Exploring the Computational Content of the Infinite Pigeonhole Principle

Diana Ratiu Trifon Trifonov

Mathematics Institute
Ludwig Maximilian University of Munich, Germany
{ratiu,trifonov}@math.lmu.de

Abstract

The use of classical logic for some combinatorial proofs, as it is the case with Ramsey's theorem, can be localized in the Infinite Pigeonhole (IPH) principle, stating that any infinite sequence which is finitely colored has an infinite monochromatic subsequence. Since in general there is no computable functional producing such an infinite subsequence, we consider a $\Pi^0_2$-corollary, proving the classical existence of a finite monochromatic subsequence of any given length. In order to obtain a program from this proof, we apply two methods for extraction: the refined A-Translation, as proposed by Berger et al., and Gödel's Dialectica interpretation. In this paper, we compare the resulting programs with respect to their behavior and complexity and indicate how they reflect the computational content of IPH.

1 Introduction

In the case of intuitionistic proofs, the semantics associated to the logical operators allows for an interpretation of proofs as programs, via the Curry-Howard correspondence. In the classical setting, on the other hand, the computational content of proofs is not explicit, because the existential quantifier can be viewed only as a shortcut for $\neg\forall\neg$. However, it has been shown that the $\Pi^0_2$-formulas $\forall x\exists y G(x, y)$, with arithmetical and decidable kernel, are equiderivable in the two systems; consequently, various methods for retrieving the computational content from the classical proofs have been proposed, beginning as early as the 1930’s with the works of Gödel and Gentzen.

Experiments with such methods have sometimes yielded surprising and counterintuitive, yet correct and efficient algorithms. However, it should not be expected that treating a non-constructive proof will generally lead to a fast program; only its correctness is theoretically guaranteed and this is the main benefit of the extraction technique. Proving a $\Pi^0_2$-statement classically can be considered as using a shortcut in an argument that can be carried out constructively. This can make the proof significantly easier and more comprehensible, at the

*The authors gratefully acknowledge financial support by MATHLOGAPS (MEST-CT-2004-504029), a Marie Curie Early Stage Training Site.
cost of hiding its computational content. Even though methods for extraction from non-constructive proofs are able to recover a correct program, it usually reflects the use of classical logic by being indirect and possibly inefficient.

In this paper, we present — and compare — two such methods: (1) a refined version of the Friedman/Dragalin A-Translation method, [Friedman(1978)], combined with the modified realizability, and (2) Gödel’s Dialectica interpretation, [Gödel(1958)]. Whereas the A-Translation isolates the computational content in the use of ⊥, treated in this work as a predicate variable, Dialectica simulates the game of providing witnesses and counterexamples for the formulas. In order to compare the two methods on a real-world problem, we have chosen a well-known example from the realm of combinatorics.

Lemma 1.1 (Infinite Pigeonhole Principle). Any infinite sequence colored with finitely many colors has (at least) one color occurring infinitely often.

This principle plays an important role in proving the infinite version of Ramsey’s Theorem, which makes it an important component of the expected computational content of this combinatorial result.

Theorem 1.2 (Infinite Ramsey’s Theorem). Let \( n \in \mathbb{N} \setminus \{0\} \) and \( S \) be an infinite set\(^1\). Let colors from some finite set be associated with the \( n \)-element subsets of \( S \). Then there exists an infinite set \( M \subseteq S \), which is homogeneous, i.e., the \( n \)-element subsets of \( M \) all have the same color.

Of course, one cannot expect to be able to associate a computable functional with the Infinite Pigeonhole Principle, since this would amount to producing the infinite subsequence by inspecting infinitely many elements of a given one, which is clearly not a decidable problem. In other words, Lemma 1.1 does not have a constructive proof, so we restrict our attention to a \( \Pi^0_2 \)-consequence.

Corollary 1.3 (Unbounded Pigeonhole Principle (UPH)). For any infinite sequence \( f \), colored with \( r \in \mathbb{N} \) colors and any \( n \in \mathbb{N} \), one can find a finite monochromatic subsequence of \( f \) of length \( n \).

The paper is structured as follows. We begin the next section with an exposition of the underlying system as a negative fragment of Heyting’s Arithmetic. Subsections 2.2 and 2.3 review the two extraction methods, emphasizing the aspects relevant to the present paper. Section 3 begins with the formulation and an informal proof of the IPH. The understanding of this proof should enable the reader to trace in the coming subsections the way in which A-Translation and Dialectica operate on the proofs, in order to synthesize the associated functionals. In Section 4 we present average runs of the extracted programs on different parameters and make a thorough comparison of the two methods from the point of view of the time complexity of the resulting programs. We also comment on the drawbacks of the methods, while at the same time presenting the advantages of one over the other. Section 5 overviews the related work and points out the contribution of this paper in correlation with the existing results. We conclude with a sketch of ongoing work, in which we take the outcomes of this paper further to an analysis of Ramsey’s Theorem. 

