Well Quasi-Orders and the Functional Interpretation

Thomas Powell

Abstract

The purpose of this article is to study the role of Gödel's functional interpretation in the extraction of programs from proofs in well quasi-order theory. The main focus is on the interpretation of Nash-Williams' famous minimal bad sequence construction, and the exploration of a number of much broader problems which are related to this, particularly the question of the constructive meaning of Zorn's lemma and the notion of recursion over the non-wellfounded lexicographic ordering on infinite sequences.

1 Introduction

When I was invited to contribute a chapter to this volume, I felt that I should write something that would reflect, as much as possible, the extraordinary richness of the theory of well quasi-orders. Anyone present at the Dagstuhl Seminar of January 2016 would have experienced first hand how this rather innocent looking mathematical object plays a central role in so many seemingly disparate areas, ranging from proof theory, computability theory and reverse mathematics on the one hand to to term rewriting, program verification and the world of automata and formal languages on the other. While I could never do justice to such diversity in one article, my hope was to at least explore a variety of interesting problems in my own field which arise from the study of well quasi-orders.

I decided, therefore, to write an essay on Gödel's functional interpretation, and the role it plays in making constructive sense of well quasi-orders. I have chosen to organise the essay around a somewhat superficial challenge, namely the development of a program which realizes Higman's lemma for boolean alphabets:

> PROBLEM. Write a program Φ which takes as input an infinite sequence u of words over a two letter alphabet, and returns a pair of indices $i < j \in \mathbb{N}$ such that u_i is embedded in u_j.

Of course, as long as one has proven that two such indices must exist one could simply write a program which carries out a blind search until they are found! However, I am interested in the question of how one can formally construct a subrecursive program which constitutes a computational analogue of Nash-Williams' famous minimal bad sequence construction - an elegant combinatorial idea which appears throughout well quasi-order theory.

It is important to stress that this relatively simple problem provides merely a narrative framework: My ulterior motive is to explore a number of much more elusive problems which lurk underneath. So while on the surface we will work towards the construction of our program Φ, the real aim of this essay is to try to address several deeper questions, chief among them being:

1. What is the computational meaning of Zorn's lemma?

2. Is it possible to sensibly define recursive functionals on chain-complete partial orders?

3. How can one describe formally extracted programs so that they can be easily understood by a human?

Each of these questions has significance far beyond Higman's lemma, and yet the fact that they are all naturally prompted by our elementary problem is, I believe, testament to the richness inherent to the theory of well quasi-orders.

1.1 Proof interpretations and well quasi-orders: A brief history

In 1958, Gödel published a landmark paper [10] which introduced his functional, or 'Dialectica' interpretation, which he had already conceived in the 1930s as a response to Hilbert's program and his own incompleteness theorems. Initially, the functional interpretation translated Peano arithmetic to a calculus of primitive recursive functionals in all finite types known as System T, thereby reducing the consistency of the former theory to the latter. In modern day parlance, System T is nothing more than a simple functional programming language which permits the construction of higher-type primitive recursive functionals. The soundness of the functional interpretation guarantees that, whenever some statement A is provable in Peano arithmetic, we can extract a total functional program in T which witnesses its translation A^I.

The functional interpretation was just one of a number of techniques designed during the mid 20th century to establish relative consistency proofs. Kreisel soon observed that these techniques could be flipped on their head and viewed from a different perspective: namely as tools for extracting computational information from non-constructive proofs [14, 15]. While the significance of this idea was not fully appreciated at the time, in recent decades the application of proof theoretic methods to extract programs from proofs has flourished, and now proof interpretations are primarily used for this purpose. Variants of Gödel's functional interpretation in particular are central to the highly successful 'proof mining' program pioneered by Kohlenbach [13], which has led to new quantitative results in several areas of mathematics. At the same time, the arrival of the computer has meant that the extraction of programs from proofs can be automated, and there are now proof assistants such as MINLOG [1] which are dedicated to this, and which implement sophisticated refinements of the traditional proof theoretic techniques.

So where do well quasi-orders feature in all of this?

The vast majority of proofs in 'normal' mathematics use only a very small amount of set theory. Often, proofs of existential theorems in mathematics analysis which officially require choice or comprehension, use it in such a limited way that it doesn't really contribute to the complexity of extracted programs. However, the theory of well quasi-orders contains a number of key theorems which *do* use choice in a crucial way, the most notorious being those such Kruskal's theorem which historically rely on variant of Nash-Williams' minimal bad sequence construction.

As a result, these theorems have become something of a focal point for research in program extraction, as canonical existential statements which come with concise, elegant, but proof theoretically non-trivial classical proofs. The question of the computational meaning of such proofs is so deep that entire theses have been dedicated to it (such as [18, 26]). By now, even comparatively simple results like Higman's lemma have an extensive body of research devoted to them. Thus the theory of well quasi-orders has firmly established a foothold in the world of proof theory, and it is from this perspective that I study them here.

1.2 The origins and purpose of this chapter

Given the popularity of Higman's lemma among researchers in proof theory, it's perhaps important to outline my own motivation in adding yet another paper to this menagerie.

My interest in well quasi-orders began when I was a doctoral student studying Gödel's functional interpretation. Paulo Oliva suggested to me that Higman's lemma might prove a useful exercise in program extraction via the functional interpretation, as up to that point this had never been done: The majority of attempts at giving a constructive proof of the lemma had utilised some form of realizability instead. So I undertook this challenge and published my work as [21].

While this indeed turned out to be a valuable for me personally, improving my own understanding of the functional interpretation and providing me with a welcome excuse to learn about well quasi-orders, in most other respects I found my work rather unsatisfactory. In order to give a computational interpretation of the instance of dependent choice used in the proof of the theorem, I resorted to the standard technique at one's disposal - a higher-type form of bar recursion. But due to the subtlety of Nash-Williams' construction, the resulting instance of bar recursion is extremely complex, leading to an extracted term whose *operational* behaviour as a program is somewhat obscure, to say the least! While after a certain amount of effort I began to see what the underlying program did, this was still very difficult to describe, and I doubt that anyone

who has read [21] will have gained any fresh insight into the computational meaning of Higman's lemma.

I believe that the shortcomings of this paper were partly due to my own inexperience at the time, and partly due to the fact that the basic technology for extracting programs from proofs is largely unchanged since its introduction over half a century ago. While admittedly a range of refinements have been developed, and proof mining in particular has produced a number of extremely powerful metatheorems which guarantee the extractability of low-complexity programs from proofs in specific areas of analysis, these do not really help us when comes to non-constructive proofs in well quasi-order theory which rely in an essential way on dependent choice.

In the years that followed I ended up thinking about much more general problems which were prompted from my analysis of Higman's lemma. In particular, I studied forms of higher-order recursion closely related to Nash-Williams' construction [22], and tried to develop notation systems which allow one to describe extracted programs in a more intuitive way [23]. And with the announcement of this book I felt that there was an opportunity for me to expand my original work in light of these developments, and present some new results in this direction.

As I have already emphasised, the 'official' goal of building a program Φ which witnesses Higman's lemma over boolean alphabets is nothing more than an organisational device. Indeed, this is by no means the first place in which such a program has been presented, and I reiterate that real content of this essay lies in the methods which we use to obtain it, and the series of theoretical results presented in Sections 7 and 8, particularly Theorems 7.8 and 8.3, which I publish here for the first time.

Much of the technical groundwork I will present here has been done elsewhere, and this allows me to adopt a lighter style of presentation, in which my priority will be to stress the key points and skim over the heavier details. I have nevertheless tried to keep everything as self-contained as possible. So, for example, the reader not familiar with Gödel's functional interpretation will be given the main definition and plenty of intuition on what it means, and should be able to follow later sections without too much confusion, although whenever a key concept is introduced I take care to include references to introductory material in which a more extensive presentation is given.

On the other hand, the expert reader may wish to skip straight ahead to later sections in which the main technical contributions are presented, and so I have indicated whenever a section is comprised mostly of background material.

In the area of program extraction, it is not uncommon to see technical achievements presented with few examples to illustrate them, and concrete case studies which give little insight into the underlying techniques on which they are based (and I have certainly been guilty of both of these at one point or another!). But my aim here is to endeavour to strike a balance between both theory and practice, and as a result I hope that this article will form a pleasant read for both specialists in proof theory as well as those with a more general interest in well quasi-orders.

2   Well quasi-orders and Zorn's lemma

Let's begin at the beginning, with the definition of a well quasi-order. There are numerous equivalent formulations of this concept - one of the simplest and most widely seen is the following:

Definition 2.1. A quasi-order $(X, \preceq)$ is a set X equipped with a binary relation $\preceq$ which is reflexive and transitive. It is a well quasi-order (or WQO) if it satisfies the additional property that for any infinite sequence of elements $x_0, x_1, x_2, \ldots$ there exists some $i < j$ such that $x_i \preceq x_j$.

It is not difficult to see that a quasi-order is a WQO iff it contains no infinite strictly decreasing chains and no infinite sequences or pairwise incomparable elements. Therefore being a WQO is a strictly stronger property than being well-founded. For example, the quasi-order $(\mathbb{N}, \mid)$ of natural numbers ordered by divisibility is well-founded, but not a WQO. The following is perhaps slightly less obvious:

Lemma 2.2. *A quasi-order $(X, \preceq)$ is a WQO iff any infinite sequence $x_0, x_1, x_2, \ldots$ contains an infinite increasing subsequence $x_{g(0)} \preceq x_{g(1)} \preceq x_{g(2)} \preceq \ldots$ (where $g(0) < g(1) < \ldots$).*

Proof. For the non-trivial direction, let $(X, \leq)$ be a WQO, and take some infinite sequence $x_0, x_1, \ldots$. Define $T \subseteq \mathbb{N}$ by $T := \{i \in \mathbb{N} \mid (\forall j > i) \neg (x_i \leq x_j)\}$. Then T must be finite, otherwise we would be able to construct a sequence contradicting the assumption that X is a WQO. Therefore there is some $N \in \mathbb{N}$ such that for all $i \geq N$ there exists some $j > i$ with $x_i \leq x_j$, which allows us to construct our infinite increasing subsequence. $\qquad\square$

Given some mathematical property, such as being well quasi-ordered, we are often interested in identifying constructions which preserve that property. WQO theory is particularly rich in such results. A simple example is the following:

Proposition 2.3. *If $(X, \leq_X)$ and $(Y, \leq_Y)$ are WQOs, then so is their cartesian product $(X \times Y, \leq_{X \times Y})$ under the pointwise ordering.*

Proof. Given an infinite sequence $\langle x_0, y_0 \rangle, \langle x_1, y_1 \rangle, \ldots$, consider the first component $x_0, x_1, \ldots$. Since X is a WQO, by Lemma 2.2 there exists an infinite increasing sequence $x_{g(0)} \leq_X x_{g(1)} \leq_X \ldots$. Now consider the sequence $y_{g(0)}, y_{g(1)}, \ldots$. Since Y is a WQO, there exists some $i < j$ with $y_{g(i)} \leq_Y y_{g(j)}$. But by transitivity we also have $x_{g(i)} \leq x_{g(j)}$, and therefore $\langle x_{g(i)}, y_{g(i)} \rangle \leq_{X \times Y} \langle x_{g(j)}, y_{g(j)} \rangle$. $\qquad\square$

A far more subtle result, which forms the basis of this article, is the following theorem, widely known as *Higman's lemma*:

Theorem 2.4 (Higman's lemma [11]). *If $(X, \leq)$ is a WQO, then so is $(X^*, \leq_*)$, the set of finite sequences over X ordered under the embeddability relation, where $[x_0, \ldots, x_{m-1}] \leq_* [y_0, \ldots, y_{n-1}]$ whenever there is a strictly increasing map f with $x_i \leq y_{f(i)}$ for all $i < m$.*

A short and extremely elegant proof of Higman's lemma was given by Nash-Williams, using the so-called minimal bad sequence construction, which is a central topic of our paper.

Proof of Higman's lemma [19]. Suppose for contradiction that X is a WQO but that there exists an infinite sequence of words $u_0, u_1, \ldots$ such that $\neg (u_i \leq_* u_j)$ for all $i < j$. We call such a sequence a 'bad sequence'. Now, using the axiom of dependent choice, pick a *minimal* bad sequence $v_0, v_1, \ldots$ as follows:

> Given that we have already constructed $[v_0, \ldots, v_{k-1}]$, define v_k to be such that $[v_0, \ldots, v_{k-1}, v_k]$ extends to some infinite bad sequence, but $[v_0, \ldots, v_{k-1}, a]$ does not for any $a \lhd v_k$, by which mean any strict prefix a of v_k.

Note that such a v_k exists by the minimum principle over the wellfounded prefix relation $\lhd$, together with the fact that $[v_0, \ldots, v_{k-1}]$ must extend to *some* bad sequence: For $k = 0$ this follows from our assumption that least one bad sequence exists, while for $k > 0$ it is true by construction.

Now, the crucial point is that this minimal sequence must itself be bad: If instead there were some $i < j$ with $v_i \leq_* v_j$, then $[v_0, \ldots, v_j]$ could not extend to a bad sequence, contradicting our construction. Therefore in particular each v_n must be non-empty, otherwise we would trivially have $v_n = [] \leq_* v_{n+1}$. This means that each v_n must be a concatenation of the form $\tilde{v}_n * \bar{v}_n$ where $\tilde{v}_n \in X^*$ and $\bar{v}_n \in X$. By Lemma 2.2 the sequence $\bar{v}_0, \bar{v}_1, \ldots$ contains some increasing subsequence $\bar{v}_{g0} \leq \bar{v}_{g1} \leq \ldots$, so let's now consider the sequence

$$w := v_0, \ldots, v_{g0-1}, \tilde{v}_{g0}, \tilde{v}_{g0+1}, \tilde{v}_{g0+2}, \ldots$$

Since $\tilde{v}_{g0} \lhd v_{g0}$, by minimality of v the sequence w must be good, which means that $w_i \leq_* w_j$ for some $i < j$. There are three possibilities: First $j < g0$ and so $v_i = w_i \leq_* w_j = v_j$, second $i < g0$ and $j = gj'$ and so $v_i \leq_* \tilde{v}_{gj'}$ which implies that $v_i \leq_* v_{gj'}$ since $\tilde{v}_{gj'} \lhd v_{gj'}$, and finally $i, j = gi', gj'$ and so $\tilde{v}_{gi'} \leq_* \tilde{v}_{gj'}$ which implies that $v_{gi'} \leq_* v_{gj'}$ since $\bar{v}_{gi'} \leq \bar{v}_{gj'}$. In all cases we have $v_i \leq_* v_j$, contradicting the fact that v is bad. Hence our original assumption was false, and we can conclude that there are no bad sequence, or equivalently that X^* is a WQO. $\qquad\square$

As an immediate consequence of Higman's lemma, we see that our main problem can, in theory, be solved:

4

Corollary 2.5. *Given an infinite sequence u of words over a two letter alphabet $\{0, 1\}$, there exists a pair of indices $i < j$ such that u_i is embedded in u_j.*

Proof. The set $(\{0, 1\}, =)$ trivially a WQO, therefore by Higman's lemma so is $(\{0, 1\}^*, =_*)$. $\qquad\square$

2.1 The minimal bad sequence construction and Zorn's lemma

The existence of a minimal bad sequence in Nash-Williams' proof of Higman's lemma can be viewed in a much broader context as a particular instance of Zorn's lemma, or equivalently, as an inductive principle over chain-complete partial orders. This was first observed by Raoult [24], and since it informs our approach to program extraction, we will explain in a little more detail what is meant by this.

Suppose that $(Y, \sqsupseteq)$ is a chain-complete partial order, where for each non-empty chain γ in Y we fix some lower bound $\bigwedge \gamma$, which is usually taken to be the greatest lower bound if it exists (note that the fact that we talk about lower rather than upper bounds is purely cosmetic, as it sounds slightly more natural when generalising the notion of a *minimal* bad sequence). The following result is essentially just the contrapositive of the principle of open induction discussed in [24]:

Proposition 2.6. *Let B be a predicate on Y which satisfies the property that for any* non-empty *chain γ,*

$$(\forall x \in \gamma)B(x) \to B\left(\bigwedge \gamma\right). \tag{1}$$

Then whenever $B(x)$ holds for some $x \in Y$, there is some minimal y such that $B(y)$ holds, but $y \sqsupset z \to \neg B(z)$.

Proof. Define $S :\equiv \{x \in Y \mid B(x)\}$. Then whenever $B(x)$ holds for some x, the set S is chain complete: For the empty chain we just take x as a lower bound, while any non-empty chain γ in S we have that $\bigwedge \gamma \in S$ by (1). Therefore by Zorn's lemma S has some minimal element y. $\qquad\square$

Now, consider some set X which comes equipped with a given strict partial order $\lhd$ on X which is wellfounded. Define the lexicographic extension $\lhd_{\text{lex}}$ of $\lhd$ by

$$u \lhd_{\text{lex}} v \quad \text{iff} \quad (\exists n)([u](n) = [v](n) \wedge u_n \lhd v_n)$$

where u and v are infinite sequences of type $X^{\mathbb{N}}$ and $[u](n) := [u_0, \dots, u_{n-1}]$ denotes the initial segment of u of length n. It is easy to show that $\lhd_{\text{lex}}$ is also (strict) a partial order. Note that $\lhd_{\text{lex}}$ is not wellfounded - for example, setting $X := 0, 1$ and defining $0 \lhd 1$ we would have

$$1, 1, 1, \dots \rhd_{\text{lex}} 0, 1, 1, \dots \rhd_{\text{lex}} 0, 0, 1, \dots \rhd_{\text{lex}} \dots$$

However, $\lhd_{\text{lex}}$ is *chain-complete*. In fact, given a chain γ in $(X^{\mathbb{N}}, \rhd_{\text{lex}})$, we can construct its greatest lower bound by defining $v_0 \in X$ to be the minimum with respect to $\rhd$ of the first components of the elements of γ, then $v_1 \in X$ to be the minimum of the second components of all elements $x \in \gamma$ with $x_0 = v_0$, then v_2 to be the minimum of the third components of all elements $x \in \gamma$ with $x_0, x_1 = v_0, v_1$ and so on, and it is not difficult to show that $\bigwedge \gamma := v$ is a greatest lower bound of γ.

