\gamma < \beta$.
(ii) $\mathcal{I}(F)$ is *defined* if there is an ordinal α such that $f^F_{\alpha+1}$ is defined and $f^F_\alpha = f^F_{\alpha+1}$
(iii) If $\mathcal{I}(F)$ is defined, and α is as in (ii), $\mathcal{I}(F)(n) = f^F_\alpha(n)$ for each $n \in \mathbb{N}$.

When the context is clear, we will talk about t-computations and p-computations.

Theorem 2.12. *There is a computable (in the sense of Turing) function ρ such that if $\{e\}_p(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = b$, then $\{\rho(e)\}_t(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = b$.*

Proof. We use the recursion theorem to define ρ, and define it by cases according to S1–S9. It is obvious what to do in all cases except application of $\mathcal{I}$. The final correctness proof will, of course, be by induction on the complexity of the $\{e\}_p$-computation (we will define the *rank* or *norm* of a terminating computation formally below, definitions that do not rely on the correctness of this theorem), but as is common for this kind of argument, we assume that ρ does the job on all subcomputations, and we define ρ by self-reference.

So assume that

$$\{e\}_p(\mathcal{I}, \vec{F}, \vec{f}, b, \vec{a}) = \mathcal{I}(F_{G_p})(b)$$

where $G_p(f) = \{d\}_p(\mathcal{I}, \vec{F}, f, \vec{f}, \vec{a})$, and that the recursion terminates as defined in Definition 2.11. Assume further, as an induction hypothesis, that we can replace G_p with the, still partial,

$$G_t(f) = \{\rho(d)\}_t(\mathcal{I}, \vec{F}, f, \vec{f}, \vec{a}).$$

We assume familiarity with the concept of a *prewellordering* R on a domain $D \subseteq \mathbb{N}$. Since ${}^2E$ and $\mathbf{S}$ are t-computable in $\mathcal{I}$, we also have that the set of prewellorderings will be t-computable in $\mathcal{I}$.

If R is a prewellordering on D, each element in the domain D will have an ordinal rank, and we let R_β be the elements in D with ordinal rank below β. We will construct a total functional H mapping prewellorderings to prewellorderings such that we can decide $b \in \mathcal{I}(F_{G_t})$ from $\mathcal{I}(H)$. The definition of $H(R)$ is as follows, observing that we only need ${}^2E$ when we know that R is a prewellordering. We let f_β be as in the definition of $\mathcal{I}(F_{G_t})$, and we *identify* f_β with $A_\beta = \{b \in \mathbb{N} : f_\beta(b) = 1\}$.

– By R-recursion, compare R_α with A_α until we either have disagreement or that $R_\alpha = A_\alpha$ with $F(A_\alpha) \subseteq A_\alpha$.

– In the first case, α must be a successor ordinal $\beta + 1$. We let $H(R)$ be R restricted to $R_\beta = A_\beta$, and then end-extended with $F(A_\beta) \setminus A_\beta$. In the other case we let $H(R)$ be R restricted to R_α.

Since we only will ask for values $F(A_\beta)$ in the computation of $H(R)$, our assumption shows that H is total. $\mathcal{I}(H)$ will be a prewellordering R, and we will have that it matches the prewellordering induced by F_{G_t}. We then have that

$$b \in \mathcal{I}(F) \Leftrightarrow b \in \operatorname{dom}\bigl(\mathcal{I}(H)\bigr).$$

It is now a matter of routine to define a suitable candidate for $\rho(\langle 8, d\rangle)$ in a computable way from d and an alleged index for ρ, so we may define a working ρ by the classical recursion theorem. $\square$

From now on, when we write $\{e\}$, we mean $\{e\}_p$.

2.5. The norm of a computation and Gandy selection

The advantage of using p-computations is that now all computation trees will be countable, and all computations will have a countable ordinal as rank. We give a direct definition of this rank. In order to simplify the readability we introduce the following as a convention: With the expression $\lambda(g,c).\{d\}(\mathcal{I},\vec{F},g,\vec{f},c,\vec{a})$ we really mean the function

$$(*)\quad F(g)(c) = \min\bigl\{1, \{d\}(\mathcal{I},\vec{F},c\hat{\ }g,\vec{f},\vec{a})\bigr\}.$$

Definition 2.13. Let $C_{\mathcal{I}}$ be the set of finite sequences $\langle e,\vec{F},\vec{f},\vec{a}\rangle$ such that for some b we have

$$\{e\}(\mathcal{I},\vec{F},\vec{f},\vec{a}) = b.$$

If $\langle e,\vec{F},\vec{f},\vec{a}\rangle \in C_{\mathcal{I}}$ we define the *rank* or *ordinal norm* $\|\langle e,\vec{F},\vec{f},\vec{a}\rangle\|$ by transfinite recursion as follows:

(i) If e corresponds to S1–S3 or S7, we let the norm be zero.

(ii) If $\{e\}(\mathcal{I},\vec{F},\vec{f},\vec{a}) = \{e_1\}(\mathcal{I},\vec{F},\vec{f},\{e_2\}(\mathcal{I},\vec{F},\vec{f},\vec{a}),\vec{a})$, where $\{e_2\}(\mathcal{I},\vec{F},\vec{f},\vec{a}) = c$, we let

$$\bigl\|\langle e,\vec{F},\vec{f},\vec{a}\rangle\bigr\| = \max\bigl\{\bigl\|\langle e_2,\vec{F},\vec{f},\vec{a}\rangle\bigr\|, \bigl\|\langle e_1,\vec{F},\vec{f},c,\vec{a}\rangle\bigr\|\bigr\} + 1.$$

The cases S6 and S9 are handled in a similar way, and are left for the reader.

(iii) If $\{e\}(\mathcal{I},\vec{F},\vec{f},\vec{a}) = F_1(g)$ where $g(b) = \{d\}(\mathcal{I},\vec{F},\vec{f},b,\vec{a})$ we let

$$\bigl\|\langle e,\vec{F},\vec{f},\vec{a}\rangle\bigr\| = \sup\bigl\{\bigl\|\langle d,\vec{F},\vec{f},b,\vec{a}\rangle\bigr\| + 1 : b \in \mathbb{N}\bigr\}$$

(iv) If $\{e\}(\mathcal{I},\vec{F},\vec{f},b,\vec{a}) = \mathcal{I}(\lambda(g,c).\{d\}(\mathcal{I},\vec{F},g,\vec{f},c,\vec{a}))(b)$, we let F be as in $(*)$, and we let α and f_β for $\beta \leqslant \alpha$ be as in the definition of $\mathcal{I}$. By the assumption, f_β is well defined and total for all $\beta \leqslant \alpha$, where $F(f_\alpha) \leqslant f_\alpha$. We let

$$\bigl\|\langle e,\vec{F},\vec{f},b,\vec{a}\rangle\bigr\| = \sup\bigl\{\bigl\|\langle d,\vec{F},c\hat{\ }f_\beta,\vec{f},\vec{a}\rangle\bigr\| + 1 : \beta \leqslant \alpha \wedge c \in \mathbb{N}\bigr\}.$$

If $\langle e,\vec{F},\vec{f},\vec{a}\rangle \notin C_{\mathcal{I}}$ we let $\|\langle e,\vec{F},\vec{f},\vec{a}\rangle\| = \aleph_1$, the first uncountable ordinal.

Lemma 2.14 (Stage Comparison). *There is a partial functional P in two variables, p-computable in $\mathcal{I}$, such that*

(i) *$P(\langle e,\vec{F},\vec{f},\vec{a}\rangle, \langle d,\vec{G},\vec{g},\vec{c}\rangle)$ terminates if at least one of $\langle e,\vec{F},\vec{f},\vec{a}\rangle$ and $\langle d,\vec{G},\vec{g},\vec{c}\rangle$ is in $C_{\mathcal{I}}$ and then*

(ii) *$P(\langle e,\vec{F},\vec{f},\vec{a}\rangle, \langle d,\vec{G},\vec{g},\vec{c}\rangle) = 1$ if $\|\langle e,\vec{F},\vec{f},\vec{a}\rangle\| \leqslant \|\langle d,\vec{G},\vec{g},\vec{c}\rangle\|$*

(iii) *$P(\langle e,\vec{F},\vec{f},\vec{a}\rangle, \langle d,\vec{G},\vec{g},\vec{c}\rangle) = 0$ if $\|\langle d,\vec{G},\vec{g},\vec{c}\rangle\| < \|\langle e,\vec{F},\vec{f},\vec{a}\rangle\|$.*

Proof. We use the recursion theorem to construct P, and the definition is split into 81 cases, according to the schemes corresponding to e and d. S8 splits into two cases, S8.2 and S8.3 for applications of F_1 and $\mathcal{I}$, while S5 is redundant and left out. This is why we have 9×9 cases. Strictly speaking there are 100 cases, because we must say what P does in cases where one or both indices do not correspond to Kleene-indices at all, but we leave these trivial cases for the reader. Fortunately, many other cases are trivial as well, in particular those where one of the indices e or d represents a basic computation via S1–S3 or S6. Moreover, all cases not involving S8.3 are covered by the literature, see e.g. [6].

We will give the details for three cases (S4, S8.3), (S8.2, S8.3) and (S8.3, S8.3). The remaining cases follow by similar, or even simpler, arguments. As is normal practise for this kind of construction/proof we define P by self reference, assuming for each case, as an induction hypothesis, that P works for the immediate subcomputations.

Case (S4, S8.3): Let

$$\{e\}(\mathcal{I},\vec{F},\vec{f},\vec{a}) = \{e_1\}\bigl(\mathcal{I},\vec{F},\vec{f},\{e_2\}(\mathcal{I},\vec{F},\vec{f},\vec{a}),\vec{a}\bigr)$$

and let

$$\{d\}(\mathcal{I}, \vec{G}, \vec{g}, b, \vec{c}) = \mathcal{I}(G)(b),$$

where $G = \lambda(g, c).\{d_1\}(\mathcal{I}, \vec{G}, c\,\hat{}\,g, \vec{g}, \vec{c})$.

Let g_α be element α in the sequence inductively defined from G. We now consider the following induction, that can easily be formalised via an inductive definition:

Use P to compare $\|\langle e_1, \vec{F}, \vec{f}, \vec{a}\rangle\|$ with the ranks needed to compute $g_0, g_1, \ldots$ until the first is bounded in norm by one of the latter computations or until the latter induction terminates.

In the first case, let $c = \{e_2\}(\mathcal{I}, \vec{F}, \vec{f}, \vec{a})$ and start over again, now comparing the computations involved in computing the g_α's with $\|\langle e_1, \vec{F}, \vec{f}, c, \vec{c}\rangle\|$.

If $\|\langle e, \vec{F}, \vec{f}, \vec{a}\rangle\| \leqslant \|\langle d, \vec{G}, \vec{g}, \vec{c}\rangle\|$, this will be verified through the two inductions, and the composition will terminate at least as fast as the induction. If $\|\langle d, \vec{G}, \vec{g}, \vec{c}\rangle\| < \|\langle e, \vec{F}, \vec{f}, \vec{a}\rangle\|$, at least one of the two inductions will result in the full induction induced by G, and we can deduce that this terminates faster than the composition.

Case (S8.2, S8.3): Let

$$\{e\}(\mathcal{I}, \vec{F}, \vec{f}, \vec{a}) = F_1(f)$$

where

$$f(a) = \{e_1\}(\mathcal{I}, \vec{F}, \vec{f}, a, \vec{a})$$

and let

$$\{d\}(\mathcal{I}, \vec{G}, \vec{g}, b, \vec{c}) = \mathcal{I}(G)(b)$$

where G is as in the previous case.

As in the previous case, we simulate the induction in the second part while, at each step, comparing the length of the computations needed with those of each $\{e_1\}(\mathcal{I}, \vec{F}, \vec{f}, a, \vec{a})$. We use ${}^2E$ in doing this. If we for each a reach a step in the induction where we need a computation that dominates the computation of $\{e_1\}(\mathcal{I}, \vec{F}, \vec{f}, a, \vec{a})$, we know that the left hand side will terminate at most with the same rank as the right hand side. If we are able to complete the induction on the right hand side before termination of all sub-computations on the left hand side, we know that the right hand side terminates first. This stepwise comparison until the value of P is settled can be expressed as an inductive definition.

Case (S8.3, S8.3): Let

$$\{e\}(\mathcal{I}, \vec{F}, \vec{f}, a, \vec{a}) = \mathcal{I}\big(\lambda\big(a', f'\big).\{e_1\}\big(\mathcal{I}, \vec{F}, a'\,\hat{}\,f', \vec{f}, \vec{a}\big)\big)(a)$$

and let

$$\{d\}(\mathcal{I}, \vec{G}, \vec{g}, b, \vec{b}) = \mathcal{I}\big(\lambda\big(b', g'\big).\{d_1\}\big(\mathcal{I}, \vec{G}, b'\,\hat{}\,g', \vec{g}, \vec{b}\big)\big)(b).$$

Notice that the norms of these computations will be independent of the choices of a and b. Let F and G be the partial functionals involved in these inductions, where at least one is sufficiently total for the induction to terminate. We now describe a simultaneous inductive definition of two increasing sequences f_α and g_β of elements of C, where we use ${}^2E$ and P to make all the comparisons involved:

* Let $f_0 = g_0$ be the constant zero.
* Assume that $f_0, \ldots, f_\alpha$ and $g_0, \ldots, g_\beta$ are constructed.
* Consider all computations involved in computing all $f_\delta(a)$ for $\delta \leqslant \alpha$ and in computing $F(f_\alpha)(a)$ for all a, and then consider all computations involved in computing all $g_\gamma(b)$ for $\gamma \leqslant \beta$ and in computing $G(g_\beta)(b)$ for all b.

- $*$ If the norm of each computation in the first set is bounded by the norm of some computation in the second set, we add $f_{\alpha+1} = \max\{f_\alpha, F(f_\alpha)\}$ and keep $g_0, \dots g_\beta$.
- $*$ On the other hand, if there is one computation in the first set whose norm strictly bounds all norms of the computations in the other set, we add $g_{\beta+1} = \max\{g_\beta, G(g_\beta)\}$ and keep $f_0, \dots, f_\alpha$.
- $*$ At least one of these two inductions will terminate through this process, and when it does, we know which one will terminate with lowest ordinal rank.

We leave the formal definition of this inductive definition for the reader. □

Theorem 2.15 (Gandy Selection)**.** *There is a p-computable* selection operator ν *such that for all e,* $\vec{F}$*,* $\vec{f}$ *and* $\vec{a}$ *we have*

$$\exists n\{e\}(\mathcal{I}, \vec{F}, \vec{f}, n, \vec{a}) \downarrow \Rightarrow \{e\}\big(\mathcal{I}, \vec{F}, \vec{f}, \nu(e, \vec{F}, \vec{f}, \vec{a}), \vec{a}\big) \downarrow .$$

Proof. This is a soft consequence of Lemma 2.14, with an argument well known in the literature, see e.g. [5, Theorem 3.1.6], [13, Theorem 3], [30, Theorem X.4.1] or the original [6]. □

For many of the inductive definitions used, we add at most one new element to the inductively defined set at each stage. Such definitions can be defined by functionals F of pure type 2, identifying $2^{\mathbb{N}}$ with the power set of $\mathbb{N}$ via characteristic functions:

Definition 2.16. Let $G : 2^{\mathbb{N}} \to \mathbb{N}$ and let $H_G : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ be defined by

$$H_G(A) = A \cup \big\{G(A)\big\}.$$

An inductive definition F is *single valued* if it is of the form H_G. We let $\mathcal{I}_0(G) = \mathcal{I}(H_G)$.

Lemma 2.17. *The functionals* $\mathcal{I}$ *and* $\mathcal{I}_0$ *are computationally equivalent modulo* ${}^2E$.