\(^1\)Ramsey’s Theorem is most commonly stated in terms of graphs: \( S \) is taken to be an infinite complete graph, \( M \) a subgraph of \( S \) and \( n \) the number of edges
2 Extraction from classical proofs

2.1 The system NAω

We work in Heyting arithmetic with finite types (HAω) (cf. [Troelstra(1973)]), restricted to the language of → and ∀, and refer the resulting system as Negative arithmetic (NAω).

Definition 2.1. Types (ρ, σ) and (object) terms (s, t) are defined inductively

\[
\begin{align*}
\rho, \sigma &::= B \mid N \mid L \mid \rho \Rightarrow \sigma \mid \rho \times \sigma \\
s, t &::= x^\rho \mid (\lambda x^\rho) t^\rho \Rightarrow \sigma \mid (s^\rho, t^\rho)\sigma \mid (s^\rho, t^\rho)\rho \times \sigma \mid (\rho \Rightarrow \sigma \Rightarrow \sigma) \mid (\rho \Rightarrow \sigma)\sigma \mid \text{ff}^B \mid \text{ff}^N \mid \text{ff}^L \mid (n^N : l^L) \\
\end{align*}
\]

The base types of booleans B, natural numbers N and lists of natural numbers L are equipped with the usual constructors and Gödel structural recursor constants.

We will use (n::) as a shortcut for (n::nil), x, y, z to denote typed object variables and P for a predicate variable of arity σ. We define ⊥ as a nullary predicate variable. The predicate at(·) allows us to consider any boolean valued function defined in our term system as a decidable predicate.\(^2\) We let F := at(ff) denote the (arithmetical) falsity.

Definition 2.2. With the above conventions, formulas (A,B) are defined to be

\[
\begin{align*}
A, B &::= \text{at}(t^B) \mid P(s^F) \mid \bot \mid A \Rightarrow B \mid \forall x^\rho A
\end{align*}
\]

We regard the classical (or weak) logical operators as abbreviations

\[
\begin{align*}
\tilde{\land} A &::= A \Rightarrow \bot \\
\tilde{\forall} x A &::= (\forall x. A \Rightarrow \bot) \Rightarrow \bot
\end{align*}
\]

We denote the conjunction of A₁, ..., Aₙ by ̃A. If ̃A is the premise of the implication, we force it to unfold to implications, by using the special operator ̃·, and thus have

\[
\tilde{A} \Rightarrow B := (A₁ \tilde{\land} ... \tilde{\land} Aₙ) \Rightarrow B := A₁ \Rightarrow ... \Rightarrow Aₙ \Rightarrow B
\]

The sets of free variables FV(t), FV(A) and free predicate variables FP(A) are defined inductively as usual. Substitution of terms for object variables s[x/t], A[x/t] and of formulas for predicate variables A[P(\(\overline{x}\))/B(\(\overline{x}\))] are always assumed to be capture-free with respect to abstraction and quantification.

Notation 2.3. ∀ and λ will be assumed to bind stronger than → and application. Occasionally, we will use a dot notation such as ∀x. A → B and λx. st to override this convention. For technical convenience we will use ε to denote a special nulltype and by abuse of notation we use ε also for the terms of this nulltype. We stipulate that the following simplifications are always carried out implicitly:

\[
\begin{align*}
\rho \times \varepsilon &\sim \rho, & \text{t}0 \sim t, & (t, \varepsilon) \sim t & p \Rightarrow \varepsilon &\sim \varepsilon, & \lambda x \varepsilon \sim \varepsilon, & \varepsilon t &\sim \varepsilon \\
\varepsilon \times \rho &\sim \rho, & \text{t}1 \sim t, & (\varepsilon, t) \sim t & \varepsilon &\sim \rho \Rightarrow \rho, & \lambda x \varepsilon t &\sim t, & t \varepsilon &\sim t & (\varepsilon) \\
\forall x \varepsilon A &\sim A, & M \varepsilon &\sim M
\end{align*}
\]

\(^2\)When we write for example n < m, we formally mean at(Lₙ n m), where Lₙ B≡nil is a term defining the decidable relation <.
The operational semantics of the object terms is given by the usual rule of β-reduction \((\lambda xs)t \mapsto s[x/t]\) and computation rules for the recursion constants:

\[
\begin{array}{|c|c|}
\hline
C \; tt \; t_1 \; t_2 & \mapsto t_1 \\
\hline
\hline
Cf \; t_1 \; t_2 & \mapsto t_2 \\
\hline
R_N \; (Sn) \; s \; t & \mapsto tn (R_N \; n \; s \; t) \\
\hline
R_L \; (n::l) \; s \; t & \mapsto tn l (R_L \; l \; s \; t)
\end{array}
\] (1)

The term system in consideration is essentially Gödel’s T and is well known to be strongly normalizing and confluent. Without loss of generality we assume that all considered terms are in normal form.

We express derivations in a natural deduction system using a similar syntax to that of object terms, in order to stress the Curry-Howard correspondence. However, proof terms are typed by their conclusions and are built from assumption variables.

