Gödel's functional interpretation and the concept of learning

Thomas Powell
University of Innsbruck
thomas.powell@uibk.ac.at

Abstract

In this article we study Gödel's functional interpretation from the perspective of learning. We define the notion of a learning algorithm, and show that intuitive realizers of the functional interpretation of both induction and various comprehension schemas can be given in terms of these algorithms. In the case of arithmetical comprehension, we clarify how our learning realizers compare to those obtained traditionally using bar recursion, demonstrating that bar recursive interpretations of comprehension correspond to 'forgetful' learning algorithms. The main purpose of this work is to gain a deeper insight into the *semantics* of programs extracted using the functional interpretation. However, in doing so we also aim to better understand how it relates to other interpretations of classical logic for which the notion of learning is inbuilt, such as Hilbert's epsilon calculus or the more recent learning-based realizability interpretations of Aschieri and Berardi.

Keywords functional interpretation, program extraction, learning, bar recursion.

1. Introduction

Gödel's functional (or Dialectica) interpretation has a rich and varied history. Originally used to establish relative consistency proofs for classical arithmetic and analysis [9, 17], it has not only become a powerful tool in proof theory [4], but has inspired research in areas as disparate as category theory [7] and classical game theory [8], and perhaps most importantly it lies at the heart of the proof mining program [10] in which it has been used to extract quantitative information from proofs in many areas of mathematics, ranging from numerical analysis to combinatorics to ergodic theory.

While the functional interpretation plays a central role in mathematical logic and its applications in computer science and mathematics, its action on classical proofs can still be extremely difficult to understand: Because extracted programs are unwound recursively over the logical structure of proofs, for anything other than the simplest proof these programs are typically giant lambda terms whose underlying *operational* meaning as a program is obscured under a complex layer of syntax.

The last few decades have seen an increased interest in understanding the semantics of classical reasoning, and one extremely fruitful approach to this has been to utilise a semantics based on the notion of *learning*. This concept dates all the way back to Hilbert and his epsilon calculus, but more modern instances include the game theoretic interpretations of Coquand [6], Avigad's update procedures [3] and notably the large body of work on learning-based realizability interpretations due to Aschieri and Berardi (see [1, 2]). At the core of all these works is the following basic idea:

(a) Non-computable objects should be interpreted by finitary approximations to these objects.

(b) These finitary approximations can be computed using some kind of learning algorithm.

The purpose of the article is to demonstrate that this idea is capable of providing us with a great deal of insight into Gödel's functional interpretation, and in particular the semantics of programs it extracts. We do this by showing that the functional interpretations of key non-constructive principles are realized by intuitive learning algorithms that compute finitary approximations of these principles. These learning algorithms can then be used instead of the usual realizers written in terms of Gödel's primitive recursor or Spector's bar recursor, with the intended result that the behaviour of extracted programs will be much more visible from their syntax, and will come equipped with a natural semantics based on learning.

We begin with some ground work, providing a brief outline of the functional interpretation and then defining what we mean by a learning algorithm. We then move on to treat the functional interpretation of classical arithmetic by focusing on a general, transfinite least element principle, whose functional interpretation we realize in a concise and natural way as the limit of a learning algorithm. The main novelty of the paper, however, is our study of comprehension principles. We move on to define an abstract operation on learning algorithms which extends *pointwise* algorithms to *sequential* ones, and demonstrate that this extension operation realizes the functional interpretation of variants of arithmetical comprehension. We then link this to the traditional bar recursive solution, and reveal the perhaps surprising fact that bar recursive realizers of arithmetical comprehension implicitly build quite intuitive approximations to comprehension functionals that are limits of 'forgetful' learning algorithms.

Throughout the paper we draw on a wide range of sources. A connection between the functional interpretation and learning was already implicitly observed in the pioneering work of Spector and Kreisel of the 1960s (see e.g. [17]), and as such we hope that this work forms a thread which makes this connection more explicit and links it to modern approaches to program extraction. More importantly, in studying the operational semantics of extracted programs, we take another step on the way to gaining a better understanding of the computational meaning of non-constructive theorems in mathematics. In particular we believe that the recent work on program extraction in subsystems of analysis, for example the functional interpretation of Ramsey's theorem [11, 15], will benefit directly from our study of comprehension principles in the second half of the paper.

2. Gödel's functional interpretation

We begin by outlining some background theory. We assume that the reader has a basic knowledge of Gödel's functional interpretation and systems of primitive recursive functionals in finite type. However, we give a brief overview of these concepts here, with the hope that a more general audience can follow the main ideas.

The finite types are defined inductively as follows: $\mathbb{B}$ and $\mathbb{N}$ are base types, and if X and Y are finite types then so are the product type $X \times Y$, function type $X \to Y$ and finite sequence type X^*. We collect together here some important notational conventions.

Notation 2.1. *We write $x : X$ or x^X when x has type X. $\mathbf{0}_X$ is the zero object of type X. We denote the length of a list by $|s|$, and the concatenation of finite sequences $s, t : X^*$ by $s * t$. If $x : X$ we write $s*x$ for $s*\langle x\rangle$. $last(s)$ denotes the last element of a list, or just $\mathbf{0}$ if s is the empty list $\langle\rangle$. For a sequence $\alpha : X^{\mathbb{N}}$, $[\alpha](n)$ denotes its initial segment of length n. Conversely, for $s : X^*$, $\hat{s} : X^{\mathbb{N}}$ denotes the extension of s with some canonical objects $\mathbf{0} : X$. For $f : X^{\mathbb{N}} \to Y$, we sometimes write $\hat{f} : X^* \to Y$ for $\hat{f}(s) := f(\hat{s})$. Given a function $f : X \to Y \times Z$, we write $f = (f_0, f_1)$ where $f_0 : X \to Y$ and $f_1 : X \to Z$ are its projections.*

By $\mathsf{E\text{-}HA}^\omega$ we mean the theory of extensional Heyting arithmetic in all finite types (here with explicit sequence types), while $\mathsf{WE\text{-}HA}^\omega$ denotes its weakly-extensional variant, and $\mathsf{E\text{-}PA}^\omega$, $\mathsf{WE\text{-}PA}^\omega$ the classical versions of these theories. For full details of these systems see e.g. [10]. In addition to the usual axioms and rules of arithmetic, these theories contain constants which allow the definition of primitive recursive functionals of arbitrary type. Quantifier-free formulas P of arithmetic are decidable, which means they can be represented by characteristic functions t_P. We use a slight abuse of notation and conflate P and t_P, treating quantifier-free predicates themselves as boolean valued functions.

For a detailed introduction to Gödel's functional interpretation the reader is directed to [4, 10], as we give only the main definition here. The basic functional interpretation of intuitionistic logic, which we denote the D-interpretation, is a translation which maps each formula A in the language of $\mathsf{WE\text{-}HA}^\omega$ to a quantifier-free formula $|A|^x_y$ where x and y are (possibly empty) tuples of variables (for simplicity encoded as a single variable here). It is defined over the logical structure of A as follows:

$$\begin{aligned} |A| &:\equiv A \text{ for } A \text{ atomic} \\ |A \wedge B|^{x,u}_{y,v} &:\equiv |A|^x_y \wedge |B|^u_v \\ |A \vee B|^{b,x,u}_{y,v} &:\equiv |A|^x_y \vee_b |B|^u_v \\ |A \to B|^{U,Y}_{x,v} &:\equiv |A|^x_{Yxv} \to |B|^{Ux}_v \\ |\exists z A(z)|^{x,u}_v &:\equiv |A(x)|^u_v \\ |\forall x A(x)|^U_{x,v} &:\equiv |A(x)|^{Ux}_v, \end{aligned} \tag{1}$$

where

$$P \vee_b Q :\equiv (b = \top \to P) \wedge (b = \bot \to Q).$$

The basic idea is that $A \leftrightarrow \exists x \forall y |A|^x_y$ over classical logic, but whenever A is provable we can extract an explicit witnessing term t satisfying $\forall y |A|^t_y$ from its proof. In order to interpret classical logic, one first uses a negative translation N to embed the classical theory into an intuitionistic one, and then applies the D-interpretation. We label this combined translation the ND-interpretation. The following result is standard.

Theorem 2.2 (Functional interpretation of Peano arithmetic)**.** *Let Δ be a set of purely universal sentences, $\mathsf{QF\text{-}AC}$ denote the axiom of choice for quantifier-free formulas and $A(x)$ be a formula in the language of $\mathsf{WE\text{-}PA}^\omega$ with only x free. Then*

$$\mathsf{WE\text{-}PA}^\omega + \mathsf{QF\text{-}AC} + \Delta \vdash A(x) \Rightarrow \mathsf{E\text{-}HA}^\omega + \Delta \vdash \forall y |A(x)^{\mathrm{N}}|^{t(x)}_y$$

where $A(x)^{\mathrm{N}}$ denotes the negative translation of $A(x)$, and t is a closed term of $\mathsf{WE\text{-}HA}^\omega$ which can be formally extracted from the proof of $A(x)$.

Theorem 2.2 can, and has been, extended to a wide range of richer theories by exploiting the modular nature of the functional interpretation. If we want to extend the interpretation to incorporate some additional axiom Γ, then it is sufficient to produce a functional F which witnesses $\exists x \forall y |\Gamma^{\mathrm{N}}|^x_y$. We can also replace an existing realizer of a principle with a new one, so that programs extracted from proofs which use this principle as a lemma are constructed using the new realizer instead.

Because this article is more concerned with algorithms which realize the ND-interpretation of classical principles rather than the functional interpretation itself, we do not go into any more details here. Although in later sections we will need to provide the ND-interpretation of certain principles, we will not give the detailed steps used to compute it. For example, we do not apply any particular version of the negative translation formally, rather in each case we just state a negative translated version of the principle in question which can be directly realized by the D-interpretation. However, we do endeavour to highlight the *intuitive* meaning of the interpretation throughout. As such, it will be useful at this stage to outline how the ND-interpretation deals with certain simple formulas.

First, Π_2-formulas $A :\equiv \forall x \exists y P(x, y)$ with P quantifier-free, are typically negative translated as $\forall x \neg\neg\exists y P(x, y)$. However, because the D-interpretation admits Markov's principle, in practice we can omit this double negation and interpret A directly as $\exists f \forall x P(x, f(x))$. This allows realizers of Π_2-formulas to be extracted even from classical proofs. On the other hand, for Π_3-formulas $B :\equiv \forall x \exists y \forall z Q(x, y, z)$ it is in general impossible to find a computable function f satisfying $\forall x, z Q(x, f(x), z)$. Formulas of this form can be negative translated into intuitionistic logic as $\forall x \neg\neg\exists y \forall z Q(x, y, z)$, whose D-interpretation before the final Skolemisation is

$$(*) \;\; \forall x, \xi \exists y Q(x, y, \xi(y))$$

and whose full interpretation is thus $\exists F \forall x, \xi Q(x, Fx\xi, \xi(Fx\xi))$. One can view this semantically as follows: while in general we cannot compute a y satisfying $\forall z Q(x, y, z)$, the ND-interpretation asks for an *approximation* to the non-computable object y which satisfies $Q(x, y, \xi(y))$, where ξ is some arbitrary functional, which in this sense calibrates how 'strong' we want the approximation to be. In what follows, we often express the ND-interpretation in the form $(*)$ before the final step in order the minimise syntax, in which case we call it the *partial* functional interpretation.