Proof. $\mathcal{I}_0$ is trivially, and outright, computable in $\mathcal{I}$. In order to prove the other direction, we let $F : 2^{\mathbb{N}} \to 2^N$ be given, and we will construct a single valued G that *simulates* F. We assume that F is nontrivial, i.e. that $F(\emptyset) \neq \emptyset$. We let G operate on sets B of (sequence numbers of) finite binary sequences, and we totally order these sequences using the standard lexicographical ordering by first comparing the first place where two sequences are different, and if this does not help, by length. This is not a well ordering, but G, as we define it, will only generate well ordered sets of sequences. There will be three cases in the definition of $G(B)$:

(1) B has no maximal element s. Let A be the set of n such that $s(n) = 1$ for at least one $s \in B$. If $F(A) \subseteq A$, let $G(B)$ be the empty sequence. If not, let n be the least number in $F(A) \setminus A$, and let $G(B) = s$ where s is the binary sequence of length $n + 1$ approximating the characteristic function of $A \cup F(A)$.

(2) If there are elements $s_1 < \cdots < s_k$ in B so that $B_1 = \{s \in B : s < s_1\}$ has no maximal element and such that

$$B = B_1 \cup \{s_1, \dots, s_k\},$$

let A be the set of n such that $s(n) = 1$ for at least one s in B_1. If the sequences $s_1, \dots, s_k$ do not approximate the characteristic function of $A \cup F(A)$, let $G(B) = 0$ (the value does not matter), while otherwise, we let $G(B) = s$ where s is the least proper extension of s_k that approximates $A \cup F(A)$.

(3) Otherwise, let $G(B) = 0$.

The induction induced by G will, one step at the time, build up approximations to the characteristic functions of the sets appearing in the induction induced by F. If $\mathcal{I}(F)$ uses α many steps, $\mathcal{I}_0(G)$ will use $\omega \cdot \alpha$ many steps. Clearly G is computable in F and ${}^2E$, and the closure set of F is arithmetical in the closure set of G. Thus $\mathcal{I}$ is computable in $\mathcal{I}_0$ and ${}^2E$. □

3. The companion of $\mathcal{I}$

In this section we will analyse the computational power of $\mathcal{I}$ in terms of set theory. Recall that a set X is *hereditarily countable* if the *transitive closure* $\mathrm{trcl}(X)$ is countable. Hereditarily countable sets X will have *codes*, essentially structures (D, R, A) where $D \subseteq \mathbb{N}$, R is a binary relation on D, $A \subseteq D$ and (D, R, A) is isomorphic to $(\mathrm{trcl}(X), \in^{\mathrm{trcl}(X)}, X)$. Such codes can further be coded as functions in $\mathbb{N}^{\mathbb{N}}$ in a standard way.

Definition 3.1. The *companion* $\mathcal{M}$ of $\mathcal{I}$ is defined as the set of sets X with codes that are computable in $\mathcal{I}$.

Remark 3.2. The companion of other functionals are defined in analogy with this. For instance, the companion of ${}^2E$ will be $\mathsf{L}_{\omega_1^{\mathrm{CK}}}$, the companion of $\mathbf{S}$ (the Suslin functional) is L_β for the first recursively inaccessible ordinal β while the companion of $\mathbb{S}$ (the Superjump) is L_ρ where ρ is the first recursively Mahlo ordinal.

Lemma 3.3. *There is a countable ordinal π such that $\mathcal{M} = \mathsf{L}_\pi$.*

Proof. Since $\{e\}_p$ is absolute for L, we have that $\mathcal{M}$ is a transitive subset of L. The set of functions computable in $\mathcal{I}$ will be closed under a certain map sending a code for an ordinal α to a code for L_α (this map is computable in ${}^2E$), so $\mathcal{M}$ will be an initial segment of L. $\square$

Lemma 3.4. *Let $F : \mathsf{C} \to \mathsf{C}$ be a partial functional computable in $\mathcal{I}$ such that $\mathcal{I}(F)$ is defined. Let α be the corresponding closure ordinal for F. Then $\alpha < \pi$.*

Proof. For each F there is an F' computable in F and ${}^2E$ such that F' generates a prewellordering R where $R_{\beta+1} = F(R_\beta) \cup R_\beta$ for each ordinal β. Then α will be the ordinal rank of the inductively definable prewellordering R so α will be computable in $\mathcal{I}$ whenever F is computable in $\mathcal{I}$. $\square$

The aim of this section is to find closure- and reflection-properties of L_π. Since $\mathbf{S}$ is computable in $\mathcal{I}$ we have that the set of codes for hereditarily countable sets is computable in $\mathcal{I}$. Given codes f_i for sets X_i, $i \in \mathbb{N}$,, we only need ${}^2E$ to unify the codes in the form of a code for $\{X_i : i \in \mathbb{N}\}$. Further, given codes $f_1, \ldots, f_n$ for sets $X_1, \ldots, X_n$, and a Δ_0-formula $\Phi(x_1 \ldots, x_n)$, ${}^2E$ can decide the truth value of $\Phi(X_1, \ldots, X_n)$. Finally, if $\Phi(x_1, \ldots, x_n, y)$ is a Δ_0-formula, $f_1, \ldots, f_n$ are codes computable in $\mathcal{I}$ for $X_1, \ldots, X_n \in \mathsf{L}_\pi$ and

$$\mathsf{L}_\pi \models \exists Y \Phi(X_1, \ldots, X_n, Y)$$

then we can use Gandy selection for $\mathcal{I}$ to compute (an index for) a code g for a set Y such that $\Phi(X_1, \ldots, X_n, Y)$. This leads to a proof of

Lemma 3.5. *L_π is an admissible structure.*

Let WO be the set of codes for countable ordinals. This is a Π^1_1- set, and it is easy to prove that the following sets are Π^1_1 as well:

(1) The set of $f \in \mathsf{WO}$ that codes ω_1^{CK}.
(2) The set of $f \in \mathsf{WO}$ that codes the first recursively inaccessible ordinal.
(3) The set of $f \in \mathsf{WO}$ that codes the first recursively Mahlo ordinal.

We say that these ordinals are Π^1_1-*characterisable*. Many ordinals of distinction are Π^1_1-characterisable, for instance all *clockable* ordinals in the sense of infinite time Turing machines ([7]), see Welch [32] for a survey and further references on such machines.

Definition 3.6. Let $\mathbf{P}$ be a class of ordinals. We say that $\mathbf{P}$ is $\mathcal{I}$-*decidable* if there is an $\mathcal{I}$-computable function $\Delta : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ such that $\Delta(f) = 0$ if and only if f codes an ordinal α and $\mathbf{P}(\alpha)$ holds.

That π is not Π^1_1-characterisable follows from the following much stronger:

Theorem 3.7. *Let* $\mathbf{P}$ *be a property on ordinals that is* $\mathcal{I}$*-decidable and such that* $\mathbf{P}(\pi)$*. Let* $X \subset \pi$ *be closed, unbounded and* Σ_1 *over* L_π*. Then there is an* $\alpha \in X$ *such that* $\mathbf{P}(\alpha)$*.*

Proof. We can code a partially enumerated set $\{f_d : d \in D\}$ of functions as the set of pairs $\langle d, \overline{f}_d(n)\rangle$ where $d \in D$ and $n \in \mathbb{N}$. The idea is to construct an inductive definition Γ that is computable in $\mathcal{I}$ and such that Γ generates a code for an ordinal both in X and satisfying $\mathbf{P}$. Γ will not be total, but sufficiently total for the induction to terminate. In defining Γ as computable in $\mathcal{I}$, we use that the Suslin functional $\mathbf{S}$ is computable in $\mathcal{I}$. We define $\Gamma(R)$ as follows:

- If R does not code an enumerated set $\{f_d : d \in D\}$ of functions, we let $\Gamma(R) = R$. Note that the empty set codes the empty set of functions.
- Assume that R codes $\{f_d : d \in D\}$. If $f_d \notin \mathsf{WO}$ for some $d \in D$, let $\Gamma(R) = R$.
- Assume now that $f_d \in \mathsf{WO}$ codes α_d for all $d \in D$, and use ${}^2E$ to compute a code g for the least upper bound α of $\{\alpha_d : d \in D\}$. If all α_d are in X, then $\alpha \in X$ since X is closed. If $\mathbf{P}(\alpha)$ we let $\Gamma(R) = R$. This is where we want the induction to close.
- Otherwise, we apply Gandy selection for $\mathcal{I}$ and search for an index e for a code g of an ordinal $\beta > \alpha$ such that $\beta \in X$. We then let

$$\Gamma(R) = R \cup \{\langle e, \overline{g}(n)\rangle : n \in \mathbb{N}\}.$$

If $\alpha < \pi$, we can use the recursion theorem for $\mathcal{I}$ to see that L_π is closed under the α-iteration of Γ, and that Γ generates codes for an increasing sequence of ordinals γ_β for $\beta < \alpha$. Since we always use an index e for an ordinal larger than those appearing at earlier stages, we do not risk to mix up codes for different ordinals. Since X is closed, all ordinals obtained during this iteration will be codes for ordinals $\gamma_\beta \in X$. Since $\mathbf{P}(\pi)$ and this induction will stop when we hit a γ_β with $\mathbf{P}(\gamma_\beta)$, and since by Lemma 3.4 no such induction will stop at π, there must be an ordinal $\gamma_\beta < \pi$ such that $\mathbf{P}(\gamma_\beta)$. □

Corollary 3.8. *The closure ordinal* π *of* $\mathcal{I}$ *is not* Π^1_1*-characterisable.*

We also have

Corollary 3.9. *The closure-ordinal* π *of* $\mathcal{I}$ *is recursively Mahlo.*

Proof. We have to prove that if $X \subseteq \pi$ is π-computable, closed and unbounded, then X contains an admissible ordinal. Since the class of countable, admissible ordinals is $\mathcal{I}$-decidable, this is a direct consequence of Theorem 3.7. □

Since being recursively Mahlo and other even stronger closure properties are also $\mathcal{I}$-decidable, we may extend this argument in order to prove that π satisfies these stronger properties, and that every closed unbounded subset of π that are Σ_1 over L_π also contain elements satisfying these stronger properties. We will not pursue this further here.

We will now consider an alternative way of expressing that π must be a "large" countable ordinal. What is "large" is of course subject to the perspective one is taking.

Definition 3.10. An ordinal γ is *reflecting* if for all formulas $\Phi(x_1, \ldots, x_n)$ and elements $X_1, \ldots, X_n$ in $\mathsf{L}(\gamma)$,

$$\mathsf{L}_\gamma \models \Phi(X_1, \ldots, X_n) \Rightarrow \exists \beta < \gamma\big[L_\beta \models \Phi(X_1, \ldots, X_n).\big]$$

Note that if γ is reflecting, then γ is admissible and recursively inaccessible.

Corollary 3.11. *The closure ordinal π of $\mathcal{I}$ is reflecting.*

Proof. If $\alpha < \beta$, $X_1, \dots, X_n$ are in L_α and $\mathsf{L}_\pi \models \Phi(X_1, \dots, X_n)$, then the set of $\gamma > \alpha$ such that $\mathsf{L}_\gamma \models \Phi(X_1, \dots, X_n)$ is $\mathcal{I}$-decidable, contains π and thus, by Theorem 3.7, contains an ordinal β with $\alpha < \beta < \pi$. □

π will not be the least reflecting ordinal:

Corollary 3.12. *Let π be the closure ordinal of $\mathcal{I}$. If $\alpha < \pi$, then there is a reflecting ordinal γ with $\alpha < \gamma < \pi$.*

Proof. This is also a consequence of Theorem 3.7, since being reflecting is $\mathcal{I}$-decidable. Indeed, if $X \subseteq \pi$ is closed, unbounded and Σ_1 over L_π, then X contains arbitrarily large reflecting ordinals. □

Remark 3.13. These results do of course not imply that L_π has an elementary substructure, or even a substructure satisfying the same first order sentences. The *theory* of L_π is not $\mathcal{I}$-decidable, so there is no way to unify these arguments to all formulas simultaneously.

4. Classes of functionals of type 3

4.1. Motivation

We introduced the functional $\mathcal{I}$ in Section 2 and illustrated its computational strength through an analysis of the companion in Section 3. In this section we will give an analysis of computations relative to $\mathcal{I}$ resembling an *operational semantics*. Our approach is inspired by the success-story of using *nested sequential procedures* for modelling functionals definable in LCF (Scott, [31]) or equivalently in PCF (Plotkin, [28]) from objects of type 1. For an introduction to nested sequential procedures, see [12, Chapter 6].

Since we are only concerned with functionals of type $\leqslant 3$ in this paper, we can forget the qualifier 'nested', while we will add the qualifier 'hyper' in order to deal with discontinuity. We will aim for more and more restricted concepts of *hyper-sequential procedures* until we find a characterisation of the functionals of type 3 that are computable in $\mathcal{I}$ and some functional of type 2. The gain will mainly be that we obtain a more civilised, and less ad hoc, way of expressing relative computability for certain functionals of type 3 than when we refer to the Kleene schemes directly. We will use this to give mathematical support to the informal claim that if realisers of classical theorems based on compactness arguments are computable in $\mathcal{I}$, then the full power of $\mathcal{I}$ is required.

A functional Φ of type 3 is *normal* if ${}^3E$ is computable in Φ, where

$$ {}^3E(F) = \begin{cases} 0 & \text{if } \forall f \in \mathbb{N}^{\mathbb{N}}(F(f) = 0) \\ 1 & \text{if } \exists f \in \mathbb{N}^{\mathbb{N}}(F(f) > 0) \end{cases}, $$

and where F is assumed to be total.

The set of functionals of type 3 that are neither normal nor computable in type 2 objects is mainly unexplored with respect to computability-theoretical properties. The classical object of this kind is Gandy's Superjump $\mathbb{S}$. $\mathbb{S}$ is of course a natural functional in the context of higher order computability theory. Recently, examples that are natural from other perspectives have emerged. In [16, §3] we introduced classes of realisers Θ for the general Heine–Borel theorem and a weaker class of functionals Λ that compute realisers for the Vitali Covering theorem. In [15] we also considered functionals Ξ that serve as realisers for the Lindelöf Lemma for Baire Space. In this paper we introduced $\mathcal{I}$, which, under the name IND, was proved in [15] to compute Lindelöf realisers Ξ. This plethora of elements in a so far unexplored class of functionals justifies a more coherent study of this class. We will return to some of these functionals in Section 6.2.

In [25] we consider some other non-normal functionals of type 3, functionals that reflect natural constructions in ordinary mathematics and that are intrinsically partial. As these functionals are not computable in $\mathcal{I}$, they fall out of the scope of this paper, but the alternative to Kleene's schemes introduced there may serve as an alternative foundation for what we do in Sections 4 and 5.

Hartley [9] investigated the fully typed hierarchy of hereditarily countably based functionals, based on a definition due to Stan Wainer, and he obtained some general results. For instance, he proved that if we assume the Continuum hypothesis together with ZFC, Φ is countably based if and only if ${}^3E$ is not computable in Φ and any functional of type 2.

The original definition of the countably based functionals is by a generalisation of the definition of the *continuous* functionals e.g. as based on *domain theory*, see [12, Chapter 10] for a recent introduction. In this paper, we will only be interested in objects of types 0, 1, 2 and 3, and we define the countably based functionals for these cases, suiting our own purposes:

Definition 4.1. All integers are countably based. Moreover

(1) All total functions $f : \mathbb{N} \to \mathbb{N}$ are countably based.
(2) All partial functionals F mapping a subset of $\mathbb{N}^{\mathbb{N}}$ to $\mathbb{N}$ are countably based.
(3) Let Φ be a partial functional taking countably based functionals of type 2 as arguments and yielding integers as values. Φ is countably based if we for each F and n such that $\Phi(F) = n$ find a countable set $A \subseteq \mathbb{N}^{\mathbb{N}}$ such that F is total on A and such that for all G of type 2, if G is total on A and agrees with F on A, then $\Phi(G) = n$.