**Definition 2.4.** Proof terms \((M,N)\) of \(\text{NA}^\omega\) are defined as follows:

\[
M, N ::= u^A | (\lambda u^A M^B)^{A \rightarrow B} | (M^{A \rightarrow B} N^A)^B | (\forall x^B M^{\forall x^B A(x)})^{A(t)} | \text{AxT} : \text{at}(tt) | \text{Ind}_{a,A(n)}^{\forall n^B, A(0)} \rightarrow \forall n^B (A(n) \rightarrow A(Sn)) \rightarrow A(n) | \text{Ind}_{b,A(b)}^{\forall n^B, A(n)} \rightarrow \forall n^B, A(b) \rightarrow A(b) | \text{Ind}_{l,A(l)}^{\forall n^B, A(n)} \rightarrow \forall n^B, A(l) \rightarrow A(l :: n) \rightarrow A(l) \rightarrow A(l) \rightarrow A(l)
\]

with the usual variable condition on \((*)\) that the object variable \(x\) does not occur freely in any of the open assumptions of \(M\). The sets of free variables \(FV(M)\) and free (open) assumption variables \(FA(M)\), as well as capture-free substitutions \(M[x/t]\) and \(M[u/N]\), are defined inductively as usual.

Since we view the symbol \(\bot\) only as a placeholder, to be later substituted with an arbitrary formula, we have no special rules for it and thus choose to work in a minimal logic setting. However, substituting \(\bot\) with the closed atom \(F\) we gain back the full power of classical logic. In particular, for every formula \(A\) with no predicate variables we can prove

\[
\text{NA}^\omega \vdash F \rightarrow A \quad (\text{efq})
\]

by meta induction on \(A\), using \(\text{AxT}\) and \(\text{Ind}_{a,A(n)}\) for the base case.

### 2.2 Modified Realizability on A-Translated Proofs

In order to interpret the classical proofs of \(\Pi^0_2\)-formulas by the modified realizability, we need to first translate them into their intuitionistic counterparts. For this, we use the refinement of the A-Translation method, as introduced in [Berger et al.(2002)], which we overview in its relevant details in what follows.

#### 2.2.1 Refined A-Translation

The method known as A-Translation is a combination of the double negation translation, as proposed by Gödel and Gentzen, and the method known as Friedman’s “trick”. The former consists, roughly speaking, in double negating all atoms of the formulas, whereas the latter proceeds on those proofs by substituting \(\bot\) with an arbitrary formula \(A\). Since \(\bot\) is defined as a predicate
variable and is viewed only as a placeholder for an arbitrary formula \( A \), special care needs to be taken for the axioms \( \text{efq} \) and \( \text{Stab} \), as pointed out in the introductory Section 1.

In order to minimize the occurrences of \( \bot \) in the proof, however, we identify the classes of formulas for which the insertion of double negations is not necessary. Moreover, we will substitute, whenever possible, \( \bot \) by the arithmetical falsity \( F \). The following constraints on the formulas will serve this purpose.

**Definition 2.5** (Decidable formulas\(^3\)). A formula is said to be **decidable** if it is built from atomic formulas, at(\( t \)), only by propositional connectives and quantifiers, which are either boolean or range over a finite set of naturals.

With these restrictions, case distinction is permitted on the decidable formulas.

**Lemma 2.6** (Case Distinction). If \( A \) is an arbitrary formula and \( B \) a decidable formula, then\(^4\)

\[
\vdash (B \rightarrow F) \rightarrow (B \rightarrow A) \rightarrow A. \quad \text{(Cases)}
\]

**Definition 2.7.** A formula \( C \) is said to be **(computationally) relevant** iff it ends with \( \bot \) and irrelevant otherwise, i.e., the following are the relevant formulas

\[
C := \bot \mid A \rightarrow C \mid \forall xC.
\]

We define **goal formulas** \( G \) and **definite formulas** \( D \) inductively. Let \( P \) range over atomic formulas and \( \bot \).

\[
G := P \mid D \rightarrow G, \quad \text{provided } D \text{ relevant or } D \text{ decidable},
\]

\[
\mid \forall xG, \quad \text{provided } G \text{ irrelevant},
\]

\[
D := P \mid G \rightarrow D, \quad \text{provided } D \text{ relevant or } G \text{ irrelevant},
\]

\[
\mid \forall xD.
\]

**Remark.** This choice for definite and goal formulas is enforced by the special treatment of \( \bot \), which is to be substituted with a formula that has computational relevance for the extracted program. By isolating the occurrences of \( \bot \) for which this substitution is not necessary, \( \bot \) can be safely substituted by \( F \), as the following lemma shows, and thus some redundancies arising from the use of \( \bot \) can be eliminated.