Moreover, this greatest lower bound v has the property that for any $n \in \mathbb{N}$, there is some $x_n \in \gamma$ which agrees with v on the first n elements i.e. $[v](n) = [x_n](n)$. This motivates the following definition:

Definition 2.7. A formula $B(u)$ on infinite sequences $u \in X^{\mathbb{N}}$ is piecewise definable, or just piecewise, if it can be expressed in the form $(\forall n)P([u](n))$ for some formula $P(s)$ on finite sequences $s \in X^*$.

Theorem 2.8. *Let $B(u) \equiv (\forall n)P([u](n))$ be a piecewise formula, and suppose that $B(u)$ holds for some u. Then there exists some minimal 'bad' sequence v such that $B(v)$ holds, but $\neg B(w)$ for any $w \lhd_{lex} v$.*

Proof. Take any non-empty chain γ such that $(\forall x \in \gamma)B(x)$, and let $v := \bigwedge \gamma$. We want to show that $B(v)$ holds i.e. $P([v](n))$ holds for all $n \in \mathbb{N}$. But as observed above, for any n there exists some $x_n \in \gamma$ with $[v](n) = [x_n](n)$, and $P([x_n](n))$ follows from $B(x_n)$. Therefore the existence of a minimal bad sequence v follows directly from Proposition 2.6. $\qquad\square$

5

Theorem 2.8 is nothing more than a generalisation of the minimal-bad-sequence construction in Nash-Williams' proof of Higman's lemma: The predicate 'u is bad' can be expressed as $B(u) :\equiv (\forall n)(\forall i < j < n)\neg(u_i \leq_* u_j)$ which is clearly a piecewise formula, and so the existence of a minimal bad sequence follows as a special case of the instance of Zorn's lemma given in Proposition 2.6, where $Y := (X^*)^{\mathbb{N}}$ and $\sqsupset$ is taken to be $\triangleright_{\text{lex}}$ over the lexicographic extension of the prefix order.

2.2 Zorn's lemma as an axiom

The reason for the short digression above is to encourage the reader to think of the minimal bad sequence construction, not as a derived result which follows from dependent choice, but as an axiomatic minimum principle over the chain-complete partial order $((X^*)^{\mathbb{N}}, \triangleright_{\text{lex}})$ which can be considered a weak form of Zorn's lemma, namely

$$(\exists u)B(u) \to (\exists v)(B(v) \land (\forall w \triangleleft_{\text{lex}} v)\neg B(w)), \tag{2}$$

where $B(u)$ ranges over *piecewise* formulas. Note that in this case, the premise of Zorn's lemma, namely chain-completeness of $S = \{u \in (X^*)^{\mathbb{N}} \mid B(u)\}$, is encoded by both the premise $(\exists u)B(u)$ and the assumption that B is piecewise.

While this slight shift of emphasis from dependent choice to Zorn's lemma might not seem significant from an ordinary mathematical perspective, it completely alters the way in which we give a computational interpretation to Nash-Williams' proof of Higman's lemma, as this depends entirely on the manner in which we choose to formalise that proof. We will discuss proof interpretations and program extraction in much more detail in Section 4 below, but since the difference between dependent choice and above formulation of Zorn's lemma motivates our formal proof in the next section, it is important to at least roughly explain why this distinction matters to us here.

The extraction of programs from proofs typically works by assigning basic programs to the axioms and rules of some mathematical theory, and then constructing general programs recursively over the structure of formal proofs. Thus the programs which interpret the axioms of our theory form our basic building blocks, and the overall size and complexity of an extracted program in terms of these blocks reflects the size and complexity of the formal proof from which it was obtained.

In the early days of proof theory, when proofs interpretations were primarily used to obtain relative consistency proofs, 'extracted programs' were nothing more than hypothetical objects which gave a computational interpretation to falsity, whose existence within some formal calculus was necessary but whose structure as programs was uninteresting and irrelevant. As such, it was sensible to work in a minimal axiomatic theory which was easy to reason about on a meta-level, but not necessarily convenient for extracting programs in practice. This was the approach taken by Spector [27], who extended Gödel's consistency proof to full mathematical analysis by showing that countable dependent choice could be interpreted by the scheme of *bar recursion* in all finite types, a form of recursion which, while elegant, can be rather abstruse when it comes to understanding its operational semantics as part of a real program.

For us, on the other hand, a proof interpretation is a *tool* for extracting an *actual* program from the proof of Higman's lemma whose algorithmic behaviour can be understood to some extent, as opposed to some obscure syntactical object which essentially acts as a black box. As a result, we want to work in a axiomatic system in which Nash-Williams' minimal bad sequence construction can be cleanly and concisely formalised. So it is natural to ask whether, to this end, we can give a more *direct* proof of Nash-Williams' construction which circumvents the use of bar recursion, and leads to a more intuitive extracted program.

Our idea will be the following: Instead of taking dependent choice as a basic axiom and using this to prove the existence of a minimal-bad-sequence, we will instead take (2) as a basic axiom, from which the existence of minimal-bad-sequences follows trivially. As a result, though, we will no longer be able to rely on Spector's computational interpretation of dependent choice, and will have to instead construct a new realizer for (2).

It is now perhaps becoming clear to the reader why the three deeper questions outlined in the introduction emerge naturally from Higman's lemma! The construction of our direct realizer for the principle (2) carried out in Sections 6-8 offers a partial solution to Question 1 for the particular instance of Zorn's lemma used

dependent choice   Zorn's lemma

(indirect)   *(direct)*

bar recursion ⟶ extracted program
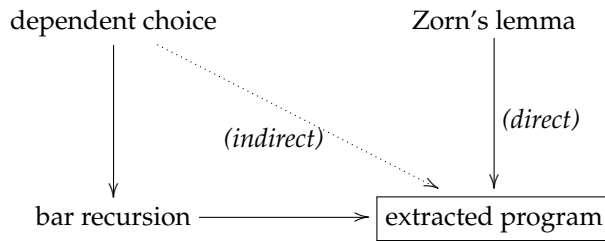
here. Section 7 will focus specifically on the special variant of recursion over $\rhd_{\mathrm{lex}}$ that will be required in order to do all this, and will therefore in turn address Question 2. Question 3 is something that will be on our minds throughout the paper, and in particular influences our description of the realizing term in terms of *learning procedures*.

Before we go on, it is important to observe that the idea of replacing dependent choice with some variant of Zorn's lemma has already been considered by Berger in the setting of modified realizability, in which a variant of Raoult's principle of open induction was given a direct realizability interpretation by a form of open recursion [4]. Here we will give an analogous interpretation for the *functional interpretation* of a principle classically equivalent open induction, and our work here differs considerably from the realizability setting in a number of crucial respects, all of which we make clear later.

3   A formal proof of Higman's lemma

As I highlighted above, in order to apply a proof interpretation to a proof, we first need to have some kind of formal representation of this proof in mind. The route from 'textbook' to formal proof is no mere preprocessing step - the structure and hence usefulness of our extracted program is entirely dependent on the way in which we make precise the logical steps encoded by our textbook proof. The power of applied proof theory is due to the fact that the careful analysis of logical subtleties in formal proofs can reveal quantitative information that is not apparent from an ordinary mathematical perspective, hence the frequent characterisation of this information as being 'hidden' in the proof. In our case, as emphasised already, the fact that we will formalise Nash-Williams' proof of Higman's lemma using an axiomatic form of Zorn's lemma is absolutely crucial to our approach.

When it comes to the application of proof interpretations, one encounters two rather distinct styles in the literature. In proof mining as conceived by Kohlenbach [13], the formal analysis of a proof is typically done 'by hand'. Here, a proof interpretation is simply a means to an end (typically a numerical bound on e.g. a rate of convergence), and as such plays the role of a tool to be wielded by a mathematician. In contrast, for automated program extraction in proof assistants such as Minlog [1], a proof interpretation forms a high level description of a procedure which has to be implemented, and programs are then extracted synthetically at the push of a button. In this case it goes without saying that the user must provide as input a full machine checkable formal proof, written within the confines of some predetermined logical system.

In this article, we take a somewhat mixed approach. On the one hand, this is not a paper on formalised mathematics (and I am certainly not a specialist in this area!): We are interested in a range of rather broad theoretical issues which we intend to present through focusing on the key features of Nash-Williams' proof, and in this sense our construction of a realizing term is based on the pen-and-paper style familiar in proof mining. On the other hand, the novelties of our approach are useful partially because they could be automated within a proof assistant, and so throughout we present our construction in a semi-formal manner in the hope that the reader clearly sees that the main ideas could be implemented at some point.

3.1 The logical system

The main logical theory in which we work will be the theory PA$^\omega$ of Peano arithmetic in all finite types. Here, we define the finite types to include the base types $\mathbb{B}$ and $\mathbb{N}$ for booleans and natural numbers respectively, and allow the construction of product $X \times Y$, finite sequence X^* and function types $X \to Y$. The theory PA$^\omega$ is just the usual theory of Peano arithmetic, but with variables and quantifiers for objects of any type. We write $x : X$ or x^X to denote that x has type X. Equality symbols $=_\mathbb{B}$ and $=_\mathbb{N}$ for base types are taken as primitive, whereas equality for other types is defined inductively terms of these, so for example $s =_{X \to Y} t :\equiv (\forall x)(sx =_Y tx)$. There are various ways of treating extensionality: for reasons which we will not go into here, the functional interpretation does not interpret the axiom of extensionality - only a weak rule form - and so if the reader prefers they can take PA$^\omega$ to be the weakly-extensional variant WE-PA$^\omega$ defined in e.g. [13, 29], although it should be stressed that extensionality is only an issue for the interpreted theory, and when it comes to *verifying* our extracted programs in later sections we freely make use of full extensionality. In any case, the exact details of the logical system are not important in this paper, as our extraction of a program is not fully formal.

What *is* important is the way in which we extend our base theory in order to deal with the minimal bad sequence argument. First note that when reasoning in higher types it is essential to be able to add to our base theory the very weak axiom of choice for quantifier-free formulas:

$$\text{QF-AC} : \ (\forall x^X)(\exists y^Y)A_0(x,y) \to (\exists f^{X \to Y})(\forall x)A_0(x,f(x)).$$

which as we will see is completely harmless from a computational point of view. In contrast, the crucial additional axiom we choose in order to formalise Nash-Williams' argument is the following syntactic variant of Theorem 2.8 already stated as (2), which we interpret as an axiom schema labelled ZL$_{\text{lex}}$,

$$\text{ZL}_{\text{lex}} : \ (\exists u^{\mathbb{N} \to X})B(u) \to (\exists v)(B(v) \wedge (\forall w \lhd_{\text{lex}} v)\neg B(w))$$

where $B(u)$ is understood to range over all *piecewise* formulas of the form $(\forall n)P([u](n))$, and $\lhd$ is some primitive recursive relation on X, transfinite induction over which is provable in PA$^\omega$. Of course, technically we should include this additional wellfounded assumption as a premise so that ZL$_{\text{lex}}$ becomes a proper axiom schema, but we will omit it for simplicity and if the reader prefers they can just imagine that ZL$_{\text{lex}}$ is defined relative to some arbitrary but fixed $(X, \rhd)$. Both here and in Chapter 9, X will actually be of the form X^* and $\rhd$ will be nothing more than the prefix relation on finite sequences which is trivially wellfounded.

Note that the contrapositive of ZL$_{\text{lex}}$ can be identified with open induction over the lexicographic ordering as treated in [4]:

$$\text{OI}_{\text{lex}} : \ (\forall v)((\forall w \lhd_{\text{lex}} v)U(w) \to U(v)) \to (\forall u)U(u)$$

where now $U(u)$ must be an open formula of the form $(\exists n)P([u](n))$ (and so in our terminology, being piecewise is the negation of being open). In the realizability setting of [4], there is a genuine difference between ZL$_{\text{lex}}$ and OI$_{\text{lex}}$: the latter is an intuitionistic principle which can be given a direct computational interpretation via open recursion, whereas the former is a non-constructive principle which cannot be realized without the use of e.g. the A-translation. On the other hand, for the functional interpretation combined with the negative translation, both are ZL$_{\text{lex}}$ and OI$_{\text{lex}}$ are interpreted by exactly the same term (informally, this is due to the fact that the functional interpretation of implication is much more intricate than that of realizability), so they are essentially interchangeable here. We choose ZL$_{\text{lex}}$ as primitive, because in our opinion the realizing term is a little more intuitive when viewed as an approximation to a minimal element v. In any case we will discuss these nuances later. For now, we proceed straight to the formal proof.

3.2 The formal proof

Suppose that X is some arbitrary finite type which comes equipped with some quasi-order $\preceq$, which we take to mean some primitive recursive function $t : X \times X \to \mathbb{B}$ for which reflexivity and transitivity are provable

in PA$^\omega$. We now introduce two predicates which represent the two equivalent definitions of a WQO which we required in Section 2:

$$\mathsf{WQO}(\preceq) :\equiv (\forall x^{\mathbb{N}\to X})(\exists i < j)(x_i \preceq x_j)$$

$$\mathsf{WQO}_{\mathrm{seq}}(\preceq) :\equiv (\forall x^{\mathbb{N}\to X})(\exists g^{\mathbb{N}\to\mathbb{N}})(\forall i < j)(g(i) < g(j) \wedge x_{g(i)} \preceq x_{g(j)}).$$

Now, given $\preceq$ we can formally define the embeddability relation $\preceq_*$ as a primitive recursive function in $\preceq$, as in order to check that $a \preceq_* b$ we simply need to carry out a bounded search over all increasing functions $\{0,\dots,|a|-1\} \to \{0,\dots,|b|-1\}$, where $|a|$ denotes the length of a. Note that reflexivity and transitivity of $\preceq_*$ is easily provable from that of $\preceq$ in PA$^\omega$. The main result of this section is the following:

Theorem 3.1. *For any fixed quasi-order $\preceq$ on X we have* PA$^\omega$ + QF-AC + ZL$_{\mathrm{lex}}$ $\vdash$ $\mathsf{WQO}_{\mathrm{seq}}(\preceq) \to \mathsf{WQO}(\preceq_*)$.

The basic strategy of Nash-Williams' proof is first to construct a hypothetical minimal bad sequence, then to deal with a number of simple but quite fiddly cases in order to derive a contradiction. It will be greatly helpful to us in later sections if we separate these two parts here, and prove the following numerically explicit form of the latter:

Lemma 3.2. *Given a sequence $v : (X^*)^{\mathbb{N}}$, define two sequences $\tilde{v} : (X^*)^{\mathbb{N}}$ and $\bar{v} : X^{\mathbb{N}}$ from v as follows:*

$$\tilde{v}_n, \bar{v}_n := \begin{cases} [], 0_X & \text{if } v_n = [] \\ [x_1,\dots,x_{k-1}], x_k & \text{if } v_n = [x_1,\dots,x_k] \end{cases}$$

where 0_X denotes some canonical element of type X. Given in addition some function $g : \mathbb{N} \to \mathbb{N}$, define the sequence $w : (X^*)^{\mathbb{N}}$ by

$$w_n := \begin{cases} v_n & \text{if } n < g(0) \\ \tilde{v}_{g(i)} & \text{if } n = g(0) + i. \end{cases}$$

Suppose that there exists some $k : \mathbb{N}$ such that

$$w_{g(0)} \triangleleft v_{g(0)} \to (\exists i < j < k)(w_i \preceq_* w_j) \tag{3}$$

where $\triangleleft$ denotes the strict prefix relation on words, and that g satisfies

$$(\forall i < j \leq k)(g(i) < g(j) \wedge \bar{v}_{g(i)} \preceq \bar{v}_{g(j)}). \tag{4}$$

Then there exists a pair of indices $i < j < g(k) + 2$ such that $v_i \preceq_* v_j$.

Proof. This is a simple case distinction that we prove in excruciating detail. First of all, we note that by induction on (4) it follows that $i \leq g(i)$ for all $i < k$, which we use below. There are two main cases: A degenerate one where $v_{g(0)} = []$, in which case $v_{g(0)} \preceq_* v_{g(0)+1}$ and so we can set $i, j = g(0), g(0) + 1 < g(0) + 2 \leq g(k) + 2$. For the non-degenerate case where $v_{g(0)} \neq []$ then we have $w_{g(0)} = \tilde{v}_{g(0)} \triangleleft v_{g(0)}$ and hence $w_i \preceq_* w_j$ for some $i < j < k$ by (3). There are three further possibilities:

(i) $i < j < g(0)$: Then $v_i = w_i \preceq_* w_j = v_j$ and $j < k \leq g(k) < g(k) + 2$.

(ii) $i < g(0) \leq j$: Then $v_i = w_i \preceq_* w_j = \tilde{v}_{g(j')}$ where $j = g(0) + j'$. Either $v_{g(j')} = []$ and so trivially $v_{g(j')} \preceq_* v_{g(j')+1}$, with $j' \leq j$ and hence $g(j')+1 \leq g(j)+1 < g(k)+2$, or $\tilde{v}_{g(j')} \triangleleft v_{g(j')}$ and therefore $v_i \preceq_* v_{g(j')}$ with $i < g(0) \leq g(j') < g(k)$.

(iii) $g(0) \leq i < j$: Then $\tilde{v}_{g(i')} = w_i \preceq_* w_j = \tilde{v}_{g(j')}$ with $i = g(0) + i', j = g(0) + j'$. If either $\tilde{v}_{g(i')} = []$ or $\tilde{v}_{g(j')} = []$ then the result follows exactly as in part (ii), and otherwise we have $v_{g(i')} = \tilde{v}_{g(i')} * \bar{v}_{g(i')}$ and $v_{g(j')} = \tilde{v}_{g(j')} * \bar{v}_{g(j')}$ and since $i' < j' \leq j < k$ it follows by (4) that $g(i') < g(j')$ and $\bar{v}_{g(i')} \preceq \bar{v}_{g(j')}$ and hence $v_{g(i')} \preceq_* v_{g(j')}$, and since $g(j') \leq g(j) < g(k)$ we're done.