In (3), a *base element* for Φ will be a countable set A together with the restriction of an F to A with the property described.

${}^3E$ will not be countably based, since in order to know that ${}^3E(O^2) = 0$ we need to know that $O^2(f) = 0$ for all $f \in \mathbb{N}^{\mathbb{N}}$.

One problem with the countably based functionals Φ is that the base elements of Φ are not well structured as individual sets, and a suitable class of base elements for Φ may not be well structured as a class. Much of the way of thinking inherited from the computability theory of the continuous functionals is useless. The aim of this section is to introduce a more restricted class, the hyper-sequential functionals, where we have added some further structure. Examples of hyper-sequential functionals will be the Superjump and $\mathcal{I}$. However, the first concept we introduce will be too general for our purpose, for instance, all functionals of type 3 computable using an *infinite time Turing machine* the way suggested by Welch [33] will be hyper-sequential.

4.2. Hyper-sequential functionals

4.2.1. The definitions

In this section we will define what we mean with a hyper-sequential procedure. A transfinite calculation using a functional F as an oracle can be viewed as a sequence of queries of the form "what is $F(f)$?", where the next query will depend on the answers to the previous ones. We will capture such deterministic procedures with our concepts defined in 4.2. Our aim will be to isolate the procedures that will correspond to computations relative to $\mathcal{I}$, and in order to fully capture those, our calculations must also contain some *documentation*. In a computation $\{e\}(\mathcal{I}, F, \vec{f}, \vec{a})$, there may be subcomputations with extra arguments g or b of type 1 or 0. Our abstract calculations will contain what we call a LOG, and the use of this LOG will be to show that procedures corresponding to Kleene-computations in $\mathcal{I}$ are definable at the level of Π^1_1. This will be made precise later, and the actual function of the LOG will be easier to grasp when we simulate Kleene-computations as procedures. In order to formally define this LOG we take the liberty to add an extra element $*$ to $\mathbb{N}$, and to claim that sets of objects involving this $*$ will be of a certain complexity, for instance Π^1_1, without going to the trouble of coding.

Definition 4.2.

(a) A *string* is a sequence $\{(f_\beta, a_\beta)\}_{\beta<\alpha}$ where α is a countable ordinal, each $f_\beta \in \mathbb{N}^{\mathbb{N}}$ and each $a_\beta \in \mathbb{N} \cup \{*\}$. We call f_β a *query*, and sometimes writes it as $F(f_\beta) =?$.
(b) A *hyper-sequential procedure* is a set Ω of strings where each string will be given an integer value, and such that whenever $\{(f_\beta, a_\beta)\}_{\beta<\alpha}$ and $\{(f'_\beta, a'_\beta)\}_{\beta<\alpha'}$ are in Ω they are either equal or there is a $\beta < \min\{\alpha, \alpha'\}$ such that $f_\beta = f'_\beta$, $a_\beta \neq a'_\beta$, $a_\beta \neq *$, $a'_\beta \neq *$ and $(f_\gamma, a_\gamma) = (f'_\gamma, a'_\gamma)$ for all $\gamma < \beta$. Formally, will let Ω be a set of pairs (t, b) where t is a string and b is the associated value.

(c) If $\{(f_\beta, a_\beta)\}_{\beta<\alpha}$ is a string and $F \in Tp(2)$, we say that the string *matches* F if $F(f_\beta) = a_\beta$ for all $\beta < \alpha$ with $a_\beta \in \mathbb{N}$.
(d) If a string t is in a hyper-sequential procedure Ω, has a value a and matches F, we call t a *calculation*, calculating $\Omega(F) = a$.
(e) If Ω is a hyper-sequential procedure, then Ω *defines* (or *computes*) the partial functional $\Phi(F) = \Omega(F)$ of type 3. When $\Phi(F)$ is defined, the calculation of $\Omega(F)$ will be unique.
(f) A *total* functional of type 3 is hyper-sequential if there is a hyper-sequential procedure that defines it.
(g) If $t = \{(f_\beta, a_\beta)\}_{\beta<\alpha}$ is a calculation, and $a_\beta = *$ we say that f_β (or β) is in the LOG of t.

We will from now on use the words *procedure* and *sequential* in the meaning of hyper-sequential procedure and hyper-sequential.

Remark 4.3. A procedure can be viewed as a *strategy* for a transfinite *game* where Player I, the computing device, plays *queries* and Player II, the input, *answers* each query using F. In some matches of the games, corresponding to the calculations, Player I wins the match in the sense of providing an output, while in other matches, Player II wins because the match either stops after countably many steps without a value, or it goes on through $\aleph_1$ many steps. We will discuss this further when we consider procedures with more structure. If we then still use the picture of games with rules, the LOG will represent places where Player I will enter a sub-game following different rules, and the LOG will help the referee to verify that the whole match is played according to the general, nested, rules of the game.

Note that if t is a string that is an initial segment of several calculations, then the next f_β will be the same for all such extensions, and if β is in the LOG of one of them, it will be in the LOG of all extensions.

Definition 4.4. Let $\{(f_\beta, a_\beta)\}_{\beta<\alpha}$ be a string.
A *sub-string* is a sequence $\{(f_\gamma, a_\gamma)\}_{\gamma<\beta}$ for some $\beta \leqslant \alpha$.

We can concatenate strings in the usual way: If we for each ordinal $\gamma < \gamma_0$ have a string $\{(f_{\gamma,\beta}, a_{\gamma,\beta})\}_{\beta<\alpha_\gamma}$ we let the *concatenation* $\{(f_\beta, a_\beta)\}_{\beta<\alpha}$ be defined by

- $\alpha = \sum_{\gamma<\gamma_0} \alpha_\gamma$
- If $\beta = \sum_{\gamma<\gamma_1} \alpha_\gamma + \beta_1$ where $\gamma_1 < \gamma_0$ and $\beta_1 < \alpha_{\gamma_1}$, then $(f_\beta, a_\beta) = (f_{\gamma_1,\beta_1}, a_{\gamma_1,\beta_1})$.

We will prove that the class of sequential functionals of type 3 is closed under Kleene-computability as defined through the schemes S1–S9. To be more precise, we will prove that if $\vec{\Phi} = (\Phi_1, \ldots, \Phi_n)$ consists of sequential functionals and

$$\lambda F.\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a})$$

is total, then it is itself sequential. To make this precise, we need to extend S8 to deal with general inputs of type 3. For the sake of notational simplicity, we assume that the arguments of our computations will be of the form as above, that we drop the scheme S6 of permutation and that we use an alternative indexing for scheme S8 so that we can read out from the index for which of the arguments in the list $\vec{\Phi}$ the oracle call is made. (Alternatively we could modify S6 to cater for permutations of the list of inputs of all four types.) We still leave out S5, primitive recursion, partly because it is redundant in the presence of S9, and partly because it can be handled in analogy to composition S4. Thus we add the following scheme to Definition 2.4, while replacing the one occurrence of $\mathcal{I}$ with a sequence $\vec{\Phi}$ of functionals of type 3:

S8 If $e = \langle 8, 3, i, d\rangle$ then

$$\{e\}(\vec{\Phi}, \vec{F}, \vec{f}, \vec{a}) = \Phi_i\big(\lambda f.\{d\}(\vec{\Phi}, \vec{F}, f, \vec{f}, \vec{a})\big)$$

In the original definition by Kleene, this is only supposed to make sense when $\{d\}(\vec{\Phi}, \vec{F}, f, \vec{f}, \vec{a})$ terminates for all $f \in \mathbb{N}^{\mathbb{N}}$, but when we are working with countably based Φ_i we normally only require that there is a base element that is a sub-function of $\lambda f.\{d\}(\vec{\Phi}, \vec{F}, \vec{f}, \vec{a})$.

As we will see in the sequel, being sequential the way we define it here is quite general, and thus the fact that this class is closed under Kleene computability may be of restricted interest. However, we will later refer to the construction of procedures imbedded in the proof of Lemma 4.7 in situations where we will show that much more restricted classes of functionals still are Kleene closed.

Since we, in this section, are primarily interested in functionals of type 3 computable in a given sequence of sequential functionals of the same type, we restrict the number of arguments of type 2 to one. We can do this because the number of type 2 arguments will not increase as we move down the paths of the computation tree. The number of arguments of type 0 and of type 1 may increase, so we need to consider arbitrarily long finite lists of such input arguments.

Definition 4.5. Let $\vec{\Phi} = (\Phi_1, \dots, \Phi_n)$ be a sequence of sequential functionals defined from the procedures $\Omega_1, \dots, \Omega_n$. Let F be of type 2 and let $\vec{f}$, $\vec{a}$ be finite sequences of objects of type 1 and 0 resp. Assume that $\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a}) = b$. By recursion on the length of this computation we define the calculation $t_{e,\vec{\Phi},F,\vec{f},\vec{a}}$ with value b as follows, where we use ˆ to denote concatenation of strings (recall that *calculations* are strings that, in the given context, have values):

- If e is an index for an initial computation, i.e. for S1, S2, S3 or S7, we let $t_{e,\vec{\Phi},F,\vec{f},\vec{a}}$ be the empty string, i.e. with $\alpha = 0$.
- If
$$\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a}) = \{e_1\}\big(\vec{\Phi}, F, \vec{f}, \{e_2\}(\vec{\Phi}, F, \vec{f}, \vec{a}), \vec{a}\big),$$
let $c = \{e_2\}(\vec{\Phi}, F, \vec{f}, \vec{a})$. Let
$$t_{e,\vec{\Phi},F,\vec{f},\vec{a}} = t_{e_2,\vec{\Phi},F,\vec{f},\vec{a}}\,\hat{}\,t_{e_1,\vec{\Phi},F,\vec{f},c,\vec{a}}.$$
- In the case of S9, we just use the calculation for the immediate subcomputation.
- Let $\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a}) = F(\lambda c.\{e_1\}(\vec{\Phi}, F, \vec{f}, c, \vec{a}))$. Let $f(c) = \{e_1\}(\vec{\Phi}, F, \vec{f}, c, \vec{a})$. Then
$$t_{e,\vec{\Phi},F,\vec{f},\vec{a}} = t_{e_1,\vec{\Phi},F,\vec{f},0,\vec{a}}\,\hat{}\,t_{e_1,\vec{\Phi},F,\vec{f},1,\vec{a}}\,\hat{}\,\underbrace{\cdots\cdots}_{\omega}\,\hat{}\,\big(f, F(f)\big).$$
- Let $\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a}) = \Phi_i(\lambda g.\{e_1\}(\vec{\Phi}, F, g, \vec{f}, \vec{a}))$. Let $H(g) = \{e_1\}(\vec{\Phi}, F, g, \vec{f}, \vec{a})$ and let $\{(g_\beta, b_\beta)\}_{\beta<\alpha}$ be the calculation in Ω_i that is matching H.
We let $t_{e,\vec{\Phi},F,\vec{f},\vec{a}}$ be the concatenation of
$$\big\{(g_\beta, *)\,\hat{}\,t_{e_1,\vec{\Phi},F,g_\beta,\vec{f},\vec{a}}\big\}_{\beta<\alpha}.$$

This ends the definition.

Remark 4.6. We inserted the pairs $(g_\beta, *)$ in the LOG in order to remind us of the fact that we at that stage are simulating a subcomputation with an extra argument g_β. We need the information about this extra argument in order to say that a string is 'correct', in a sense made precise later.

Lemma 4.7. *Let $e, \vec{\Phi}$, $\vec{f}$ and $\vec{a}$ be fixed as in Defintion 4.5. Then the set*
$$\big\{(t_{e,\vec{\Phi},F,\vec{f},\vec{a}}, b) : \{e\}(\vec{\Phi}, F, \vec{f}, \vec{a}) = b\big\}$$
will be a procedure.

Proof. Assume that both $\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a})$ and $\{e\}(\vec{\Phi}, G, \vec{f}, \vec{a})$ terminate. We prove by induction on the ordinal ranks of the computations that the corresponding calculations satisfy Definition 4.2(b). The proof is split into cases corresponding to the Kleene schemes.

If e is an index for an initial computation, the claim is trivial, and the induction step is trivial in the case of application of S9.

Let e be an index for composition, and let e_1 and e_2 be as in the construction. If the calculations for $\{e_2\}(\vec{\Phi}, F, \vec{f}, \vec{a})$ and $\{e_2\}(\vec{\Phi}, G, \vec{f}, \vec{a})$ are different, then by the induction hypothesis they split at a first point, and there the f-parts are the same while the a-parts differ. Since these calculations are initial segments of the calculations under consideration, the concatenated calculations also satisfy the definition.

If the calculations for the e_2-computation are equal, then, by the indiction hypothesis, the values are the same, c, and then our conclusion follows from the induction hypothesis for $\{e_1\}(\vec{\Phi}, F/G, \vec{f}, \vec{a})$.

Application of F/G: In this case, we construct calculations as the concatenation of $\omega + 1$ items, first the corresponding calculations for each $c \in \mathbb{N}$, and at the end, pairs $(f, F(f))$ and $(f', G(f'))$ respectively. If there is a least c where the corresponding two calculations differ, the f-parts will agree while the a-parts will differ at a minimal location in these calculations, by the induction hypothesis. Then the f-parts will agree and the a-parts will differ at the corresponding minimal location in the concatenated calculation. If the two concatenations of the calculations inherited for each c are equal, it follows from the induction hypothesis that the arguments f and f' for F and G resp. are equal, so at the top pair $(f, F(f))$ and $(f', G(f'))$ we will have that the query parts are equal.

Application of Φ_i: Let $\{(g_\beta, b_\beta)\}_{\beta<\alpha}$ and $\{(g'_\beta, b'_\beta\}_{\beta<\alpha'}$ be the two calculations in Ω_i matching the corresponding functionals H and H' as in the definition in this case. First we see that if the two concatenated calculations agree as far as they both go, we can use the induction hypothesis, sub-induction on $\beta < \min\{\alpha, \alpha'\}$ and the fact that Ω_i is a procedure to show that $g_\beta = g'_\beta$ and that $H(g_\beta) = H'(g'_\beta)$ for all β. Since Ω_i is a procedure, it follows that $\alpha = \alpha'$, that the two concatenated calculations are equal and that the values are the same.

If the two concatenated calculations differ, there will be a least $\beta < \min\{\alpha, \alpha'\}$ such that they differ in the sections computing $H(g_\beta)$ and $H'(g'_\beta)$. Then $g_\beta = g'_\beta$, so by the induction hypothesis there is a least location in those sections where they differ, and there the f parts are equal while the a-parts differ. So, the calculations constructed will satisfy the definition. □

Theorem 4.8. *The class of hyper-sequential functionals of type 3 is closed under relative Kleene-computability.*

Proof. Immediate from Lemma 4.7. □

Remark 4.9. We can deduce, from the proof of Theorem 4.8, that all functionals of type 3 computable in functionals of lower types will be hyper-sequential.

4.2.2. Mixed types

Some of the objects we are interested in are of types at level $\leqslant 3$ that are not pure, $\mathcal{I}$ is one prominent example. There are two natural ways to extend the concept of sequential functionals to objects of such types. One is to identify such types as the fixed points of computable retracts on the corresponding pure types, the retracts being explicitly definable as Kleene-computable where the schemes S5 and S9 are not needed. Then an object will be, by definition, sequential if the representation in the pure type is so.