**Lemma 2.8.** Let \( A^F \) denote \( A[\bot := F] \). For definite formulas \( D \) and goal formulas \( G \) we have derivations from \( F \rightarrow \bot \) of\(^5\)

\[
\vdash D^F \rightarrow D, \quad \text{(2)}
\]

\[
\vdash G \rightarrow (G^F \rightarrow \bot) \rightarrow \bot. \quad \text{(3)}
\]

Furthermore, (3) can be extended for a conjunction of formulas \( \vec{G} \) to

\[
\vdash \vec{G} \rightarrow (\vec{G}^F \rightarrow \bot) \rightarrow \bot.
\]

---

\(^3\)This definition is in the same fashion as in [Seisenberger(2003)].

\(^4\)For the proof, please refer to [Berger et al.(2002)].

\(^5\)Please refer to [Schwichtenberg(2007)] for the complete statement and proof of this lemma.
Having “eliminated” $\bot$ from the definite and goal formulas, we can now substitute $A := \exists y \, G(x, y)$ for the remaining occurrences of $\bot$, as in the Friedman-Dragalin A-Translation method\(^6\). The following lemma summarizes this mechanism of “refined A-Translation”.

**Lemma 2.9.** Consider that we are given

\[
\begin{align*}
\vdash \forall x. \vec{D} \to \vec{H} \to \exists y \, G(x, y), \\
\end{align*}
\]

where $G$ is a goal, $\vec{D}$ are definite and $\vec{H}$ are arbitrary formulas. Then

\[
\begin{align*}
\vdash \forall x. \vec{D}^F \to \vec{H}[\bot := \exists y \, G^F(x, y)] \to \exists y \, G^F(x, y).
\end{align*}
\]

**Proof.** (Sketch) Suppose we are given the proof of

\[
\begin{align*}
\forall x. \vec{D} \to \vec{H} \to (\forall y. \, G(x, y) \to \bot) \to \bot,
\end{align*}
\]

with $\vec{D}$ definite assumptions and $\vec{H}$ arbitrary formulas.

By Lemma 2.8, this can be transformed to a derivation of

\[
\begin{align*}
(F \to \bot) \to \forall x. \vec{D}^F \to \vec{H} \to (\forall y. \, G^F(x, y) \to \bot) \to \bot.
\end{align*}
\]

By substituting in the latter $\exists y \, G^F(x, y)$ for $\bot$, and since

\[
\begin{align*}
\vdash F \to \exists y \, G^F(x, y) \text{ and } \vdash \forall y. \, G^F(x, y) \to \exists y \, G^F(x, y),
\end{align*}
\]

we obtain (5). \(\square\)

The transformed proof can be now interpreted by means of the modified realizability, which we describe in the next subsection. By this, we will synthesize the algorithmic content, recovered from the initial classical proof by the A-Translation mechanism.

### 2.2.2 Realizability

Let in the following $|A|^r$, which reads “the term $r$ realizes the formula $A$”, denote \((Kreisel’s)\) modified realizability. We assign a computational type $\alpha_P$ and a (new) predicate variable $P^r$ of arity $(\alpha_P, \vec{\sigma})$ to every predicate variable $P$ of arity $\vec{\sigma}$ and define the modified realizability inductively.

\[
|P(\vec{s})|^r = \begin{cases} 
P^r(r, \vec{s}) & \text{if } P \text{ is a predicate variable with comp. type } \alpha_P \\
P(\vec{s}) & \text{if } P \text{ is a predicate constant} 
\end{cases}
\]

\[
|\forall x. A|^r = \forall x \, |A|^x^x
\]

\[
|A \to B|^r = \forall x. \, |A|^x \to |B|^x
\]

\(^6\)The original A-Translation method uses Friedman’s trick on double-negation translated proofs, substituting $\bot$ by an arbitrary formula $A$. 

Terminology and notations  We assign to every formula $A$ its computational type $\tau(A)$. In the special case when $A$ is computationally irrelevant, i.e., the proof $M$ of $A$ has no computational content, we mark this by assigning the type $\varepsilon$ and let $\llbracket M \rrbracket := \varepsilon$, with $\llbracket M \rrbracket$ denoting the program extract from the proof of $A$. When $\tau(A) \neq \varepsilon$, $A$ is called a computationally relevant formula. The definition of the extracted program below makes use of the rules ($\varepsilon$).

**Definition 2.10.** The extracted term of a derivation is given by

\[
\llbracket uA \rrbracket := x_{\tau(u)}^{\tau(A)} (x_{\tau(u)}^{\tau(A)} \text{ uniquely associated with } uA),
\]

\[
[(\lambda uA.M)^{A \rightarrow B}] := \lambda x_{\tau(u)}^{\tau(A)}[M],
\]

\[
[M^{A \rightarrow B}] := [M][N],
\]

\[
[\lambda x^\varepsilon M] := \lambda x^\varepsilon[M],
\]

\[
[M^{\forall x^A r}] := [M]^r,
\]

In particular, for our induction schemes, when $\sigma := \tau(A) \neq \varepsilon$, we have

\[
[\text{Ind}_{\varepsilon, A}] := \mathcal{R}_{\varepsilon}^{\neg \varepsilon \Rightarrow \sigma \Rightarrow (\neg \varepsilon \Rightarrow \sigma) \Rightarrow \sigma}, \quad [\text{Ind}_{\lambda, A}] := \mathcal{R}_{\lambda}^{\neg \varepsilon \Rightarrow \sigma \Rightarrow (\neg \varepsilon \Rightarrow \sigma) \Rightarrow \sigma},
\]

with the conversion rules from (1) in Section 2.1.