9

In all cases we have found some $i'' < j'' < g(k) + 2$ with $v_{i''} \leq_* v_{j''}$. □

Proof of Theorem 3.1. First of all, let $\vartriangleleft$ denote the strict prefix relation on words, so that $a \vartriangleleft b$ iff $|a| < |b|$ and $(\forall i < |a|)(a_i = b_i)$. This is clearly wellfounded, and we can assume for argument's sake that it is decidable, which is automatically the case when X is a base type. Now, define the piecewise predicate $B(u)$ on infinite sequences of words by $B(u) :\equiv (\forall n)P([u](n))$, where

$$P(s) :\equiv (\forall i < j < |s|)(s_i \not\leq_* s_j).$$

Suppose that $(\exists u)B(u)$. Then by $\mathrm{ZL}_{\mathrm{lex}}$ there exists some $v : (X^*)^{\mathbb{N}}$ such that

$$(*) \quad B(v) \wedge (\forall w \vartriangleleft_{\mathrm{lex}} v)\neg B(w).$$

Now let $\tilde{v}, \bar{v}$ and w be defined as in Lemma 3.2, where g is the function satisfying

$$(\forall i < j)(g(i) < g(j) \wedge \bar{v}_{g(i)} \leq \bar{v}_{g(j)})$$

which exists by $\mathrm{WQO}_{\mathrm{seq}}(\leq)$. Then (3) holds for some k by minimality of v, since if $w_{g(0)} \vartriangleleft v_{g(0)}$ then $w \vartriangleleft_{\mathrm{lex}} v$, and (4) holds for *any* k, therefore by Lemma 3.2 there exists $i < j$ such that $v_i \leq_* v_j$, contradicting $B(v)$. Therefore $(\exists u)B(u)$ is false, which implies that for all u there exists some $i < j$ such that $u_i \leq_* u_j$, which is $\mathrm{WQO}(\leq_*)$. Hence we have shown that $\mathrm{WQO}_{\mathrm{seq}}(\leq) \to \mathrm{WQO}(\leq_*)$. □

While the derivation above is not fully formal in the sense that would be expected were we to formally extract a program using a proof assistant, in contrast to the textbook proof given in Section 1 it makes explicit important quantitative information which will guide us in constructing a realizing term, as we will see later.

Now to our main problem, which is to prove that $(\{0, 1\}, =_*)$ is a WQO. From now on we will equate the two letter alphabet $\{0, 1\}$ with our type $\mathbb{B}$. Suppose that $\leq$ is now just $=_{\mathbb{B}}$, which is clearly a WQO. In order to be able to apply Higman's lemma, we need to establish $\mathrm{WQO}_{\mathrm{seq}}(=_{\mathbb{B}})$, either by formalising Lemma 2.2 or by a direct argument. We choose the latter, since for the simple case of $=_{\mathbb{B}}$ a single instance of the law of excluded-middle suffices, whereas the general case given as Lemma 2.2 would require some form of comprehension.

Theorem 3.3. $\mathrm{PA}^{\omega} + \mathrm{QF\text{-}AC} \vdash \mathrm{WQO}_{\mathrm{seq}}(=_{\mathbb{B}})$

Proof. We will first show that
$$(\forall x^{\mathbb{N} \to X})(\exists b^{\mathbb{B}})(\forall n)(\exists k \geq n)(x_k = b). \tag{5}$$
Fix some sequence $x : \mathbb{B}^{\mathbb{N}}$. By the law of excluded middle we have

$$(\exists n)(\forall k \geq n)(x_k = 0) \vee (\forall n)(\exists k \geq n)(x_k = 1).$$

If the left hand side of the disjunction holds we set $b := 0$. We have that there is some N such that $x_k = 0$ for all $k \geq N$, and so for an arbitrary number n, setting $k := \max\{N, n\}$ yields $k \geq n$ and $x_k = 0$. If the right hand side holds, we set $b := 1$ and we are done by definition.

So we have proved (5). To establish $\mathrm{WQO}_{\mathrm{seq}}(=_{\mathbb{B}})$, take some x and let b be such that $(\forall n)(\exists k \geq n)(x_k = b)$. By QF-AC there exists some $f : \mathbb{N} \to \mathbb{N}$ satisfying

$$(\forall n)(f(n) \geq n \wedge x_{f(n)} = b).$$

Now, define $g : \mathbb{N} \to \mathbb{N}$ via primitive recursion as

$$g(0) := f(0) \quad \text{and} \quad g(n + 1) := f(g(n) + 1).$$

Then it is clear that $g(n) < g(n + 1)$ and $x_{g(n)} = b$, and therefore

$$(\forall i < j)(g(i) < g(j) \wedge x_{g(i)} = b = x_{g(j)})$$

and we're done. □

10

Theorem 3.3

$$
\cfrac{
  \begin{array}{c}
  \vdots \\
  \mathrm{WQO_{seq}}(=_{\mathbb{B}})
  \end{array}
  \qquad
  \cfrac{
    \cfrac{(\exists u)B(u)}{(\exists v)(B(v) \wedge (\forall w \lhd_{\mathrm{lex}} v)\neg B(w))}\ \mathrm{ZL_{lex}}
  }{}
}{
  \cfrac{\bot}{\mathrm{WQO}(=_{\mathbb{B},*})}
}\ \mathrm{Lemma\ 3.2}
$$

Figure 1: A map of the formal proof

Now, putting together Theorems 3.1 and 3.3, we have:

Corollary 3.4. $\mathrm{PA}^{\omega} + \mathrm{QF\text{-}AC} + \mathrm{ZL_{lex}} \vdash \mathrm{WQO}(=_{\mathbb{B},*})$.

We summarise the main structure of our proof of $\mathrm{WQO}(=_{\mathbb{B},*})$ in Figure 1. There are three main parts to the proof, each of which will be treated somewhat separately in what follows, namely:

(1) A proof of $\mathrm{WQO_{seq}}(=_{\mathbb{B}})$ given as Theorem 3.3, which uses an instance of the law of excluded middle for Π_2^0 formulas.

(2) A single instance of $\mathrm{ZL_{lex}}$ applied to the piecewise formula $(\forall i < j)(u_i \not\preceq_* u_j)$, set out in the main proof of Theorem 3.1.

(3) The derivation of a contradiction from $\mathrm{WQO_{seq}}(\preceq)$ combined with the existence of a minimal bad sequence, which is Lemma 3.2.

Having now introduced the theory of WQOs and given a formal proof of our main result, the remainder of the paper will be dedicated to constructing a program which finds an embedded pair of words in an arbitrary input sequence. We will introduce our main tool - Gödel's functional interpretation - in the next section, then each of the three main components will be analysed in turn in Sections 5, 6-8 and 9, respectively.

4 Gödel's functional interpretation

We now put well quasi-orders aside for a moment, and introduce the second main topic of this paper: Gödel's functional (or 'Dialectica') interpretation. The reader already familiar with this may wish to skip ahead to Section 6, which continues with the computational interpretation of $\mathrm{ZL_{lex}}$.

Presenting the functional interpretation is something of a challenge for an author: The interpretation is one of those syntactical objects - particularly common in proof theory - whose basic definition can be given in a few lines and whose characterising theorem (in this case soundness) can be set up in a couple of pages, and yet none of this is necessarily remotely helpful in giving the unacquainted reader any real insight into what it actually does! In reality, the functional interpretation is an extraordinarily subtle idea which continues to be studied from a range of perspectives, and the fact that it forms one of the central techniques of the highly successful proof mining program is testament to its power. For a comprehensive treatment of the functional interpretation and its role in program extraction, the reader is encouraged to consult the standard textbook [13], or alternatively the shorter chapter [3].

Nevertheless, in an effort to make this essay as accessible as possible it is important that I say something about the interpretation here. So my plan is as follows: in Sections 4.1-4.3 below I will begin by defining the interpretation, and will state without proof the main results on program extraction. This will all be standard material. Then in Section 4.4 I will employ the slightly unconventional tactic of explaining on a high level how the functional interpretation treats a series of formulas of a specific logical shape, which appear several times in the remainder of this work. Finally, in Section 5, I will present in quite some detail the extraction of a simple program from the proof of Theorem 3.3, which will conveniently serve simultaneously as a illustration of the functional interpretation in action and the first step in our main challenge!

4.1 The basics

In one sentence, Gödel's functional interpretation is a syntactic translation which takes as input a formula A in some logical theory $\mathcal{L}$ and returns a new formula $A' := (\exists x)(\forall y)|A|_y^x$ where x and y are sequences of potentially higher type variables, and $|A|_y^x$ is in some sense computationally neutral, which in this article will just mean quantifier-free and hence decidable (since characteristic functions for all quantifier-free formulas can be constructed in PA^ω). The idea behind the translation is that $A \leftrightarrow A'$ over some reasonable higher-type theory, but the latter can be witnessed by some term in a calculus T. We say that the functional interpretation soundly interprets $\mathcal{L}$ in T, if for any formula in the language of $\mathcal{L}$ we have

$$\mathcal{L} \vdash A \Rightarrow \text{ there exists some closed term } t \text{ of } T \text{ such that } \mathcal{T} \vdash |A|_y^t$$

where $\mathcal{T}$ represents some verifying theory which allows us to reason about terms in our calculus T. Crucially, the soundness proof comes equipped with a method of constructing such a realizer t from the proof of A. The direct approach above typically works for *intuitionistic* theories $\mathcal{L}$ extended with some weak semi-classical axioms (for example Markov's principle), but for theories $\mathcal{L}_c$ based on full classical logic, we need to precompose the functional interpretation with a negative translation $A \mapsto A^{\neg\neg}$. Therefore from now on, soundness of the functional interpretation for classical theories refers to the following:

$$\mathcal{L}_c \vdash A \Rightarrow \text{ there exists some closed term } t \text{ of } T \text{ such that } \mathcal{T} \vdash |A^{\neg\neg}|_y^t.$$

In this article, our $\mathcal{L}_c$ will be $\mathrm{PA}^\omega + \mathrm{QF\text{-}AC}$, later extended with $\mathrm{ZL}_{\mathrm{lex}}$. But before we go further, we need to introduce our functional calculus T.

4.2 The programming language

Our interpreting calculus will be a standard variant of Gödel's system T, extended with product and finite sequence types as with our variant of PA^ω. System T is well-known enough that we feel no need to give a proper definition here: in any case full details can be found in many places, including the aforementioned sources [3, 13, 29]. In a sentence: System T is a simply typed lambda calculus which allows the definition of functionals via primitive recursion in all higher types. We summarise the basic constructions of the calculus below, if only to allow the reader to become familiar with our notational conventions. We take the types of T to be the same as those in our logical system PA^ω, namely those build from $\mathbb{B}$ and $\mathbb{N}$ via product, sequence and arrow types. Terms of the calculus include the following:

- **Functions.** We allow the construction of terms via lambda abstraction and application: if $x : X$ and $t : Y$ then $\lambda x.t : X \to Y$, while if $t : X \to Y$ and $s : X$ then $ts : Y$, and these satisfy the usual axioms, e.g. $(\lambda x.t[x])(s) = t[x\backslash s]$.

- **Canonical objects.** For each type X we define a canonical 'zero object' $0_X : X$ in the standard manner, with $0_\mathbb{N} = 0$, $0_\mathbb{B} = 0$, $0_\mathbf{1} = ()$, $0_{X \times Y} = \langle 0_X, 0_Y \rangle$, $0_{X^*} = []$ and $0_{X \to Y} = \lambda x.0_Y$.

- **Products.** Given $z : X \times Y$ we often write just z_0, z_1 for the projections $\pi_0 z : X$, $\pi_1 z \in Y$. This will also be the case for sequences, where for $z : (X \times Y)^\mathbb{N}$, $z_0 : X^\mathbb{N}$ is defined by $(z_0)_n := \pi_0 z_n$ and so on. For $x : X$ and $y : Y$ we have a pairing operator $\langle x, y \rangle : X \times Y$.

- **Sequences.** As before, given $s : X^*$ we denote by $|s|$ the length of s, for $x : X$ we define $s * x : X^*$ by $[s_0, \ldots, s_{k-1}, x]$ i.e. the concatenation of s with x, and use this also for the concatenation of s with another finite sequence $s * t : X^\mathbb{N}$ or an infinite sequence $s * \alpha : X^\mathbb{N}$. For $\alpha : X^\mathbb{N}$ we let $[\alpha](n) := [\alpha_0, \ldots, \alpha_{n-1}]$.

- **Recursors.** For each type we have a recursor Rec_X which has the defining equations

$$\mathrm{Rec}_X^{a,h}(0) =_X a \quad \text{and} \quad \mathrm{Rec}_X^{a,h}(n+1) =_X hn(\mathrm{Rec}^{a,h}(n)).$$

  for parameters $a : X$ and $h : \mathbb{N} \to X \to X$.

Note that having access to recursors of arbitrary finite type means that along with all normal primitive recursive functions we can define e.g. the Ackermann function (using $\mathrm{Rec}_{\mathbb{N}\to\mathbb{N}}$). Indeed, the closed terms of type $\mathbb{N}\to\mathbb{N}$ definable in T are the provably recursive functions of Peano arithmetic, a fact which follows from the soundness of the functional interpretation.

There are a couple of further remarks to be made. First, we have presented system T as a equational calculus, but of course we could have instead used a conversion rule $\to_T$, in which case system T can be viewed as a fragment of PCF consisting only of total objects. More concretely, any term of system T can be straightforwardly written as a functional program, and we encourage the reader to think of system T in this manner, as a high level means of describing real programs.

Finally, in the previous section we wrote $\mathcal{T} \vdash |A|_y^t$, which implies that T also comes equipped with a logic $\mathcal{T}$ for verifying programs. There are various ways of defining the underlying logic of system T - traditionally it is presented as a minimal quantifier-free calculus with an induction axiom, although alternatively we can just identify $\mathcal{T}$ with the fully extensional variant of PA^ω, extended with additional axioms whenever we need them. Such distinctions are more relevant for foundational issues such as relative consistency proofs, where it was the goal to make $\mathcal{T}$ as weak as possible. Here we have no such concerns, and so we reason about the correctness of our extracted programs in a fairly free manner.

4.3 The interpretation

The functional interpretation $|A|_y^x$ of a formula A in the language of PA^ω is defined inductively over the logical structure of A as follows:

(i) $|A| :\equiv A$ if A is prime

(ii) $|A \wedge B|_{y,v}^{x,u} :\equiv |A|_y^x \wedge |B|_v^u$

(iii) $|A \vee B|_{y,v}^{b^{\mathbb{B}},x,u} :\equiv |A|_y^x \vee_b |B|_v^u$

(iv) $|A \to B|_{x,v}^{f,g} :\equiv |A|_{gxv}^x \to |B|_v^{fx}$

(v) $|\exists t^X A(t)|_y^{z,x} :\equiv |A(z)|_y^x$

(vi) $|\forall t^X A(t)|_{z,y}^{f} :\equiv |A(z)|_y^{fz}$

where in clause (iii) we define
$$P \vee_b Q :\equiv (b = 0 \to P) \wedge (b = 1 \to Q).$$

At first glance, the functional interpretation looks very much like a standard BHK interpretation, with the exception of the treatment of implication (iv), which is in many ways the characterising feature of the interpretation. Though (iv) may appear to be a little mysterious, it should be viewed as the 'least non-constructive' Skolemization of the formula

$$(\exists x)(\forall y)|A|_y^x \to (\exists u)(\forall v)|B|_v^u$$

which goes via

$$(\forall x)(\exists u)(\forall v)(\exists y)(|A|_y^x \to |B|_v^u)$$

as an intermediate step. In the original 1958 paper [10], Gödel proved that the usual first order theory of Heyting arithmetic could be soundly interpreted in System T. It follows directly that Peano arithmetic can also be interpreted in T when precomposed with the negative translation, and in fact it is not difficult at all to extend these results to the higher-order extensions of arithmetic with quantifier-free choice:

Theorem 4.1. *Let $A(a)$ be a formula in the language of (weakly-extensional)* PA^ω *containing only a free. Then*

$$\mathrm{PA}^\omega + \mathrm{QF\text{-}AC} \vdash A(a) \Rightarrow \mathcal{T} \vdash |A(a)^{\neg\neg}|_y^{ta}$$

where t is a closed term of T which can be formally extracted from the proof of A(a).

A modern presentation and proof of this result can be found in [13], which also discusses the various theories $\mathcal{T}$ in which soundness can be formalised. As simple as Theorem 4.1 appears, understanding how the combination of negative translation and functional interpretation treats even simple logical formulas is far from straightforward, and is often a stumbling block when one first encounters the ideas of applied proof theory. We now try to provide some insight into this.

4.4 The *meaning* of the interpretation

In order that the reader not familiar with the functional interpretation and program extraction can follow the main part of the paper, it is important that we highlight how the functional interpretation combined with the negative translation treats a handful of key formulas. Note that we have not yet stated which negative translation we use. Unless one wants to formalise the translation this is not so important: Typically what we do is take some arbitrary choice of $A^{\neg\neg}$ and use a range of semi-intuitionistic laws which are admitted by the functional interpretation to rearrange it into a simpler formula which is easier to interpret.

4.4.1 Π_2 *formulas* $A :\equiv (\forall x)(\exists y)B(x, y)$

We begin with one of the key properties of the functional interpretation, which makes it so useful for program extraction. The negative translation of a Π_2 formula is equivalent to $(\forall x)\neg\neg(\exists y)B(x, y)$. However, it is not difficult to see that the functional interpretation translates $\neg\neg(\exists y)B(x, y)$ to $(\exists y)B(x, y)$, and so in particular the functional interpretation admits Markov's principle. Therefore

$$(\forall x)\neg\neg(\exists y)B(x, y) \text{ is translated to } (\exists f)(\forall x)B(x, fx)$$

and therefore in theory we can extract a program directly witnessing a Π_2 formula, even when that formula is proven classically. Note that the statement WQO($\preceq$) is a Π_2 formula, and so even though we use a number of non-constructive principles in its proof, we can still hope to extract a program Φ witnessing it!

4.4.2 Σ_2 *formulas* $A :\equiv (\exists x)(\forall y)B(x, y)$

In contrast to Π_2 formulas, Σ_2 formulas are more problematic, as there provable Σ_2 formulas which cannot in general be directly witnessed by a computable function. Here, the negative translation is equivalent to $\neg\neg(\exists x)(\forall y)B(x, y)$, and functional interpretation acts as follows

$$\begin{aligned}
\neg\neg(\exists x)(\forall y)B(x, y) &\mapsto \neg(\exists f)(\forall x)\neg B(x, fx) \\
&\mapsto (\forall f)(\exists x)\neg\neg B(x, fx) \\
&\mapsto (\forall f)(\exists x)B(x, fx) \quad (*) \\
&\mapsto (\exists \Phi)(\forall f)B(\Phi f, f(\Phi f)).
\end{aligned}$$

Note that we can omit double negations in front of $B(x, y)$ as this is a quantifier-free formula. Nevertheless, the interpretation of our original formula gives us something 'indirect', which in this case coincides with Kreisel's well-known 'no-counterexample' interpretation (although in general the functional interpretation is different). The intuitive idea is that f is a function which attempts to witness falsity of A i.e. $(\forall x)(\exists y)\neg B(x, fx)$. Then the functional Φ takes any proposed 'counterexample function' and shows that it must fail. Over classical logic, the existence of such a functional Φ is equivalent to the existence of some x satisfying $(\forall y)B(x, y)$, but unlike x it can be directly constructed.