The other alternative is to extend the intuition of sequentiality to objects of these general types. A type like this will be of the form

$$\sigma_1, \dots, \sigma_n \to \mathbb{N},$$

where each σ_i has level $\leqslant 2$. Thus a calculation will be a well-ordered set of queries with answers where each query is of the form $F_i(\vec{f}) =?$ for some i, varying with the query. Each $\vec{f}$ will consist of functions and/or integers, and the functions may be of one or several number variables. To keep track of all this in its full generality will require some heavy notation, but there will be no genuine mathematical problems. Given this, we can define what we mean with a *procedure* adjusted to each type, and then the sequential objects of that type. It is obvious that the two approaches are equivalent, but not being pressed, we prefer to omit all details. In some of our examples, we will use the latter, intuitive approach.

4.2.3. Examples

Our first example is what motivated us to isolate the concept of hyper-sequential functionals:

Theorem 4.10. *The functional $\mathcal{I}$ is hyper-sequential.*

Proof. Let $F : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$, $F'(a \,\hat{}\, f) = F(f)(a)$ and let f_0^F be the constant 0. We find $f_1^F = F(f_0^F)$ through the ω-series of queries $F'(a \,\hat{}\, f_0^F) =?$, then $f_2^F = F(f_1^F) \cup f_1^F$ (identifying a characteristic function with the corresponding set) through the ω-sequence of queries $F'(a \,\hat{}\, f_1^F) =?$ and so on. This is clearly a hyper-sequential procedure. □

In his CiE-2019-paper [33], Philip Welch introduced infinite time Turing machines that can take functionals F of type 2 as oracles. The idea is to have a special oracle tape, and whenever the oracle F is called upon, we consider the oracle tape as the input information, and what the consequence of the oracle call will be will depend on the precise ITTM-model we are using. We have

Theorem 4.11. *Every ITTM-computable functional is sequential.*

We leave this theorem without a proof, since the proof is easy, but requires familiarity with the ITTM-model.

Clearly, all sequential functionals are countably based. To what extent the converse is true is unknown, but we do have:

Theorem 4.12. *If the continuum hypothesis CH holds, all countably based total functionals will have extensions to partial functionals that are sequential.*

Proof. We work within ZFC + CH. Let $\{f_\alpha\}_{\alpha<\aleph_1}$ be an enumeration of $\mathbb{N}^{\mathbb{N}}$. Let Φ be countably based and let X be set of base elements for Φ. The elements of X will be triples (A, ϕ, a) where $A \subseteq \mathbb{N}^{\mathbb{N}}$ is countable, $\phi : A \to \mathbb{N}$ and $a \in \mathbb{N}$. The significance is that whenever F extends ϕ to all of $\mathbb{N}^{\mathbb{N}}$, then $\Phi(F) = a$, and that for each F there will be at least one $(A, \phi, a) \in X$ where F is an extension of ϕ.

The sequential procedure will then be to compute $F(f_\alpha)$ up to the first α_0 where there is some $(A, \phi, a) \in X$ such that

- $A \subseteq \{f_\alpha : \alpha < \alpha_0\}$.
- For $f_\alpha \in A$ we have that $F(f_\alpha) = \phi(f_\alpha)$.

We will have that $\Phi(F) = a$ independent of which (A, ϕ, a) we chose with this property. □

4.3. Denotation procedures

There is no reason to believe that the continuum hypothesis can be avoided in Theorem 4.12, but the theorem still suggests that the concept of hyper-sequential functional is too general to be of interest, and the intention is to investigate possible refinements of the concept. Now we will consider procedures that will include some extra information, a number or *denotation* d_β for each β in the index ordinal of a calculation. In its full generality, this does not restrict the class of functionals definable from procedures, but it gives us a tool for discussing the complexity of them. Thus, in the theorems of this section, the constructions of the procedures with denotations used to prove them will be as important as the theorems themselves.

Definition 4.13. A *denotation procedure* Ω, *d-procedure* for short, will be a set Ω of *calculations with denotations*

$$\big(\{(f_\beta, a_\beta, d_\beta)\}_{\beta<\alpha}, c\big)$$

where each $a_\beta \in \mathbb{N} \cup \{*\}$ and

(1) The *denotations* d_β are in $\mathbb{N}$.
(2) The corresponding set of calculations without the denotations is a procedure.
(3) For each $(\{(f_\beta, a_\beta, d_\beta)\}_{\beta<\alpha}, c) \in \Omega$, if $\beta < \gamma < \alpha$, then $d_\beta \neq d_\gamma$.

By abuse of terminology, we will use Ω both for a d-procedure and for the corresponding procedure, making it clear in each case if we consider the denotations or not. Clearly, all d-procedures will define sequential, partial functionals as well. In fact we have

Observation 1. *By the axiom of choice, all procedures Ω can be extended to d-procedures.*

We simply use the axiom of choice to select one enumeration of α for each calculation $(\{(f_\beta, a_\beta)\}_{\beta<\alpha}, c) \in \Omega$ and use this to define the additional d_β s for each calculation. There is of course no extra knowledge to be harvested from this argument, but it illustrates an aspect that we have to bring under control in the d-procedures that we construct:

Definition 4.14. Let Ω be a d-procedure, let $\{(f_\beta, a_\beta, d_\beta)\}_{\beta<\alpha}$ be a calculation in Ω and let $\beta < \alpha$. The *delay* of the denotation of the calculation at point β is the least ordinal γ such that for all other calculations $\{(f'_\delta, a'_\delta, d'_\delta)\}_{\delta<\alpha'}$ in Ω, if $(f_\delta, a_\delta) = (f'_\delta, a'_\delta)$ for all $\delta < \beta + \gamma$, then $d_\delta = d'_\delta$ for all $\delta \leqslant \beta$.

The delay tells us for how much longer we must run a calculation before we can tell what the denotation will be.

A key property of a d-procedure is that we can use the denotations to code the procedure in a manageable way as a subset of the continuum.

Definition 4.15. (a) Let Ω be a d-procedure. The *representation* of Ω will be the set of quadruples $(D, \prec, \{(f_d, a_d)\}_{d\in D}, c)$ derived from calculations $(\{(f_\beta, a_\beta, d_\beta)\}_{\beta<\alpha}, c)$ in Ω as follows:

(i) D is the set of d_β for $\beta < \alpha$ and $\prec$ is the corresponding ordering on D.
(ii) When $d = d_\beta$, then f_d is the f_β and a_d is the a_β of the calculation.
(iii) c is the value of the calculation.

We code these items as elements of $\mathbb{N}^{\mathbb{N}}$ in some standard way.

(b) We say that a d-procedure Ω is Π^1_1 if the representation of Ω is a Π^1_1-set.
(c) If $(D, \prec, \{(f_d, a_d)\}_{d\in D}, c)$ is a calculation in a d-procedure and $(D', \prec', \{(g_d, b_d)\}_{d\in D'})$ satisfies that $\prec'$ is a well ordering of D', each g_d is of type 1 and each b_d is of type 0, we say that $(D', \prec', \{(g_d, b_d,\}_{d\in D'})$ is *isomorphic to* an initial segment of $(D, \prec, \{(f_d, a_d)\}_{d\in D})$ if there is an order isomorphism ρ from D' to an initial segment of D such that $f_{\rho(d)} = g_d$ and $a_{\rho(d)} = b_d$ for all $d \in D'$.

Lemma 4.16. *The functional $\mathcal{I}$ is definable from a d-procedure that is Π^1_1.*

Proof. For each b, we will construct a procedure for the 0–1-valued function

$$\lambda G.\mathcal{I}(F_G)(b),$$

where $F_G(f)(a) = \min\{1, G(a\,\hat{}\, f)\}$ is as in Remark 2.3. The only difference between these procedures will be in the value part, the c in each string.

Let G be given. We will describe the calculation with denotation that will match G and conclude with the value $\mathcal{I}(F_G)(b)$. Let $\{f_\beta\}_{\beta\leqslant\alpha}$ be the sequence constructed while defining $\mathcal{I}(F_G)$.

For each $\beta \leqslant \alpha$, let $g_{a,\beta} = a\,\hat{}\,f_\beta$. We see that in order to "compute" $\mathcal{I}(F_G)(b)$ we have to evaluate G on all functions $g_{a,\beta}$ for all $\beta \leqslant \alpha$, a sequence of queries of order type $\omega(\alpha+1)$. So, we define the calculation matching G as

$$\left(\left\{(h_\gamma, b_\gamma, d_\gamma)\right\}_{\gamma<\omega(\alpha+1)}, c\right)$$

where

- $h_{\omega\cdot\beta+a} = g_{a,\beta}$ for $\beta \leqslant \alpha$ and $a < \omega$.
- $b_{\omega\cdot\beta+a} = G(g_{a,\beta})$ for β and a as above.
- $d_{\omega\cdot\alpha+a} = \langle 0, a\rangle$ for $a \in \omega$.
- $d_{\omega\cdot\beta+a} = \langle x+1, a\rangle$, where x is minimal such that $f_{\beta+1}(x) = 1$ while $f_\beta(x) = 0$, if $\beta < \alpha$ and $a \in \omega$.
- $c = f_\alpha(b)$.

It remains to prove that the representation is Π^1_1. We do this through the following steps:

(1) Since the set of pairs $(D, \prec)$ where $D \subseteq \mathbb{N}$ and $\prec$ is a well ordering of D is Π^1_1, the set Ω_1 of quadruples $(D, \prec, \{(h_d, b_d)\}_{d\in D}, c)$ where $(D, \prec)$ is a well ordering as above is Π^1_1. We call the elements in Ω_1 *strings*.
(2) If Ω_2 is the set of strings in Ω_1 where the order type of $(D, \prec)$ equals $\omega\cdot(\alpha+1)$ for some α, we still have a Π^1_1-set.
(3) Let Ω_3 be the strings in Ω_2 that corresponds to a possible evaluation of $\mathcal{I}$ on some G. This requires that the calculation is *locally correct*, i.e. that each (h_d, a_d) is in relation to its $(D, \prec)$-predecessors as prescribed by the recursion step. This can be decided arithmetically, so Ω_3 is also Π^1_1.
(4) For a string in Ω_3, we can arithmetically decide if the enumeration $(D, \prec)$ is as in the construction above, so the representation Ω_4 of the calculations with denotations in the procedure for $\mathcal{I}$ will also be Π^1_1. □

Remark 4.17. We introduce delays in this construction. Whenever we simulate one step in the induction, we must wait until we know if we are at the final step or not before we can decide what the denotation will be, and this involves a delay of length ω.

Definition 4.18. Let Φ be a total functional of type 3. We say that Φ is Π^1_1-*definable* if Φ is definable from a Π^1_1 d-procedure.

Lemma 4.19. *The class of Π^1_1-definable total functionals of type 3 is closed under relative Kleene computability.*

Proof. We build on the proof of Theorem 4.8 and the construction in Definition 4.5. We just have to show how to add the denotations d_β to each item in the calculation, and then show that the complexity of the representation is preserved. We define the d-procedure as follows:

In the cases of initial computations there are no ordinals to be denoted, and in the case of S9 we can keep the denotations as they are.

In the case of composition, we can use $d \mapsto \langle 0, d\rangle$ to denote the items in the first part and the map $d \mapsto \langle 1, d\rangle$ to denote the items in the second part.

When we compute g and then apply F to g, we use the map $d \mapsto \langle c+1, d\rangle$ to denote the items coming from the calculation computing $g(c)$ and end the full subcalculation with $(g, F(g), \langle 0, 0\rangle)$.

In the case where we apply the procedure Ω_i for Φ_i to a partial functional H of type 2 for which we have an index, our calculation will be the concatenation of the calculations related to the computations of $H(g_\beta) = a_\beta$, where we also inserted $(g_\beta, *)$ in front of each such local calculation. If the Ω_i-denotation for the pair (g_β, a_β) in the calculation evaluating $\Phi_i(H)$ is d_1, we use $\langle d_1, 0\rangle$ to denote $(g_\beta, *)$ in the calculation we construct, and if an item in the calculation defined from the computation of $H(g_\beta) = a_\beta$ is d_2, we let $\langle d_1, d_2+1\rangle$ be the denotation of the corresponding item in the concatenated calculation.

It is clear that if two calculations, as in Definition 4.5 are equal, the denotations will be the same as well. This defines a d-procedure.

It remains to show that the representation of this d-procedure will be Π^1_1 when the representations of the d-procedures for $\Phi_1, \dots, \Phi_n$ are Π^1_1. This will be the hard, technical part of our proof, and we first give a brief explanation of what we aim to do:

We let the Π^1_1-representations for $\vec{\Phi}$ be given. Using the recursion theorem for computing relative to ${}^2E$, we will design an algorithm that, given e, $\vec{f}$, $\vec{a}$ and a representation

$$\bigl(D, \prec, \bigl\{(f_d, a_d)\bigr\}_{d\in D}\bigr)$$

of a string with denotations (a d-string for short) will semi-check, in the sense of providing an algorithm relative to ${}^2E$ that terminates when the property holds, if there is some F such that this string matches F and that the d-string gives us the representation of the calculation we constructed for the computation $\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a})$. In addition, if our algorithm finds the representation $(D, \prec, \{(f_d, a_d)\}_{d\in D})$ adequate as the representation of a d-calculation, it will produce the value of the computation $\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a})$, which then will be the same for any F matching the given d-string (which by now is confirmed as a d-calculation). Since termination of ${}^2E$-algorithms is of complexity Π^1_1, this will prove the lemma. Without stressing this point everywhere needed, we assume that the given d-string matches itself, in the sense that if both (f, a) and (f, a') occur, maybe at different places, then $a = a'$.

As usual, our ${}^2E$-procedure will be defined by cases following S1–S9, where we only focus on the nontrivial cases.

If e is an index for an initial computation, we check if the given string is empty. If so, it is fine as a calculation, and we can read off the value from the index, the given $\vec{f}$ and $\vec{a}$.

Composition:

$$\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a}) = \{e_1\}\bigl(\vec{\Phi}, F, \vec{f}, \{e_2\}(\vec{\Phi}, F, \vec{f}, \vec{a}), \vec{a}\bigr).$$

First we check if $(D, \prec)$ is of the form $(D_2, \prec_2) + (D_1, \prec_1)$ where each $d \in D_2$ is of the form $\langle 0, d'\rangle$ and each $d \in D_1$ is of the form $\langle 1, d'\rangle$.

Let $D_2' = \{d' : \langle 0, d'\rangle \in D_2\}$ and consider the corresponding string inherited from the given one. If this is ok for the computation $\{e_2\}(\vec{\Phi}, F, \vec{f}, \vec{a})$, we compute the value c', and now ask if the $(D_1, \prec_1)$ is ok for $\{e_1\}(\vec{\Phi}, F, \vec{f}, c', \vec{a})$ in the same sense.

Application of F:

$$\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a}) = F\bigl(\lambda c.\{e_1\}(\vec{\Phi}, F, \vec{f}, c, \vec{a})\bigr).$$

First we check if the given string has a last element $(g, a, \langle 0, 0\rangle)$ and if what comes before can be seen as an ω-sum of intervals I_c where the denotations are of the form $\langle c + 1, d\rangle$.

If this is the case, the given string generates, in analogy with the case for composition, strings t_c, and we check for each of them if they are ok for the computation $\{e_1\}(\vec{\Phi}, F, \vec{f}, c, \vec{a})$, and with value $g(c)$.

If they are all ok we accept the given string as a calculation, and see that the value of $F(\lambda c.\{e_1\}(\vec{\Phi}, F, \vec{f}, c, \vec{a}))$ must be a.

Application of Φ_i:

$$\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a}) = \Phi_i\bigl(\lambda g.\{e_1\}(\vec{\Phi}, F, g, \vec{f}, \vec{a})\bigr).$$

This is where we need the extra information stored in the LOG. We proceed as follows:

In the given string, first check if $(D, \prec)$ is the union of intervals where the first element of the interval is of the form $\langle d_1, 0\rangle$ and the rest are of the form $\langle d_1, d_2 + 1\rangle$.