3. Learning algorithms

Having given a fairly normal overview of the first main subject of the article, we now move on to a much more non-standard treatment of the second key concept – learning. We define below precisely what we mean by a learning algorithm, and while our definition is related to those found in e.g. [2, 3], here it is more general and tailored to our specific situation.

Definition 3.1 (Learning algorithm). A *learning algorithm* $\mathcal{L}$ of types X, Y, Z is a tuple $(Good, \delta, \xi, \oplus)$, where

- *Good* is a decidable predicate on X, treated as a boolean-valued functional $X \to \mathbb{B}$;
- $\delta : X \to Z$ and $\xi : X \to Y$ are functionals;
- $\oplus : X \times Y \to X$ is a functional which we view as an operation and write in infix notation as $x \oplus y$.

Definition 3.2 (Learning procedures and limits). For any point $x : X$ the learning algorithm $\mathcal{L}$ triggers a *learning procedure* $\mathcal{L}[x]$, which is the sequence (x_i) recursively defined by

$$x_0 := x \quad \text{and} \quad x_{i+1} := \begin{cases} x_i & \text{if } Good(x_i) \\ x_i \oplus \xi(x_i) & \text{if } \neg Good(x_i). \end{cases}$$

We say that a learning procedure *terminates* if there exists some x_k satisfying $Good(x_k)$, in which case we say that x_k is a *limit point* and $\delta(x_k)$ the *limit* of the learning procedure, writing

$$\lim \mathcal{L}[x] := \delta(x_k).$$

Note that this is well-defined when it exists since any x_k satisfying $Good(x_k)$ must be equal to the first such point in the learning procedure.

The components of learning algorithms are intended to have the following intuitive meaning:

- *Good* is a predicate which decides whether or not an element of X is 'good' in some specific context which will vary throughout the paper;
- The operation
$$x_i \mapsto x_i \oplus \xi(x_i)$$
is an instance of *learning*, namely it takes x_i and updates it with some additional piece of information $\xi(x_i)$;
- δ is simply a function which, assuming it is eventually provided with a good $x_k : X$ returns a final value $\delta(x_k) : Z$.

A learning procedure starting at x_0 is an attempt at finding a good element of X, guided by ξ. At each stage in the procedure either x_i is good, in which case we have reached a limit point and return $\delta(x_i)$, or x_i is bad and we respond by updating it with some new piece of information $\xi(x_i)$ in the hope that $x_{i+1} = x_i \oplus \xi(x_i)$ is good.

Underneath the mathematical syntax, a learning algorithm $\mathcal{L}$ is essentially nothing more than a simple **while** loop (Algorithm 1), with $\mathcal{L}[x]$ just a trace of the variable y given some initial value x,

Algorithm 1 Computing $\lim \mathcal{L}[x]$

```
1: input $y = x$
2: while $\neg Good(y)$
3:     $y \to y \oplus \xi(y)$
4: return $\delta(y)$
```

and $\lim\ \mathcal{L}[x]$ the output of the program when it terminates. We will utilise this informal association of a learning procedure with a imperative program throughout the paper.

Example 3.3. Take an arbitrary pair of functions $f, g : \mathbb{N} \to \mathbb{N}$. Then there is some integer n such that

$$f(n) \leq f(g(n)). \tag{2}$$

This follows by classical logic by defining n to be a least element of the set $(\mathbb{N}, \prec)$ ordered by $x \prec y :\equiv f(x) < f(y)$. However, n can also be computed as the limit of a learning procedure. Let $\mathcal{L} := (Good, \iota, g, \oplus)$ for $Good(x) :\equiv f(x) \leq f(g(x)), x \oplus y = y$ and ι the identity function. Then for any x, $\mathcal{L}[x]$ terminates and $n := \lim \mathcal{L}[x]$ satisfies (2).

To see this, let $(x_i) = \mathcal{L}[x]$. Then as long as $f(x_i) > f(g(x_i))$ we have $x_{i+1} = x_i \oplus g(x_i) = g(x_i)$ and thus $\mathcal{L}[x]$ is just the sequence

$$x, g(x), g^{(2)}(x), \ldots, g^{(k)}(x), g^{(k)}(x), g^{(k)}(x), \ldots$$

where k is the first point satisfying $f(g^{(k)}(x)) \leq f(g^{(k+1)}(x))$. Thus $n = \lim \mathcal{L}[x] = g^{(k)}(x)$ works.

In this example, we computed the minimal element n by first making some arbitrary guess x. Either this was good with $f(x) \leq f(g(x))$, or we were able to learn from the failure of x and update $x \mapsto g(x)$ with $x \succ g(x)$. In this way, by well-foundedness of $\prec$ we eventually arrived at a good guess $x_k = g^{(k)}(x)$ for n. The simple example illustrates a fundamental idea already stated in the introduction, namely:

> Learning algorithms compute approximations to non-computable objects which arise from classical reasoning.

Our purpose is to show that the functional interpretation interpretation of key classical principles can be directly realized by typed learning procedures in the sense of Definition 3.1. In this way we gain some insight into the *algorithmic behaviour* of extracted programs, which can often become obscured beneath the complex forms of higher-type recursion traditionally associated with the functional interpretation. However, before we go on, we want to be able to ensure that learning procedures and their limits can be defined within a standard calculus of recursive functionals.

Definition 3.4. The learning algorithm $\mathcal{L} := (Good, \delta, \xi, \oplus)$ reduces with respect to some binary relation $\prec$ on X, if for any $x : X$, $\neg Good(x)$ implies $x \oplus \xi(x) \prec x$. In particular, if $\prec$ is well-founded then $\lim \mathcal{L}[x]$ exists for any x.

Lemma 3.5. *Let Δ be a (possibly empty) extension of $\mathsf{E\text{-}HA}^\omega$ such that $\mathsf{E\text{-}HA}^\omega + \Delta$ allows the definition of functionals by recursion over $\prec$. If $\mathcal{L}$ reduces with respect to $\prec$, then $\lim \mathcal{L}[x]$ exists for all x provably in $\mathsf{E\text{-}HA}^\omega + \Delta$, and the functional $\lambda x. \lim \mathcal{L}[x]$ can be defined as a term of $\mathsf{E\text{-}HA}^\omega + \Delta$.*

Proof. Define the term $l_{\mathcal{L}} : X \to X^*$ via recursion over $\prec$ as

$$l_{\mathcal{L}}(x) = \begin{cases} \langle x \rangle & \text{if } Good(x) \\ \langle x \rangle * l_{\mathcal{L}}(x \oplus \xi(x)) & \text{otherwise.} \end{cases}$$

Then $l_{\mathcal{L}}(x)$ is an initial segment of $\mathcal{L}[x]$ up to its limit point and $\lambda x. \lim \mathcal{L}[x]$ is defined by $\lambda x. \delta(\mathrm{last}(l_{\mathcal{L}}(x)))$. ☐

Before we go into technical details in the next section, it will be helpful to state a simple, abstract result which underpins much of what follows. Let $App(x)$ be some arbitrary predicate on X, the intended meaning being that x is an approximation of some nonconstructive object, and let $\exists z P(z)$ be some target formula that we want to realize. Suppose that we have

$$\forall x \begin{cases} App(x) \to Good(x) \to \exists z P(z) \\ App(x) \to \neg Good(x) \to \exists y App(x \oplus y), \end{cases} \tag{3}$$

which can be read as: if x is an approximation which is sufficiently good then we can infer our goal $\exists z P(z)$, and if it fails to be sufficiently good, then from this information we can learn an additional piece of information y such that $x \oplus y$ is a *better* approximation of x. Then $\exists z P(z)$ can be realized by a learning procedure.

Theorem 3.6. *Suppose that (3) is realized by $\delta : X \to Z$ and $\xi : X \to Y$ i.e.*

$$\forall x \begin{cases} App(x) \to Good(x) \to P(\delta(x)) \\ App(x) \to \neg Good(x) \to App(x \oplus \xi(x)) \end{cases}$$

and that the learning algorithm $\mathcal{L} := (Good, \delta, \xi, \oplus)$ reduces with respect to some well-founded ordering $\prec$. Then the formula

$$\forall x (App(x) \to \exists z P(z))$$

is realized by the functional

$$\lambda x. \lim \mathcal{L}[x]$$

Proof. Take some x satisfying $App(x)$ and assume that $\mathcal{L}[x]$ is given by $x_0, \ldots, x_k, x_k, \ldots$ where x_k is its limit point. Then

$App(x_0)$ holds by definition, and since $\forall i < k \neg Good(x_i)$ and thus $App(x_i) \to App(x_i \oplus \xi(x_i)) \to App(x_{i+1})$ we have $App(x_k)$ by induction. Thus from $Good(x_k)$ we obtain $P(\delta(x_k))$ i.e. $P(\lim \mathcal{L}[x])$. □

4. The functional interpretation of the minimum principle

The first main contribution of the paper is to demonstrate that learning procedures as defined in the previous section give a very natural realizer of the ND-interpretation of a generalisation of the well-known minimum principle on $\mathbb{N}$. Let $\prec$ be a well-founded, decidable relation on X, and $\oplus : X \times Y \to X$ a binary operation. Given some predicate A on X, the minimum principle $\mathsf{Min}_{A,\prec,\oplus}$ is defined by

$$\mathsf{Min}_{A,\prec,\oplus} \ : \ \exists \bar{x} A_{\bar{x}} \to \exists x (A_x \wedge \forall y (x \oplus y \prec x \to \neg A_{x \oplus y})).$$

Note that the usual minimum principle on $\mathbb{N}$ is just an instance of Min for $\prec = <$ and $x \oplus y = y$. However, our formulation above is not only more general but syntactically richer, and allows us to work over more interesting co-inductive data structures such as extensions of finite lists (as illustrated in Section 4.2), thereby leading to the possibility of being able to extract more natural and efficient programs from proofs which are based on such structures.

This section is inspired by the work of Schwichtenberg [16], who shows that the functional interpretation of well-founded induction can be realized by well-founded recursion. In fact, a realizer for well-founded induction also follows directly from our result, since induction just the contrapositive of the minimum principle and its functional interpretation is the same. Although completely equivalent to induction from this point of view, we prefer to work with the minimum principle since it fits more visually into our framework, in which our main realizer can be viewed directly as a program which builds an approximation to a minimal element through learning. Apart from this reformulation, the key differences between here and [16] are syntactic, namely the presence of the operator $\oplus$ and our construction of the realizer as the limit of a learning procedure rather than a via a well-founded recursor.