Another way of understanding the meaning of Φ is as a program which constructs an *approximation* to the non-constructive object x. In general, we cannot compute an x which satisfies $B(x, y)$ for all y, but given some f we can find an x which satisfies $B(x, fx)$. In this setting, f should be seen as a function which calibrates *how good* the approximation should be. We make extensive use of this intuition later, where we explain how the functional interpretation of the minimal bad sequence construction can be viewed as the statement that arbitrarily good 'approximate' minimal bad sequences exist.

We also note that throughout this paper, we will often express the interpretation of Π_2 formulas in its penultimate form $(\forall f)(\exists x)B(x, fx)$ indicated by $(*)$ above. This is for no other reason than that it is much easier to talk about x instead of Φf, and so we avoid a lot of rather messy notation!

4.4.3   Implication $A :\equiv (\exists x)(\forall y)B(x, y) \rightarrow (\exists u)(\forall v)C(u, v)$

Finally, it is important to sketch what happens when we want to infer the existence of a non-constructive object u from the existence of another non-constructive object x. In this case, the negative translation of A is intuitionistically equivalent to

$$(\exists x)(\forall y)B(x, y) \rightarrow \neg\neg(\exists u)(\forall v)C(u, v),$$

Now, by the previous section, the functional interpretation of the conclusion yields

$$(\exists x)(\forall y)B(x, y) \rightarrow (\forall f)(\exists u)C(u, fu).$$

and so interpreting the implication as a whole following clause (iv) we get

$$(\exists F, G)(\forall x, f)(B(x, Gfx) \rightarrow C(Fxf, f(Fxf))).$$

In terms of our discussion above, F and G can be read as follows: For any given x, Fx is a functional which computes an approximation to the conclusion of the implication i.e. $(\forall f)(\exists u)C(u, fu)$, where now it uses that $(\forall y)B(x, y)$ holds. The functional G tells us, in a certain sense, how good the approximation of the *premise* has to be in order to build an approximation of the conclusion: this is given by Gf. Note that whenever we have a functional Φ which builds an approximation to the premise in this way i.e. $B(\Phi(Gf), Gf(\Phi(Gf)))$ we can use it to construct an approximation to the conclusion.

While all this may sound extremely intricate, it will hopefully become clearer when we see some concrete examples in the sequel.

5   Interpreting the proof of $\mathrm{WQO}_{\mathrm{seq}}(=_{\mathbb{B}})$

We now give our first illustration of the functional interpretation in action. In Theorem 3.3 we showed that the proof of $\mathrm{WQO}_{\mathrm{seq}}(=_{\mathbb{B}})$ could be formalised in $\mathrm{PA}^{\omega} + \mathrm{QF\text{-}AC}$, and hence by Theorem 4.1 we know for sure that we can construct a program in T which witnesses its functional interpretation. However, actually doing so, and ending up with a program whose behaviour can be comprehended is another matter, and in what follows we outline the philosophy emphasised later in the paper of combining formal program extraction with intuition. Note that nothing in this section is new, and if the reader prefers they can simply glance at the program we obtain in Section 5.3 and move straight on to Section 6.

The proof of Theorem 3.3 has three main components. The first is an obviously non-constructive axiom, namely the law of excluded middle for Σ_2 formulas applied to $(\exists n)(\forall k \geq n)(x_k = 0)$. The second is the derivation of the auxiliary statement (5) from this instance of the law of excluded-middle, and the final is the derivation of $\mathrm{WQO}_{\mathrm{seq}}(=_{\mathbb{B}})$ from (5) by constructing the necessary primitive recursive function. We will treat each of these in turn. Before we do so, it is worth spelling out explicitly what our goal is. First note that $\mathrm{WQO}_{\mathrm{seq}}(=_{\mathbb{B}})$ can be equivalently formulated as the Π_3 formula

$$(\forall x)(\exists g)(\forall n)(\forall i < j \leq n)(g(i) < g(j) \wedge x_{g(i)} = x_{g(j)}).$$

Note that we have merged the two quantifiers $\forall i, j$ into a single 'real' quantifier $\forall n$ so that they are now bounded and hence decidable. In particular, this means they are now ignored by the functional interpretation. The negative translation of this formulas is equivalent to

$$(\forall x)\neg\neg(\exists g)(\forall n)(\forall i < j \leq n)(g(i) < g(j) \wedge x_{g(i)} = x_{g(j)}).$$

Therefore, referring back to Section 4.4.2 our challenge is to produce a program Φ which takes as input x together with some 'counterexample functional' $\omega : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$ and witnesses $(\exists g)$ in the formula

$$(\forall x, \omega)(\exists g)(\forall i < j \leq \omega g)(g(i) < g(j) \wedge x_{g(i)} = x_{g(j)}). \tag{6}$$

In other words, while we cannot hope to effectively construct a monotone subsequence g in general, we can always do the next best thing and construct an approximation to it which works for all $i < j \leq \omega g$. Then when it comes to using $\mathrm{WQO}_{\mathrm{seq}}(=_{\mathbb{B}})$ as a lemma in the proof of $\mathrm{WQO}(=_{\mathbb{B},*})$ we will need to calibrate exactly how big this approximation needs to be, in other words construct some concrete ω such that $\mathrm{WQO}(=_{\mathbb{B},*})$ follows from (6).

5.1 The law of excluded-middle for Σ_2^0 formulas

Our first step when interpreting a classical proof is to interpret the main non-constructive axioms which are needed. When interpreting $\mathrm{WQO}_{\mathrm{seq}}(\leq) \to \mathrm{WQO}(\prec)$ later, our focus will be on $\mathrm{ZL}_{\mathrm{lex}}$. Here we must deal with the somewhat simpler law of excluded-middle for Σ_2^0 formulas:

$$(\exists n)(\forall k)P(n,k) \vee (\forall m)(\exists l)\neg P(m,l).$$

The functional interpretation Skolemizes this as the Σ_1 formula

$$(\exists b,n,h)(\forall k,m)[P(n,k) \vee_b \neg P(m,hm)]$$

and following Section 4.4.2 the interpretation of the double negation of this formula is given by

$$(\forall \phi, \psi)(\exists b,n,h)[P(n,\phi bnh) \vee_b \neg P(\psi bnh, h(\psi bnh))]. \tag{7}$$

This already looks rather complex thanks to all the function dependencies, but the way to think of ϕ and ψ is again as counterexample functionals which represent the quantifiers $(\forall k)$ and $(\forall m)$ respectively. Now, we have two options in front of us: We can either carefully analyse the formal derivation of the negative translation of the law of excluded-middle in intuitionistic logic in order to produce a realizer for (7), or we can try to do this directly using our intuition. We choose the latter - and it is this kind of thing that characterizes our 'semi-formal' approach to program extraction.

Let's look more closely at (7). Our boolean b is just a marker which tells us which side of the conjunction holds, so essentially what we must do is find a pair n_L, n_R and h_L, h_R which satisfy either $P(n_L, \phi 0 n_L h_L)$ or $\neg P(\psi 1 n_R h_R, h_R(\psi 1 n_R h_R))$. In the first case we can then define b,n,h to be $0, n_L, h_L$, and in the second to be $1, n_R, h_R$. In order to do this, we want to define these so that

$$P(n_L, \phi 0 n_L h_L) \leftrightarrow P(\psi 1 n_R h_R, h_R(\psi 1 n_R h_R))$$

so that $P(n_L, \phi 0 n_L h_L) \vee \neg P(\psi 1 n_R h_R, h_R(\psi 1 n_R h_R))$ follows directly from the law of excluded-middle for quantifier-free formulas. Note that we can force this equivalence to hold if

$$n_L = \psi 1 n_R h_R \quad \text{and} \quad \phi 0 n_L h_L = h_R(\psi 1 n_R h_R).$$

So can we solve these equations? Well, the first thing we notice is that n_R and h_L do not depend on anything and so can be freely chosen, so we just set these to be canonical elements $n_R, h_L := 0_{\mathbb{N}}, 0_{\mathbb{N}\to\mathbb{N}}$ (note that this makes sense intuitively, since n only plays a role in the left disjunct, and h only in the right). We can then define $n_L := \psi 1 0 h_R$. It remains to find some h_R which satisfies

$$h_R(\psi 1 0 h_R) = \phi 0 n_L 0 = \phi 0 (\psi 1 0 h_R) 0,$$

where the latter equality follows by substituting in our definition for n_L. But this is easily achieved if we set $h_R := \lambda i. \phi 0 i 0$. So we're done, and to summarise, (7) is solved by setting

$$b,n,h := \begin{cases} 0, \psi 1 0 h_R, 0 & \text{if } P(\psi 1 0 h_R, \phi 0 (\psi 1 0 h_R) 0) \\ 1, 0, h_R & \text{otherwise} \end{cases}$$

The reader can now easily check that this is indeed a solution by substituting it back into (7). Note that while we use the law of excluded-middle in a very specific way in our proof, the above would work for *any* instance of Σ_2^0 law of excluded middle (in fact we don't even need the quantifiers to be of lowest type).

5.2 Interpreting $(\forall x)(\exists b)(\forall n)(\exists k \geq n)(x_k = b)$

We now use the realizing term given above to witness the functional interpretation of our intermediate result $(\forall x)(\exists b)(\forall n)(\exists k \geq n)(x_k = b)$. In order to distinguish this b from that of the previous section, we relabel it as c. Taking into account the negative translation, what we mean is to interpret is

$$(\forall x)\neg\neg(\exists c, f)(\forall n)(fn \geq n \wedge x_{fn} = c)$$

and hence

$$(\forall x, \xi)(\exists c, f)(f(\xi cf) \geq \xi cf \wedge x_{f(\xi cf)} = c).$$

Again, this looks somewhat intricate, but the term ξcf simply represents the quantifier $(\forall n)$. Now, in order to prove this statement we used the law of excluded-middle for the formula $P(n, k) :\equiv (k \geq n \rightarrow x_k = 0)$ given some fixed sequence x. What we need to do is work out exactly how this was used, and following our discussion in Section 4.4.3 this means realizing the implication

$$(\exists b, n, h)(\forall k, m)(P(n, k) \vee_b P(m, hm)) \rightarrow (\forall \xi)(\exists c, f)(f(\xi cf) \geq \xi cf \wedge x_{f(\xi cf)} = c) \tag{8}$$

and therefore

$$(\forall b, n, h, \xi)(\exists k, m, c, f)[P(n, k) \vee_b P(m, hm) \rightarrow f(\xi cf) \geq \xi cf \wedge x_{f(\xi cf)} = c].$$

This is much easier than it looks! Let us fix b, n, h, ξ. There are two possibilities. If $b = 0$ then we must find some k, c, f (we can set $m = 0$) such that the conclusion follows from $P(n, k)$. It's sensible to choose $c := 0$, then it remains to find k, f satisfying

$$(k \geq n \rightarrow x_k = 0) \rightarrow f(\xi 0 f) \geq \xi 0 f \wedge x_{f(\xi 0 f)} = 0.$$

Following our formal proof, let's define $f(i) := \max\{n, i\}$ and

$$k := f(\xi 0 f) = \max\{n, \xi 0 f\} = \max\{n, \xi 0(\lambda i.\max\{n, i\}))\}.$$

Then clearly $f(\xi 0 f) \geq n, \xi 0 f$ and $x_{f(\xi 0 f)} = 0$ follows from the premise.

In the second case $b = 1$, setting $c := 1$ (and this time $k = 0$) we must establish the conclusion from $\neg P(m, hm)$, i.e. find m, f satisfying

$$hm \geq m \wedge x_{hm} = 1 \rightarrow f(\xi 1 f) \geq \xi 1 f \wedge x_{f(\xi 1 f)} = 1.$$

But this is more straightforward: $f := h$ and $m := \xi 1 h$ work, so we're done. In other words, defining

$$\phi_\xi 0 n h := \max\{n, \xi 0(\lambda i.\max\{n, i\}))\} \quad \text{and} \quad \psi_\xi 1 n h := \xi 1 h$$

with $\phi_\xi 1 n h = \psi_\xi 0 n h = 0$, we can eliminate the quantifiers $(\forall m, k)$ in (8), and we have proven that

$$(\forall b, n, h, \xi)[P(n, \phi_\xi b n h) \vee_b P(\psi_\xi b n h, h(\psi_\xi b n h)) \rightarrow f(\xi cf) \geq \xi cf \wedge x_{f(\xi cf)} = c]$$

for

$$c, f := \begin{cases} 0, \lambda i.\max\{n, i\} & \text{if } b = 0 \\ 1, h & \text{otherwise.} \end{cases}$$

But we know how to find b, n, h which solve the premise for ϕ_ξ and ψ_ξ, so substituting those solutions in the definition above we have

$$(\forall \xi)(f(\xi cf) \geq \xi cf \wedge x_{f(\xi cf)} = c) \tag{9}$$

for

$$c, f := \begin{cases} 0, \lambda i.\max\{\psi_\xi 10 h_R, i\} & \text{if } P(\psi_\xi 10 h_R, \phi_\xi 0(\psi_\xi 10 h_R)0) \\ 1, h_R & \text{otherwise} \end{cases} \tag{10}$$

where ϕ_ξ, ϕ_ξ, P and h_R are defined as above.

5.3 Simplifying the realizing term

The solution given above for finding c and f in ξ is perfectly valid, but still somewhat tricky to understand, as it is couched in terms of the abstruse functionals which arise from our formal proof. So while an automated extraction may produce something like this, for a human being it is desirable to simplify everything and see if there is an underlying pattern.

It is immediately clear by inspecting the definition (10) above, that there are three key terms which play a role, namely h_R, $\psi_\xi 10h$ and $\phi_\xi 0i0$, with substitutions $h \mapsto h_R$ and $i \mapsto \psi_\xi 10h_R$. So it makes sense to unwind each of these terms. First, notice that from the definitions of ϕ_ξ, ψ_ξ we have

$$h_R(i) = \phi_\xi 0i0 = \max\{i, \xi 0(\lambda j.\max\{i, j\})\} \quad \text{and} \quad \psi_\xi 10h = \xi 1h$$

and so in particular

$$\psi_\xi 10h_R = \xi 1(\lambda i.\max\{i, \xi 0(\lambda j.\max\{i, j\})\}) =: a$$
$$\phi_\xi 0(\psi_\xi 10h_R)0 = h_R(a)$$

and our solution can already be simplified to

$$c, f := \begin{cases} 0, \lambda i.\max\{a, i\} & \text{if } x_{\max\{a, \xi 0(\lambda j.\max\{a, j\})\}} = 0 \\ 1, \lambda i.\xi 0(\lambda j.\max\{i, j\}) & \text{otherwise.} \end{cases}$$

where we use the fact that $P(a, \max\{a, \xi 0(\lambda j.\max\{a, j\})\}) \leftrightarrow x_{\max\{a, \xi 0(\lambda j.\max\{a, j\})\}} = 0$.

We now see that, far from being the syntactic mess it appeared, our realizing term can be expressed in a very natural way. By looking closer, an interesting structure emerges: Given a function $q : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, and a pair of functions $\varepsilon, \delta : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$, define the pair $(\varepsilon \otimes \delta)(q) := \langle a, b[a] \rangle$ where

$$b[i] := \delta(\lambda j.q(i, j))$$
$$a := \varepsilon(\lambda i.q(i, b[i])).$$

This is the so-called *binary product of selection functions* studied by Escardó and Oliva in [7]. Intuitively it gives a optimal play in a two player sequential game, where q is assigns an outcome to each pair of moves, and ε, δ dictate the strategy of the first and second players respectively. Using this new notation, our realizer becomes

$$c, f := \begin{cases} 0, \lambda i.\max\{a, i\} & \text{if } x_{\max\{a, b[a]\}} = 0 \\ 1, \lambda i.\xi 0(\lambda j.\max\{i, j\}) & \text{otherwise.} \end{cases}$$

where $\langle a, b[a] \rangle = (\xi 1 \otimes \xi 0)(\max)$. An extension of this idea for nested sequences of the law of excluded-middle were first considered in [20], and generalisations of the product of selection functions to so-called 'unbounded' games have been used to give computational interpretations to choice principles, thereby opening up a fascinating bridge between functional interpretations and game theory [7, 8, 9].

5.4 The final step

It now remains to compute our realizer g in x and ω satisfying (6). For this we need to define a suitable functional ξ such that (6) follows from (9). The key here is the following trick. Suppose that $\tilde{\xi}$ is another functional, and define

$$\xi cf := (\mu i \leq \tilde{\xi} cf)\neg(f(i) \geq i \wedge x_{f(i)} = c).$$

where $(\mu i \leq n)P(i)$ denotes the least $i \leq n$ satisfying $P(i)$, and just 0 if none exist. Then

$$f(\xi cf) \geq \xi cf \wedge x_{f(\xi cf)} = c \to (\forall i \leq \tilde{\xi} cf)(f(i) \geq i \wedge x_{f(i)} = c).$$

So, to this end, define $g_f(0) := f(0)$ and $g_f(n + 1) := f(g_f(n) + 1)$, and let

$$\xi_{x,\omega} cf := (\mu i \leq \tilde{\xi}_\omega cf)\neg(f(i) \geq i \wedge x_{f(i)} = c).$$

for
$$\tilde{\xi}_\omega cf := \max_{k \le \omega g_f}\{g_f(k) + 1\}.$$

Then we claim that

$$f(\xi_{x,\omega}cf) \ge \xi_{x,\omega}cf \wedge x_{f(\xi_{x,\omega}cf)} = c \rightarrow (\forall i < j \le \omega g_f)(g_f(i) < g_f(j) \wedge x_{g_f(i)} = x_{g_f(j)}).$$

Take $i < \omega g_f$. Then $g_f(i) + 1 \le \tilde{\xi}_\omega cf$ and therefore $g_f(i + 1) = f(g_f(i) + 1) \ge g_f(i) + 1 > g_f(i)$ and $x_{g_f(i+1)} = c$, from which we can infer that for any $i < j$ we have $g_f(i) < g_f(j)$ and $x_{g_f(i)} = x_{g_f(j)} = c$. Therefore, by the previous sections, we have

$$(\forall x, \omega)(\forall i < j \le \omega g)(g(i) < g(j) \wedge x_{g(i)} = x_{g(j)})$$

where $g = g_f$ for the f satisfying (9) relative to $\xi_{x,\omega}$ defined above.