The interpretation of the realiser for the case distinction, “Cases”,

\[
[\text{Cases}_{n, A}] := C_{n, \sigma}(n, f, g) = \text{if } (n = 0) \text{ then } f \text{ else } g(n - 1),
\]

\[
C_{l, \sigma}(l, f, g) = \text{if } (l = \text{nil}) \text{ then } f \text{ else } g(l_0, \text{cdr } l),
\]

with (cdr $l$) denoting the list $l$ without its head element $l_0$.

**Theorem 2.11 (Soundness [Berger et al.(2002)])**. Assume that $M$ is a derivation of $B$. Then there is a derivation of $\llbracket B \rrbracket^{\llbracket M \rrbracket}$ from the assumptions

\[
\{ [C]^r_{\tau(C)} \mid uC \in \text{FA}(M) \}.
\]

Let $\hat{M}$ be the derivation of (4) and $\llbracket M \rrbracket$ denote the program extracted from it. In order to determine the realizer for (5), let us first assume that we have terms $s$ and $t$ realizing $H$ and $D$, respectively, and the proofs that they indeed are the realizers

\[
\vdash H \rightarrow \llbracket H[\perp / \exists yG^F(y)] \rrbracket^s, \quad \vdash D \rightarrow \llbracket D^F \rrbracket^t.
\]

Then, by the Soundness Theorem for realizability we can derive

\[
H \rightarrow D \rightarrow \llbracket G^F([M]^s t) \rrbracket^{[M]^s t}. \]

and further

\[
H \rightarrow D \rightarrow \llbracket G^F([M]^s t 1) \rrbracket^{[M]^s t 0}.
\]
2.3 Dialectica interpretation

Gödel’s functional (Dialectica) interpretation [Gödel(1958)] was originally intended to embed classical logic in a quantifier-free system employing functionals of higher finite types. In its usual formulation, one first applies the Gödel-Gentzen negative translation to a classical proof, thus obtaining a proof in minimal logic. Then a refining process is used, in order to separate the two components: the positive computational information (a tuple of witnessing functionals) depending on negative parameters (a tuple of challenging variables) on one hand, and a proof of its quantifier-free specification on the other.

We follow the treatment from [Schwichtenberg(2008)] and use pair types instead of tuples, thus translating each formula $A$ to a quantifier-free formula $|A|^r$, relating a realizing variable $x : A^+$ to a challenging variable $y : A^-$. We refer to the types $A^+$ and $A^-$ as the positive and the negative computational types of $A$, respectively. The Dialectica interpretation starts from a proof $M$ in minimal logic and produces a witnessing term $t$, not containing the challenging variable $y$ freely, together with a verifying proof of $\forall y \, |A|^r_y$.

**Definition 2.12.** For a formula $A$ without predicate variables we define the positive ($A^+$) and negative ($A^-$) computational types and the Dialectica translation $|A|^r_s$ by simultaneous induction

\[
\begin{array}{c|c|c|c}
A & A^+ & A^- & |A|^r_s \\
\hline
\text{at}(b) & \varepsilon & \varepsilon & \text{at}(b) \\
A \to B & (A^+ \Rightarrow B^+) \times A^+ \times B^- & (A^+ \Rightarrow B^- \Rightarrow A^-) & |A|^r_{B_{s_0}(s_1)}(r_0)(s_0) \\
\forall \rho A(x) & \rho \Rightarrow A^+ & \rho \times A^- & |A(s)|_{s_0}^{r_0}(s_1)
\end{array}
\]

We make the silent convention that whenever we use the weak logical symbols (defined in Subsection 2.1) in a proof to be treated by Dialectica, we have substituted all predicate variables $\bot$ with the boolean falsity $F$.

The Dialectica interpretation can be viewed as extending realizability with negative content, where premises switch polarity. More specifically, while the realizability translation of $\forall x A$ preserves the quantification on the variable $x$, in Dialectica this is singled out as a part of the negative content of the formula $(s_0)$. As a result, we are able to track witnessing terms, which are used to eliminate negatively appearing $\forall$. In particular, from a proof of a weak existential formula $\exists x \rho A$ we can extract a witness for $x$. A similar effect is achieved by $A$-Translation by utilizing the computational content of $\bot$ to accumulate negative witnesses. In fact, as pointed out in [Hernest and Oliva(2008)], by following certain restrictions we can make local choices as to how to treat negative content, while effectively interleaving realizability and Dialectica in the same proof.

The soundness of the interpretation was earlier proved in a Hilbert-style setting [Troelstra(1973), Avigad and Feferman(1998)]. We followed a more recent treatment using natural deduction [Hernest(2006), Schwichtenberg(2008)].