4.1 Interpreting $\mathsf{Min}_{A,\prec,\oplus}$ for quantifier-free A

We begin by interpreting the special case of the minimum principle for quantifier-free, and hence decidable A. Not only is this case interesting in its own right, but it will help motivate the general interpretation in Section 4.3. Even for quantifier-free A, it is impossible to give a general computable witness for the D-interpretation of Min, since the existence of a minimal element requires classical reasoning. Therefore in order to interpret the principle we must first apply a negative translation. As remarked earlier, we do not formally apply the negative translation. Rather, we simply observe that $\mathsf{Min}_{A,\prec,\oplus}$ is classically equivalent to the following double negated form:

$$\exists \bar{x} A_{\bar{x}} \to \neg\neg \exists x (A_x \wedge \forall y (x \oplus y \prec x \to \neg A_{x \oplus y})). \qquad (4)$$

which can be formally obtained from any standard double negation translation using intuitionistic logic plus at semi-intuitionistic laws like Markov's principle that admit a direct D-interpretation. Thus in contrast to $\mathsf{Min}_{A,\prec,\oplus}$ we are able to realize the D-interpretation of (4). The reader could now just directly apply the defining equations (1) to the negated formula (4) in order to obtain its functional interpretation, although we provide below a sequence of informal steps in order to make the interpretation slightly more intuitive.

Let us first focus on the conclusion of $\mathsf{Min}_{A,\prec,\oplus}$, which is partially (i.e. with the final Skolemisation step $\forall \xi \exists x \to \exists \Phi \forall \xi$ omitted) interpreted via the following intuitive steps

$$\begin{aligned} &\neg\neg \exists x (A_x \wedge \forall y (x \oplus y \prec x \to \neg A_{x \oplus y})) \\ \rightsquigarrow &\neg\neg \exists x \forall y (A_x \wedge (x \oplus y \prec x \to \neg A_{x \oplus y})) \\ \rightsquigarrow &\forall \xi^{X \to Y} \exists x (\underbrace{A_x \wedge (x \oplus \xi(x) \prec x \to \neg A_{x \oplus \xi(x)})}_{P^A_\xi(x)}). \end{aligned}$$

Thus $\mathsf{Min}^N_{A,\prec,\oplus}$ is partially interpreted via

$$\begin{aligned} &\exists \bar{x} A_{\bar{x}} \to \forall \xi \exists x P^A_\xi(x) \\ \rightsquigarrow &\forall \bar{x}, \xi \exists x (A_{\bar{x}} \to P^A_\xi(x)) \end{aligned}$$

for $P^A_\xi(x)$ defined as above. The full interpretation of (4) is therefore

$$\exists \Phi^{X \to (X \to Y) \to X} \forall \bar{x}, \xi (A_{\bar{x}} \to P^A_\xi(\Phi \bar{x} \xi)).$$

Whereas the minimum principle states that if $A_{\bar{x}}$ holds for some $\bar{x}$ then there is some minimal element x satisfying A_x, in line with our comments in Section 2 the ND-interpretation states that from any $\bar{x}$ satisfying $A_{\bar{x}}$, we can compute an approximately minimal element x relative to an arbitrary functional $\xi : X \to Y$. Now let us try to produce a realizer Φ which does this. First, define the predicates $Good_\xi(x)$ and $App_{\bar{x}}(x)$ by

- $Good^A_\xi(x) :\equiv Min(A, \xi, x)$, and
- $App^A_{\bar{x}}(x) :\equiv A_{\bar{x}} \to A_x$.

where

$$Min(A, \xi, x) :\equiv x \oplus \xi(x) \prec x \to \neg A_{x \oplus \xi(x)}$$

Then it is easy to show

$$\forall x \begin{cases} (A_{\bar{x}} \to A_x) \to Min(A, \xi, x) \to (A_{\bar{x}} \to P^A_\xi(x)) \\ (A_{\bar{x}} \to A_x) \to \neg Min(A, \xi, x) \to (A_{\bar{x}} \to A_{x \oplus \xi(x)}) \end{cases}$$

and hence by Theorem 3.6 we have

$$\forall x ((A_{\bar{x}} \to A_x) \to A_{\bar{x}} \to P^A_\xi(\lim \mathcal{L}^A_\xi[x]))$$

for $\mathcal{L}^A_\xi :\equiv (Min(A, \xi, -), \iota, \xi, \oplus)$ (for ι the identity function). Note that $\neg Min(A, \xi, x) \to x \oplus \xi(x) \prec x$ and thus $\mathcal{L}^A_\xi$ reduces with respect to $\prec$. Obviously $A_{\bar{x}} \to A_{\bar{x}}$ holds and thus we have

$$A_{\bar{x}} \to P_\xi(\lim \mathcal{L}^A_\xi[\bar{x}]).$$

Theorem 4.1 (Functional interpretation of $\mathsf{Min}_{A,\prec,\oplus}$ for quantifier-free A)**.** *The ND-interpretation of* $\mathsf{Min}_{A,\prec,\oplus}$ *given (in partial form) by*

$$\forall \bar{x}, \xi \exists x (A_{\bar{x}} \to P^A_\xi(x))$$

is realized by the functional

$$\lambda \bar{x}, \xi \ . \ \lim \mathcal{L}^A_\xi[\bar{x}]$$

for $\mathcal{L}^A_\xi :\equiv (Min(A, \xi, -), \iota, \xi, \oplus)$.

Remark 4.2. Note that by Lemma 3.5 this realizer is definable in any extension of $\mathsf{E\text{-}HA}^\omega$ which admits $\prec$-recursion, and so in particular if there is a measure $\mu : X \to \mathbb{N}$ such that $x \prec x' \leftrightarrow \mu(x) <_{\mathbb{N}} \mu(y)$ then it is definable in $\mathsf{E\text{-}HA}^\omega$, coinciding with the standard result that the ND-interpretation of the minimum principle for $<_{\mathbb{N}}$ can be realized by a primitive recursive functional.

The main advantage of Theorem 4.1 is that it not only provides a realizer of a general induction principle, but the *algorithmic meaning* of the realizer can be clearly understood from its syntax. The approximately minimal object x is constructed by a learning procedure, which starts at a point $\bar{x}$ for which we know $A_{\bar{x}}$ holds, and at each stage x_i test whether or not x_i is approximately minimal with respect to ξ. If it is then we are done and x_i is our limit, and if not then we have discovered some strictly smaller element $x_{i+1} := x_i \oplus \xi(x_i)$ satisfying $A_{x_i \oplus \xi(x_i)}$, and we can continue. We can take advantage of this intuitive reading in order to understand the behaviour of programs extracted from proofs that use Min as a lemma, as we will now illustrate.

4.2 A worked example: The Euclidean algorithm

In [16] Schwichtenberg uses well-founded recursion to extract a program from a classical proof of the statement that a common divisor of any pair of natural numbers can be expressed a linear combination of the two, and highlights the fact that this program is closely related to the Euclidean algorithm. We follow this example and use our learning-based realizer of Min to extract a program along the same lines, although the proof we use exploits our formulation of the minimum principle and is based on a slightly different well-founded ordering to that in [16].

Theorem 4.3. *Given any two natural numbers $a \geq b > 0$ there exist integers n, m such that $am + bn \mid a, b$.*

Proof. Define the measure $\mu : \mathbb{N}^2 \to \mathbb{N}$ by $\mu\vec{x} := \vec{x} \cdot (a, b) = x_1 a + x_2 b$, and the ordering $\prec$ on $(\mathbb{N}^2)^*$ by

$$\langle\vec{x}\rangle \prec \langle\rangle \text{ and } \mu\vec{x} < \mu\vec{y} \Rightarrow s * \vec{y} * \vec{x} \prec s * \vec{y}.$$

where $(\mathbb{N}^2)^*$ denotes the type of finite sequences of vectors and $s * \vec{x} = \langle s_0, \ldots, s_{k-1}, \vec{x}\rangle$ the concatenation of s with a single vector $\vec{x}$. It is clear that $\prec$ is well-founded, and moreover that $t \prec s$ iff $t = s * \vec{x}$ for some $\vec{x}$. Let $\text{rem}(n, m)$ denote the remainder of n when divided by m, and $e_0, e_1 = \langle 1, 0\rangle, \langle 0, 1\rangle$ be the usual unit vectors. Now, define the decidable predicate A on $(\mathbb{N}^2)^*$ by

$$A_s :\equiv |s| \geq 2 \wedge \begin{cases} i = 0, 1 \Rightarrow s_i = e_i \\ i \geq 2 \Rightarrow \mu s_i > 0 \wedge \mu s_i = \text{rem}(\mu s_{i-2}, \mu s_{i-1}) \end{cases}$$

where $|s|$ denotes the length of s (cf. Notation 2.1). Then $A_{\langle e_0, e_1\rangle}$ trivially holds, and thus by $\mathsf{Min}_{A,\prec,*}$ there is some minimal s satisfying A_s i.e.

$$\exists s(A_s \wedge \forall \vec{x}(s * \vec{x} \prec s \to \neg A_{s*\vec{x}})).$$

Denote $l = |s|$ and suppose that $r = \text{rem}(\mu s_{l-2}, \mu s_{l-1})$. Define $\vec{r} := s_{l-2} - \text{quot}(\mu s_{l-2}, \mu s_{l-1}) s_{l-1}$, where $\text{quot}(n, m)$ denotes the quotient of n and m. This is just a vector representation of r, so that $\mu\vec{r} = r$. By A_s this is all well defined and we must have $\mu\vec{r} < \mu s_{l-1}$ and hence $s * \vec{r} \prec s$.