5.5 Summary

Our aim in this section was to lead the reader through an actual example of program extraction, from a much simpler classical principle than that about to be considered below. Rather than just presenting an extracted term, our hope was to illustrate how by analysing extracted programs and applying a degree of ingenuity, one can devise descriptions of these programs which are of interest in their own right. Here, the observation by Escardó and Oliva that the functional interpretation of the law of excluded-middle concealed a natural game-theoretic construction which could be extended to encompass much stronger principles led to a large body of research in a somewhat unexpected direction. Later in this paper, a notion of a *learning procedure* will play somewhat analogous role to the product of selection functions above, in the sense that it will describe a very natural computational pattern that underlies our realizer, and helps us understand its behaviour.

6 The functional interpretation of $\mathrm{ZL}_{\mathrm{lex}}$ - Part 1

The basic soundness proof of the functional interpretation (Theorem 4.1) guarantees that we are able to extract a program from any proof which can be formalised in $\mathrm{PA}^\omega + \mathrm{QF\text{-}AC}$. However, our formalisation of the minimal bad sequence construction involves something stronger, namely an instance of $\mathrm{ZL}_{\mathrm{lex}}$. The following three sections contain the chief novelty of our approach, namely the solution of the functional interpretation of $\mathrm{ZL}_{\mathrm{lex}}$ via a form of *open recursion*, which will allow us in Section 9 to extract a program witnessing $\mathrm{WQO}(=_{\mathbb{B},*})$.

So what exactly is the functional interpretation of $\mathrm{ZL}_{\mathrm{lex}}$? Let's begin by writing out the axiom in full, where now replace $B(u)$ with the piecewise formula $(\forall n)P([u](n))$, and for the remainder of the paper we now assume that $P(s)$ is quantifier-free, as it is in the case of Theorem 3.1. In order to avoid nested expressions such as $P([[v](m)](n))$ we will use the notation $\bar{P}(u, n) :\equiv P([u](n))$. Then $\mathrm{ZL}_{\mathrm{lex}}$ becomes

$$(\exists u)(\forall n)\bar{P}(u, n) \rightarrow (\exists v)((\forall n)\bar{P}(v, n) \wedge (\forall w \lhd_{\mathrm{lex}} v)(\exists n)\neg \bar{P}(w, n)).$$

Now, there is still an additional quantifier implicit in $(\forall w \lhd_{\mathrm{lex}} v)$, but note that

$$(\forall w \lhd_{\mathrm{lex}} v)A(w) \leftrightarrow (\forall m, w)(w_0 \lhd v_m \rightarrow A([v](m) * w))$$

and so $\mathrm{ZL}_{\mathrm{lex}}$ can be written out in a fully explicit form as

$$(\exists u)(\forall n)\bar{P}(u, n) \rightarrow (\exists v)((\forall n)\bar{P}(v, n) \wedge (\forall m, w)(w_0 \lhd v_m \rightarrow (\exists n)\neg \bar{P}([v](m) * w, n))). \tag{11}$$

Of course, we want to apply the functional interpretation to the negative translation of (11), which is equivalent to

$$(\exists u)(\forall n)\bar{P}(u, n) \rightarrow \neg\neg(\exists v)((\forall n)\bar{P}(v, n) \wedge (\forall m, w)(w_0 \lhd v_m \rightarrow (\exists n)\neg \bar{P}([v](m) * w, n))). \tag{12}$$

Since this is a rather intricate formula, let's break its interpretation up into pieces. Focusing on the conclusion first, and applying the interpretation under the double negation only, we obtain

$$\neg\neg(\exists v, \gamma^{\mathbb{N} \to X^{\mathbb{N}} \to \mathbb{N}})(\forall n, m, w)(\bar{P}(v, n) \wedge \underbrace{(w_0 \lhd v_m \to \neg \bar{P}([v](m) * w, \gamma m w))}_{C(v, \gamma, m, w)})) \tag{13}$$

where from now on we will use the abbreviation

$$C(v, \gamma, m, w) :\equiv w_0 \lhd v_m \to \neg \bar{P}([v](m) * w, \gamma m w)$$

as indicated in (13). Now, applying the functional interpretation to (13) and referring back to the discussion in Section 4.4.2 we arrive at

$$(\forall N, M, W)(\exists v, \gamma)(\bar{P}(v, N v \gamma) \wedge C(v, \gamma, M v \gamma, W v \gamma)) \tag{14}$$

where $N, M : X^{\mathbb{N}} \to (\mathbb{N} \to X^{\mathbb{N}} \to \mathbb{N}) \to \mathbb{N}$ and $W : X^{\mathbb{N}} \to (\mathbb{N} \to X^{\mathbb{N}} \to \mathbb{N}) \to X^{\mathbb{N}}$. Substituting (14) back into (12) and referring to Section 4.4.3 our challenge is to witness the following expression:

$$(\forall u, N, M, W)(\exists n, v, \gamma)(\bar{P}(u, n) \to \bar{P}(v, N v \gamma) \wedge C(v, \gamma, M v \gamma, W v \gamma))). \tag{15}$$

So what does the expression (15) *intuitively* mean? In Section 4.4 we characterised the functional interpretation as a translation which takes fundamentally non-constructive existence statements and converts them into 'approximate' existence statements, which in theory can be given a direct computational interpretation. In its original form, $\mathrm{ZL}_{\mathrm{lex}}$ simply states that

> if there exists a bad sequence u then there exists a bad sequence v which is minimal with respect to $\lhd_{\mathrm{lex}}$,

where we call u bad whenever $(\forall n)\bar{P}(u, n)$ holds. Now, very roughly, we can read the interpreted statement (15) as saying something like

> for any sequence u and counterexample functionals N, M, W, there exists n, v and γ such that $\bar{P}(u, n)$ implies that v is approximately bad with respect to N, and γ witnesses that it is approximately minimal with respect to M and W.

When using $\mathrm{ZL}_{\mathrm{lex}}$ as a lemma in the proof of a Π_2 statement, as we do in Corollary 3.4, the task of extracting a program from this proof involves calibrating exactly what kind of approximations we need.

6.1 A rough idea

So how do we go about solving (15) - in other words computing a suitable n, v and γ in terms of u, N, M and W? A natural idea might be to simply use trial and error, as follows. Given some initial sequence u, we could first just try $v := u$. Let's also set $\gamma := \gamma_u$, where γ_u is some function that we will need to sensibly define later, and put $n := N u \gamma_u$. Now suppose that $\bar{P}(u, N u \gamma_u)$ holds. There are two possibilities: Either u is approximately minimal in the sense that $C(u, \gamma_u, M u \gamma_u, W u \gamma_u)$ holds, and then we're done, or $\neg C(u, \gamma_u, M u \gamma_u, W u \gamma_u)$ i.e.

$$(W u \gamma_u)_0 \lhd u_{M u g_u} \wedge \bar{P}([u](M u \gamma_u) * W u \gamma_u, \gamma(M u \gamma_u)(W u \gamma_u)).$$

But in this case, we have found a sequence $u_1 := [u](M u \gamma_u) * W u \gamma_u$ which is lexicographically less that u and approximately bad, so could we just set $v := u_1$ and repeat this process, generating a sequence $u \rhd_{\mathrm{lex}} u_1 \rhd_{\mathrm{lex}} u_2 \rhd_{\mathrm{lex}} \ldots \ldots \rhd_{\mathrm{lex}} u_k$ until we reach some u_k which works? Of course, there are a lot of details to be filled in here, in particular a formal definition of γ, but the aim of Section 8 will be to demonstrate that this informal idea *does* actually work.

However, the obvious problem we face is that we seem to be carrying out recursion over the non-wellfounded ordering $\rhd_{\mathrm{lex}}$, and so first we must establish a set of conditions under which this kind of recursion is well-defined. This is the purpose of Section 7 which follows. Before we get into the technical details, though, we want to pause for a moment and explore the general pattern hinted at above, and introduce the notion of a learning procedure, which we have alluded to several times earlier.

20

6.2 Learning procedures

Our challenge in the next Sections is to take some initial sequence u which is 'approximately bad' and produce a v which is also approximately bad, but in addition approximately minimal. For simplicity, let's forget for a moment that we're working with infinite sequences and the lexicographic ordering, and just consider a set X which comes equipped with two decidable predicates $P_0(x)$ and $C_0(x)$.

Of course, in our case $P_0(x)$ represents that x is approximately bad while $C_0(x)$ represents that it's approximately minimal, but here everything is greatly simplified and we do not assume anything about these formulas beyond the following property, which states that if x is not minimal then there must be some $y \prec x$ satisfying $P_0(y)$:

$$(\forall x)(\neg C_0(x) \rightarrow (\exists y \prec x)P_0(y)). \tag{16}$$

Therefore on an *abstract* level, the algorithmic problem we face is the following: Given some initial x which satisfies $P_0(x)$, to find some y which satisfies both $P_0(y)$ and $C_0(y)$, where the *failure* of $C_0(x)$ always leads to a 'better' guess y - this is captured by (16). In other words, using (16) we want to produce a y satisfying

$$(\forall x)(P_0(x) \rightarrow (\exists y)(P_0(y) \wedge C_0(y))). \tag{17}$$

It is not too hard to come up with an algorithm which takes us from a realizer of (16) to a realizer of (17).

Lemma 6.1. *Suppose that $\xi : X \rightarrow X$ is a function which satisfies*

$$(\forall x)(\neg C_0(x) \rightarrow x \succ \xi(x) \wedge P_0(\xi(x))). \tag{18}$$

For any $x : X$, the learning procedure $\mathcal{L}_{\xi,C_0}[x]$ *starting at x denotes the sequence $(x_i)_{i \in \mathbb{N}}$ given by*

$$x_0 := x \quad and \quad x_{i+1} := \begin{cases} x_i & \text{if } C_0(x_i) \\ \xi(x_i) & \text{otherwise.} \end{cases}$$

Whenever $\succ$ is wellfounded, there exists some k such that $C_0(x_k)$ holds, and we call the minimal such x_k the limit of $\mathcal{L}_{\xi,C_0}[x]$, which we denote by

$$\lim \mathcal{L}_{\xi,C_0}[x].$$

Then the functional $\lambda x.\lim \mathcal{L}_{\xi,C_0}[x]$ is definable using wellfounded recursion over $\succ$, and realizes (17) in the sense that

$$(\forall x)(P_0(x) \rightarrow P_0(\lim \mathcal{L}_{\xi,C_0}[x]) \wedge C_0(\lim \mathcal{L}_{\xi,C_0}[x])). \tag{19}$$

Proof. To formally construct the limit, given C_0 and ξ define the function $L_{\xi,C_0} : X \rightarrow X^*$ by

$$L_{\xi,C_0}(x) := \begin{cases} [x] & \text{if } C_0(x) \\ [x] * L_{\xi,C_0}(\xi(x)) & \text{otherwise,} \end{cases}$$

which is definable via wellfounded recursion over $\succ$ since the recursive call $L_{\xi,C_0}(\xi(x))$ is only made in the event that $\neg C_0(x)$ and so $x \succ \xi(x)$ by (18). A simple induction over the length of $L_{\xi,C_0}(x)$ then establishes that $\lim \mathcal{L}_{\xi,C_0}[x]$ is the last element of $L_{\xi,C_0}(x)$.

That the limit satisfies (19) essentially follows from the definition. If $P_0(x_i)$ but $\neg C_0(x_i)$ then we have $x_{i+1} = \xi(x_i)$ with $x_i \succ x_{i+1}$ and $P_0(x_{i+1})$. So it follows that if $P_0(x)$ then $P_0(x_i)$ for all $i \in \mathbb{N}$. Then by the existence of a limit x_k satisfying $C_0(x_k)$ we're done, since we then have $P_0(x_k) \wedge C_0(x_k)$. □

Algorithms of the above kind can be characterised as 'learning procedures' because we start with some initial attempt x_0 for our minimal element, and either this works or it fails, in which case we replace x_0 with some 'improved' guess $x_0 \prec x_1$ which we have learned from the failure of x_0 and continue in this way until we have produced an attempt x_k which works and satisfies $P_0(x_k) \wedge C_0(x_k)$.

The next two sections involve adapting this basic idea to the more complex situation of constructing a realizer for the functional interpretation of ZL_{lex}, where the predicates P_0 and C_0 will need to take into

account the counterexample functionals which determine precisely what an approximation constitutes. Moreover, we will need to adapt Lemma 6.1 so that it applies to the non-wellfounded ordering $\triangleright_{\mathrm{lex}}$.

Learning procedures as described above form the main subject of the author's paper [23], which in particular contains a solution to the functional interpretation of the least element principle for wellfounded $>$ that essentially forms a simple version of the realizer we construct here. Moreover, learning procedures even for certain non-wellfounded orderings are discussed in [23, Section 5], although none of this encompasses the variant of recursion over $\triangleright_{\mathrm{lex}}$ which we require below. Note that these learning procedures should not be confused with the learning realizability of Aschieri and Berardi (e.g. [2]), which although based on a similar idea, takes place in a very different setting.

7   Recursion over $\triangleright_{\mathrm{lex}}$ in the continuous functionals

In order to give a functional interpretation of $\mathrm{ZL}_{\mathrm{lex}}$, it is necessary that we extend system T with some form of recursion over the relation $\triangleright_{\mathrm{lex}}$. Since $\triangleright_{\mathrm{lex}}$ is not wellfounded, it is clear that naively introducing a general recursor over $\triangleright_{\mathrm{lex}}$ will lead to problems. However, just as $\mathrm{ZL}_{\mathrm{lex}}$ is equivalent to an induction principle $\mathrm{OI}_{\mathrm{lex}}$ over $\triangleright_{\mathrm{lex}}$, which comes with the caveat that formulas must be *open* (cf. Section 3.1), we will show that we can define a recursor over $\triangleright_{\mathrm{lex}}$ which exists in continuous models of higher-type functionals, provided that we introduce an analogous restriction for the recursor.

The notion of recursion over $\triangleright_{\mathrm{lex}}$ is not new: In particular this forms the main topic of Berger's analysis of open induction in the framework of modified realizability [4]. However, the functional interpretation requires a non-trivial adaptation of these ideas, which is the main purpose of this section.

7.1   The problem with recursion over $\triangleright_{\mathrm{lex}}$

We begin by highlighting why a naive lexicographic recursor does not behave in the same way as Gödel's wellfounded recursors Rec, as identifying the problems provides some insight into how we can potentially circumvent them. Suppose that given some pair $(X, \triangleright)$ where X is a type and $\triangleright$ a wellfounded decidable relation on X, together with output type Y, we add to our programming language T an open recursor $\mathrm{ORec}_{(X,\triangleright),Y}$ which has the defining equation

$$\mathrm{ORec}^H_{(X,\triangleright),Y}(u) =_Y Hu(\lambda n, v \,.\, \mathrm{ORec}^H([u](n) * v) \text{ if } v_0 \lhd u_n)$$

where 'if $v_0 \lhd u_n$' is short for 'if $v_0 \lhd u_n$, else 0_Y'. Does our recursor give rise to well-defined functionals?

Let's consider the very simple case $X = \mathbb{B}$ where $b_0 \triangleright b_1$ only holds in the case $1 \triangleright 0$, and set the output type $Y := \mathbb{N}$. Define the closed functional $\Phi : (\mathbb{B}^{\mathbb{N}} \to (\mathbb{N} \to \mathbb{B}^{\mathbb{N}} \to \mathbb{N}) \to \mathbb{N}) \to \mathbb{N}$ by

$$\Phi H := \mathrm{ORec}^H_{(\mathbb{B},\triangleright),\mathbb{N}}(\lambda k.1).$$

Then we can show that the type structure of all set-theoretic functionals is no longer a model of $\mathsf{T} +$ $(\mathrm{ORec}_{(\mathbb{B},\triangleright),\mathbb{N}})$. To see this, consider the functional $H : \mathbb{B}^{\mathbb{N}} \to (\mathbb{N} \to \mathbb{B}^{\mathbb{N}} \to \mathbb{N}) \to \mathbb{N}$ defined by

$$Huf := \begin{cases} 1 + fn(0,1,1,\ldots) & \text{for the least } n \text{ with } u_n = 1 \\ 0 & \text{if no such } n \text{ exists.} \end{cases}$$

Suppose that $\Phi H = N$ for some natural number N. Then unwinding the defining equation of $\mathrm{ORec}^H_{(\mathbb{B},\triangleright),\mathbb{N}}$ we get

$$N = \Phi H = 1 + \mathrm{ORec}^H(0,1,1,\ldots) = 2 + \mathrm{ORec}^H(0,0,1,1,\ldots) = \ldots = N + 1 + \mathrm{ORec}^H(\underbrace{0,\ldots,0}_{N+1 \text{ times}},1,1,\ldots) \geq N + 1,$$

a contradiction. Here it is not necessarily surprising that we run into problems. But suppose that we demand that H be *continuous*, in the sense that we can determine the value of Huf based on a finite initial

22

segment of u and f. Unfortunately, it turns out that if we increase the output type to $Y := \mathbb{N} \to \mathbb{N}$ then not even continuity (or indeed even computability) can save us: Let $G : \mathbb{B}^{\mathbb{N}} \to (\mathbb{N} \to \mathbb{B}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}) \to \mathbb{N}^{\mathbb{N}}$ be defined by

$$Gufn := 1 + fn(0,1,1,\ldots)(n+1),$$

and let $N : \mathbb{N}$ be given by $N := \mathrm{ORec}^G(\lambda k.0)(0)$. Then similarly to before, we have

$$N = 1 + \mathrm{ORec}^G(0,1,1,\ldots)(1) = 2 + \mathrm{ORec}^G(0,0,1,1,\ldots)(2) = \ldots = N+1+\mathrm{ORec}^G(\underbrace{0,\ldots,0}_{N+1 \text{ times}},1,1,\ldots)(N+1) \geq N+1,$$

which is again inconsistent with the axioms of Peano arithmetic. So even though N is a closed term in $\mathsf{T} + (\mathrm{ORec}_{(\mathbb{B},\rhd),\mathbb{N}^{\mathbb{N}}})$ of base type, there is no natural interpretation of N in even in continuous models. So what does it take to ensure that recursion over $\rhd_{\mathrm{lex}}$ *does* have an interpretation in continuous models? To this end we will discuss two possible restrictions, namely:

- Leave the defining equation of the recursor unchanged but restrict Y to being a base type.