**Theorem 2.13** (Soundness theorem). Let $M$ be a proof of the formula $A$ from assumptions $u_i : C_i$. Let $x_i : C_i^+$ be fresh witnessing variables for the assumptions and $y : A^-$ be a fresh challenging variable for the conclusion. Then there
are terms $[M]^-_i : C_i^- \text{ and } [M]^+_i : A^+$, such that $y \notin \text{FV}([M]^+_i)$ and $[A]^+_{y[M]^+_i}$ is provable from assumptions $|C_i|^x_i_{[M]^-_i}$.

Proof. The proof goes by induction on $M$.


Case $\lambda uA^+ N^B$. By IH we have a proof of $[B]_{\zeta}^{[N]^-}$ from assumptions $|C_i|^{x_i}_i_{[N]^-}$ and $[A]^+_{y[M]^+_i}$. We will show that $\xi$ is provable from assumptions $|C_i|_{[N]^-}$ and $[A]^+_{[N]^-}$. Substituting $\xi = [y_0, y_1/x_0, z]$ we can prove the formula

$$[A]^{y_0}_{[M]^-} \xi \rightarrow [B]^{[N]^-}_{\zeta}$$

from $|C_i|^{x_i}_i_{[N]^-} \xi$.

Hence, we set $[M]^- := [N]^- \xi$ and $[M]^+ := (\lambda x_0[N]^+, \lambda x_0, z[N]^-)$. Case $\lambda \rho N^A(x)$. By IH we have proofs of

$$[A \rightarrow B]^{[N]^-} \xi \text{ from assumptions } |C_i|^{x_i}_i_{[N]^-}$$

$$[A]^{\rho}_{[N]^-} \text{ from assumptions } |C_i|^{x_i}_i_{[N]^-}$$

Here we substitute $\xi_1 = \{([N]_1^+, y)/z\}$ and $\xi_2 = ([N]_1^+)^{1}[N]_2^+ y/u)$. However, if the two subproofs share assumption variables, we need that $|C_i|^{x_i}_i_{[N]^-}$ implies $|C_i|^{x_i}_i_{[N]^-} \xi_2$ for $j = 1, 2$ in order to be able to use the IH. At this point we should take advantage of the quantifier-free Dialectica translation. Consider the following case distinction:

$$t_1 \triangleright t_2 := \begin{cases} t_3 - j; & \text{if } u_i : C_i \notin FA(N_j) \\ t_1 \triangleright t_2 = t_2 & \text{otherwise.} \end{cases}$$

We will show that $|C_i|^{x_i}_i_{t_1 \triangleright t_2}$ implies $|C_i|^{x_i}_i_{t_2}$ for $j = 1, 2$ and $u_i \in FA(N_1) \cap FA(N_2)$ by case distinction. If $|C_i|^{x_i}_i_{t_1} \rightarrow F$, then $t_1 \triangleright t_2 = t_1$. Hence, we can derive $F$ and finish by using $\text{eq}$. Finally, we set $[M]^- := [N]^-0[N]_2^+; \quad [M]^+ := ([N]_i^-; t_1) \triangleright ([N]_2^-; \xi_2)$.

Case $\lambda \rho N^A(x)$. By IH we have a proof of $[A(x)]^{[N]^-}_{y}$ from assumptions $|C_i|^{x_i}_i_{[N]^-}$. Substituting $\xi = [y_0, y_1/x, z]$ we can prove the formula $[A(y_0)]^{y_1}_{y} \xi$ from $|C_i|^{x_i}_i_{[N]^-} \xi$. Hence, we can set $[M]^- := [N]_2^+ \xi$ and $[M]^+ := \lambda x[N]^+$.

Case $N^A \rightarrow A(x)\rho$. By IH we have a proof of $[A(z_0)]^{[N]^-}_{z_1}$ from assumptions $|C_i|^{x_i}_i_{[N]^-}$. Hence, we can set $[M]^- := [N]^+t$ and $[M]^+ := [N]_2^+ ([t, y]/z)$.

Case $\text{AxT; at(t)}$. Since at has no computational content, $[\text{AxT}]^+ := \varepsilon$.

Instead of providing witnesses for the induction axioms, it is technically easier to treat the logically equivalent induction rules. Hence, we assume that $\text{Ind}_A$ axioms are always applied to a sufficient number of arguments.

Case $\text{Ind}_A(b) N_1 N_2$. It is easy to check that we can set $[M]^- := [N]_1^+ \triangleright [N]_2^+; \quad$ and $[M]^+ := \lambda y [N]_1^+ \text{ else } [N]_2^+$. Cases $\text{Ind}_{A(n)} n N_1 N_2$ and $\text{Ind}_{A(l)} l N_1 N_2$. The extracted terms for $\text{Ind}_A$ are:

$$[M]^+ := R_A n [N]_1^+ (\lambda n. [N]_2^+ n 0)$$

$$[M]^- := R_A n (\lambda y [N]_1^-) (\lambda n, p, y. ([N]_2^+ \xi (p([N]_2^+ n 1 [M]^+_2 y))) y,$
where $\xi = [\langle n, [M]^\dagger, y \rangle / z]$. For a complete treatment the reader is referred to [Schwichtenberg(2008)]. The case of $\text{Ind}_n$ is similar.