But by minimality of s this means that $\neg A_{s*\vec{r}}$, which is only possible if $\mu\vec{r} = r = 0$ and hence $\mu s_{l-1} \mid \mu s_{l-2}$, and thus by induction along s we must have more generally $\forall i < s(\mu s_{l-1} \mid \mu s_i)$, and in particular for $i = 0, 1$ we have $\mu s_{l-1} \mid \mu e_0, \mu e_1$ i.e. $\mu s_{l-1} \mid a, b$. Therefore we can set $\langle m, n\rangle = s_{l-1}$. ☐

We now (quite informally), extract a program from this classical proof. There are two main components: namely an instance of Min and then the construction of m, n from a minimal s. The implication

$$\exists s A_s \wedge \forall \vec{x}(s * \vec{x} \prec s \to \neg A_{s*\vec{x}}) \to \exists n, m(am + bn \mid a, b)$$

requires realizing terms ξ and δ satisfying

$$\forall s((A_s \wedge (s * \xi(s) \prec s \to \neg A_{s*\xi(s)})) \to \delta(s) \cdot (a, b) \mid a, b)$$

which can be read off from the proof as

$$\xi(s) := s_{l-2} - \text{quot}(\mu s_{l-2}, \mu s_{l-1}) s_{l-1} \text{ and } \delta(s) := \text{last}(s)$$

for $l = |s|$. Now, observe that μ, A, ξ and δ are all implicitly parametrised by a, b, which we now add as subscripts. Since $(A_{a,b})_{\langle e_0, e_1\rangle}$ holds for arbitrary a, b, by Theorem 4.1 we have

$$\forall a, b(h(a, b) \cdot (a, b) \mid a, b)$$

for $h := \lambda a, b\ .\ \text{last}(\lim \mathcal{L}^A_{a,b}[\langle e_0, e_1\rangle])$ and

$$\mathcal{L}_{a,b} :\equiv (Min(A_{a,b}, \xi_{a,b}, -), \iota, \xi_{a,b}, *).$$

But expanding definitions we can show (although for the sake of space we do not give details here) that for any s satisfying $(A_{a,b})_s$ we have

$$Min(A_{a,b}, \xi_{a,b}, s) \Leftrightarrow \text{rem}(\mu_{a,b} s_{l-2}, \mu_{a,b} s_{l-1}) = 0.$$

Therefore, written out analogously to Algorithm 1, our realizer essentially performs the procedure given by Algorithm 2, which is

Algorithm 2 $\forall a, b \exists m, n(am + bn \mid a, b)$

```
1: input $a, b$
2: $s = \langle e_0, e_1\rangle$
3: while rem$(\mu_{a,b} s_{l-2}, \mu_{a,b} s_{l-1}) > 0$
4:     $\vec{r} \to s_{l-2} - \text{quot}(\mu_{a,b} s_{l-2}, \mu_{a,b} s_{l-1}) s_{l-1}$
5:     $s \to s * \vec{r}$
6: return last$(s)$
```

nothing more than the Euclidean algorithm in vector form. This example is not only of interest in that a program extracted using our general learning realizer produces an inherently intuitive and efficient program, but also from the reverse perspective that Euclid's algorithm can be derived formally from a general computational interpretation of the minimum principle. It would be interesting in the future to formalize the extraction using learning realizers properly, along the lines of [16].

4.3 Interpreting $\mathsf{Min}_{A,\prec,\oplus}$ for general A

We now define a learning-based realizer for the ND-interpretation of our minimum principle for arbitrary formulas A. This section follows the basic outline of Section 4.1, but is naturally somewhat more complex.

First, as before, we realize the D-interpretation of the following negative-translated version of $\mathsf{Min}_{A,\prec,\oplus}$:

$$\begin{aligned} &\exists \bar{x}, \bar{a} \forall c |A^N_{\bar{x}}|^{\bar{a}}_c \\ &\to \neg\neg\exists x, a(\forall c |A^N_x|^a_c \wedge \forall y, b(x \oplus y \prec x \to \neg\forall c |A^N_{x\oplus y}|^b_c)) \end{aligned}$$

where $\exists a \forall c |A^N_x|^a_c$ is the functional interpretation of the negative translation of A_x. Observe that setting $B_{x,a} :\equiv \forall c |A^N_x|^a_c$ and defining $(x, a) \oplus' (y, b) := (x \oplus y, b)$ and $(z, b) \prec' (x, a) := z \prec' x$ (note that $\prec'$ is well-founded whenever $\prec$ is), by encoding (x, a) as one variable x this can be reduced to the minimum principle for the universal formula $\forall c |B_x|_c$ i.e. it is sufficient to interpret

$$\exists \bar{x} \forall c |B_{\bar{x}}|_c \to \neg\neg\exists x(\forall c |B_x|_c \wedge \forall y(x \oplus y \prec x \to \neg\forall c |B_{x\oplus y}|_c)).$$

We can now follow the steps of Section 4.1 closely. We first consider the D-interpretation of the conclusion, which we obtain as follows:

$$\begin{aligned} &\neg\neg\exists x(\forall c |B_x|_c \wedge \forall y(x \oplus y \prec x \to \neg\forall c |B_{x\oplus y}|_c)) \\ \rightsquigarrow &\neg\neg\exists x(\forall c |B_x|_c \wedge \exists g \forall y(x \oplus y \prec x \to \neg |B_{x\oplus y}|_{gy})) \\ \rightsquigarrow &\neg\neg\exists x, g \forall c, y(|B_x|_c \wedge (x \oplus y \prec x \to \neg |B_{x\oplus y}|_{gy})) \\ \rightsquigarrow &\forall \xi \exists x, g(\underbrace{|B_x|_{\xi_0 xg} \wedge (x \oplus \xi_1 xg \prec x \to \neg |B_{x\oplus \xi_1 xg}|_{g(\xi_1 xg)})}_{P^B_\xi(x,g)}) \end{aligned}$$

where the functional ξ combines the output variables c, y and ξ_0, ξ_1 are its components. Therefore analogously to the decidable case, the minimum principle is partially interpreted as

$$\begin{aligned} &\exists \bar{x} \forall c |B_{\bar{x}}|_c \to \forall \xi \exists x, g P^B_\xi(x, g) \\ \rightsquigarrow &\forall \bar{x}, \xi \exists c, x, g(|B_{\bar{x}}|_c \to P^B_\xi(x, g)) \end{aligned}$$

and the full interpretation is

$$\exists \Phi \forall \bar{x}, \xi (|B_{\bar{x}}|_{\Phi_0 \bar{x} \xi} \to P^B_\xi(\Phi_1 \bar{x} \xi, \Phi_2 \bar{x} \xi)).$$

Whereas for the decidable case we were able to put this directly into the form where it could be solved using Theorem 3.6, for the general case we need a trick, which is to deal with the functional part g of $P^B_\xi(x, g)$ first. For any x define the functional $g^{\xi_0}_x$ by $\prec$-recursion on x by

$$g^{\xi_0}_x := \lambda y \text{ . if } x \oplus y \prec x \text{ then } \xi_0(x \oplus y, g^{\xi_0}_{x \oplus y}), \text{ else } \mathbf{0}.$$

Now define

- $Good^B_\xi(x) :\equiv Min^*(B, \xi, x)$, and
- $App^B_{\bar{x},\xi_0}(x) :\equiv |B_{\bar{x}}|_{\xi_0 \bar{x} g^{\xi_0}_{\bar{x}}} \to |B_x|_{\xi_0 x g^{\xi_0}_x}$.

where

$$Min^*(B, \xi, x) :\equiv x \oplus \xi_1 x g^{\xi_0}_x \prec x \to \neg |B_{x \oplus \xi_1 x g^{\xi_0}_x}|_{g^{\xi_0}_x(\xi_1 x g^{\xi_0}_x)}.$$

Lemma 4.4. *Define the functionals $\delta_{\bar{x},\xi}$ and $\tilde{\xi}$ by*

$$\delta_{\bar{x},\xi}(x) := (\xi_0 \bar{x} g^{\xi_0}_{\bar{x}}, x, g^{\xi_0}_x) \quad \textit{and} \quad \tilde{\xi}(x) := \xi_1 x g^{\xi_0}_x.$$

and the predicate $Q(z)$ by

$$Q(z) :\equiv |B_{\bar{x}}|_{z_0} \to P^B_\xi(z_1, z_2).$$

Then for $App_{\bar{x},\xi}$ defined as above, we have

$$\forall x \begin{cases} App^B_{\bar{x},\xi_0}(x) \to Min^*(B, \xi, x) \to Q(\delta_{\bar{x},\xi}(x)) \\ App^B_{\bar{x},\xi_0}(x) \to \neg Min^*(B, \xi, x) \to App^B_{\bar{x},\xi_0}(x \oplus \tilde{\xi}(x)) \end{cases}$$

Proof. The first line just follows directly from expanding the definitions. For the second part, $\neg Min(B, \xi, x)$ implies both $x \oplus \xi_1 x g^{\xi_0}_x = x \oplus \tilde{\xi}(x) \prec x$ and $|B_{x \oplus \tilde{\xi}(x)}|_{g^{\xi_0}_x(\tilde{\xi}(x))}$. But, using $x \oplus \tilde{\xi}(x) \prec x$, we have

$$g^{\xi_0}_x(\tilde{\xi}(x)) = \xi_0(x \oplus \tilde{\xi}(x), g^{\xi_0}_{x \oplus \tilde{\xi}(x)})$$

by the defining equation of g^{ξ_0}, and hence

$$|B_{x \oplus \tilde{\xi}(x)}|_{\xi_0(x \oplus \tilde{\xi}(x), g^{\xi}_{x \oplus \tilde{\xi}(x)})},$$

from which $App^B_{\bar{x},\xi_0}(x \oplus \tilde{\xi}(x))$ trivially follows. □

Theorem 4.5 (Functional interpretation of the minimum principle)**.** *The formula*

$$\forall \bar{x}, \xi \exists c, x, g(|B_{\bar{x}}|_c \to P^B_\xi(x, g))$$

is satisfied by the functional

$$\lambda \bar{x}, \xi \text{ . } \lim \mathcal{L}^B_{\bar{x},\xi}[\bar{x}]$$

for $\mathcal{L}^B_{\bar{x},\xi} := (Min^*(B, \xi, -), \delta_{\bar{x},\xi}, \tilde{\xi}, \oplus)$, where $\delta_{\bar{x},\xi}$ and $\tilde{\xi}$ as defined as in Lemma 4.4.

Proof. By Lemma 4.4 and Theorem 3.6 we have

$$\forall x(App^B_{\bar{x},\xi_0}(x) \to Q(\lim \mathcal{L}^B_{\bar{x},\xi}[x])),$$

and since $App^B_{\bar{x},\xi_0}(\bar{x})$ trivially holds, the result follows by definition of Q. □

5. Learning algorithms on infinite sequences

We now come to the main part of the paper, in which we define a simple operation on countable sequences of learning algorithms that combines them into a single learning algorithm on infinite sequences. This operation allows us to build approximations to choice functions and thereby give a computational interpretation to mathematical analysis, as we will prove in Section 6 below. In Section 7 we then compare these learning-based realizers to those obtained using Spector's bar recursion. However, we begin by simply defining our operation and providing an informal argument that the resulting learning procedures are well-founded in continuous models.

Definition 5.1. Suppose that $\alpha : X^\mathbb{N}$ and $(n, x) : \mathbb{N} \times X$. Then $\alpha[n \mapsto x] : X^\mathbb{N}$ denotes the function β defined by $\beta(m) = \alpha(m)$ for $m \neq n$ and $\beta(n) = x$.