- Allow Y to be an arbitrary type but introduce an explicit 'control functional' into the defining equation.

The former is the approach taken by Berger in [4] and works well in the setting of modified realizability. However, for the functional interpretation we need a recursor whose output type Y can be arbitrary, and so we appeal to the second strategy which we will describe in detail in Section 7.3. However, to put our solution in context, first we will quickly sketch Berger's solution.

7.2 The continuous functionals and Berger's open recursor

In order to extend functional interpretations to subsystems of mathematical analysis, it is traditionally necessary to extend the usual interpreting calculus of functionals with a strong form of recursion, which is typically only satisfiable the *continuous* models. This was originally the case with Spector's bar recursion, and also here with our variants of open recursion.

In this section we assume a basic knowledge of the type structures of partial and total continuous functionals, as a full presentation here is beyond the scope of our paper. Continuous type structures of functionals were formally constructed from the 1960s onwards: The total continuous functionals being conceived simultaneously by Kleene [12] and Kreisel [16] and the partial model, developed independently by Scott [25] and Ershov [6]. Variants of the latter play an important role in domain theory, where in particular they are used to give a denotational semantics to abstract functional programming languages such as PCF. For an up-to-date presentation of these things and much more in this direction, the reader is encouraged to consult [17].

Very roughly, the continuous functionals $C^\omega_{X \to Y}$ of type $X \to Y$ consist of functionals F from X to Y which satisfy the property that

> in order to determine a finite amount of information about $F(x)$ one only needs a finite amount of information about x,

where the notion of finiteness is made precise by introducing a suitable topology for each type. Note that continuity is a strictly weaker property than being *computable*: In particular *any* function $f : \mathbb{N} \to \mathbb{N}$ is continuous by definition, since $f(n)$ only depends on a natural number n, and natural numbers are here considered to be finite pieces of information. On the other hand, not all functionals $F : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ are continuous, in fact the continuous functionals $C^\omega_{\mathbb{N}^{\mathbb{N}} \to \mathbb{N}}$ of type 2 are precisely those such that for any $\alpha : \mathbb{N}^{\mathbb{N}}$ there exists some N such that for all β, if $[\alpha](N) = [\beta](N)$ then $F(\alpha) = F(\beta)$. Both of the aforementioned properties can be generalised in the following way:

(i) The continuous functionals $C^\omega_{\mathbb{N} \to X}$ consist of *all* sequences $\mathbb{N} \to C^\omega_X$, and so in particular the type structure of continuous functionals is a model of countable dependent choice.

(ii) Any $F \in C^\omega_{X^\mathbb{N} \to \mathbb{N}}$ satisfies the following property:

$$\text{CONT} : (\forall \alpha)(\exists N)(\forall \beta)([\alpha](N) =_{X^*} [\beta](N) \to F(\alpha) = F(\beta)).$$

Note that for $X = \mathbb{N}$ this property is equivalent to F being continuous, whereas for X a higher type, it is strictly weaker (since F could depend on an infinite amount of information from $\alpha(0)$ but still satisfy CONT, for example).

The *partial* continuous functionals $\hat{C}^\omega$ are similar to the total continuous functionals described above, with the crucial difference that they allow functionals which are undefined in places, and so the $\hat{C}^\omega_X$ are represented by a *domains* which in particular come equipped with a bottom element $\bot$ denoting an undefined value. The partial continuous functionals have the key property that every continuous functional $X \to X$ has a continuous fixed point, which means in particular that any recursively defined functional has a natural interpretation in $\hat{C}^\omega$ (although this need not be total). The partial continuous functionals are related to the total continuous functionals in that C^ω is the extensional collapse of the total elements of $\hat{C}^\omega$ [6]. What this means in practice is that in order to show that a recursively defined functional has an interpretation in C^ω, it is enough to show that its interpretation in $\hat{C}^\omega$ as a fixpoint is total.

Theorem 7.1 (Berger [4]). *Let $\rhd$ be a well-founded, decidable relation on X. Then any fixpoint of the defining equation of $\text{ORec}_{(X,\rhd),\mathbb{N}}$ is total, and hence $\text{ORec}_{(X,\rhd),\mathbb{N}}$ exists in the total continuous functionals C^ω.*

Proof. While in [4, Proposition 5.1] this is proven using a variant of open induction, we appeal to the classical minimal bad sequence construction, to emphasise already the deep connection with Nash-Williams' proof of Higman's lemma. Suppose for contradiction that there are total arguments H and u such that $\text{ORec}^H(u)$ is not total. Using dependent choice, which is valid for total objects of sequence type, construct the minimal bad sequence v of total elements of type X as follows:

If $[v_0, \ldots, v_{k-1}]$ has already been constructed, define v_k to be a total element of $\hat{C}^\omega_X$ such that $\text{ORec}^H([v_0, \ldots, v_{k-1}, v_k] * w)$ is not total for some total extension w, but $\text{ORec}^H([v_0, \ldots, v_{k-1}, a] * w)$ is total for all total w whenever $a \lhd v_k$.

Now consider $\text{ORec}^H(v) = Hv(\lambda n, w . \text{ORec}^H([v](n) * w)$ if $w_0 \lhd v_n) = H\alpha$ where

$$\alpha_n := \langle v_n, \lambda w.\text{ORec}^H([v](n) * w) \text{ if } w_0 \lhd v_n \rangle,$$

and note that we use a slight abuse of types here, informally identifying the type $X^\mathbb{N} \to (\mathbb{N} \to X^\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ of H with $(X \times (X^\mathbb{N} \to \mathbb{N}))^\mathbb{N} \to \mathbb{N}$. But by minimality of v, the sequence α_n is total, and hence $H\alpha$ is total and then by CONT applied to the total objects H and α there exists some N such that whenever $[\alpha](N) = [\beta](N)$ then $H\alpha = H\beta$. But now consider the sequence $[v](N)$. By construction there exists some w such that $\text{ORec}^H([v](N) * w)$ is not total. But $\text{ORec}^H([v](N) * w) = H\beta$ for

$$\beta_n := \langle ([v](N) * w)_n, \lambda w'.\text{ORec}^H([[v](N) * w](n) * w') \text{ if } w'_0 \lhd ([v](N) * w)_n \rangle$$

and we have $\alpha_n = \beta_n$ for all $n < N$ and hence $H\beta = H\alpha$ which is total, a contradiction. Hence our original assumption was wrong and $\text{ORec}^H(u)$ must be total, and since H and u were arbitrary we have that ORec is total. $\square$

We have given this proof in great detail as we want to compare it to the corresponding totality proof of our explicit open recursor given in the next section. We now conclude our overview of Berger's open recursion by stating the main result of [4], namely:

Theorem 7.2 (Berger [4]). *There is a functional definable in $\mathsf{T} + (\text{ORec}_{(X,\rhd),\mathbb{N}})$ such that Φ satisfies the modified realizability interpretation of the axiom of open induction OI_{lex} for Σ^0_1-piecewise formulas, provably in $\text{PA}^\omega + \text{CONT} + \text{OI}_{\text{lex}} + (\text{ORec}_{(X,\rhd),\mathbb{N}})$.*

Theorem 8.3 below forms an analogue of this for the functional interpretation.

7.3 The explicitly controlled open recursor

Berger's variant of open recursion uses in an essential way the fact that total continuous functions of type $Z^{\mathbb{N}} \to \mathbb{N}$ (where Z is arbitrary) only consider a finite initial segment of their input. In this way we avoid the problems encountered earlier in the chapter. However, as we will see, having open recursive functionals whose output type Y is arbitrary is essential for the functional interpretation of $\mathrm{ZL_{lex}}$, and Berger's variant is no longer total in this case as we cannot rely on continuity to 'implicitly' control the recursion. Therefore we require some other way of ensuring that the recursor only depends on a finite initial segment of its input. We accomplish this by adding an additional parameter F to the recursor which is responsible for 'explicitly' controlling the recursion, in a sense that will be made clear below. We start with some definitions.

Definition 7.3. Suppose that $\alpha : Z^{\mathbb{N}}$ and $m : \mathbb{N}$. Then the infinite sequence $[\alpha]_m : Z^{\mathbb{N}}$ is defined by

$$[\alpha]_m := \lambda n. \begin{cases} \alpha_n & \text{if } n < m \\ 0_Z & \text{otherwise.} \end{cases}$$

Now suppose in addition that $F : Z^{\mathbb{N}} \to \mathbb{N}$. Then the infinite sequence $\{\alpha\}_F : Z^{\mathbb{N}}$ is defined by

$$\{\alpha\}_F := \lambda n. \begin{cases} 0_Z & \text{if } (\exists m \leq n)(F([\alpha]_m) < m) \\ \alpha_n & \text{otherwise.} \end{cases}$$

Note that both $[\alpha]_m$ and $\{\alpha\}_F$ are primitive recursively definable.

Note that the sequence $\{\alpha\}_F$ uses the well-known stopping condition $F([\alpha]_m) < m$ due to Spector [27], which ensures that his variant of bar recursion is well-founded.

Lemma 7.4. *Given some $F : Z^{\mathbb{N}} \to \mathbb{N}$ and $\alpha : Z^{\mathbb{N}}$, whenever there exists some $m : \mathbb{N}$ such that $F([\alpha]_m) < m$ then*

$$\{\alpha\}_F = [\alpha]_{m_0}$$

where m_0 is the least such m. If no such m exists then $\{\alpha\}_F = \alpha$.

Proof. This follows directly from the definition: For the first case, by minimality of m_0 we have $\{\alpha\}_F(n) = \alpha_n$ for all $n < m_0$, and $\{\alpha\}_F(n) = 0_Z$ otherwise, which is exactly the definition of $[\alpha]_{m_0}$. □

Lemma 7.5. *For any functional $F : Z^{\mathbb{N}} \to \mathbb{N}$ satisfying CONT, then for each $\alpha : Z^{\mathbb{N}}$ there exists some m such that $F([\alpha]_m) < m$.*

Proof. Suppose that N is the point of continuity of F which exists by CONT, and define $m := \max\{N, F\alpha + 1\}$. Then $[[\alpha]_m](N) = [\alpha](m)$ since $N \leq m$, and therefore $F([\alpha]_m) = F\alpha < m$. □

Theorem 7.6. *Given $F : Z^{\mathbb{N}} \to \mathbb{N}$ and $\alpha : Z^{\mathbb{N}}$, the following facts are provable assuming CONT:*

(i) $\{\alpha\}_F = [\alpha]_{m_0}$ where m_0 satisfies $F([\alpha]_{m_0}) < m_0$ and is the least such number;

(ii) for any β satisfying $[\alpha](m_0) = [\beta](m_0)$ we have $\{\alpha\}_F = \{\beta\}_F$;

(iii) $\{\{\alpha\}_F\}_F = \{\alpha\}_F$.

Proof. Part (i) follows directly from Lemma 7.4 together with Lemma 7.5. For part (ii), we observe that for all $n \leq m_0$ we have $[\alpha]_n = [\beta]_n$, from which it follows that the first m_1 satisfying $F([\beta]_{m_1}) < m_1$ is just m_0. Therefore by part (i) again we have $\{\beta\}_F = [\beta]_{m_0} = [\alpha]_{m_0} = \{\alpha\}_F$. Part (iii) now follows easily, since by part (i) we have $\{\{\alpha\}_F\}_F = \{[\alpha]_{m_0}\}_F$, and since $[[\alpha]_{m_0}](m_0) = [\alpha](m_0)$ then $\{[\alpha]_{m_0}\}_F = \{\alpha\}_F$ by part (ii). □

Now we are ready to define our 'explicit' recursor. Given $H : (X \times (X^{\mathbb{N}} \to Y))^{\mathbb{N}} \to Y$ and $F : (X \times (X^{\mathbb{N}} \to Y))^{\mathbb{N}} \to \mathbb{N}$, we define

$$\text{EORec}^{H,F}_{(X,\rhd),Y}(u) =_Y H(\{\alpha\}_F) \quad \text{for} \quad \alpha :=_{(X \times (X^{\mathbb{N}} \to Y))^{\mathbb{N}}} \lambda n \, . \, \langle u_n, \lambda v \, . \, \text{EORec}^{H,F}([u](n) * v) \text{ if } v_0 \lhd u_n \rangle.$$

This is a form of lexicographic recursion just as before, but with the crucial difference that the recursor now comes equipped with some additional functional $F : Z^{\mathbb{N}} \to \mathbb{N}$ which determines how much of the sequence α is 'relevant'. As soon as we have found some m satisfying the condition $F([\alpha]_m) < m$ then we declare that we are not interested in α_n for $n \geq m$. Our introduction of this 'control' functional F allows us to provide an analogue of CONT for H, even though the output type of H is arbitrary.

Lemma 7.7. *Suppose that $H : Z^{\mathbb{N}} \to Y$ and that $F : Z^{\mathbb{N}} \to \mathbb{N}$ satisfies* CONT. *Then H satisfies the following property:*

$$\text{CONT}^* : \ (\forall \alpha)(\exists N)(\forall \beta)([\alpha](N) = [\beta](N) \to H(\{\alpha\}_F) =_Y H(\{\beta\}_F)).$$

Proof. Let m_0 be the least number satisfying $F([\alpha]_{m_0}) < m_0$, which exists by CONT, and define $N := m_0$. Then if $[\alpha](m_0) = [\beta](m_0)$ then $\{\alpha\}_F = \{\beta\}_F$ by part (ii) above, and therefore $H(\{\alpha\}_F) = H(\{\beta\}_F)$. $\square$

We can use this result to show, analogously to Theorem 7.1, that $\text{EORec}_{(X,\rhd),Y}$ exists in the total continuous functionals for any Y.

Theorem 7.8. *Let $\rhd$ be a well-founded, decidable relation on X. Then the fixpoint of the defining equation of $\text{EORec}_{(X,\rhd),Y}$ is total, and hence $\text{EORec}_{(X,\rhd),Y}$ exists in the total continuous functionals C^ω.*

Proof. This follows analogously to the proof of Theorem 7.1. Suppose for contradiction that there are total arguments H, F and u such that $\text{EORec}^{H,F}(u)$ is not total. Using dependent choice, construct a minimal bad sequence v as follows:

If $[v_0, \ldots, v_{k-1}]$ has already been constructed, define v_k to be a total element of $\hat{C}_X^\omega$ such that $\text{EORec}^{H,F}([v_0, \ldots, v_{k-1}, v_k]*w)$ is not total for some total extension w, but $\text{EORec}^{H,F}([v_0, \ldots, v_{k-1}, a]*w)$ is total for all total w whenever $a \lhd v_k$.

Now consider $\text{EORec}^{H,F}(v) = H(\{\alpha\}_F)$ for

$$\alpha := \lambda n \, . \, \langle v_n, \lambda w \, . \, \text{EORec}^{H,F}([v](n) * w) \text{ if } w_0 \lhd v_n \rangle.$$

Then by construction of v, α and hence $H(\{\alpha\}_F)$ must be total, and by CONT* applied to the total objects α, F and H hence there exists some N such that for any total $\beta : (X \times (X^{\mathbb{N}} \to Y))^{\mathbb{N}}$, if $[\alpha](N) = [\beta](N)$ then $H(\{\alpha\}_F) = H(\{\beta\}_F)$. Now consider the sequence $[v](N)$. By construction there exists some w such that $\text{EORec}^{H,F}([v](N) * w)$ is not total. But $\text{EORec}^{H,F}([v](N) * w) = H(\{\beta\}_F)$ where

$$\beta := \lambda n.\langle ([v](N) * w)_n, \lambda w' \, . \, \text{EORec}^{H,F}([[v](N) * w](n) * w') \text{ if } w'_0 \lhd ([v](N) * w)_n \rangle$$

and we have $\beta(n) = \alpha(n)$ for all $n < N$ and hence by CONT* we have $\text{EORec}^{H,F}([v](N)*w) = H(\{\beta\}_F) = H(\{\alpha\}_F)$ which is total, a contradiction. Therefore $\text{EORec}^{H,F}(u)$ must be total, and since H, F and u were arbitrary total objects then EORec is total. $\square$

8   The functional interpretation of ZL_{lex} - Part 2

We will now make formal the intuitive idea presented in Section 6. We begin by setting up an analogue of Lemma 6.1, but this time the objects x of our learning procedures are sequences $u : X^{\mathbb{N}}$ (and so $P_0(u)$ and $C_0(u)$ are decidable predicates over $X^{\mathbb{N}}$) and u_{i+1} is defined as $\xi(\{u_i\}_\phi)$ for some $\phi : X^{\mathbb{N}} \to \mathbb{N}$. As a result, we end up with a sequence of the form

$$u_0 \mapsto \{u_0\}_\phi \rhd_{\text{lex}} u_1 \mapsto \{u_1\}_\phi \rhd_{\text{lex}} u_2 \mapsto \ldots$$

and so in order to guarantee that $P_0(u_i)$ holds for all i we will require an additional condition, namely that the property P_0 is preserved under the map $\{\cdot\}_\phi : X^\mathbb{N} \to X^\mathbb{N}$. We will now just state and prove the result, but the reader is strongly encouraged to simultaneously refer back to the much simpler Lemma 6.1 and its proof, not only so that it is easier to grasp what is going on here, but because the differences in the formulation of the two lemmas are extremely informative.

Remark 8.1. For the remainder of the paper, we request that the canonical element 0_X of type X is minimal with respect to $\rhd$. This condition is not essential and could be circumvented by other means, but it makes what follows a little easier and allows us to avoid some additional syntax. In practice this assumption is completely benign, and in particular in Chapter 9 where our type X will actually be a type X^* of finite words and $\lhd$ will denote the prefix relation, then the normal choice of $0_{X^*} = []$ is also minimal.