3 The Infinite Pigeonhole Principle

3.1 Statement and proof

**Lemma 3.1** (Infinite Pigeonhole Principle). Any infinite sequence that is colored with finitely many colors has an infinite monochromatic subsequence.

**Proof.** Let $f^{\exists n \exists q}$ encode the infinite sequence colored by $r$ colors, i.e., we make the assumption $\forall n f(n) < r$. We need to show that infinitely many positions on $f$ are colored by some color $q$, i.e., $\exists q \forall n \exists k. n \leq k \land f(k) = q$. We prove this claim by induction on the number of colors $r$.

**Base case** If $r = 0$, then the assumption $f(n) < r$ is $\top$, so we use $\text{efq}$.

**Step** Assume that the statement to prove is true for $r$, i.e.,

$$\forall f. \forall n f(n) < r \rightarrow \exists q \forall n \exists k. n \leq k \land f(k) = q$$  \hspace{1cm} (IH)

Let us fix an arbitrary $f$ and assume

$$\forall n f(n) < Sr$$  \hspace{1cm} (StepH)

$$\forall q. (\forall n \exists k. n \leq k \land f(k) = q) \rightarrow \bot$$  \hspace{1cm} (NegG)

It remains to derive a contradiction ($\bot$) from the above.

We make a case distinction as to whether there is an infinite subsequence of color $r$ or not. If $r$ appears infinitely often in the given sequence, i.e., if

$$\forall n \exists k. n \leq k \land f(k) = r$$  \hspace{1cm} (Inf)

then this trivially leads to a contradiction, if we take $q$ to be $r$ in (NegG).

If this is not the case, then

$$\exists n \forall k. n \leq k \rightarrow f(k) = r \rightarrow \bot$$  \hspace{1cm} (NegInf)

which means that there exists an index $n'$, from which on the color $r$ does not appear anymore in the sequence. By (StepH) this means that from $n'$ on we have only $r$ distinct colors. We therefore look at a variant of the sequence $f$, in which we overwrite the positions up to $n'$ by $f(n')$, i.e., we take the sequence

$$f' := \lambda nf(n \sqcup n'),$$

where $x \sqcup y$ denotes the maximum of $x$ and $y$. The (IH) on $f'$, colored by $r$ colors, provides a color $q$ for which

$$\forall n \exists k. n \leq k \land f(k \sqcup n') = q,$$

which contradicts (NegG).

Note that in the above proof there is an implicit use of double negation elimination: (NegInf) actually derives $f(k) = r \rightarrow \bot$ and we need $f(k) \neq r$,
which together with (StepH) gives us the premise of (IH). For the proof to go through in the minimal logic setting required for A-Translation, we need to weaken our assumption to

$$\forall n \exists \tilde{n} f(n) < r,$$

thus proving a slightly stronger claim.

Since, as argued in the Introduction, Lemma 1.1 cannot have a constructive proof, in order to retrieve a computable functional from the classical proof, we analyze a corollary of it.

**Corollary 3.2.** For any infinite sequence $f$ colored with a finite number of colors and any $n \in \mathbb{N}$ one can find a finite monochromatic subsequence of $f$ of length $n$.

**Proof.** This follows trivially from Lemma 1.1 by taking the first $n$ elements of the infinite monochromatic subsequence. However, since we are interested in the way this sequence is produced, we will analyze the formalization of the proof in more detail.

**Step 1** At this point, we show that for any color $q$,

$$\forall n \exists k. n \leq k \wedge f(k) = q,$$

(Step1H)

then

$$\forall n \exists l. G(f, n, l, q),$$

(Step1G)

with $G(f, n, l, q)$ being

$$|l| = n \wedge \left(\forall m. S_m < n \rightarrow l_{S_m} < l_m\right) \wedge \left(\forall m. m < n \rightarrow f(l_m) = q\right).$$

(7)

We construct the list by induction on its length $n$. For $n = 0$ our only choice is nil. For $n = 1$ we take a list containing a single element, obtained from (Step1H) applied at 0. Finally, for the step case, assume that we have already a list $l$ with $n > 0$ elements. (Step1H) on $S(l_0)$ gives a fresh element $k > l_0$, which we add to the beginning of $l$.

**Step 2.** Let us first observe that it is straightforward to infer

$$\forall q. (\forall m. m < n \rightarrow f(l_m) = q) \rightarrow (\forall m. S_m < n \rightarrow f(l_m) = f(l_{S_m})).$$

(8)

Since the color of the monochromatic subsequence is not relevant to us, we consider a modified goal formula $G'(f, n, l)$:

$$|l| = n \wedge \left(\forall m. S_m < n \rightarrow l_{S_m} < l_m\right) \wedge \left(\forall m. S_m < n \rightarrow f(l_m) = f(l_{S_m})\right).$$

(9)

We therefore prove that for any sequence $f$ and number of colors $r$:

$$(\forall n. (f(n) < r \rightarrow \perp) \rightarrow \perp) \rightarrow \forall n \exists l. G'(f, n, l)$$

by cutting Step 1 with IPh and using (8).