Definition 5.2. Let X and Y be arbitrary types, and suppose that we have a sequence of decidable predicates $Good_n(\xi, x)$ on $(X \to Y) \times X$ together with a sequence of binary operations $\oplus_n : X \times Y \to X$. Define the decidable predicate $Good_\infty(E, \alpha)$ on $(X^\mathbb{N} \to \mathbb{N} \times Y) \times X^\mathbb{N}$ and the binary operation $\oplus_\infty : X^\mathbb{N} \times (\mathbb{N} \times Y) \to X^\mathbb{N}$ by

(a) $Good_\infty(E, \alpha) := Good_{E_0 \alpha}(\lambda x.E_1(\alpha[E_0 \alpha \mapsto x]), \alpha(E_0 \alpha))$;
(b) $\alpha \oplus_\infty (n, y) := \alpha[n \mapsto \alpha n \oplus_n y]$.

The idea behind these constructions is the following: for any $n : \mathbb{N}, \xi : X \to Y$ and $\delta : X \to Z$, we can construct a 'pointwise' learning algorithm

$$\mathcal{L}^n_{\xi,\delta} := (Good_n(\xi, -), \delta, \xi, \oplus_n)$$

of type X, Y, Z, while for any $E : X^\mathbb{N} \to \mathbb{N} \times Y$ and $D : X^\mathbb{N} \to Z'$ we can combine these algorithms into a 'global' learning algorithm

$$\mathcal{L}^\infty_{E,D} := (Good_\infty(E, -), D, E, \oplus_\infty)$$

of type $X^\mathbb{N}, \mathbb{N} \times Y, Z'$. The usefulness of this construction will become clear in the next section, but first we want to justify that it produces learning procedures that terminate in some reasonable setting.

Lemma 5.3. *Suppose that for any ξ, $\mathcal{L}^n_{\xi,\delta}$ reduces with respect to the ordering $\prec_n$, in other words*

$$\forall n, \xi, x(\neg Good_n(\xi, x) \to x \oplus_n \xi(x) \prec_n x). \qquad (5)$$

Then for any E, $\mathcal{L}^\infty_{E,D}$ reduces with respect to the ordering $\prec^{E_0}_\infty$ defined by

$$\beta \prec^F_\infty \alpha := \exists x(\beta = \alpha[F\alpha \mapsto x] \wedge x \prec_{F\alpha} \alpha(F\alpha)),$$

for $F : X^\mathbb{N} \to \mathbb{N}$ a parameter. In other words

$$\forall E, \alpha(\neg Good_\infty(E, \alpha) \to \alpha \oplus_\infty E(\alpha) \prec^{E_0}_\infty \alpha).$$

Proof. By definition we have

$$\neg Good_\infty(E, \alpha) \to \neg Good_{E_0 \alpha}(\xi, \alpha(E_0 \alpha))$$

for $\xi := \lambda x.E_1(\alpha[E_0 \alpha \mapsto x])$, and setting $n = E_0 \alpha$ and $x = \alpha(E_0 \alpha)$ in (5) this implies that

$$\alpha(E_0 \alpha) \oplus_{E_0 \alpha} \xi(\alpha(E_0 \alpha)) \prec_{E_0 \alpha} \alpha(E_0 \alpha).$$

But since $\xi(\alpha(E_0 \alpha)) = E_1(\alpha[E_0 \alpha \mapsto \alpha(E_0 \alpha)]) = E_1 \alpha$ it follows that

$$\alpha(E_0 \alpha) \oplus_{E_0 \alpha} E_1 \alpha \prec_{E_0 \alpha} \alpha(E_0 \alpha),$$

and since $\alpha \oplus_\infty E\alpha = \alpha[E_0 \alpha \mapsto \alpha(E_0 \alpha) \oplus_{E_0 \alpha} E_1 \alpha]$ we must have $\alpha \oplus_\infty E\alpha \prec^{E_0}_\infty \alpha$. □

At this point we need to take a little care, since even if the $\prec_n$ are provably well-founded in $\mathsf{E\text{-}HA}^\omega$, it is not too difficult to define a set-theoretic functional $F : X^\mathbb{N} \to \mathbb{N}$ for which $\prec^F_\infty$ is not well-founded. For example, we can take $X = (\mathbb{N}, <)$ and $F\alpha$ to be the least n such that $\alpha(n) > 0$ and 0 otherwise. Then

$$1, 1, 1, \ldots \succ^F_\infty 0, 1, 1, \ldots \succ^F_\infty 0, 0, 1, \ldots \succ^F_\infty \ldots$$

However, we can overcome this problem by requiring F to be *continuous*, so that in particular the computation of $F\alpha$ only depends on finitely many values of α i.e.

$$\exists N \forall \beta (\forall n < N(\alpha n = \beta n) \to F\alpha = F\beta). \tag{6}$$

This continuity principle is satisfied, for example, by all F of type $X^{\mathbb{N}} \to \mathbb{N}$ in the standard Kleene-Kreisel type structure $\mathcal{C}^\omega$ of continuous functionals [12].

To see that $\prec_\infty^F$ is well-founded in this case, suppose for contradiction that there is some infinite descending chain (α_i) which satisfies $\alpha_{i+1} \prec_\infty^F \alpha_i$ for all i i.e.

$$\alpha_{i+1} = \alpha_i[F\alpha_i \mapsto x_i] \text{ and } x_i \prec_{F\alpha_i} \alpha_i(F\alpha_i)$$

for some sequence of elements (x_i). For any fixed n, we have $\alpha_{i+1}(n) \preceq_n \alpha_i(n)$ for all i, and since $\prec_n$ is well-founded the sequence $(\alpha_i(n))_{i\in\mathbb{N}}$ eventually stabilizes, so we can define a limit function $\tilde{\alpha}$ by

$$\tilde{\alpha}(n) := \alpha_{i_n}(n) \text{ for } i_n \text{ such that } \alpha_i(n) = \alpha_{i_n}(n) \text{ for all } i \geq i_n.$$

Let N be the point of continuity we obtain from (6) applied to $\tilde{\alpha}$, and let $I := \max_{n \leq F\tilde{\alpha}, N}\{i_n\}$. Then for all $n < N$ we have $I \geq i_n$ and thus $\alpha_I(n) = \alpha_{i_n}(n) = \tilde{\alpha}(n)$, which by (6) implies that $F\alpha_I = F\tilde{\alpha}$. But then $\alpha_{I+1}(F\tilde{\alpha}) = x_I \prec_{F\tilde{\alpha}} \alpha_I(F\tilde{\alpha})$, which contradicts the fact that $\alpha_{I+1}(F\tilde{\alpha}) = \alpha_I(F\tilde{\alpha}) = \alpha_{i_{F\tilde{\alpha}}}(F\tilde{\alpha})$.

Following Lemma 4.4, we can define realizers of the form $\lambda x.\lim\ \mathcal{L}_E^\infty[\alpha]$ in any extension of $\mathsf{E\text{-}HA}^\omega$ with a recursion operator over $\prec_\infty^F$, such as

$$\Phi^{\phi,F}(\alpha) := \phi(\lambda x\ .\ \Phi^{\phi,F}(\alpha[F\alpha \mapsto x]) \text{ if } x \prec_{F\alpha} \alpha(F\alpha) \text{ else } \mathbf{0}).$$

This is essentially a variant of the *open recursion* operator studied by Berger in [5]. By adapting the well-foundedness argument above, one can then show that Φ defines a total continuous functional and therefore is satisfied in the model $\mathcal{C}^\omega$, analogously to [5, Section 5]. However, we omit any details here. In the next section we will simply show that realizers for the ND-interpretation of classical analysis can be given as limits of learning procedures of the form $\mathcal{L}^\infty$, and our goal here was simply to convince the reader that these limits make sense in standard continuous models of higher-type functionals.

6. The functional interpretation of classical analysis

We now demonstrate that our construction on learning algorithms gives a very natural way of building approximations to choice functions. The running example we use to illustrate this is arithmetical comprehension for Σ_1-formulas i.e.

$$\Sigma_1\text{-}\mathsf{CA} : \exists g^{\mathbb{N}\to\mathbb{B}} \forall n(gn = \top \leftrightarrow \exists x P_n(x))$$

where $P_n(x)$ is quantifier-free, although our construction allows us to interpret a much more general class of principles.

As we recount in the next section, following Spector [17], the traditional way of realizing the ND interpretation of Σ_1-CA is to realize the D-interpretation of the stronger *double negation shift*

$$\mathsf{DNS} : \forall n \neg\neg A_n \to \neg\neg\forall n A_n$$

using an extension of the primitive recursive functionals with *bar recursion*. In this section we take a more intuitive approach that bears a lot in common with Hilbert's epsilon elimination procedure and the recent work on learning realizability of Aschieri et al. [2]. Rather than giving an interpretation to the DNS as a whole, we show that our operation on learning algorithms interprets a certain rule form of the DNS: namely that whenever its premise can be realized by a countable collection of learning algorithms $\mathcal{L}^n$, its conclusion can be realized by $\mathcal{L}^\infty$. Although less general than Spector's approach, we still obtain a realizer of the ND-interpretation of Σ_1-CA (a principle already strong enough to formalise a large portion of mathematical analysis) as a very simple instance of $\mathcal{L}^\infty$, and our realizers enjoy the advantage that their *algorithmic* behaviour can be directly understood in terms of a simple **while**-loop.

Theorem 6.1. *Suppose that $\exists x^X \forall y^Y |A_n|_y^x$ is the D-interpretation of A_n, and App_n is some arbitrary sequence of predicates such that*

$$\forall n, \xi^{X\to Y}, x \begin{cases} App_n(x) \to Good_n(\xi, x) \to |A_n|_{\xi(x)}^x \\ App_n(x) \to \neg Good_n(\xi, x) \to App_n(x \oplus_n \xi(x)) \end{cases} \tag{7}$$

holds. Then the formula

$$\forall \bar{x}, n, \xi \exists x (App_n(\bar{x}) \to |A_n|_{\xi(x)}^x) \tag{8}$$

is realized by the functional

$$\lambda \bar{x}, n, \xi\ .\ \lim \mathcal{L}_\xi^n[\bar{x}]$$

for $\mathcal{L}_\xi^n := (Good_n(\xi, -), \iota, \xi, \oplus_n)$, assuming $\mathcal{L}_\xi^n$ reduces with respect to some well-founded ordering, and the formula

$$\forall \bar{\alpha}, \omega^{X^{\mathbb{N}}\to\mathbb{N}}, \phi^{X^{\mathbb{N}}\to Y} \exists \alpha (\forall n App_n(\bar{\alpha}n) \to |A_{\omega\alpha}|_{\phi\alpha}^{\alpha(\omega\alpha)}) \tag{9}$$

is realized by the functional

$$\lambda \bar{\alpha}, \omega, \phi\ .\ \lim \mathcal{L}_{(\omega,\phi)}^\infty[\bar{\alpha}]$$

with $\mathcal{L}_{(\omega,\phi)}^\infty := (Good_\infty((\omega,\phi), -), \iota, (\omega,\phi), \oplus_\infty)$ as in Definition 5.2, assuming $\mathcal{L}_{(\omega,\phi)}^\infty$ reduces w.r.t. some well-founded ordering. In particular, for any sequence (x_n) satisfying $\forall n App_n(x_n)$, the D-interpretation of $\forall n \neg\neg\exists x \forall y |A_n|_y^x$ is realized by

$$\lambda n, \xi\ .\ \lim \mathcal{L}_\xi^n[x_n]$$

and the D-interpretation of $\neg\neg\forall n \exists x \forall y |A_n|_y^x$ is realized by

$$\lambda \omega, \phi\ .\ \lim \mathcal{L}_{(\omega,\phi)}^\infty[\lambda n.x_n].$$

Proof. The realizer of (8) follows directly by (7) and Theorem 3.6. In order to verify the realizer of (9), we prove

$$\begin{cases} App_\infty(\alpha) \to Good_\infty((\omega,\phi),\alpha) \to |A_{\omega\alpha}|_{\phi\alpha}^{\alpha(\omega\alpha)} \\ App_\infty(\alpha) \to \neg Good_\infty((\omega,\phi),\alpha) \to App_\infty(\alpha \oplus_\infty (\omega,\phi)(\alpha)) \end{cases}$$

for all ω, ϕ, α, where $App_\infty(\alpha) :\equiv \forall n App_n(\alpha n)$, in which case correctness follows from Theorem 3.6.