Lemma 8.2. *Suppose that $\xi : X^\mathbb{N} \to X^\mathbb{N}$ is defined by*

$$\xi(u) := [u](\xi_0(u)) * \xi_1(u)$$

where $\xi_0 : X^\mathbb{N} \to \mathbb{N}$ and $\xi_1 : X^\mathbb{N} \to X^\mathbb{N}$, and that ξ satisfies

$$(\forall u)(\neg C_0(u) \to u_{\xi_0(u)} \rhd \xi_1(u)_0 \wedge P_0(\xi(u))). \tag{20}$$

Moreover, suppose that $\phi : X^\mathbb{N} \to \mathbb{N}$ is an additional functional which satisfies

$$(\forall u)(P_0(u) \to P_0(\{u\}_\phi)). \tag{21}$$

For any $u : X^\mathbb{N}$, the controlled learning procedure $\mathcal{LC}^\phi_{\xi,C_0}[u]$ starting at u denotes the sequence $(u_i)_{i\in\mathbb{N}}$ given by

$$u_0 := u \quad and \quad u_{i+1} := \begin{cases} \{u_i\}_\phi & if\ C_0(\{u_i\}_\phi) \\ \xi(\{u_i\}_\phi) & otherwise. \end{cases}$$

Then provably from CONT, *firstly there always exists some k such that $C_0(\{u_k\}_\phi)$ holds, and we call $\{u_k\}_\phi$ for the minimal such u_k the limit of $\mathcal{LC}^\phi_{\xi,C_0}[u]$, which we denote by*

$$\lim \mathcal{LC}^\phi_{\xi,C_0}[u],$$

secondly the functional $\lambda u.\lim \mathcal{LC}^\phi_{\xi,C_0}[u]$ is definable in $\mathsf{T} + (\mathrm{EORec}_{(X,\rhd)})$, and finally we have

$$(\forall u)(P_0(u) \to P_0(\lim \mathcal{LC}^\phi_{\xi,C_0}[u]) \wedge C_0(\lim \mathcal{LC}^\phi_{\xi,C_0}[u])). \tag{22}$$

Proof. We first formally construct the limit functional, which the reader can skip if they like since this is nothing more that a somewhat intricate unwinding of definitions. First, we define $L^\phi_{\xi,C_0} : X^\mathbb{N} \to (X^\mathbb{N})^*$ by $L^\phi_{\xi,C_0}(u) := \mathrm{EORec}^{H,F}_{(X,\rhd),(X^\mathbb{N})^*}(u)$ where

$$F\alpha := \phi(\alpha_0)$$

$$H\alpha := \begin{cases} [\alpha_0] & if\ C_0(\alpha_0) \\ [\alpha_0] * \alpha_1\xi_0(\alpha_0)\xi_1(\alpha_0) & otherwise. \end{cases}$$

Here we denote by α_0 the sequence $\lambda n.\pi_0\alpha(n)$ and similarly for α_1. Then unwinding the definition, we have $L^\phi_{\xi,C_0}(u) = H(\{\alpha\}_F)$ for $\alpha = \lambda n.\langle u_n, \lambda v.L^\phi_{\xi,C_0}([u](n) * v)\rangle$ if $v_0 \lhd u_n$. But since $F\alpha$ depends only on the first component $\alpha_0 = u$, we have (by Lemma 7.6) $\{\alpha\}_F = [\alpha]_{m_0}$ where m_0 is the least number satisfying

$F([\alpha]_{m_0}) = \phi([u]_{m_0}) < m_0$. This means that $\{u\}_\phi = [u]_{m_0}$ and so $(\{\alpha\}_F)_0 = ([\alpha]_{m_0})_0 = [u]_{m_0} = \{u\}_\phi$ and $(\{\alpha\}_F)_1 = \lambda n < m_0, v.L^\phi_{\xi,C_0}([u](n) * v)$ if $v_0 \lhd u_n$, and so

$$H(\{\alpha\}_F) = \begin{cases} [\{u\}_\phi] & \text{if } C_0(\{u\}_\phi) \\ [\{u\}_\phi] * (\{\alpha\}_F)_1 \xi_0(\{u\}_\phi)\xi_1(\{u\}_\phi) & \text{otherwise.} \end{cases}$$

But now by (20), if $\neg C_0(\{u\}_\phi)$ then $(\{u\}_\phi)_{\xi_0(\{u\}_\phi)} \rhd \xi_1(\{u\}_\phi)_0$, and since 0_X was chosen to be minimal with respect to $\rhd$ (cf. Remark 8.1) this can only mean that $\xi_0(\{u\}_\phi) < m_0$ (else $(\{u\}_\phi)_{\xi_0(\{u\}_\phi)} = 0_X$) and therefore $(\{u\}_\phi)_{\xi_0(\{u\}_\phi)} = u_{\xi_0(\{u\}_\phi)}$. Substituting all this information into the $(\{\alpha\}_F)_1$ we have

$$(\{\alpha\}_F)_1 \xi_0(\{u\}_\phi)\xi_1(\{u\}_\phi) = L^\phi_{\xi,C_0}([\{u\}_\phi](\xi_0(\{u\}_\phi)) * \xi_1(\{u\}_\phi)) = L^\phi_{\xi,C_0}(\xi(\{u\}_\phi)).$$

So to summarise, the functional L^ϕ_{ξ,C_0} satisfies (repressing subscripts)

$$L^\phi_{\xi,C_0}(u) = \begin{cases} [\{u\}_\phi] & \text{if } C_0(\{u\}_\phi) \\ [\{u\}_\phi] * L^\phi_{\xi,C_0}(\xi(\{u\}_\phi)) & \text{otherwise,} \end{cases}$$

and so by induction on the length of $L^\phi_{\xi,C_0}(u)$ one establishes that $\lim \mathcal{L}C^\phi_{\xi,C_0}[u]$ exists and is the last element of $L^\phi_{\xi,C_0}(u)$.

To verify (22) is similar to the proof of Lemma 6.1. We first show by induction that if $P_0(u)$ holds then $P_0(\{u_i\}_\phi)$ holds for all $i \in \mathbb{N}$. For $i = 0$ this follows by (21) applied to $u = u_0$, and otherwise if $P_0(\{u_i\}_\phi)$ is true then either $\{u_{i+1}\}_\phi = \{\{u_i\}_\phi\}_\phi = \{u_i\}_\phi$ by Lemma 7.6 or $\neg C(\{u_i\}_\phi)$ and then by (20) we have $P_0(u_{i+1})$ and hence $P_0(\{u_{i+1}\}_\phi)$ by (21). Therefore, if $\{u_k\}_\phi$ is the limit of the learning procedure, then $P_0(\{u_k\}_\phi) \wedge C_0(\{u_k\}_\phi)$ holds, and we're done. □

Our final step is now to produce a realizer for the functional interpretation of $\mathrm{ZL}_{\mathrm{lex}}$. Let's briefly recall from Section 6 what this means: We are given as input a sequence $\bar{u}$ (we use this new notation as we want u to denote a separate variable below), a pair of functionals $M, N : X^{\mathbb{N}} \to (\mathbb{N} \to X^{\mathbb{N}} \to \mathbb{N}) \to \mathbb{N}$, together with $W : X^{\mathbb{N}} \to (\mathbb{N} \to X^{\mathbb{N}} \to \mathbb{N}) \to X^{\mathbb{N}}$, and we must produce some $n : \mathbb{N}$, $v : X^{\mathbb{N}}$ and $\gamma : (X^{\mathbb{N}} \to \mathbb{N})^{\mathbb{N}}$ satisfying

$$\bar{P}(\bar{u}, n) \to \bar{P}(v, Nv\gamma) \wedge C(v, \gamma, Mv\gamma, Wv\gamma), \tag{23}$$

where as before $C(v, \gamma, m, w) :\equiv w_0 \lhd v_m \to \neg\bar{P}([v](m) * w, \gamma m w)$. We first need some definitions. Define the functional $\Psi^N : X^{\mathbb{N}} \to \mathbb{N}$ by

$$\Psi^N(u) := \mathrm{EORec}^{\tilde{N},\tilde{N}}_{(X,\rhd),\mathbb{N}}(u)$$

where $\tilde{N}\alpha := N\alpha_0\alpha_1$. Using Ψ, for each $u : X^{\mathbb{N}}$ define $\gamma_u : \mathbb{N} \to X^{\mathbb{N}} \to \mathbb{N}$ by

$$\gamma_u := \lambda n, v \,.\, \Psi^N([u](n) * v) \text{ if } v_0 \lhd u_n.$$

Finally, define parameters $\phi : X^{\mathbb{N}} \to \mathbb{N}$, $\xi_0 : X^{\mathbb{N}} \to \mathbb{N}$ and $\xi_1 : X^{\mathbb{N}} \to X^{\mathbb{N}}$, together with predicates P_0 and C_0, by

$$\phi(u) := Nu\gamma_u$$
$$\xi_0(u) := Mu\gamma_u$$
$$\xi_1(u) := Wu\gamma_u$$
$$P_0(u) :\equiv \bar{P}(\bar{u}, \Psi^N(\bar{u})) \to \bar{P}(u, \Psi^N(u))$$
$$C_0(u) :\equiv C(u, \gamma_u, Mu\gamma_u, Wu\gamma_u).$$

Now it is perhaps becoming clear to the reader what will come next: We will set up a controlled learning procedure $\mathcal{L}C^\phi_{\xi,C_0}[\bar{u}]$ on these parameters exactly as in Lemma 8.2, and the limit $v := \{u_k\}_\phi$ of $\mathcal{L}C^\phi_{\xi,C_0}[\bar{u}]$ will satisfy $P_0(v) \wedge C_0(v)$, or in other words, $\bar{P}(\bar{u}, \Psi^N(\bar{u})) \to \bar{P}(v, \Psi^N(v))$ and $C(v, \gamma_v, Mv\gamma_v, Wv\gamma_v)$, from which we will be able to construct our realizer of $\mathrm{ZL}_{\mathrm{lex}}$. Let's make this formal.

Theorem 8.3. *Let n, v and γ be defined in terms of $\bar{u}, N, M$ and W by*

$$n := \Psi^N(\bar{u})$$

$$v := \lim \mathcal{LC}^{\phi}_{\xi, C_0}[\bar{u}]$$

$$\gamma := \gamma_v.$$

Then (provably in CONT*) these satisfy (23) and therefore solve the functional interpretation of* $\mathrm{ZL}_{\mathrm{lex}}$.

Proof. We use Lemma 8.2, which means that we must check that each of (20) and (21) hold for our choice of P_0 and C_0, i.e. we must prove

$$(\forall u)[\neg C(u, \gamma_u, M\gamma_u, W\gamma_u) \to u_{\xi_0(u)} \rhd \xi_1(u)_0 \wedge (\bar{P}(\bar{u}, \Psi^N(\bar{u})) \to \bar{P}(\xi(u), \Psi^N(\xi(u))))] \tag{24}$$

and

$$(\forall u)[(\bar{P}(\bar{u}, \Psi^N(\bar{u})) \to \bar{P}(u, \Psi^N(u))) \to (\bar{P}(\bar{u}, \Psi^N(\bar{u})) \to \bar{P}(\{u\}_{\phi}, \Psi^N(\{u\}_{\phi})))]. \tag{25}$$

For the first condition, note that $\neg C(u, \gamma_u, M\gamma_u, W\gamma_u)$ is just $\neg C(u, \gamma_u, \xi_0(u), \xi_1(u))$, which implies both $\xi_1(u)_0 \lhd u_{\xi_0(u)}$ and $\bar{P}([u](\xi_0(u)) * \xi_1(u), \gamma_u \xi_0(u) \xi_1(u))$. But since

$$[u](\xi_0(u)) * \xi_1(u) = \xi(u) \quad \text{and} \quad \gamma_u \xi_0(u) \xi_1(u) = \Psi^N(\xi(u))$$

we have established $\bar{P}(\xi(u), \Psi^N(\xi(u)))$, and hence the conclusion of (24).

The second condition is more subtle: Either $\neg \bar{P}(\bar{u}, \Psi^N(\bar{u}))$ and we're done, or it suffices to prove $\bar{P}(u, \Psi^N(u)) \to \bar{P}(\{u\}_{\phi}, \Psi^N(\{u\}_{\phi}))$. We now need to unwind the definition of $\Psi^N(u)$: First note that we have

$$\Psi^N(u) = \tilde{N}(\{\alpha\}_{\tilde{N}})$$

where (using the definition of γ_u)

$$\alpha := \lambda n. \langle u_n, \lambda v \, . \, \Psi^N([u](n) * v) \text{ if } v_0 \lhd u_n \rangle = \lambda n. \langle u_n, \gamma_{u,n} \rangle.$$

and so in particular we have (by the definitions of $\tilde{N}$ and ϕ)

$$\tilde{N}([\alpha]_m) = N([u]_m)([\gamma_u]_m) = N([u]_m)(\gamma_{[u]_m}) = \phi([u]_m) \tag{26}$$

where for the central equality we use the assumption that 0_X is minimal with respect to $\rhd$ and so

$$\gamma_{[u]_m} = \lambda n, v. \Psi^N([[u]_m](n) * v) \text{ if } v_0 \lhd ([u]_m)_n = \lambda n \, . \, \begin{cases} \lambda v. \Psi^N([u](n) * v) \text{ if } v_0 \lhd u_n & \text{if } n < m \\ \lambda v. \Psi^N([u](n) * v) \text{ if } v_0 \lhd 0_X & \text{otherwise} \end{cases}$$

$$= \lambda n \, . \, \begin{cases} \lambda v. \gamma_u n v & \text{if } n < m \\ \lambda v. 0 & \text{otherwise} \end{cases}$$

$$= [\gamma_u]_m.$$

Let m_0 be the least number such that $\tilde{N}([\alpha]_{m_0}) < m_0$ (which exists since we are assuming CONT), and therefore by (26) the also the least number such that $\phi([u]_{m_0}) < m_0$. Then by Lemma 7.6 we have that for all u:

$$\Psi^N(u) = \tilde{N}(\{\alpha\}_{\tilde{N}}) \overset{\text{L. 7.6(i)}}{=} \tilde{N}([\alpha]_{m_0}) \overset{(26)}{=} \phi([u]_{m_0}) \overset{\text{L. 7.6(i)}}{=} \phi(\{u\}_{\phi}) = N\{u\}_{\phi} \gamma_{\{u\}_{\phi}}. \tag{27}$$

In particular, by Lemma 7.6 (iii) we have

$$\Psi^N(\{u\}_{\phi}) \overset{(27)}{=} \phi(\{\{u\}_{\phi}\}_{\phi}) \overset{\text{L. 7.6(iii)}}{=} \phi(\{u\}_{\phi}) \overset{(27)}{=} \Psi^N(u) \tag{28}$$

and

$$\Psi^N(u) \overset{(27)}{=} \tilde{N}([\alpha]_{m_0}) < m_0 \tag{29}$$

and therefore

$$[\{u\}_\phi](\Psi^N(\{u\}_\phi)) \overset{(28)}{=} [\{u\}_\phi](\Psi^N(u)) \overset{L.\ 7.6(i)}{=} [[u]_{m_0}](\Psi^N(u)) \overset{(29)}{=} [u](\Psi^N(u))$$

and thus $P([u](\Psi^N(u)))$ implies $P([\{u\}_\phi](\Psi^N(\{u\}_\phi)))$, which establishes (25).

It now follows from Lemma 8.2 that

$$P_0(\bar{u}) \rightarrow P_0(v) \wedge C_0(v)$$

and since $P_0(\bar{u})$ is trivially true we have established

$$\bar{P}(\bar{u}, \Psi^N(\bar{u})) \rightarrow \bar{P}(v, \Psi^N(v)) \quad \text{and} \quad C(v, \gamma_v, Mv\gamma_v, Wv\gamma_v).$$

We can now prove (23). Suppose that $\bar{P}(u, n)$ holds. Then since $n = \Psi^N(\bar{u})$ from the left hand side we have $\bar{P}(v, \Psi^N(v))$. Now, since $v = \{u_k\}_\phi$ for some element in the learning procedure $\mathcal{LC}^\phi_{\xi, C_0}[\bar{u}]$, by (28) we have

$$\Psi^N(v) = \Psi^N(\{u_k\}_\phi) = \phi(\{u_k\}_\phi) = N\{u_k\}_\phi \gamma_{\{u_k\}_\phi} = Nv\gamma_v = Nv\gamma$$

and so we have established $\bar{P}(v, Nv\gamma)$. Then, since $C(v, \gamma, Mv\gamma, Wv\gamma)$ is given to us automatically, we have proven

$$\bar{P}(u, n) \rightarrow \bar{P}(v, Nv\gamma) \wedge C(v, \gamma, Mv\gamma, Wv\gamma)$$

which is exactly (23), and so we're done. □

The results of this section mark the technical climax of the paper, and in particular form our broadest and most widely applicable contribution. While the proofs above are perhaps somewhat difficult to navigate, it is important to emphasise that most of the technical details are bureaucratic in nature, in the unwinding of all the definitions and the careful use of Lemma 7.6. The *intuition* behind our realizer, on the other hand, should hopefully be clear from the somewhat more informal discussion in Section 6. In any case, now that the hard work is done, a computational interpretation of Nash-William's proof of Higman's lemma follows relatively easily.

9 Interpreting the proof of $\mathrm{WQO}(=_{\mathbb{B},*})$

We are now finally ready to produce our realizer for the statement that $=_{\mathbb{B}^*}$ is a WQO. In fact we do something more general, namely give a computational interpretation to the proof that $\mathrm{WQO}_{\mathrm{seq}}(\preceq) \rightarrow \mathrm{WQO}(\preceq_*)$, which is valid for *any* well quasi-order $\preceq$. Recall that the functional interpretation of $\mathrm{WQO}_{\mathrm{seq}}(\preceq)$ is given by

$$(\forall x^{X^{\mathbb{N}}}, \omega^{(\mathbb{N}\rightarrow\mathbb{N})\rightarrow\mathbb{N}})(\exists g)(\forall i < j \leq \omega g)(g(i) < g(j) \wedge x_{g(i)} \preceq x_{g(j)}). \tag{30}$$

Lemma 3.2 makes precise exactly how we use the assumption $\mathrm{WQO}_{\mathrm{seq}}(\preceq)$ to prove $\mathrm{WQO}(\preceq_*)$: Namely given a hypothetical minimal bad sequence of words v, we take the sequence $\bar{v}$ and require that our monotone subsequence g be valid up to the point k, where k is such that $P(w, k)$ holds for $w = [v](g(0)) * \tilde{v}_g$ as defined in Lemma 3.2. If minimality of v is witnessed by some functional γ then such a k would be given by $\gamma(g0)(\tilde{v}_g)$. This motivates the following:

Lemma 9.1. *Suppose that G is a realizer for (30):*

$$(\forall x, \omega)(\forall i < j < \omega G_{x,\omega})(G_{x,\omega}(i) < G_{x,\omega}(j) \wedge x_{G_{x,\omega}(i)} \preceq x_{G_{x,\omega}(j)}). \tag{31}$$

Then from this we can construct a functional $H : X^{\mathbb{N}} \rightarrow (\mathbb{N} \rightarrow X^{\mathbb{N}} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}^{\mathbb{N}}$ satisfying

$$(\forall v, \gamma)(\forall i < j < \gamma(H_{v,\gamma}(0))(\tilde{v}_{H_{v,\gamma}}))(H_{v,\gamma}(i) < H_{v,\gamma}(j) \wedge \bar{v}_{H_{v,\gamma}(i)} \preceq \bar{v}_{H_{v,\gamma}(j)}) \tag{32}$$

where $v_{H_{v,\gamma}}$ is shorthand for $\lambda i.v_{v_{H_{v,\gamma}(i)}}$, and $\tilde{v}, \bar{v}$ are defined as in Lemma 3.2.