Then check for each of these intervals, where $g = f_{\langle d_1, 0\rangle}$, if the rest of this interval, after replacing $\langle d_1, d_2 + 1\rangle$ with d_2, is ok for $\{e_1\}(\vec{\Phi}, F, g, \vec{f}, \vec{a})$, and if so, compute the value a.

Finally, we collect these pairs (g, a) with denotation d_1 into a string, and check if this is a calculation in Ω_i with some value c. For this, we use Gandy selection, and we then find the correct value as well.

In order to complete the argument we must prove that if this process works, then the computation in question, relative to any F matching the given string, will terminate with the chosen value, and prove that if the computation terminates for a total F, then our process terminates on the corresponding representation of the d-calculation, and again, that it gives the right value. Both arguments are by induction on the length of computations, the first for ${}^2E$-computations and the latter for the computation of $\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a})$. The details are trivial. □

5. Inductive procedures

As a consequence of Lemmas 4.16 and 4.19 we see that all total functionals of type 3 computable in $\mathcal{I}$ will be definable from a d-procedure that is Π^1_1, but the converse is not true, see Theorem 6.1.

The aim of this section is to narrow down a subclass of the d-procedures further in order to approach a characterisation of the class we are primarily interested in, the functionals computable in $\mathcal{I}$.

5.1. Computability in ${}^2E$

Matters are trivial if the d-procedure is hyperarithmetical:

Theorem 5.1. *Let Φ be of type 3. Then Φ is computable in ${}^2E$ if and only if Φ is definable from a d-procedure with a Δ^1_1-representation.*

Proof. First let Φ be definable from the d-procedure Ω, and assume that the representation is Δ^1_1. By the boundedness theorem for Σ^1_1-sets of codes for ordinals, see e.g. [30, Exercise II 5.9], there will be a computable ordinal λ such that all calculations in Ω have order-types bounded by λ. Let $(X, \lhd)$ be a computable well-ordering of length λ, and for each $x \in X$, let $X_x = \{y \in X : y \lhd x\}$. For each F, and by recursion on the $\lhd$-rank of $x \in X$, we will use F and ${}^2E$ to *compute* a string indexed by X_x that matches F and is, modulo the choice of denotations, isomorphic to the calculation in Ω matching F, until Ω tells us what the value $\Phi(F)$ must be. We use the recursion theorem, and explain the step from x to its $\lhd$-successor x'. So, as an induction hypothesis, we assume that we have constructed the string $t = \{(f_y, a_y, y)\}_{y \lhd x}$. This string is *isomorphic* to an initial segment of $((D, \prec, \{(f'_d, a'_d)\}_{d \in D}, c)$ if there is a $d \in D$ with the same rank as x, and the corresponding isomorphism r from X_x to D_d will satisfy that $f_y = f'_{r(y)}$ and $a_y = a'_{r(y)}$ for all $y \in D_x$. Now, the set Ω_t of calculations in Ω such that the string t is isomorphic to an initial segment will be Δ^1_1 relative to t.

In order to know what to do next, we first have to split between the two cases: are we able to give out a value for $\Phi(F)$ or must we continue the evaluation, that is, identifying, up to isomorphism, a larger part of the calculation matching F? Since Ω has a calculation for $\Phi(F)$, we know that there is at least one calculation in Ω of which t is isomorphic to an initial segment. Moreover, if t is actually isomorphic to a calculation in Ω, this is unique, and Ω provides us with the value. So, in order to decide between the two cases, we ask a "$\Delta^1_1(t)$"-question, i.e. a $\Sigma^1_1(t)$-question and a $\Pi^1_1(t)$-question that are equivalent. Those are:

'is t isomorphic to a proper initial segment of some element of Ω_t?'

and

'is t isomorphic to proper initial segments of all elements in Ω_t?'.

In the case the answer is 'no', we have constructed a copy of the calculation matching F. Then we can compute the unique value $\Phi(F) = c$ from the data. On the other hand, if the answer is 'yes', there will be a next query f_x that will be unique for all calculations in Ω_t. $\{f_x\}$ is a Δ^1_1-singleton relative to t, and we can compute each $f_x(n)$ from t and ${}^2E$. In both cases, we can rely on Gandy selection. This proves the theorem one way.

Now assume that $\Phi(F) = \{e\}({}^2E, F)$. We can construct a d-procedure Ω for Φ in analogy with the one we constructed in the proof of Lemma 4.19, without relativizing the construction to a set of $\vec{\Phi}$ with Π^1_1-procedures. That the result now will be Δ^1_1 can be seen from the following consideration:

(i) Whenever $\{e\}(^2E, F) \downarrow$ we can compute the corresponding calculation

$$\big(D, \prec, \big\{(f_d, a_d)\big\}_{d \in D}, c\big)$$

uniformly in e, F and $\exists^2$, by use of the recursion theorem. It is worth noticing that there will be no delay here, given $\{f_d, a_{d'}\}_{d' \prec d}$, we do not only have a unique value for the next f, but also for its denotation d, even if the calculations are matching different F s.

(ii) Next we observe that when checking if a representation $(D, \prec, \{f_d, a_d\}_{d \in D})$ of a d-string is a real representation of a real calculation of a value, we can relax the requirement that $(D, \prec)$ is a well ordering. The checking the way we did it in the proof of Lemma 4.19 is partially computable by the recursion theorem, and can be proved to terminate for a given e under the assumption that there is at least one total F matching the given d-string such that $\{e\}(^2E, F) \downarrow$.

This shows that the d-procedure will be Δ^1_1 in this case. □

Remark 5.2. We have essentially used that the element of a Δ^1_1-singleton is itself hyperarithmetical, and this implicitly provides us with a NEXT-function in this case.
There is no delay in the Δ^1_1-d-procedure constructed in the above argument, the denotations in an initial segment of a calculation is uniquely determined by the segment itself.

5.2. The class $\mathbf{W}(\mathcal{I})$

In this section we will give a closer analysis of the class of functionals of type 3 that are computable in $\mathcal{I}$. We will do so by investigating additional properties of the elements in the following class:

Definition 5.3. Let $\mathbf{W}(\mathcal{I})$ be the set of d-procedures for functionals Φ computable in $\mathcal{I}$ as constructed in the proofs of Lemma 4.16 and Lemma 4.19.

5.2.1. Tame d-procedures

In this sub-section we will introduce two properties shared by all d-procedures in $\mathbf{W}(\mathcal{I})$.

Definition 5.4. Let Ω be a procedure defining a total functional.

(a) Let Ω_{pre} be the set of $(D, \prec, \{(f_d, a_d)\}_{d \in D})$ that are isomorphic to an initial segment of a calculation in Ω.
(b) Let NEXT_Ω be the function mapping $t \in \Omega_{\text{pre}}$ to the disjoint union of $\mathbb{N}^{\mathbb{N}} \times \{0, 1\}$ and $\mathbb{N}$ such that

- If t is isomorphic to a calculation t' in Ω, then $\text{NEXT}_\Omega(t)$ is the value of this calculation
- If t is isomorphic to a proper initial segment t' of a calculation t'' in Ω, then $\text{NEXT}_\Omega(t) = (f, c)$ where $F(f) =?$ is the next query after t' in t'' (independent of the choice of t'') and $c \in \{0, 1\}$. Moreover, if $c = 0$, then the next pair in t'' after t' will be of the form $(f, *)$ while if $c = 1$, we continue t' with a pair (f, a) in t'' for some $a \in \mathbb{N}$.

Definition 5.5. Let Ω be a d-procedure for a total functional. We say that Ω is *tame* if Ω is Π^1_1, Ω_{pre} is Π^1_1 and NEXT_Ω is partially computable in ^2E.

Lemma 5.6. *Let Ω be the d-procedure for $\mathcal{I}$. Then Ω is tame.*

Proof. We need the full complexity of Π^1_1 to formulate that we are dealing with well orderings $(D, \prec)$, the rest is actually arithmetical. Each step in the underlying recursion takes ω many steps when we evaluate according to Ω. It is clearly arithmetical to decide if any ordered set of pairs (f, a) indexed over $\mathbb{N}$ locally satisfies the recursion in $\mathcal{I}$, so checking if a d-string is in Ω_{pre} is arithmetical when we know that the representation is well ordered. If a string is locally correct, the next query is arithmetically defined if there is one, and the value is arithmetically expressible from the list of queries answers if the d-string corresponds to a calculation, so NEXT_Ω is computable in ^2E as requested. □

Lemma 5.7. *The class of functionals definable from tame d-procedures Ω is closed under Kleene computability.*

Proof. Let $\vec{\Phi} = \Phi_1, \dots, \Phi_n$ be defined from the tame d-procedures $\Omega_1, \dots, \Omega_n$. We already know that the d-procedure for any Φ computable in $\vec{\Phi}$ is Π^1_1.

We use the recursion theorem to define, for each index e and extra inputs $\vec{f}$ and $\vec{a}$, a set $X_{e,\vec{f},\vec{a}}$ of d-strings

$$\bigl(\bigl(D, \prec, \bigl\{(f_d, a_d)\bigr\}_{d\in D}\bigr)$$

that is Π^1_1 uniformly in e, $\vec{f}$, $\vec{a}$, together with the function

$$\text{NEXT}_{e,\vec{f},\vec{a}}$$

defined on $X_{e,\vec{f},\vec{a}}$, and show that

(i) If F is of type 2 and $\{e\}(\Phi_1, \dots, \Phi_n, F, \vec{f}, \vec{a}) = c$ and $(D, \prec, \{(f_d, a_d)\}_{d\in D}, c)$ is the associated d-calculation obtained from the proof of Lemma 4.19, then any string isomorphic to an initial segment of $(D, \prec, \{(f_d, a_d)\}_{d\in D})$ is in $X_{e,\vec{f},\vec{a}}$.

(ii) If a string is in $X_{e,\vec{f},\vec{a}}$ and matches some F for which $\{e\}(\Phi_1, \dots, \Phi_n, F, \vec{f}, \vec{a})$ terminates, then the string is isomorphic to an initial segment of the d-calculation obtained through the proof of Lemma 4.19.

(iii) $\text{NEXT}_{e,\vec{f},\vec{a}}$ is computable in ${}^2E$ uniformly in the parameters and acts as specified.

We define $X_{e,\vec{f},\vec{a}}$ and $\text{NEXT}_{e,\vec{f},\vec{a}}$ by cases according to the scheme e represents. Note that since we are dealing with semi-decidable sets, we cannot take NO for an answer, and search-procedures have to use Gandy selection:

- e is an index for an initial computation given by S1-S3, S7: $X_{e,\vec{f},\vec{a}}$ will consist of the empty string only. $\text{NEXT}_{e,\vec{f},\vec{a}}$ is trivially given in all these cases.

- Composition:

$$\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a}) = \{e_1\}\bigl(\vec{\Phi}, F, \vec{f}, \{e_2\}(\vec{\Phi}, F, \vec{f}, \vec{a}), \vec{a}\bigr).$$

Given $(D, \prec, \{(f_d, a_d)\}_{d\in D})$, where $(D, \prec)$ is a well ordering, we use $(D, \prec)$–recursion to test if the initial segments of $(D, \prec, \{(f_d, a_d)\}_{d\in D})$ are in $X_{e_2,\vec{f},\vec{a}}$ until we either find an initial segment that is in $X_{e_2,\vec{f},\vec{a}}$ and with a value c or we find that the given string is in $X_{e_2,\vec{f},\vec{a}}$, and thus in $X_{e,\vec{f},\vec{a}}$. If this search fails, the given string is not in $X_{e,\vec{f},\vec{a}}$, and non-termination is not a problem. If this search ends with a proper substring that is in $X_{e_2,\vec{f},\vec{a}}$ and with a value c, we compare the rest of the string with $X_{e_1,\vec{f},c,\vec{a}}$ in the same way. The NEXT-function for e will be inherited from the NEXT-functions of e_2 and e_1, c, and these can be used to check that the given string does not go too far, beyond where we should have a valued string.

– We leave the cases for permutation and S9 for the reader, as those cases are even simpler.

– Application of F:

$$\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a}) = F\bigl(\lambda a.\{e_1\}(\vec{\Phi}, F, \vec{f}, a, \vec{a})\bigr).$$

This will be in analogy with how we dealt with composition, and technically we use transfinite recursion over $(D, \prec)$. We first compare the given string poin-by-point with elements in $X_{e_1,\vec{f},0,\vec{a}}$. If the given string is in $X_{e_1,\vec{f},0,\vec{a}}$, then it is acceptable. If an initial segment of the given string is a maximal element of $X_{e_1,\vec{f},0,\vec{a}}$, then we compare the remaining part with $X_{e_1,\vec{f},1,\vec{a}}$ and so forth. If any of those local comparisons fails, then our global comparison fails. If the comparison is successful, and we find some n and an end segment of our given string in $X_{e_1,\vec{f},n,\vec{a}}$, we accept the string. The last possibility is that the given string has an initial segment that is a concatenation of maximal strings from each $X_{e_1,\vec{f},n,\vec{a}}$ in increasing order. If this is the full given string, it is also acceptable. If not, we must check that it contains a final (g, b) at the end, and in this case, we must also check for each a if $g(a)$ is the value of the string from $X_{e_1,\vec{f},a,\vec{a}}$ used in the concatenation before accepting the given string. When accepted, we inherit the NEXT-function in the obvious way.

– Application of Φ_i:

$$\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a}) = \Phi_i\big(\lambda g.\{e_1\}(\vec{\Phi}, F, g, \vec{f}, \vec{a})\big).$$

Here it may be useful to look back on Definition 4.5. We explain informally how we, point by point, compare the initial segments of the given string $(D, \prec, \{(f_d, a_d)\}_{d\in D})$ with the requirements for Ω_i and the various sets

$$X_{e_1,g,\vec{f},\vec{a}}$$

where we may assume that we have defined these sets as a part of the induction hypothesis:

- If d_0 is the $(D, \prec)$-least element, f_{d_0} has to be the first query g_0 in Ω_i, given to us by NEXT_{Ω_i}, with $a_{d_0} = *$.
- We check the next segment of $(D, \prec, \{(f_d, a_d)\}_{d\in D})$ for membership in $X_{e_1,g,\vec{f},\vec{a}}$ until we have reached a value or until we have reached the end of the given string.
- In the latter case, the string is in $X_{e,\vec{f},\vec{a}}$ and in the first case, we let b_0 be the value, check that the pair (g_0, b_0) is in $\Omega_{i\,\text{pre}}$ and use the NEXT-function of Ω_i to verify that the next query in Ω_i will be the next query in $(D, \prec, \{(f_d, a_d)\}_{d\in D})$.
- By transfinite recursion on $(D, \prec)$ we can continue this comparison until we have reached the end of the given string or until it does not compare with strings in Ω_i (whenever we have found a new value there, and can use its NEXT-function to find the new g_β), or with the strings in the sets $X_{e_1,g_\beta,\vec{f},\vec{a}}$.

It is now routine to verify the properties (i)–(iii). □

5.2.2. Blocking

It is not the case that all functionals definable by a tame d-procedures will be computable in $\mathcal{I}$, see Theorem 6.1. The point with the denotations is that they may make it easier to design non-monotone inductions that are copying evaluations in a procedure, but the obstacle will be that we will not, in general, be able to tell from a part of a calculation what the correct denotation of the next query will be, there may be a *delay* as defined in 4.14. We find this phenomenon in the procedure for $\mathcal{I}$, where we first must establish, within each in a series of ω-length sub-calculations, if we reached the final fixed point or not before reading off the correct denotation. That the situation would be simpler without this obstacle, is seen from the following lemma:

Lemma 5.8. *Let Ω be a d-procedure for a total functional such that for any sub-string $\{(f_\beta, a_\beta, d_\beta)\}_{\beta<\alpha}$ of a calculation in Ω, the denotations d_β are uniquely given by $\{(f_\beta, a_\beta)\}_{\beta<\alpha}$.*

If Ω in addition is tame, and the unique choice of d_β is computable from $\{(f_\gamma, a_\gamma\}_{\gamma\leqslant\beta}$ and ${}^2E$ uniformly at each stage, then the functional defined by Ω is computable in $\mathcal{I}$.