First, note that by Definition 5.2 we have

$$Good_\infty((\omega,\phi),\alpha) \leftrightarrow Good_{\omega\alpha}(\xi, \alpha(\omega\alpha))$$

for $\xi := \lambda x.\phi(\alpha[\omega\alpha \mapsto x])$. Therefore if $Good_\infty((\omega,\phi),\alpha)$ holds, then since $App_\infty(\alpha) \to App_{\omega\alpha}(\alpha(\omega\alpha))$ then by (7) we obtain $|A_{\omega\alpha}|_{\xi(\alpha(\omega\alpha))}^{\alpha(\omega\alpha)}$, and since $\xi(\alpha(\omega\alpha)) = \phi\alpha$ this is equivalent to $|A_{\omega\alpha}|_{\phi\alpha}^{\alpha(\omega\alpha)}$. On the other hand, if $\neg Good_\infty((\omega,\phi),\alpha)$ then by (7) we obtain $App_{\omega\alpha}(\alpha(\omega\alpha) \oplus_{\omega\alpha} \xi(\alpha(\omega\alpha)))$ i.e. $App_{\omega\alpha}(\alpha(\omega\alpha) \oplus_{\omega\alpha} \phi\alpha)$, which along with the assumption $App_\infty(\alpha)$ proves that $App_\infty(\alpha \oplus_\infty (\omega,\phi)(\alpha))$, and we're done. ☐

6.1 Interpreting Σ_1-CA

We now show how the ND interpretation of Σ_1^0-CA can be obtained as a simple consequence Theorem 6.1. By a quick calculation using equations (1) we can show that the ND-interpretation of Σ_1^0-CA is equivalent to that of $\neg\neg\exists\alpha^{\mathbb{N}\to\mathbb{B}\times X}\forall n, y(P_n(\alpha_1 n) \vee_{\alpha_0 n} \neg P_n(y))$, where α_0 represents the comprehension function g and α_1 a sequence such that $\alpha_1 n$ realizes $\exists x P_n(x)$ whenever $\alpha_0 n = \top$. This in turn is identical to the D-interpretation of $\neg\neg\forall n A_n$ for $A_n :\equiv \exists x, b \forall y (P_n(x) \vee_b \neg P_n(y))$, so we can apply Theorem 6.1 directly.

Lemma 6.2. *Define*

(a) $App_n(b, x) :\equiv (b = \top \to P_n(x))$*;*
(b) $Good_n(\xi, b, x) :\equiv (b = \bot \to \neg P_n(\xi bx))$ *and*
(c) $(b, x) \oplus_n y := (\top, y)$.

Then the D-interpretation of

$$\forall n \neg\neg \exists b, x \forall y (P_n(x) \vee_b \neg P_n(y))$$

which is

$$\forall n, \xi \exists b, x (P_n(x) \vee_b \neg P_n(\xi bx))$$

is realized by the learning procedure

$$\lambda n, \xi \ . \lim \mathcal{L}^n_\xi[(\bot, \mathbf{0})]$$

for $\mathcal{L}^n_\xi := (Good_n(\xi, -), \iota, \xi, \oplus_n)$*, while the D-interpretation of*

$$\neg\neg \forall n \exists b, x \forall y (P_n(x) \vee_b \neg P_n(y))$$

and hence Σ_1-CA*, which is*

$$\forall \omega, \phi \exists \alpha (P_{\omega\alpha}(\alpha_1(\omega\alpha)) \vee_{\alpha_0(\omega\alpha)} \neg P_{\omega\alpha}(\phi\alpha))$$

is realized by the learning procedure

$$\lambda \omega, \phi \ . \lim \mathcal{L}^\infty_{(\omega,\phi)}[\lambda n.(\bot, \mathbf{0})]$$

for $\mathcal{L}^\infty_{(\omega,\phi)} := (Good_\infty((\omega, \phi), -), \iota, (\omega, \phi), \oplus_\infty)$.

Proof. By Theorem 6.1 it suffices to show that (7) holds, which after unwinding the definitions is completely trivial. Bearing in mind that $App_n(\bot, \mathbf{0})$ also holds for any n, the result follows. □

As a consequence of expressing it in terms of a learning algorithm, the underlying procedural behaviour of our realizer for Σ_1-CA is much easier to understand. In informal terms, it produces a functional α of type $\mathbb{N} \to \mathbb{B} \times X$ which we can imagine as encoding a partial function of type $\mathbb{N} \to X$, where the first component $\alpha_0 n : \mathbb{B}$ informs us if n is in the domain of α, while the second $\alpha_1 n : X$ contains the actual value. Thus the initial element of the procedure $\lambda n.(\bot, \mathbf{0})$ just encodes the empty partial function $\emptyset$, and unfolding all the definitions the test predicate $Good_\infty((\omega, \phi), \alpha)$ can be expressed as $\omega\alpha \notin \text{domain}(\alpha) \to \neg P_{\omega\alpha}(\phi\alpha)$. Therefore the underlying learning procedure is equivalent to the simple **while**-loop sketched in Algorithm 3. We start with an empty approximation $\emptyset$ to a comprehension function α, and gradually build up a better approximation by adding realizers of $\exists x P_n(x)$ whenever we find them. Eventually, the program terminates at a stage where either $\omega\alpha \in \text{domain}(\alpha)$, and by definition $\alpha_1(\omega\alpha)$ is a realizer of $\exists x P_{\omega\alpha}(x)$, or $\omega\alpha \notin \text{domain}(\alpha)$ and $\neg P_{\omega\alpha}(\phi\alpha)$.

Algorithm 3 $\forall \omega, \phi \exists \alpha (P_{\omega\alpha}(\alpha_1(\omega\alpha)) \vee_{\alpha_0(\omega\alpha)} \neg P_{\omega\alpha}(\phi\alpha))$

```
1: input $\omega, \phi$
2: $\alpha = \emptyset$
3: while $\omega\alpha \notin \text{domain}(\alpha) \wedge P_{\omega\alpha}(\phi\alpha)$
4:     $\alpha \to \alpha[\omega\alpha \mapsto (\top, \phi\alpha)]$
5: return $\alpha$
```

This way of building approximations to a comprehension function is by no means new: algorithms of this form underlie Hilbert's epsilon substitution method, are studied by Avigad in [3] where they are known as *update procedures*, and are essential to the learning-realizability of [2]. The novelty here is that it forms an instance of a general framework for obtaining learning-based realizers for choice functions via the ND interpretation. Interestingly, this solution in the case of Σ^0_1-CA was briefly considered by Spector as an alternative to bar recursion in [17, 12.1]. According to the posthumous footnotes of Kreisel, Spector planned to extend this to solve more general instances of double negation shift, as we have done to an extent here. Sadly, it is not clear precisely what Spector's intentions were as he died before he was able to carry them out.

7. Bar recursion and 'forgetful' learning

Spector did, however, give a full computational interpretation to the DNS using bar recursion. We conclude the article by showing that, underneath the complex syntax, simple cases of bar recursion produce limits of a special kind of learning procedure which forget everything that they have learned above the point that is being updated. This section is the most technically involved of the paper, and casual readers not already familiar with bar recursion may prefer to skim over the details of the section up until 7.1, where analogously to Section 6.1 we illustrate the main points by looking at the simple case of Σ_1-CA.

As mentioned earlier, Spector's realizer for comprehension principles was obtained by realizing the D-interpretation of the full double negation shift. The interpretation is given as follows:

$$\begin{aligned} &\forall n \neg\neg \exists x \forall y |A_n|^x_y \to \neg\neg \exists \alpha \forall n, y |A_n|^{\alpha n}_y \\ &\rightsquigarrow \exists L \forall n, \xi |A_n|^{Ln\xi}_{\xi(Ln\xi)} \to \forall \omega, \phi \exists \alpha |A_{\omega\alpha}|^{\alpha(\omega\alpha)}_{\phi\alpha} \\ &\rightsquigarrow \forall L, \omega, \phi \exists n, \xi, \alpha (|A_n|^{Ln\xi}_{\xi(Ln\xi)} \to |A_{\omega\alpha}|^{\alpha(\omega\alpha)}_{\phi\alpha}). \end{aligned}$$

In his landmark paper, Spector witnessed this last line by using the bar recursive functional $\mathsf{BR}^{L,\omega,\phi} : X^* \to X^{\mathbb{N}}$ defined as (see Notation 2.1)

$$\mathsf{BR}^{L,\omega,\phi}(s) = \begin{cases} \hat{s} & \text{if } \hat{\omega}(s) < |s| \\ \mathsf{BR}^{L,\omega,\phi}(s * y_s) & \text{otherwise,} \end{cases}$$

where $y_s :=_X L|s|(\lambda x \ . \ \phi(\mathsf{BR}^{L,\omega,\phi}(s * x)))$. He demonstrated that $\alpha := \mathsf{BR}^{L,\omega,\phi}(\langle\rangle)$, $n := \omega\alpha$ and $\xi := \lambda x.\phi(\mathsf{BR}^{L,\omega,\phi}([\alpha](n) * x))$ satisfy the equations

$$\alpha(\omega\alpha) = Ln\xi \quad \text{and} \quad \phi\alpha = \xi(Ln\xi)$$

thus solving the functional interpretation of DNS. We do not go any further into the details of this general solution of DNS (for these the reader is directed to e.g. [10, 13]). Rather we try to link everything to the previous section and focus on the underlying *algorithmic* meaning of the resulting realizer for $\neg\neg\forall n A_n$, in cases where the premise of DNS is realized by a simple kind of learning procedure, namely

$$L^a n\xi := \lim \mathcal{L}^n_\xi[an]$$

for

$$\mathcal{L}^n_\xi := (T_n(-, \xi(-)), \iota, \xi, \oplus_n)$$

for some decidable predicate T_n on $X \times Y$, operation $\oplus_n$ and $a : X^{\mathbb{N}}$. Note that this is more restrictive than the learning algorithms of the previous section, but is still general enough to include Σ_1-CA as a very simple case. In addition we assume that $\mathcal{L}^n_\xi$ reduces with respect to some well-founded order $\prec_n$, and thus we can express $L^a n\xi$ as a $\prec_n$-recursive function i.e. $L^a n\xi = l_{n,\xi}(an)$ where $l_{n,\xi}$ is defined by

$$l_{n,\xi}(x) := \begin{cases} x & \text{if } T_n(x, \xi(x)) \\ l_{n,\xi}(x \oplus_n \xi(x)) & \text{otherwise.} \end{cases}$$

We claim that for L^a of this form, the realizer $\mathsf{BR}^{L^a,\omega,\phi}(\langle\rangle)$ is the limit of a learning procedure, which we define now. We first need some preliminary definitions.