Proof. Define $\omega_{v,\gamma}g := \gamma(g(0))(\tilde{v}_g)$ and then $H_{v,\gamma} := G_{\tilde{v},\omega_{v,\gamma}}$. Then (32) follows directly from (31). □

We will now give a computational version of Lemma 3.2 as a whole:

Lemma 9.2. *Suppose that H satisfies (32) and that v and γ satisfy $C(v, \gamma, H_{v,\gamma}(0), \tilde{v}_{H_{v,\gamma}})$, which analogously to before abbreviates*

$$\tilde{v}_{H_{v,\gamma}(0)} \lhd v_{H_{v,\gamma}(0)} \to \underbrace{(\exists i < j < \gamma(H_{v,\gamma}(0))(\tilde{v}_{H_{v,\gamma}}))(([v](H_{v,\gamma}(0)) * \tilde{v}_{H_{v,\gamma}})_i \leq_* ([v](H_{v,\gamma}(0)) * \tilde{v}_{H_{v,\gamma}})_j)}_{\neg \bar{P}([v](H_{v,\gamma}(0))*\tilde{v}_{H_{v,\gamma}}, \gamma(H_{v,\gamma}(0))(\tilde{v}_{H_{v,\gamma}}))} . \tag{33}$$

Then we have $\neg \bar{P}(v, H_{v,\gamma}(\gamma(H_{v,\gamma}(0))(\tilde{v}_{H_{v,\gamma}})) + 2)$ i.e.

$$(\exists i < j < H_{v,\gamma}(\gamma(H_{v,\gamma}(0))(\tilde{v}_{H_{v,\gamma}})) + 2)(v_i \leq_* v_j).$$

Proof. This follows directly from Lemma 3.2. First of all, we define $g := H_{v,\gamma}$, then the sequence w in Lemma 3.2 becomes identified with $[v](H_{v,\gamma}(0)) * \tilde{v}_{H_{v,\gamma}}$, and so setting $k := \gamma(H_{v,\gamma}(0))(\tilde{v}_{H_{v,\gamma}})$, the equation (33) is just (3), while (32) is just (4), and so by the lemma there exists some $i < j < g(k) + 2$ such that $v_i \leq_* v_j$. But since $g(k) + 2 = H_{v,\gamma}(\gamma(H_{v,\gamma}(0))(\tilde{v}_{H_{v,\gamma}})) + 2$ we're done. □

What we have shown above is that if $C(v, \gamma, H_{v,\gamma}(0), \tilde{v}_{H_{v,\gamma}})$ then $\neg \bar{P}(v, H_{v,\gamma}(\gamma(H_{v,\gamma}(0))(\tilde{v}_{H_{v,\gamma}})) + 2)$, or in other words,

$$\bar{P}(v, H_{v,\gamma}(\gamma(H_{v,\gamma}(0))(\tilde{v}_{H_{v,\gamma}})) + 2) \wedge C(v, \gamma, H_{v,\gamma}(0), \tilde{v}_{H_{v,\gamma}})$$

must be false. But this is just the conclusion of the function interpretation of $\mathrm{ZL}_{\mathrm{lex}}$ for $Nv\gamma = H_{v,\gamma}(\gamma(H_{v,\gamma}(0))(\tilde{v}_{H_{v,\gamma}})) + 2$, $Mv\gamma = H_{v,\gamma}(0)$ and $Wv\gamma = \tilde{v}_{H_{v,\gamma}}$, and so for any v, γ and n satisfying (23) we must have $\neg \bar{P}(u, n)$, which is exactly what we want! Let's make this formal.

Theorem 9.3. *Define $N, M : (X^*)^{\mathbb{N}} \to (\mathbb{N} \to (X^*)^{\mathbb{N}} \to \mathbb{N}) \to \mathbb{N}$ and $W : (X^*)^{\mathbb{N}} \to (\mathbb{N} \to (X^*)^{\mathbb{N}} \to \mathbb{N}) \to (X^*)^{\mathbb{N}}$ by*

$$Nv\gamma := H_{v,\gamma}(\gamma(H_{v,\gamma}(0))(\tilde{v}_{H_{v,\gamma}})) + 2$$
$$Mv\gamma := H_{v,\gamma}(0)$$
$$Wv\gamma := \tilde{v}_{H_{v,\gamma}}$$

where H is some functional which satisfies (32). Define $P(s) :\equiv (\forall i < j < |s|)(s_i \not\leq_* s_j)$ so that $\bar{P}(u, k) \equiv (\forall i < j < k)(u_i \not\leq_* u_j)$, and let n, v and γ be such that they satisfy the functional interpretation of $\mathrm{ZL}_{\mathrm{lex}}$ relative to N, M, W defined above:

$$\bar{P}(u, n) \to \bar{P}(v, Nv\gamma) \wedge C(v, \gamma, Mv\gamma, Wv\gamma) \tag{34}$$

Then we have $\neg \bar{P}(u, n)$ and hence

$$(\forall u)(\exists i < j < n)(u_i \leq_* u_j).$$

Proof. As shown above, if H satisfies (32) and $C(v, \gamma, Mv\gamma, Wv\gamma)$, then by Lemma 9.2 we have $\neg \bar{P}(v, Nv\gamma)$, which implies $\neg(\bar{P}(v, Nv\gamma) \wedge C(v, \gamma, Mv\gamma, Wv\gamma))$, and so by the contrapositive of (34) we have $\neg \bar{P}(u, n)$. □

Therefore any program which computes v, γ and n on any u, N, M and W, in particular that of Theorem 8.3, can be converted to a program which realizes $\mathrm{WQO}(\leq_*)$:

Corollary 9.4. *Suppose that H satisfies (32), and define $Nv\gamma := H_{v,\gamma}(\gamma(H_{v,\gamma}(0))(\tilde{v}_{H_{v,\gamma}})) + 2$. Then provably in CONT the functional $\Phi : (X^*)^{\mathbb{N}} \to \mathbb{N}$ defined by*

$$\Phi(u) := \Psi^N(u)$$

where $\Psi^N(u)$ is defined as in Theorem 8.3 witnesses $\mathrm{WQO}(\leq_*)$ i.e.

$$(\exists i < j < \Phi(u))(u_i \leq_* u_j).$$

Corollary 9.5. *Let H be defined as in Lemma 9.1 for G as defined in Section 5. Then $\Phi : (\mathbb{B}^*)^{\mathbb{N}} \to \mathbb{N}$ as defined in Corollary 9.4 witnesses* $\mathrm{WQO}(=_{\mathbb{B},*})$ *i.e.*

$$(\exists i < j < \Phi(u))(u_i =_{\mathbb{B},*} u_j).$$

Remark 9.6. To construct our realizer for $\mathrm{WQO}(\leq_*)$ we have only used the first component n of the full functional interpretation n, v, γ of $\mathrm{ZL}_{\mathrm{lex}}$. This makes sense: We actually prove via Lemma 9.2 that $(\exists v, \gamma)Q(v, \gamma) \to \bot$ where $Q(v, \gamma)$ abbreviates the conclusion of (34), and so to realize Higman's lemma we in fact only need to produce some functional Φ such that $\bar{P}(u, \Phi(u)) \to (\exists v, \gamma)Q(v, \gamma)$, and so the full computational interpretation of $\mathrm{ZL}_{\mathrm{lex}}$ via learning procedures was not strictly necessary. However, this simply emphasises the fact that we have achieved much more that a realizer for Higman's lemma - Theorem 8.3 allows us to extract a program from *any* proof which uses $\mathrm{ZL}_{\mathrm{lex}}$, and in general this program may well need a specific v and γ satisfying $Q(v, \gamma)$, even though here that was not the case. There is a further point to be made in this direction - namely that the concrete witnesses for v and γ enables us to *verify* our realizer $\Phi(u)$ in a quantifier-free theory, a fact that is relevant to those inclined towards foundational issues.

Before we conclude, it is worth pausing for a moment and trying to explain from an algorithmic point of view what the realizer we get in Corollary 9.4 actually does. Note that all of the following is essentially just an informal recapitulation of ideas contained in the preceding results. Roughly speaking, $\Phi(u)$ encodes a program which works by recursion on the lexicographic ordering $\rhd_{\mathrm{lex}}$. First, it finds the point m_0 such that

$$N([u]_{m_0})([\gamma_u]_{m_0}) < m_0$$

where $\gamma_u := \lambda n, w \,.\, \Phi([u](n) * w)$ if $w_0 \lhd u_n$, and so in particularly it only looks at the sequence u at points $n < m_0$. For simplicity let's define $u', \gamma' := [u]_{m_0}, [\gamma_u]_{m_0}$. Now, using any program H which realizes $\mathrm{WQO}_{\mathrm{seq}}(\leq)$ on the sequence $\bar{u}'$ we find a sufficiently large approximation $H_{u',\gamma'}$ to a constant subsequence, which works up to the point $\gamma'(H_{u',\gamma'}(0))(\tilde{u}'_{H_{u',\gamma'}})$.

Now if $\tilde{u}'_{H_{u',\gamma'}(0)} \lhd u'_{H_{u',\gamma'}(0)}$ then we must have $H_{u',\gamma'}(0) < m_0$ (using our assumption that 0_X is chosen to be minimal with respect to $\lhd$) and so $\gamma'(H_{u',\gamma'}(0))(\tilde{u}'_{H_{u',\gamma'}}) = \Phi([u'](H_{u',\gamma'}(0)) * \tilde{u}'_{H_{u',\gamma'}})$. Assuming inductively that this returns a bound for $[u'](H_{u',\gamma'}(0)) * \tilde{u}'_{H_{u',\gamma'}}$ being a good sequence, then using reasoning as in Lemma 3.2 this means that u' becomes a good sequence before point $H_{u',\gamma'}(\Phi([u'](H_{u',\gamma'}(0)) * \tilde{u}'_{H_{u',\gamma'}})) + 2 = Nu'\gamma'$. But since $\Phi(u) = Nu'\gamma' < m_0$ and $u' = [u]_{m_0}$ then this means also that u is good before $\Phi(u)$.

To verify that $\Phi([u'](H_{u',\gamma'}(0)) * \tilde{u}'_{H_{u',\gamma'}})$ returns a bound, we can repeat this argument for $u_1 := [u'](H_{u',\gamma'}(0)) * \tilde{u}'_{H_{u',\gamma'}}$, and we end up with a learning procedure as in Section 8. Eventually, this learning procedure will terminate with a minimal sequence v such that $\Phi(v)$ is guaranteed to witnesses that v is good.

10 Conclusion

I will conclude by tying up everything that we've done and outlining some directions for future work. On the route to Corollary 9.5 we took what we hope was a pleasant and instructive detour through many different areas which connect proof theory and well quasi-order theory, the most important of which I will now summarise.

Right at the start, in Chapters 2-3, we discussed various nuances that arise when giving an axiomatic formalisation of results in WQO theory, in particular how the distinction between dependent choice or Zorn's lemma plays an important role in the context of program extraction. The full formalisation of Nash-Williams' minimal bad sequence argument has already been studied in e.g. [26] (in MINLOG) and [28] (in Isabelle/HOL), and we hope that our formal proof sketched in Chapter 3 may prove informative to those working in a more hands-on manner on the formalisation of WQO theory in proof assistants.

In Chapters 4-5 we took the opportunity to present Gödel's functional interpretation in a way that would appeal to readers not already familiar with it. In Section 4.4 we placed particular emphasis on explaining how the interpretation behaves in *practice*, and in this vein we gave a carefully worked out case study in

Section 5, which also formed a key Lemma in our proof of WQO($=_{\mathbb{B},*}$). It is my sincere hope that these chapters will be a general help to those interested in how proof interpretations work, independently of the rest of the paper.

Chapters 6 and 8 contain our main technical contribution, namely the solution of the functional interpretation of ZL_{lex}. While as a direct consequence this enables us to extract a program witnessing WQO($=_{\mathbb{B},*}$), our work in these chapters is much broader, and provides us with a method of giving a computational interpretation to *any* proof that can be formalised in $PA^{\omega} + QF\text{-}AC + ZL_{\text{lex}}$, where moreover ZL_{lex} can involve any relation $(X, \rhd)$ which is provably wellfounded. In particular, this paves the way for the extraction of programs from much more complex proofs in WQO theory, such as Kruskal's theorem, and we intend to address this in future work.

Hidden in Chapters 6 and 8 is also a small extension of my work on *learning procedures* [23]. While in this paper they play the role of making our computational interpretation of ZL_{lex} more intuitive, Lemma 8.2 is of interest in its own right, as it demonstrates that we can extend the notion of a learning procedure as introduced in [23] to the non-wellfounded ordering $\rhd_{\text{lex}}$. We anticipate that a number of variants of open induction or Zorn's lemma over $\rhd_{\text{lex}}$ could be given computational interpretations by appealing directly to Lemma 8.2 as a intermediate result, and this was part of our motivation for stating it explicitly here.

Sandwiched between these sections is Chapter 7, which itself forms a small essay on higher-type computability theory, and the various ways of carrying out recursion over $\rhd_{\text{lex}}$ in the continuous functionals. There are a number of interesting questions to be answered in this direction. Firstly, what is the relationship between EORec, Berger's open recursion and the many variants of bar recursion which have been devised in the context of proof theory? I have already shown that Berger's open recursion is primitive recursively equivalent to *modified bar recursion* [5] and thus strictly stronger than Spector's original bar recursion, but I conjecture that in contrast, EORec is equivalent to Spector's bar recursion and thus weaker than Berger's open recursion. This would also imply that EORec exists in the type structure of $\mathcal{M}^{\omega}$ strongly majorizable functionals, and so does not necessarily rely on continuity to be a wellfounded form of recursion. It would be interesting to explore some of these issues in the future.

Finally, we should not forget that in Chapter 9 we gave a new program which witnesses Higman's lemma, that works not just for the $=_{\mathbb{B}}$, but for *any* WQO for which a realizer of WQO$_{\text{seq}}(\leq)$ can be given. Moreover, in contrast to [21], this realizer encodes a recursive algorithm which seems to do something fundamentally intuitive. The precise relationship between this algorithm and the many others which have been offered over the years is a question we leave open for now, although the presence of the control functional in the explicit open recursor leads me to conjecture that it is genuinely different to most of them. But for now we simply hope that, among other things, we have provided a little more insight into the computational meaning of Nash-Williams' elegant classical proof.

References

[1] http://www.mathematik.uni-muenchen.de/~logik/minlog/. Official homepage of MINLOG, as of January 2018.

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

[3] 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.

[4] U. Berger. A computational interpretation of open induction. In *Proceedings of LICS 2004*, pages 326–334. IEEE Computer Society, 2004.

[5] U. Berger and P. Oliva. Modified bar recursion and classical dependent choice. *Lecture Notes in Logic*, 20:89–107, 2005.

[6] Y. L. Ershov. Model *C* of partial continuous functionals. In *Logic Colloquium*, pages 455–467. North Holland, Amsterdam, 1977.

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

[8] M. Escardó and P. Oliva. Sequential games and optimal strategies. *Royal Society Proceedings A*, 467:1519–1545, 2011.

[9] M. Escardó and P. Oliva. Computing Nash equilibria of unbounded games. In *Proceedings of the Turing Centenary Conference, Manchester*, volume 10 of *EPiC Series*, pages 53–65, 2012.

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

[11] G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 2:326–336, 1952.

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

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

[14] G. Kreisel. On the interpretation of non-finitist proofs, Part I. *Journal of Symbolic Logic*, 16:241–267, 1951.

[15] G. Kreisel. On the interpretation of non-finitist proofs, Part II: Interpretation of number theory. *Journal of Symbolic Logic*, 17:43–58, 1952.

[16] G. Kreisel. Interpretation of analysis by means of functionals of finite type. In A. Heyting, editor, *Constructivity in Mathematics*, pages 101–128. North-Holland, Amsterdam, 1959.

[17] J. Longley and D. Normann. *Higher-Order Computability*. Theory and Applications of Computability. Springer, 2015.

[18] C. R. Murthy. *Extracting Constructive Content from Classical Proofs*. PhD thesis, Ithaca, New York, 1990.

[19] C. St. J. A. Nash-Williams. On well-quasi-ordering finite trees. *Proceedings of the Cambridge Philosophical Society*, 59:833–835, 1963.

[20] 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.

[21] T. Powell. Applying Gödel's Dialectica interpretation to obtain a constructive proof of Higman's lemma. In *Proceedings of Classical Logic and Computation '12*, volume 97 of *EPTCS*, pages 49–62, 2012.

[22] T. Powell. The equivalence of bar recursion and open recursion. *Annals of Pure and Applied Logic*, 165(11):1727–1754, 2014.

[23] T. Powell. Gödel's functional interpretation and the concept of learning. In *Proceedings of Logic in Computer Science (LICS 2016)*, pages 136–145. IEEE Computer Society, 2016.

[24] J.-C. Raoult. Proving open properties by induction. *Information Processing Letters*, 29:19–23, 1988.

[25] D. S. Scott. Outline of a mathematical theory of computation. In *4th Annual Princeton Conference on Information Sciences and Systems*, pages 169–176, 1970.

[26] M. Seisenberger. *On the Constructive Content of Proofs*. PhD thesis, Ludwig Maximilians Universität München, 2003.

[27] 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.

[28] C. Sternagel. Certified Kruskal's tree theorem. In *Proceedings of the 3rd International Conference on Certified Programs and Proofs (CPP '13)*, volume 8307 of *LNCS*, pages 178–193. 2013.

[29] A. S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer, Berlin, 1973.