Proof. We code an entry $(f_\beta, a_\beta, d_\beta)$ as the set

$$\big\{\langle \bar{f}_\beta(n), a_\beta, d_\beta\rangle : n \in \mathbb{N}\big\}.$$

These sets will be disjoint, so we may code each initial segment of a string as a pair of sets, where one is the union X_β of such single codes and the other is the corresponding ordering of the denotations. We will use that the Suslin functional **S** is computable in $\mathcal{I}$.

Given F, we design a non-monotone inductive definition Γ_F computable in **S** that simulates the evaluation of the calculation matching F:

(1) Given X, we use **S** to check if X codes an initial segment of a calculation in Ω as coded above. If not, let $\Gamma_F(X) = X$, and if it does, continue.

(2) Use NEXT_Ω to find the next query f_α, use F to find $a_\alpha = F(f_\alpha)$ and finally the ${}^2E$ algorithm that gives us the unique denotation d_α.
(3) Let $\Gamma_F(X)$ be X extended with the code for $(f_\alpha, a_\alpha, d_\alpha)$ in the set to the left and all pairs $\langle d_\beta, d_\alpha\rangle$ for $\beta < \alpha$ in the set to the right.

It is clear that the set $\mathcal{I}(\Gamma_F)$ will code the calculation in Ω matching F, together with the ordering of all the denotations used in that calculation, and we can use NEXT_Ω to compute $\Omega(F)$.

Further details are left for the reader. □

We will now add further structure to d-procedures, *blocks*. It will be like inserting commands of the form \begin{`block`} and \end{`block`} bracketing blocks and sub-blocks. These imaginary commands must satisfy, for each calculation, the standard rules of bracketing, allowing for infinite branchings in the length, but only finite nesting in depth. Where to put these commands will be determined by the initial segment of the string up to where the command is, and the use will be to mark that there is now an uncertainty to what the denotations will be at the end, and that we have to carry out a sub-procedure, or evaluate the calculation for F a bit further, in order to find the true denotations of the calculation. We will consider two examples before giving the abstract definition of a tame d-procedure with blockings:

Example 2. Let Ω be the d-procedure for $\mathcal{I}$. Recall that, given G, $\Omega(G)$ will iterate the induction given by F_G, generating the sequence $\{f_\beta\}_{\beta\leqslant\alpha}$ by evaluating G on $0\,\hat{}\,f_0, 1\,\hat{}\,f_0, \ldots_\omega 0\,\hat{}\,f_1, 1\,\hat{}\,f_1 \ldots_{\omega(\alpha+1)}$.

Each ω-sequence will be a block in this case, and after each block we know what the denotation for the query $G(n\,\hat{}\,f_\beta) =?$ will be, but not while we are inside a block. However, in order to view the calculation within a block as a sub-procedure, we only need denotations that are unique for queries within this block, and ignore the larger picture. It is not hard to modify the proof of Lemma 5.8 to see that we can simulate the procedure for $\mathcal{I}$ using $\mathcal{I}$.

Example 3. In the case of composition we constructed the calculations as concatenations of strings for the two parts, and when defining the new denotations, we paired the denotations from the first part with 0 and from the second part with 1. We may consider the first part as one block and the other part as another one, but if we know from the larger picture that we are entering a composition, there is no need for this. There is no delay in deciding what the denotations are inherited in the construction of denotations for compositions, as there is for the construction of denotations for transfinite recursions with an unknown end.

We will need the blocking structure on calculations to characterise functionals of type 3 computable in $\mathcal{I}$ in terms of procedures.

The blocks will be organised in a nested way, with some blocks being sub-blocks of others. The point is that, within each block, we will define denotations along the way, and when the need of a delay is observed, we enter a sub-block where we form temporary denotations that at the end of the block will be rewritten to the true ones. The nesting of the blocks will reflect that there may be delays within a period of delay, so the rewriting of denotations may go through several levels.

We will now give the full definition:

Definition 5.9. Let Ω be a tame d-procedure.

(a) A *block* in Ω is an interval t_2 in a calculation $t_1\,\hat{}\,t_2\,\hat{}\,t_3$ in Ω. Blocks t_2 in $t_1\,\hat{}\,t_2\,\hat{}\,t_3$ and t_2' in $t_1'\,\hat{}\,t_2'\,\hat{}\,t_3'$ are considered to be equal only when $t_2 = t_2'$ and we in addition have that $t_1 = t_1'$.
(b) A *blocking* of Ω is a set of blocks for each calculation t in Ω satisfying:

   (i) Given two blocks in t, they are either disjoint or one is included in the other.
   (ii) For each calculation, all chains of blocks totally ordered by inclusion will be finite.
   (iii) t is one of the blocks in t.
   (iv) The *level* of a block t_1 in t is the number of other blocks in t properly containing t_1 as a substring.

(v) If $t = t_1 \,\hat{}\, t_2$ and $t' = t_1 \,\hat{}\, t_2'$ are two calculations in Ω with a common initial substring t_1, and if a block of level m in t starts at the beginning of t_2, then a block of level m starts in t' at the beginning of t_2'. Moreover, if the two blocks coincide until one of them ends, they are equal.

(iii) above just makes the rest easier to express.

(c) The blocking is *tame* if we in addition have

(i) There is a partial function BLOCK_Ω computable in ${}^2E$ such that for each string t in Ω_{pre}, $\mathrm{BLOCK}_\Omega(t)$ decides for each m if there is a block of level m starting at the next query $\mathrm{NEXT}_\Omega(t)$ and decides the levels of the blocks, if any, ending before the next query.

(ii) For each block $s = \{(f_\delta, a_\delta)\}_{\gamma \leqslant \delta < \beta}$ in a calculation t there is a unique injective denotation $\{d^s_\delta\}_{\gamma \leqslant \delta < \alpha}$, where these denotations are computed using the two functions DENOTE and REDENOTE (with subscript Ω if needed) both computable in ${}^2E$ and where

- if $t_1 \,\hat{}\, (f_\beta, a_\beta)$ is an initial substring of the calculation t and s is the block of highest level containing (f_β, a_β) then $\mathrm{DENOTE}(t_1, a_\beta)$ will be the denotation d^s_β.
- If s is a block of level $m > 0$ contained in the calculation $t = t_1 \,\hat{}\, s_1 \,\hat{}\, s \,\hat{}\, s_2 \,\hat{}\, t_3$, where $t_2 = s_1 \,\hat{}\, s \,\hat{}\, s_2$ is the surrounding block of level $m - 1$, then REDENOTE with input $t_1 \,\hat{}\, s_1 \,\hat{}\, s$ and m will give us d^{t_2} restricted to $s_1 \,\hat{}\, s$.

Comment 1. It is (c), (ii) that captures the essence of blocking, a block represents the local delay of deciding what the denotation one level up will be like, and will make it possible to simulate the evaluation of a procedure as a nested application of $\mathcal{I}$.

Definition 5.10. An *Inductive Procedure* is a tame Π^1_1-procedure with a tame blocking.

Theorem 5.11. *A functional Φ of type 3 is definable from an inductive procedure if and only if Φ is computable in $\mathcal{I}$.*

Proof. We first show that if Φ is definable from an inductive procedure, then Φ is computable from $\mathcal{I}$. We use a nested version of the argument in Lemma 5.8, using everywhere that the functions NEXT, BLOCK, DENOTE and REDENOTE are computable in ${}^2E$, and thus in $\mathcal{I}$. We use the recursion theorem for Kleene computations to make the following precise:

Given F, we construct the inductive definition Γ_F as in Observation 5.8, using the denotation as it comes, until we hit the beginning of a block s. Then we start the execution of a sub-procedure simulating the evaluation of F along s as an inductive definition Γ^s_F in the same way. This sub-procedure will come to an end when the full evaluation along s is simulated. At this stage we can describe the next step for Γ_F: From the output of Γ^s_F, ${}^2E$ and the assumption on blocks, we can compute the correct denotations along the string up to the end of s, and Γ_F just ads the full simulation of the evaluation of F in Ω using this denotation.

If s has sub-blocks, then Γ^s_F will have sub-procedures in a similar way, this is why we need the recursion theorem to formally describe this procedure.

In order to prove the other direction we elaborate on the proofs that the class of functionals satisfying our requirements is closed under Kleene computability, and assume that $\Phi_1, \ldots, \Phi_n$ now are defined from inductive procedures.

The only case we need to consider is that of application of Φ_i,

$$\{e\}(\vec{\Phi}, F, \vec{f}, \vec{a}) = \Phi_i\big(\lambda g.\{e_1\}(\vec{\Phi}, F, g, \vec{f}, \vec{a})\big).$$

Let $G(g) = \{e_1\}(\vec{\Phi}, F, g, \vec{f}, \vec{a})$. Then there is a calculation $\{(g_\beta, b_\beta)\}_{\beta<\alpha}$ in Ω_i matching G. For each $\beta < \alpha$, the pair (g_β, b_β) will be replaced by a substring of the composed calculation as follows: $(g_\beta, *)$ will just be preserved as it is, while (g_β, b_β) is replaced by a string starting with $(g_\beta, *)$ and continued with the calculation of $F(g_\beta)$.

When we defined the denotations for these composed calculations, we gave them directly from the denotations in the Ω_i-calculation and from the denotations in the calculations of $G(g_\beta)$ without adding any further delay. Thus we can inherit the blocking structure of the Ω_i-calculation, and whenever (g_β, a_β) is in one of these blocks before we compose all the substrings, we let all blocks in t, where $(g_\beta, *)\hat{\ }t$ is inserted for (g_β, a_β) and t is the calculation of $G(g_\beta)$, be new blocks of a higher level.

When we constructed the d-procedure in this case, we gave the rules for transforming the denotations in this simulating block to denotations of the corresponding items in the full calculation, and this clearly can be relativised to the blocks in Ω_i.

In order to tie the whole thing up showing that the definability and computability requirements of what we construct are satisfied, we need to use the recursion theorem for ${}^2E$, induction on the ordinal lengths of $\vec{\Phi}$, F-computations and induction/recursion on the level of blocks in a string. The details are tedious, but simple. $\square$

5.3. The $\mathcal{I}$-computable functions revisited

In this section we will consider pure computations

$$\{e\}(\mathcal{I}, \vec{a}),$$

without function and functional parameters.

In our definition of a procedure, we used the parameter F to give values to queries, but at certain points we also inserted elements of the LOG, functions appearing as arguments in sub-computations but not necessarily as arguments in the main computable function or functional we design the procedure for.

When transforming a computation in $\mathcal{I}$ without functional arguments to a procedure, this LOG will now be the backbone of the calculation. Since there will be no genuine queries anymore, we can even drop the $*$ for marking element-hood in the LOG. Thus a *pure string* will be a triple $(D, \prec, \{g_d\}_{d\in D})$ where $(D, \prec)$ is a well ordering and each $g_d \in \mathbb{N}^{\mathbb{N}}$. We will consider such strings that are Π^1_1-singletons, where the set of other strings isomorphic to initial segments of the given one is Π^1_1, where we have a NEXT-function computable in ${}^2E$ and where we have functions BLOCK, DENOTE and REDENOTE as before, making this one-point set an inductive procedure. We call this a *pure inductive calculation*, and these pure inductive calculations will reflect $\mathcal{I}$-computations with integer inputs.

Remark 5.12. While transforming a computation $\{e\}(\mathcal{I}, \vec{a}) = b$ to a pure inductive calculation we first of all linearised the computation. Then we hid some of the indexing in the function NEXT, and also, in the way we designed the denotations, we actually "hid" all intermediate "Kleene-calculations" that do not involve the scheme S8. However, for transfinite computations, this *hiding* will not significantly reduce the length of a computation. On the other hand, when we translate a pure inductive calculation to a Kleene-computation, we may add to the length of the computation, partly because it takes time to compute NEXT, BLOCK, DENOTE and REDENOTE whenever needed, and partly because we have to add time to the time-span of an induction in order to verify that the induction comes to a halt when it does.

Recall the definition of π as the first ordinal with no code computable in $\mathcal{I}$. We have the following characterisation.

Lemma 5.13. *π is the supremum π^* of the ordinal lengths of the pure inductive calculations.*

Proof. First, we will prove that $\pi \leqslant \pi^*$. Let $\alpha < \pi$, and let $(X, \prec)$ be $\mathcal{I}$-computable and a well ordering of ordinal length α. We prove this direction by constructing a pure inductive calculation with at least α many steps. We will use the pure inductive calculations deciding $x \in X$ and $x \prec y$ as building blocks, and with the help of those we simulate, in the form of a grand pure inductive calculation, the induction building up X one point at each step, a process that needs exactly α many steps.

Then we prove that $\pi^* \leqslant \pi$. Let t be a pure inductive calculation. Using the same strategy we used when showing that a functional definable from an inductive procedure is computable in $\mathcal{I}$, a strategy involving the recursion theorem for $\mathcal{I}$, we can show that there is a nested computation relative to $\mathcal{I}$ that computes a code for the ordinal length of t, whenever we enter a block, we compute the length of that block as a subcomputation, and then at the end of the block, ad a copy of this code to the well ordering we are building up. Actually, it will be the denotations with their ordering we compute, and in a block, the local denotations within that block. □

Theorem 5.14. *Let $\alpha < \pi$. If α is the rank of a pure inductive calculation t, then α is Π^1_1-characterisable.*

Proof. If $f \in \mathsf{WO}$ has rank β, we can decide if $\beta = \alpha$ as follows: By recursion on the wellordering coded by f we can use $\textsc{next}_t$ and the ${}^2E$-algorithm organising the blockings to compute the corresponding stages in t with full information about where in the blocking structure we are at each step. If this simulation terminates exactly at the end of t, we accept f, otherwise we refute it. The set of f s accepted will be Π^1_1. □

6. Tracing computations in $\mathcal{I}$

6.1. Partial procedures

In order to make constructions of procedures smoother, we have not insisted on the natural requirement that for each f and calculation, there is at most one occurrence of the query $F(f) =?$. However, when it does appear several times, the calculation will be based on the same answer everywhere. When we refer to a query $F(f) =?$ we will always mean the first occurrence. When $\{e\}(\mathcal{I}, F, \vec{f}, \vec{a}) \downarrow$ for all F, it is clear that the associated procedure will lead to terminating calculations for all F. This means that when f appears in a query $F(f) =?$ there will be an extension into a calculation for all $a \in \mathbb{N}$. Conversely, we can prove that for every calculation $(\{(f_\beta, a_\beta)\}_{\beta<\alpha}, c)$ constructed in the procedure for $\lambda F.\{e\}(\mathcal{I}, F, \vec{f}, \vec{a})$, if F matches this string, then c will be the value of this computation.

We will now discuss what happens if we consider computations $\{e_0\}(\mathcal{I}, F)$ that do not terminate for all inputs F. In this case, we can still define a set Ω_{pre} of strings with denotations, in these strings we may enter blocks and sub-blocks, and they behave as required for inductive procedures, since we have established these properties for each e, F, $\vec{f}$ and $\vec{a}$ such that $\{e\}(\mathcal{I}, F, \vec{f}, \vec{a}) \downarrow$.

We need to consider parameters $\vec{f}$ and $\vec{a}$, since such parameters occur in subcomputations, so we reason within this generality.

If we consider the construction of the inductive procedure more carefully, we can observe more closely what we construct in the case of non-termination, again by cases according to what the index e is like:

If e is an index for an initial computation, we constructed a trivial procedure yielding the correct output without any queries made.