Definition 7.1. Let $s : X^*$ be a finite sequence, $a : X^{\mathbb{N}}$ an infinite sequence and $\omega : X^{\mathbb{N}} \to \mathbb{N}$ a functional. Define $N_{s,a,\omega}$ to be the least $n \geq |s|$ such that

$$\hat{\omega}(s * \langle a_{|s|}, \ldots, a_{n-1} \rangle) < n$$

and define

$$s_{a,\omega} := s * \langle a_{|s|}, \ldots, a_{N_{s,a,\omega}-1} \rangle$$

The point $N_{s,a,\omega}$ always exists for continuous ω, and as shown in e.g. [14] is definable in $\mathsf{E\text{-}HA}^\omega + (\mathsf{BR})$.

Definition 7.2. Let the interval $[m, n)$ denote the set $\{m, \dots, n-1\}$ if $m < n$ and $\emptyset$ otherwise. Given a decidable predicate $P(n)$ and finite $X \subset \mathbb{N}$ let $\bar{\mu}n \in X$. $P(n)$ denote the greatest $n \in X$ such that $P(n)$ holds, and just 0 if no such n exists.

Definition 7.3. Given a collection of predicates $T_n(x, y)$ on $X \times Y$ together with a collection of operations $\oplus_n : X \times Y \to X$ which partially define $\mathcal{L}^n$ as above, for each $m \in \mathbb{N}$ define the learning algorithm

$$\mathcal{L}^{\infty,m}_{a,\omega,\phi} := (Good^m_{a,\omega,\phi}, D^m_{a,\omega,\phi}, E^m_{a,\omega,\phi}, \oplus_a)$$

of type $X^*, \mathbb{N} \times Y, X^{\mathbb{N}}$ parametrised by $a : X^{\mathbb{N}}$, $\omega : X^{\mathbb{N}} \to \mathbb{N}$ and $\phi : X^{\mathbb{N}} \to Y$ as

(a) $Good^m_{a,\omega,\phi}(s) := \forall n \in [m, |s_{a,\omega}|)\ T_n((s_{a,\omega})_n, \hat{\phi}(s_{a,\omega}))$;
(b) $D^m_{a,\omega,\phi} s := \widehat{s_{a,\omega}}$;
(c) $(E^m_{a,\omega,\phi})_0 s := \bar{\mu}n \in [m, |s_{a,\omega}|)\ .\ \neg T_n((s_{a,\omega})_n, \hat{\phi}(s_{a,\omega}))$;
(d) $(E^m_{a,\omega,\phi})_1 s := \hat{\phi}(s_{a,\omega})$;
(e) $s \oplus_{a,\omega} (n, y) := [s_{a,\omega}](n) * ((s_{a,\omega})_n \oplus_n y)$ if $n < |s_{a,\omega}|$ else $\langle\rangle$.

Theorem 7.4. *Let $L^a n\xi := l_{n,\xi}(an)$ for l as defined above (with respect to some T_n, $\oplus_n$). Then for any infinite sequence a, the limit of $\mathcal{L}^{\infty,0}_{a,\omega,\phi}[\langle\rangle]$ (w.r.t. the same T_n, $\oplus_n$) exists and we have*

$$\lim \mathcal{L}^{\infty,0}_{a,\omega,\phi}[\langle\rangle] = \mathsf{BR}^{L^a,\omega,\phi}(\langle\rangle)$$

In particular, the realizer of $\neg\neg\forall n A_n$ given by Spector's bar recursion in this case is just $\lambda\omega, \phi.\lim \mathcal{L}^{\infty,0}_{a,\omega,\phi}[\langle\rangle]$.

Proof. The proof consists of a single instance of the following variant of *bar induction*:

$$\forall \beta^{X^{\mathbb{N}}} \exists N B([\beta](N)) \wedge \forall s(\forall x B(s * x) \to B(s)) \to B(\langle\rangle),$$

which is classically valid for any B by the axiom of dependent choice. We use it with $B(s)$ defined as the formula

$$\forall a (\text{the limit of } \mathcal{L}^{\infty,|s|}_{a,\omega,\phi}[s] \text{ exists and is equal to } \mathsf{BR}^{L^a,\omega,\phi}(s)).$$

The theorem then follows directly from $B(\langle\rangle)$. Thus we need to establish the two premises of bar induction. Because the parameters ω and ϕ remain fixed throughout, we omit these from $\mathcal{L}^{\infty,m}_{a,\omega,\phi}$ and its components from now on.

(i) $\forall \beta \exists N B([\beta](N))$

Take some infinite sequence β. Then assuming we are working in a model of $\mathsf{E\text{-}HA}^\omega$ + (BR) such as the continuous functionals, there exists some N such that $\hat{\omega}([\beta](N)) < N$. In this case $Good^N_a([\beta](N))$ is trivially true, since $[\beta](N)_{a,\omega} = [\beta](N)$ and thus $[N, |[\beta](N)_{a,\omega}|) = \emptyset$, and therefore

$$\lim \mathcal{L}^{\infty,N}_a[[\beta](N)] = \widehat{[\beta](N)_{a,\omega}} = \widehat{[\beta](N)} = \mathsf{BR}^{L^a}([\beta](N)),$$

the last equality following from the defining equation of $\mathsf{BR}^{L^a}(s)$ for $\hat{\omega}(s) < |s|$. This establishes $B([\beta](N))$.

(ii) $\forall s(\forall x B(s * x) \to B(s))$

Fixing s, we can assume that $\hat{\omega}(s) \geq |s|$, for if $\hat{\omega}(s) < |s|$ then we have $\lim \mathcal{L}^{\infty,|s|}_a[s] = \hat{s} = \mathsf{BR}^{L^a}(s)$ by exactly the same reasoning as with the case (i) above. We now come to the core of the proof, which is to show that for fixed $a : X^{\mathbb{N}}$ we have

$$\underbrace{\lim \mathcal{L}^{\infty,|s|}_a[s * x] \text{ exists and is equal to } \mathsf{BR}^{L^a}(s * l_{|s|,\xi}(x)))}_{Q_a(x)}$$

for all x, with

$$\xi := \lambda x\ .\ \phi(\mathsf{BR}^{L^a}(s * x)).$$

Note that the learning procedure in $Q_a(x)$ is $\mathcal{L}^{\infty,|s|}_a$ and not $\mathcal{L}^{\infty,|s*x|}_a$, and so this is not the same as proving $B(s * x)$! We prove $Q_a(x)$ by induction on x with respect to the ordering $\prec_{|s|}$. We fix x and assume $\forall x' \prec_{|s|} x Q_a(x')$.

Let $(u_i), (v_i) : (X^*)^{\mathbb{N}}$ be, respectively, the learning procedures $\mathcal{L}^{\infty,|s|}_a[s * x]$ and $\mathcal{L}^{\infty,|s*x|}_a[s * x]$. By the main *bar induction* hypothesis $B(s * x)$ we can assume that (v_i) has a limit point v_j and that

$$D^{|s*x|}_a(v_j) = \widehat{(v_j)_{a,\omega}} = \mathsf{BR}^{L^\alpha}(s * x). \tag{10}$$

We claim that for all $i \leq j$

(a) $u_i = v_i$, and
(b) $[(u_i)_{a,\omega}](|s| + 1) = [(v_i)_{a,\omega}](|s| + 1) = s * x$.

We use a side induction from $i = 0$ to $i = j$. For the base case we clearly have $u_0 = s * x = v_0$ and $[(s * x)_{a,\omega}](|s| + 1) = s * x$ by definition. Now assume that (a) and (b) hold for $i < j$. Since v_i is not a limit point, we must have $\neg Good^{|s*x|}_a(v_i)$, which implies there is some $k \in [|s * x|, |(v_i)_{a,\omega}|)$ such that

$$\neg T_k(((v_i)_{a,\omega})_k, \hat{\phi}((v_i)_{a,\omega})).$$

But since $u_i = v_i$ then this k is also the greatest in $[|s|, |(u_i)_{a,\omega}|)$ satisfying

$$\neg T_k(((u_i)_{a,\omega})_k, \hat{\phi}((u_i)_{a,\omega})),$$

and thus $\neg Good^{|s|}_a(u_i)$ also holds with $E^{|s|}_a(u_i) = E^{|s*x|}_a(v_i) = (k, \hat{\phi}(u_i)_{a,\omega})$. Therefore

$$u_{i+1} = u_i \oplus_a E^{|s|}_a(u_i) = v_i \oplus_a E^{|s*x|}_a(v_i) = v_{i+1}.$$

Moreover, since $|u_{i+1}| = (E^{|s|}_a)_0(u_i) + 1 = k + 1 > k > |s|$ and $((u_i)_{a,\omega})_{|s|} = x$ we must also have

$$[(u_{i+1})_{\alpha,\omega}](|s|+1) = [u_{i+1}](|s|+1) = [(u_i)_{\alpha,\omega}](|s|+1) = s * x$$

the last step using the side induction hypothesis (b) for u_i. This proves the claim. Now, since v_j *is* a limit point we have $Good^{|s*x|}_a(v_j)$ and thus since $u_j = v_j$

$$Good^{|s|}_a(u_j) \leftrightarrow T_{|s|}(((u_j)_{a,\omega})_{|s|}, \hat{\phi}((u_j)_{a,\omega})).$$

But by our claim (a), (b) for $i = j$ we have

$$((u_j)_{a,\omega})_{|s|} \overset{(b)}{=} x$$

and

$$\hat{\phi}((u_j)_{a,\omega}) \overset{(a)}{=} \hat{\phi}((v_j)_{a,\omega}) \overset{(10)}{=} \phi(\mathsf{BR}^{L^a}(s * x)) = \xi(x) \tag{11}$$