In the case of composition, we first constructed the procedure for the inner component, and for the calculations in this procedure (the strings that give us an answer), we concatenated with calculations from the procedure of the corresponding outer component. In the case the composed computation does not terminate, we do construct a string modelling the leftmost non-terminating subcomputation.

In the cases where there will be exactly one subcomputation, what we do is using the string of that one.

In the cases of application of F or application of $\mathcal{I}$, we are doing exactly as above, in case of non-termination we build up the string until we reach the leftmost non-terminating subcomputation, and ending the string in Ω with a copy of a string for this leftmost one.

In the case that e is not an index at all, the procedure will be trivial, but with non-termination as the conclusion.

If $\{e\}(\mathcal{I}, F, \vec{f}, \vec{a}) \uparrow$, there will be a *leftmost* Moschovakis witness, a descending sequence of unsettled computations such that every computation to the left will terminate, and it will be exactly the string corresponding to evaluate $\{e\}(\mathcal{I}, F, \vec{f}, \vec{a})$ along this descending sequence of subcomputations that will be constructed in this case. If Ω is constructed like this, we will simply have some strings where the conclusion must be $\perp$ instead of a proper value. However, since being a Moschovakis witness is semi-decidable (this was the original point with them), we see that Ω_{pre} will still be Π^1_1. We will also have functions $\textsc{next}$, $\textsc{block}$, $\textsc{denote}$ and $\textsc{redenote}$ computable in ${}^2E$. What may be lacking is that blocks may be entered without ever being left, that we may have an infinite descending sequence of blocks (that will then not have end points) and that we will not have a ${}^2E$-computable way to define the denotations for the blocks unless they have an end. So, the procedure will not be an inductive procedure. This is as it has to be, since we can define the characteristic function of a set of functions of type 2 that is complete semi-computable in $\mathcal{I}$ using a procedure like this, replace the value $\perp$ with 0 and the value $a \in \mathbb{N}$ with 1 as the values of calculations.

These considerations contain the proof of

Theorem 6.1. *There is a total procedure Ω such that Ω_{pre} is Π^1_1, and such that there is a partial function* NEXT_Ω *that is computable in ${}^2E$, but where the functional defined by Ω is not computable in $\mathcal{I}$.*

Proof. The only property left is that we must be able to decide, using a ${}^2E$-algorithm, if a string t in Ω_{pre} is maximal or not, and in case it is maximal, if it has a value or not. By recursion on the indexing of t we can follow the points in the computation tree corresponding to the points in t. If this point in the computation tree has an index that is none of the indices of S1–S9, we can conclude that there is no value. If the blocking depth is infinite, we can conclude that the string represents non-termination. This can be checked by ${}^2E$, using BLOCK and calculating the $\limsup$ of the block level of the items of the string. In all other cases, there will be a next query or there will be a value given to us by the original NEXT-function □

We also have

Theorem 6.2. *There is a non-terminating computation $\{e\}(\mathcal{I}, \vec{a})$ such that the length of the string simulating the leftmost Moschovakis witness will have length at least $\pi (= \omega_1^{\mathcal{I}})$.*

Proof. If this were not the case we can use Gandy selection for $\mathcal{I}$-computations and make $\mathcal{I}$-semidecidable equivalent to $\mathcal{I}$-decidable, obtaining the standard contradiction by diagonalisation. □

Remark 6.3. Moschovakis witnesses were introduced in [14], and they were significant for the understanding of higher order computing relative to normal functionals and in set recursion. They are also introduced in [12, Section 5.2.2], and they were applied in the proof of [20, Theorem 6.6].

6.2. Functionals computable in $\mathcal{I}$

In a series of papers [16–22,27], written jointly with Sam Sanders, we investigate classes of functionals of type 3 that serve as realisers for classical theorems in analysis, and where each class contains elements computable in $\mathcal{I}$. There are unsettled question concerning the relative computability of the elements of these classes. In this section we will see that in the case that elements of these classes are computable in $\mathcal{I}$, we can "almost" compute the Suslin functional from them, and consequently, they will "almost" be computationally equivalent to $\mathcal{I}$ itself. We will make the *"almost"* precise by replacing a functional Φ computable in $\mathcal{I}$ with one that traces the history of the computation, not just gives the value. We call this the *honest version of* Φ

There is an analogue with what we do in complexity theory where the complexity of a set often is measured by the resources required to decide membership in the set and not by what we can decide using small resources combined with the set as an oracle. In a mathematically precise way we will see that if we compute realisers for some classical theorems of analysis from $\mathcal{I}$, we need the full power of $\mathcal{I}$ in doing so.

Definition 6.4. Let $\Omega \in \mathbf{W}(\mathcal{I})$ be the inductive procedure constructed from a functional Φ as computable in $\mathcal{I}$. We define the *honest version* $\mathcal{H}(\Phi)$ as the functional (of mixed type) that sends a functional F of type 2 to the fixed point of the inductive definition Γ_F as constructed from Ω in the proof of Theorem 5.11, i.e. as the *history* of the evaluation of $\Phi(F)$ from $\mathcal{I}$.

An *open covering* of a set X in a topological space $\mathcal{T}$ is normally defined as a set $\mathcal{C}$ of open sets in $\mathcal{T}$ whose union is a superset of X. However, if we make use of the concept of realisers, a *realiser* of the open covering will be a map sending $x \in X$ to an open set $O_x \in \mathcal{T}$ such that $x \in O_x$. When Borel [3] gave an attempt of a direct proof of the Heine–Borel theorem, he actually, without knowing the concept, used this idea of a realiser; with free translation he expressed his assumption as follows:

($*$) Assume that we have a way of attaching an open neighbourhood O_x of x to each $x \in [a,b]$.

In the papers with Sanders, we have considered coverings and related concepts primarily over the *Cantor space* $\mathsf{C} = \{0,1\}^{\mathbb{N}}$ and the *Baire space* $\mathsf{B} = \mathbb{N}^{\mathbb{N}}$ given as functionals F of type 2, where $F(f)$ *defines* the neighbourhoods $\mathsf{C}_{\overline{f}(F(f))}$ and $\mathsf{B}_{\overline{f}(F(f))}$ of extensions of $\overline{f}(F(f))$, depending on which space we consider f to be an element of.

We have been looking at the following three classes:

Definition 6.5. (a) A *strong realiser* for the Heine–Borel theorem will be a functional Θ such that whenever $F : \mathsf{C} \to \mathbb{N}$, then $\Theta(F) = \{f_1, \dots, f_n\}$ such that

$$\mathsf{C} \subseteq \mathsf{C}_{\overline{f_1}(F(f_1))} \cup \dots \cup \mathsf{C}_{\overline{f_n}(F(f_n))}.$$

(b) A *weak realiser* for the Heine–Borel theorem will be a functional θ such that whenever $F : \mathsf{C} \to \mathbb{N}$ then $\theta(F) = \{s_1, \dots, s_n\}$ where each s_i is a finite binary sequence, $\{\mathsf{C}_{s_1}, \dots, \mathsf{C}_{s_n}\}$ is a covering of C and for each $i = 1, \dots, n$ there is an $f_i \in C$ such that $\overline{f_i}(F(f_i)) = s_i$.

(c) A *Pincherle realiser* will be a functional M such that whenever $F : \mathsf{C} \to \mathbb{N}$, then $M(F) = N \in \mathbb{N}$ and N satisfies:

($-$) If $G : \mathsf{C} \to \mathbb{N}$ satisfies that $G(g) \leqslant F(f)$ whenever $\overline{g}(F(f)) = \overline{f}(F(f))$ (F is considered as a realiser for local boundedness) then G is bounded by N on C.

The following lemma is trivial:

Lemma 6.6. *Every strong realiser for the Heine–Borel theorem computes a weak one, and every weak realiser for the Heine–Borel theorem computes a Pincherle realiser.*

The proof is left for the reader.

Lemma 6.7. *Let M be a Pincherle realiser that is countably based. Let $F : \mathsf{C} \to \mathbb{N}$ be arbitrary, and let $X \subset \mathsf{C}$ be countable such that $M(G) = M(F)$ for all G such that F and G are equal when restricted to X. Then $\{\mathsf{C}_{\overline{f}(F(f))} : f \in X\}$ is a covering of C.*

Proof. Assume not, let $M(F) = N$ and define

$$F_1(g) = \begin{cases} F(g) & \text{if } g \in \bigcup_{f \in X} C_{\overline{f}(F(f))} \\ N+1 & \text{if } g \notin \bigcup_{f \in X} C_{\overline{f}(F(f))} \end{cases}$$

Then $M(F) = M(F_1)$ because the two functions agree on X. However, if we define

$$G(g) = \begin{cases} 0 & \text{if } g \in \bigcup_{f \in X} C_{\overline{f}(F(f))} \\ N+1 & \text{if } g \notin \bigcup_{f \in X} C_{\overline{f}(F(f))} \end{cases}$$

then G obviously satisfies the boundedness condition induced by F_1, but is not bounded by N, contradicting that $N = M(F_1)$. $\square$

Theorem 6.8. *Let M be a Pincherle realiser that is computable in $\mathcal{I}$. Then the Suslin functional* $\mathbf{S}$ *is computable in $\mathcal{H}(M)$ and ${}^2E$.*

Proof. In [16, Theorem 5.1] it is proved that there is a functional F computable in ${}^2E$ such that the collection of neighbourhoods defined from F and the hyperarithmetical binary sequences is not a covering of C. The construction easily relativizes to an arbitrary $f \in \mathbb{N}^{\mathbb{N}}$ so it suffices to show how we can compute a complete Π^1_1-set from $\mathcal{H}(M)$, F and ${}^2E$.

For a general procedure Ω and an arbitrary G, the calculation of $\Omega(G)$ will form a countable basis for $\Omega(G)$. If F and M are as given, Lemma 6.7 then shows that the calculation of $\Omega_M(F)$ must contain queries that are not hyperarithmetical. However, in an inductive procedure, if the input functional is computable in ${}^2E$, then all queries appearing at the level of a computable ordinal must also be computable in ${}^2E$. This follows from the assumption that the NEXT-function is computable in ${}^2E$. So, the calculation of $\Omega_M(F)$ must have a rank that goes beyond ω_1^{CK}. The set of (indices for the) computable well-orderings will then be both Σ^1_1 and Π^1_1 in this calculation, and thus computable from this calculation using ${}^2E$. The calculation itself is computable from F and $\mathcal{H}(M)$, so we are through. □

In [15] it is proved that $\mathcal{I}$ (under the name of IND) is computable in the Suslin functional $\mathbf{S}$ and any strong realiser for the Heine–Borel theorem. We can improve this as

Lemma 6.9. *Let M be a Pincherle realiser. Then $\mathcal{I}$ is computable in M and* $\mathbf{S}$.

Proof. Let $F : \mathsf{C} \to \mathsf{C}$ be given, and consider F as an inductive definition, defining the sequence $\{f_\beta\}_{\beta<\alpha}$. This set is coded as a *prewellordering* $(A, \preceq)$ where $x \preceq y$ if $f_\alpha(x) = f_\alpha(y) = 1$ and we for all $\beta \leqslant \alpha$ have that $f_\beta(y) = 1 \to f_\beta(x) = 1$. Identify $\preceq$ with $\{\langle x, y\rangle : x \preceq y\}$. We will see how to compute $\preceq$ from F, $\mathbf{S}$ and M. We let x, y, z, w, n, m etc. range over $\mathbb{N}$.

The idea is, for each n, x, y to construct a functional $G_{n,x,y}$ such that if $x \preceq y$ then $M(G_{n,x,y}) \geqslant n$ and such that $G_{n,x,y}$ is independent of n otherwise. We will then have that

$$x \preceq y \leftrightarrow \forall m \exists n (M(G_{n,x,y}) > m).$$

We will now define $G_{n,x,y}(X)$ in cases, where we in all cases except in the last one have defined $G_{n,x,y}(X)$ independently of n, x and y, and so large that $\preceq$ will be different from X for at least one argument $z < G_{n,x,y}(X)$. We rename X to $\preceq_X = \{\langle z, w\rangle : \langle z, w\rangle \in X\}$. Let $\langle z, w\rangle \in X_\prec$ if $\langle z, w\rangle \in \preceq_X$ and $\langle w, z\rangle \notin \preceq_X$.

For all cases below, we assume that none of the earlier cases apply.

(1) If $\preceq_X$ is not a preordering, there is a finite initial binary subsequence s of (the characteristic function of) X such that no extension of s is a preordering. In this case, let $G_{n,x,y}(X)$ be the length of the least such s.

(2) Let W^X be the domain of the well founded part of $\prec_X$ (computable in the data using $\mathbf{S}$), and for each $z \in W^X$ let f^X_z be the characteristic function of $\{w \in W^X : w \prec_X z\}$ and g^X_z be the characteristic function of $\{w \in W^X : w \preceq_X z\}$.
If there is a $\prec_X$-least w such that $g^X_w \neq \max\{f^X_w, F(f^X_w)\}$, we know that $\preceq_X$ differs from $\preceq$, and we need to find a finite approximation to (the characteristic function of) X where this is manifested. Choose the numerically least such w. There will be two sub-cases:
- There is a z such that $f^X_w(z) = 0$, $F(f^X_w)(z) = 0$ but $g^X_w(z) = 1$. Select the numerically least such z. Then we cannot have both $z \preceq w$ and $w \preceq z$, while we have $z \preceq_X w$ and $w \preceq_X z$, so we let

$$G_{n,x,y}(X) = \max\bigl\{\langle z, w\rangle, \langle w, z\rangle\bigr\} + 1.$$

– There is no such z. Then there is a z such that $f^X_w(z) = 0$, $F(f^X_w)(z) = 1$, but $g^X_w(z) = 0$. Then we do have $z \preceq w$ and not $z \preceq_X w$, so we let $G_{n,x,y}(X) = \langle z, w\rangle + 1$ for the numerically least such z.

(3) If we get to this point, the well-founded part of $\preceq_X$ is an initial segment of $\preceq$, and we want to decide if this initial segment is proper or not. This is tested by letting g be the characteristic function of W^X: the initial segment is proper unless $F(g) \leqslant g$. If the initial segment is proper, we can find z such that $z \preceq z$ but $z \notin W^X$, recognised by $g(z) < F(g)(z)$. If $\langle z, z\rangle \notin X$, we let $G_{n,x,y}(X) = \langle z, z, \rangle + 1$. If $\langle z, z\rangle \in X$, z is still not in the well founded part of $\prec_X$ so there will be a w such that $w \prec z$ and w is not in the well founded part of $\prec_X$. Since z is of minimal rank in $\prec$ outside X^W, we cannot have that $w \prec z$ when $w \notin X^W$. We can find such z and w using ${}^2E$ and search over $\mathbb{N}$, and we let $G_{n,x,y}(X) = \max\{\langle z, w\rangle, \langle w, z\rangle\} + 1$.

(4) So far, we have defined $G_{n,x,y}(X)$ independently of n, x and y. If we have reached this far, we know that $\mathcal{I}(F) = W^X$, and we let

- $G_{n,x,y}(X) = n$ if $x \in W^X$, $y \in W^X$ and $x \preceq_X y$
- $G_{n,x,y}(X) = 0$ otherwise.

Through items (1)–(3) above, we have constructed $G_{n,x,y}$ such that unless X is a preordering with $\preceq$ as its well founded part, $G_{n,x,y}(X)$ is such that $\preceq$ is not in the neighbourhood of X induced by $G_{n,x,y}(X)$. Moreover $G_{n,x,y}(X)$ is independent of n (and of x and y) in this case. We further have that $G_{n,x,y}$ is independent of n if we do not have $x \preceq y$, while the function

$$F_{n,x,y}(X) = \begin{cases} n & \text{if } X = \preceq \\ 0 & \text{otherwise} \end{cases}$$

will satisfy the bounding condition induced by $G_{e,x,y}$ if $x \preceq y$. In this case we must have that $M(F_{e,x,y}) \geqslant n$. This is what we aimed to obtain. □

Corollary 6.10. *Let M be a Pincherle realiser that is computable in $\mathcal{I}$. Then $\mathcal{I}$ and $(\mathcal{H}(M), {}^2E)$ are computationally equivalent.*

7. Relativisations

It is a matter of routine to relativise concepts of computability to functions $f : \mathbb{N} \to \mathbb{N}$ or to subsets of $\mathbb{N}$. Our characterisation of the functionals of type 3 computable in $\mathcal{I}$ using inductive procedures do relativise directly to objects of type 1. We even had to do so in order to cope with inductive procedures themselves, since there will be function parameters in subcomputations of the form $\{e\}(\mathcal{I}, \vec{a})$.

In this section we will briefly discuss how the concept of an inductive procedure relativises to parameters $\vec{F}$ of type 2, without going into any technical details. The key observation is that we can easily extend the definition of procedures to cope with multiple inputs $\vec{F}$, or, if we are interested in functionals of type 3 computable in a fixed functional G of type 2, to input pairs F, G. We only have to add the coordinate of each query when asked during a calculation.

Given a partial functional $\lambda(F, H)\{e\}(\mathcal{I}, F, H)$ there will be a procedure Ω as before, where Ω and Ω_{pre} are Π^1_1, there are functions NEXT, BLOCK, DENOTE and REDENOTE computable in ${}^2E$. The calculations/strings corresponding to terminating computations will have a blocking accepting the axioms we gave, respecting the rules of bracketing and with no infinitely deep chains of blocks. Our main problem in describing what we mean with an inductive G-procedure for a total functional $\lambda F.\{e\}(F, G)$ is to find the right relativisation of Π^1_1 to G. Another, minor problem is that we must allow information about G in the LOG of a calculation, or in some other way, see Remark 7.4.

Definition 7.1. Let G be a functional of type 2.

(a) For $g \in \mathbb{N}^{\mathbb{N}}$, let $\{g^p_i\}_{i \in \mathbb{N}}$ be an enumeration of the set of functions primitive recursive in g, where the enumeration is uniformly computable in g.
A *weakly arithmetical formula* $\Phi(g, G)$ is a formula that is arithmetical in g and $\lambda i.G(g^p_i)$.

(b) $X \subseteq \mathbb{N}^{\mathbb{N}}$ is *weakly arithmetical* in G if it is defined by

$$g \in X \leftrightarrow \Phi(g, G)$$

for some weakly arithmetical formula Φ.

(c) $X \subseteq \mathbb{N}^{\mathbb{N}}$ is $\Pi^1_1[G]$ if X is the intersection of a Π^1_1-set and a set that is weakly arithmetical in G.

Definition 7.2. Let Ω be a procedure for a total functional $\Phi(F)$. Let G be of type 2. Ω is an *inductive G-procedure* if the following are satisfied:

(1) Ω and Ω_{pre} are $\Pi^1_1[G]$.
(2) There is a function NEXT_Ω partially computable in ${}^2E$ and terminating on Ω_{pre}.
(3) The calculations in Ω have blockings, and the blocking structure is guided by the partial functions BLOC, DENOTE and REDENOTE, computable in ${}^2E$ and with the standard properties.

Theorem 7.3. *Let G be of type 2 and Φ of type 3. Then Φ is computable in $\mathcal{I}$ and G if and only if Φ is definable by an inductive G-procedure.*

Proof. One way is proved exactly in the same way that we proved that if Φ is definable by an inductive procedure, then Φ is computable in $\mathcal{I}$.

For the other direction, let $\Phi(F) = \{e\}(\mathcal{I}, F, G)$. We consider the procedure Ω^+ for the partial functional $\lambda F, H.\{e\}(\mathcal{I}, F, H)$, where both Ω^+ and Ω^+_{pre} are Π^1_1 and where the functions NEXT, BLOCK, DENOTE and REDENOTE are computable in ${}^2E$. Intersecting with a set that is weakly arithmetical in G we get the procedures in Ω^+ and the strings in Ω^+_{pre} that are matching G. Those will be our Ω and Ω_{pre}. □

Remark 7.4. Pairs $(g, G(g))$ will still be present in the calculations. They may be considered to be elements of the LOG of the G-calculations. This would actually require a re-definition of our concepts of *procedure* and *calculation*, but we leave how to do it for the reader in case of interest.

It may be possible to avoid appearances of pairs $(g, G(g))$ in the procedure for $\lambda F.\{e\}(F, G)$, but then at the cost of the complexity of the NEXT-function and the other functions guiding the blockings. These will then have to be computed by $\mathcal{I}$ and G, and not just by ${}^2E$.

8. Summary and open problems

In this paper[2] we have investigated non-monotone induction as given by a functional $\mathcal{I}$ of type 3 from the perspective of higher order computability theory. We have established strong closure properties for the *companion* L_π of the set of functions computable in $\mathcal{I}$, and we have represented computations relative to $\mathcal{I}$ and parameters of type 2 in the form of inductive procedures and sequential calculations. Computations relative to $\mathcal{I}$ can be linearised in a natural way, since application of $\mathcal{I}$ can be seen as the result of a linear process indexed by some ordinal, and the ordinal rank of a calculation reflects the length of a computation seen as a linear process. We have shown that the length of any terminating computation, with integer inputs in addition to $\mathcal{I}$, is Π^1_1-characterisable. There are two open problems related to this:

Problem 1. *Are all ordinals $\alpha < \pi$ Π^1_1-characterisable?*

Problem 2. *Are there ordinals computable in $\mathcal{I}$ that are not the length of any computation $\{e\}(\mathcal{I}, \vec{a})$?*

A positive solution to Problem 1 would give us a nice characterisation of the closure ordinal π, but we conjecture that the answer is negative. We also believe that when the two problems are solved, the solution will show that they are connected.

Problem 2 asks if there is a gap-structure for computing with $\mathcal{I}$ as it is for infinite time Turing machines, see Hamkins and Lewis [7] or Welch [33], and for recursion in ${}^3E$, and not as for computing relative to the Superjump $\mathbb{S}$. In case there are gaps, it will be of interest to see how the gap structure coincides with the gap structure of infinite time Turing machines computing in time bounded by π.

This problem also suggests that there is a distinction between various functionals of type 3 similar to the one between predicative and non-predicative arguments in mathematics: in order to compute $\mathbb{S}(F)$ we need to generate the 1-section of F, the set of functions computable in F, and we need that F is total on its own 1-section, and then we have enough information to deduce what $\mathbb{S}(F)$ will be. This will also work when F is partial, as long as it is

[2]The paper was written in 2020, and has, since then, only undergone cosmetic changes and updates of the bibliography.

not so partial that it is undefined for an input it is able to compute. In order to compute $\mathcal{I}(F)$ for a partial F we need that F is total on functions computable from F and $\mathcal{I}$, including the final product of the 'computation'. In our main theorem for establishing closure properties of π, it is essential for the argument that we construct an induction where π tells us to stop, and that we thus must stop before π. We can only consider partial inductive definitions computable in $\mathcal{I}$ when they also make sense in the case when the recursion lasts π steps in order to deduce that they must stop at an earlier stage. This is a kind of non-predicativity.

We will end this paper with an example of a partial functional $F : \mathsf{C} \to \mathsf{C}$ that is computable in $\mathcal{I}$ and total on the set of $f \in \mathsf{C}$ that are computable in $\mathcal{I}$, but where the closure set X of the associated inductive definition is not computable in $\mathcal{I}$ because $F(X)$ is undefined.

Example 4. We define F as a partial function from $\mathcal{P}(\mathbb{N} \times \mathbb{N})$ to $\mathcal{P}(\mathbb{N} \times \mathbb{N})$:

- If X is a well ordering, use Gandy selection for $\mathcal{I}$ to find an index e for a well ordering $Y \subseteq \mathbb{N} \times \mathbb{N}$ with domain B and of length extending that of X, and let

$$F(X) = X \cup \big\{\big(a, \langle e, b\rangle\big) : a \in X \wedge b \in B\big\} \cup \big\{\big(\langle e, b\rangle, \langle e, c\rangle\big) : (b, c) \in Y\big\}.$$

- If X is not a well ordering, we let $F(X) = X$.

During the induction, a new index e must be found each time, so $F(X_\beta)$ will be an end-extension of X_β with a well ordering of the order-type of some Y_β with index e_β for all $\beta < \pi$. The recursion will stop after π steps because then $F(X_\pi)$ is undefined. Clearly, X_π is not computable in $\mathcal{I}$.

Acknowledgements

I thank Sam Sanders for involving me in the project this paper is a spin-off from, for reading a preliminary version of this paper, and for giving valuable feedback on the exposition. Our joint project started with him asking me if I could say anything about the computational properties of some weird-looking functionals of type 3, of which Θ_0 is one example. The rest is history.

I am grateful to John Hartley for his comments on the exposition.

I am grateful to editors and anonymous referees of other papers from our joint project, their sharp comments often helped me think more clearly about how to present higher order computability in the context of those papers, and then of this one.

I also thank the participants of the seminar on mathematical logic at the University of Oslo for attending my informal talks on the subjects of this paper, and giving valuable feedback.

References

[1] S.O. Aanderaa, Inductive definitions and their closure ordinals, in: *Generalized Recursion Theory*, J.E. Fenstad and P.G. Hinman, eds, North-Holland, 1974, pp. 207–220.

[2] P. Aczel and W. Richter, Inductive definitions and analogues of large cardinals, in: *Conference in Mathematical Logic, London'70*, Springer Verlag, 1971, pp. 1–10.

[3] E. Borel, Sur quelques points de la théorie des fonctions, *Ann. Sci. École Norm. Sup.* **12**(3) (1895), 9–55. doi:10.24033/asens.406.

[4] D. Cenzer, Ordinal recursion and inductive definitions, in: *Generalized Recursion Theory*, J.E. Fenstad and P.G. Hinman, eds, North-Holland, 1974, pp. 221–264.

[5] J.E. Fenstad, *Generalized Recursion Theory*, Springer Verlag, 1980.

[6] R.O. Gandy, General recursive functionals of finite type and hierarchies of functions, *Ann. Fac. Sci. Univ. Clermont-Ferrand No.* **35** (1967), 5–24.

[7] J.D. Hamkins and A. Lewis, Infinite time Turing machines, *The Journal of Symbolic Logic* **65**(2) (2000), 567–604. doi:10.2307/2586556.

[8] L. Harrington, The superjump and the first recursively Mahlo ordinal, in: *Generalized Recursion Theory*, J.E. Fenstad and P.G. Hinman, eds, North-Holland, 1974, pp. 43–52.

[9] J.P. Hartley, The countably based functionals, *Journal of Symbolic Logic* **48** (1983), 458–474. doi:10.2307/2273562.

[10] S.C. Kleene, Recursive functionals and quantifiers of finite types I, *Trans. Amer. Math. Soc.* **91** (1959), 1–52.

[11] U. Kohlenbach, *Higher Order Reverse Mathematics 2001*, Lecture Notes in Logic, Vol. 211, Association of Symbolic Logic, 2005, pp. 281–295.

[12] J.R. Longley and D. Normann, *Higher Order Computability*, Spinger Verlag, 2015.

[13] J. Moldestad, *Computations in Higher Types*, Lecture Notes in Mathematics, Vol. 574, Springer Verlag, 1977.

[14] Y.N. Moschovakis, Hyperanalytic predicates, *Transactions of the American Mathematical Society* **129**(2) (1967), 249–282. doi:10.1090/S0002-9947-1967-0236010-2.

[15] D. Normann, Functionals of type 3 as realisers of classical theorems in analysis, in: *Sailing Routes in the World of Computation*, F. Manea, R.G. Miller and D. Nowotka, eds, Lecture Notes in Computer Science, Vol. 10936, Springer Verlag, 2018, pp. 318–327. doi:10.1007/978-3-319-94418-0_32.

[16] D. Normann and S. Sanders, Computability theory, nonstandard analysis and their connections, *Journal of Symbolic Logic* **84**(4) (2019), 1422–1465. doi:10.1017/jsl.2019.69.

[17] D. Normann and S. Sanders, The strength of compactness in computability theory an nonstandard analysis, *Annals of Pure and Applied Logic* **170**(11) (2019), 1–42. doi:10.1016/j.apal.2019.05.007.

[18] D. Normann and S. Sanders, On the mathematical and foundational significance of the uncountable, *Jour. Math. Log.* **19**(1) (2019). doi:10.1142/S0219061319500016.

[19] D. Normann and S. Sanders, Pincherle's theorem in reverse mathematics and computability theory, *Annals of Pure and Applied Logic* **171**(5) (2020). doi:10.1016/j.apal.2020.102788.

[20] D. Normann and S. Sanders, Open sets in reverse mathematics and computability theory, *Journal of Logic and Computation* **30**(8) (2020), 1639–1676. doi:10.1093/logcom/exaa049.

[21] D. Normann and S. Sanders, The axiom of choice in computability theory and reverse mathematics, with a cameo for the continuum hypothesis, *Journal of Logic and Computation* **31**(1) (2021), 297–325. doi:10.1093/logcom/exaa080.

[22] D. Normann and S. Sanders, On the uncountability of $\mathbb{R}$, *Journal of Symbolic Logic* **87**(4) (2022), 1–45. doi:10.1017/jsl.2022.27.

[23] D. Normann and S. Sanders, Betwixt Turing and Kleene, in: *Logical Foundations of Computer Science*, Lecture Notes in Computer Science, S. Artemov and A. Nerode, eds, Vol. 13137, 2022, pp. 236–252. doi:10.1007/978-3-030-93100-1_15.

[24] D. Normann and S. Sanders, On robust theorems due to Boltzano, Weierstrass, Jordan and Cantor, *Journal of Symbolic Logic* (2022). doi:10.10177jsl2022.17.

[25] D. Normann and S. Sanders, On the computational properties of basic mathematical notions, *Journal of Logic and computation* **32**(8) (2022), 1747–1795. doi:10.1093/logcom/exaa075.

[26] D. Normann and S. Sanders, The biggest five of reverse mathematics, *Journal of Mathematical Logic* (2022). doi:10.1142/S0211190613245000077.

[27] D. Normann and S. Sanders, Representations in measure theory: Between a non-computable rock and hard to prove place, submitted arXiv:1902.02756.

[28] G.D. Plotkin, LCF considered as a programming language, *Theoretical Computer Science* **5**(3) (1977), 223–255. doi:10.1016/0304-3975(77)90044-5.

[29] W. Richter and P. Aczel, Inductive definitions and reflecting properties of admissible ordinals, in: *Generalized Recursion Theory*, J.E. Fenstad and P.G. Hinman, eds, North-Holland, 1974, pp. 301–381.

[30] G.E. Sacks, *Higher Recursion Theory*, Perspectives in Mathematical Logic, Springer-Verlag, Berlin-Heidelberg, 1990.

[31] D.S. Scott, A type-theoretical alternative to ISWIM, CUCH, OWHY, unpublished note (1969). Edited version in: A collection of contributions in honor of Corrado Böhm on the occasion of his 70th birthday, *Theoretical Computer Science* **121**(1–2) (1993), 411–440. doi:10.1016/0304-3975(93)90095-B.

[32] P.D. Welch, Transfinite machine models, in: *Turing's Legacy, Developments from Turing's Ideas in Logic*, R. Downey, ed., Lecture Notes in Logic, Vol. 42, Cambridge University Press, 2014, pp. 493–529. doi:10.1017/CBO9781107338579.015.

[33] P.D. Welch, Higher type recursion for transfinite machine theory, in: *Computing with Foresight and Industry*, F. Manea, B. Martin, D. Paulusma and G. Primeiro, eds, Springer Lecture Notes in Computer Science, Vol. 11558, 2019, pp. 72–83. doi:10.1007/978-3-030-22996-2_7.