And thus

$$Good^{|s|}_\alpha(u_j) \leftrightarrow T_{|s|}(x, \xi(x)).$$

There are two cases to consider: Either $T_{|s|}(x, \xi(x))$ holds and thus

$$\lim \mathcal{L}^{\infty,|s|}_a[s * x] = \widehat{(u_j)_{a,\omega}} \overset{(a)}{=} \widehat{(v_j)_{a,\omega}} \overset{(10)}{=} \mathsf{BR}^{L^a}(s * x) \tag{12}$$

or $\neg T_{|s|}(x, \xi(x))$, in which case $E^{|s|}_a(u_j) = (|s|, \hat{\phi}((u_j)_{a,\omega})) \overset{11}{=} (|s|, \xi(x))$ and thus

$$\begin{aligned} u_{j+1} &= u_j \oplus_a E^{|s|}_a(u_j) \\ &= [(u_j)_{\omega,a}](|s|) * (((u_j)_{\omega,a})_{|s|} \oplus_{|s|} \xi(x))) \\ &\overset{(b)}{=} s * (x \oplus_{|s|} \xi(x)) \end{aligned}$$

Now $x' := x \oplus_{|s|} \xi(x) \prec_{|s|} x$ since $\neg T_{|s|}(x, \xi(x))$ and therefore

$$\lim \mathcal{L}^{\infty,|s|}_a[s * x] = \lim \mathcal{L}^{\infty,|s|}_a[u_{j+1}] = \lim \mathcal{L}^{\infty,|s|}_a[s * x']$$

so by the induction hypothesis $Q_a(x')$ we obtain

$$\lim \mathcal{L}^{\infty,|s|}_a[s * x'] = \mathsf{BR}^{L^\alpha}(s * l_{|s|,\xi}(x')). \tag{13}$$

Thus together with (12) we have

$$\lim \mathcal{L}_a^{\infty,|s|}[s*x] = \begin{cases} \mathsf{BR}^{L^a}(s*x) \text{ if } T_{|s|}(x,\xi(x)) \\ \mathsf{BR}^{L^a}(s*l_{|s|,\xi}(x \oplus_{|s|} \xi(x))) \text{ otherwise} \end{cases}$$
$$= \mathsf{BR}^{L^a}(s*l_{|s|,\xi}(x)).$$

This completes the auxiliary induction step, and so we have established that $\forall x Q_a(x)$, and therefore in particular $Q_a(a(|s|))$ i.e.

$$\begin{aligned}\lim \mathcal{L}_a^{\infty,|s|}[s*a(|s|)] &= \mathsf{BR}^{L^a}(s*l_{|s|,\xi}(a(|s|))) \\ &= \mathsf{BR}^{L^a}(s*L^a|s|\xi) \\ &= \mathsf{BR}^{L^a}(s)\end{aligned}$$

the last equality following from definition of ξ and the defining equation of BR^{L^a} for $\hat{\omega}(s) \geq |s|$. All that remains is to show that

$$\lim \mathcal{L}_a^{\infty,|s|}[s] = \lim \mathcal{L}_a^{\infty,|s|}[s*a(|s|)] \tag{14}$$

and we're done, since then $\lim \mathcal{L}_a^{\infty,|s|}[s] = \lim \mathcal{L}_a^{\infty,|s|}[s*a(|s|)] = \mathsf{BR}^{L^a}(s)$, and since a was arbitrary this establishes $B(s)$, and we're done with the second bar induction premise. But (14) is easy (though tedious) to prove. Since $\hat{\omega}(s) \geq |s|$ we must have $s_{a,\omega} = (s*a(|s|))_{a,\omega}$ and thus since $Good_a^{|s|}(t), D_a^{|s|}(t), E_a^{|s|}(t)$ and $t \oplus_a (n,y)$ depend only on $t_{a,\omega}$ we have either

$$\lim \mathcal{L}_a^{\infty,|s|}[s] = D_a^{|s|}(s) = D_a^{|s|}(s*a(|s|)) = \lim \mathcal{L}_a^{\infty,|s|}[s*a(|s|)]$$

in the case that $Good_a^{|s|}(s)$ (equivalently $Good_a^{|s|}(s*a(|s|))$) holds, or otherwise

$$\lim \mathcal{L}_a^{\infty,|s|}[s] = \lim \mathcal{L}_a^{\infty,|s|}[t] = \lim \mathcal{L}_a^{\infty,|s|}[s*a(|s|)]$$

for $t = s \oplus_a E_a^{|s|}(s) = (s*a(|s|)) \oplus_a E_a^{|s|}(s*a(|s|))$. $\square$

7.1 Interpreting Σ_1-CA, revisited

Let $A_n :\equiv \exists x, b \forall y (P_n(x) \vee_b \neg P_n(y))$ as in Section 6.1. It is easy to check that $\forall n \neg\neg A_n$ is realized by $L^{\emptyset} n \xi = l_{n,\xi}(\bot, \mathbf{0})$ with

$$l_{n,\xi}(b,x) = \begin{cases} (b,x) & \text{if } b = \bot \to \neg P_n(\xi b x) \\ (\top, \xi b x) & \text{otherwise,} \end{cases}$$

(denoting $\emptyset = \lambda n.(\bot, \mathbf{0})$ as before) and thus by Spector's result the ND interpretation of $\neg\neg\forall n A_n$ can be given by $\lambda\omega,\phi.\mathsf{BR}^{L^{\emptyset},\omega,\phi}(\langle\rangle)$. But by Theorem 7.4 we have $\mathsf{BR}^{L^{\emptyset},\omega,\phi}(\langle\rangle) = \lim \mathcal{L}_{\emptyset,\omega,\phi}^{\infty,0}[\langle\rangle]$ for $T_n((b,x),y) :\equiv b = \bot \to \neg P_n(y)$ and $(b,x) \oplus_n y := (\top, y)$. Assuming the convention $\mathbf{0}_{\mathbb{B}\times X} = (\bot, \mathbf{0}_X)$, it is not to difficult to show that $s_{\emptyset,\omega}$ as in Definition 7.1 is just the sequence $s*\langle \mathbf{0},\ldots,\mathbf{0}\rangle$ of length N_s, for

$$N_s := \left(\begin{cases} |s| & \text{if } \hat{\omega}(s) < |s| \\ \hat{\omega}(s)+1 & \text{otherwise} \end{cases} \right) = \max\{|s|, \hat{\omega}(s)+1\}.$$

Using this, we can unwrap all the definitions (we do not give the details here, though again this is all fairly straightforward) and see that the underlying learning algorithm is given as in Algorithm 4, where we use the notation $n \notin \text{domain}(s) :\equiv (s_n)_0 = \bot$ as in Section 6.1. It is highly illuminating to compare this with Algorithm 3. The basic idea of starting with an empty approximation to a comprehension function and updating with realizers for $\exists x P_n(x)$ is the same, but the manner in which it is done is completely different. In particular, when updating with a realizer at point n, all realizers for $P_m(x)$ with $m > n$ are forgotten.

Algorithm 4 $\forall \omega, \phi \exists \alpha (P_{\omega\alpha}(\alpha_1(\omega\alpha)) \vee_{\alpha_0(\omega\alpha)} \neg P_{\omega\alpha}(\phi\alpha))$

```
1: input $\omega, \phi$
2: $s = \langle\rangle$
3: while $\exists n < \max\{|s|, \hat{\omega}(s)+1\} \,.\, n \notin \text{domain}(s) \wedge P_n(\phi(\hat{s}))$
4:     $s \to [\hat{s}](n) * (\top, \phi(\hat{s}))$ for the greatest such $n$
5: return $\alpha := \hat{s}$
```

Acknowledgments

I am grateful to the anonymous referees for their many useful comments and corrections.

References

[1] F. Aschieri. *Learning, Realizability and Games in Classical Arithmetic*. PhD thesis, Universita degli Studi di Torino and Queen Mary, University of London, 2011.

[2] F. Aschieri and S. Berardi. Interactive learning-based realizability for Heyting arithmetic with $\mathsf{EM1}$. *Logical Methods in Computer Science*, 6(3), 2010.

[3] J. Avigad. Update procedures and the 1-consistency of arithmetic. *Mathematical Logic Quarterly*, 48(1):3–13, 2002.

[4] J. Avigad and S. Feferman. Gödel's functional ("Dialectica") interpretation. In S. R. Buss, editor, *Handbook of Proof Theory*, volume 137 of *Studies in Logic and the Foundations of Mathematics*, pages 337–405. North Holland, Amsterdam, 1998.

[5] U. Berger. A computational interpretation of open induction. In F. Titsworth, editor, *Proceedings of the Nineteenth Annual IEEE Symposium on Logic in Computer Science*, pages 326–334. IEEE Computer Society, 2004.

[6] T. Coquand. A semantics of evidence for classical arithmetic. *Journal of Symbolic Logic*, 60:325–337, 1995.

[7] V. de Paiva. *The Dialectica Categories*. PhD thesis, University of Cambridge, 1991. Published as Technical Report 213, Computer Laboratory, University of Cambridge.

[8] M. Escardó and P. Oliva. Selection functions, bar recursion and backward induction. *Mathematical Structures in Computer Science*, 20(2):127–168, 2010.

[9] K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *dialectica*, 12:280–287, 1958.

[10] U. Kohlenbach. *Applied Proof Theory: Proof Interpretations and their Use in Mathematics*. Monographs in Mathematics. Springer, 2008.

[11] A. Kreuzer. *Proof mining and Combinatorics : Program Extraction for Ramsey's Theorem for Pairs*. PhD thesis, TU Darmstadt, 2012.

[12] D. Normann. The continuous functionals. In S. Abramsky, S. Artemov, R. A. Shore, and A. S. Troelstra, editors, *Handbook of Computability Theory*, volume 140 of *Studies in Logic and the Foundations of Mathematics*, pages 251–275. North Holland, Amsterdam, 1999.

[13] P. Oliva. Understanding and using Spector's bar recursive interpretation of classical analysis. In A. Beckmann, U. Berger, B. Löwe, and J. V. Tucker, editors, *Proceedings of CiE'2006*, volume 3988 of *LNCS*, pages 423–234, 2006.

[14] P. Oliva and T. Powell. On Spector's bar recursion. *Mathematical Logic Quarterly*, 58:356–365, 2012.

[15] P. Oliva and T. Powell. A constructive interpretation of Ramsey's theorem via the product of selection functions. *Mathematical Structures in Computer Science*, 25(8):1755–1778, 2015.

[16] H. Schwichtenberg. Dialectica interpretation of well-founded induction. *Mathematical Logic Quarterly*, 54(3):229–239, 2008.

[17] C. Spector. Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles in current intuitionistic mathematics. In F. D. E. Dekker, editor, *Recursive Function Theory: Proc. Symposia in Pure Mathematics*, volume 5, pages 1–27. American Mathematical Society, Providence, Rhode Island, 1962.