---

7The list will be strictly decreasing rather than increasing, to avoid using “append” for constructing it.
3.2 Results with A-Translation

We first need to ensure that the specification conforms to our requirements. Since the Infinite Pigeonhole Principle

\[ \forall r, f. \forall n f(n) < r \rightarrow \exists q \forall n \exists k. n \leq k \land f(k) = q \]  

(IPH)

is clearly not a \( \Pi^0_2 \)-formula and its constructive version (with \( \exists \) instead of \( \exists \)) not provable, the A-Translation mechanism cannot be applied in a modular manner to it. However, the corollary

\[ \forall r, f. (\forall n. (f(n) < r \rightarrow \bot) \rightarrow \bot) \rightarrow \forall n \exists l G'(f, n, l), \]  

(UPH)

with \( G'(f, n, l) \) the conjunction from (9), is of the form (4).

We first observe that \( D := \forall n. (f(n) < r \rightarrow \bot) \rightarrow \bot \) is a relevant formula and that \( G := G'(f, n, l) \) consists of irrelevant conjuncts only. Thus, in the first step of eliminating the irrelevant occurrences of \( \bot \) by Lemma 2.8, we have \( G^F := G \), so no transformation on the goal is needed. \( D \) is, however, relevant and we therefore need Lemma 2.8 to derive (2): \( DF \rightarrow D \). Since \( f(n) < r \) is an atomic formula, thus decidable, we can apply Lemma 2.6 (Cases) to obtain

\[ ((f(n) < r \rightarrow F) \rightarrow \bot) \rightarrow (f(n) < r \rightarrow \bot) \rightarrow \bot \]

With this and \( \text{efq} \), we can easily infer (2). It already becomes obvious that the core of the program component associated to this part of the proof will be the test \( f(n) < r \).

Next, we replace throughout the proof the remaining occurrences of \( \bot \) by the strong goal \( \exists l G'(f, n, l) \). The constructive counterpart

\[ \forall f, r. (\forall n. (f(n) < r \rightarrow F) \rightarrow F) \rightarrow \forall n \exists l G'(f, n, l) \]  

(UPH-\( \exists \))

of our initial (classical) proof is now provable by plugging together the above transformations and the derivations (6).

The proof is now ready to be interpreted by the modified realizability, in the way suggested by the Soundness Theorem 2.11; using the rules from 2.10, we are able to automatically extract a program. Rather than displaying the raw object term, in Figure 1 we present two extracted programs corresponding to the proofs of IPH and UPH in a more readable form with suggestive variable names and separate definitions for functional arguments. In order to understand the program code better, it is helpful to consider Table 1, which summarizes the computational types associated with the relevant subformulas of the two statements.

With \( \bot \) being a predicate variable, we regard here its computational type as being \( \text{abstract} \), i.e., a type variable that can be substituted with any type.\(^8\) Since \( \bot \) is to be substituted with the formula \( \exists x G \), we can in fact think of the computational type of \( \bot \) as the type of the final result \( x \), which in our case is determined by Corollary 3.2. Since computationally relevant formulas always end in \( \bot \), they correspond to computations producing an output of the abstract type, i.e., a candidate for a final result, as shown in the “Output” column in

\(^8\text{By abuse of notation we will denote by } \bot \text{ the predicate variable, its computational type, and object variables of this type.}\)}
Table 1. The abstract view of \( \perp \) restricts the possible form of the extracted programs: they can either return a dummy value, or refer to some of their (functional) parameters, whose type ends in \( \perp \). Thus the programs adhere to the continuation passing style (CPS).

Now let us regard the separate computational components in more details.

- The program \( \text{FC} \) (Finitely Colored) is given an index \( n \) and a candidate for the final result \( \perp \), which is correct if the color of \( f \) at \( n \) is below \( r \).

- The program \( \text{SE} \) (Sequence Extension) has to produce the final result given an index \( k \geq n \), at which the sequence \( f \) is colored by \( q \).

- The program \( \text{IS} \) (Infinite Sequence) takes an index \( n \) and is expected to provide a next occurrence \( k \) of the color \( q \) in \( f \). Note that \( \text{IS} \) cannot return \( k \) directly; instead it should return a final result by passing a candidate for \( k \) to its parameter \( \text{SE} \) (traditionally called a continuation).

- Similarly, \( \text{IMS} \) (Infinite Monochromatic Sequence) takes a color \( q \) and computes a final result by invoking \( \text{IS} \) for some indices \( n \). Note that \( \text{IMS} \) will not obtain the final result “for free”: in order to invoke \( \text{IS} \) it needs to be provided with a program \( \text{SE} \), which specifies how to continue the computation after receiving an answer \( k \) with \( f(k) = q \).

- The main program \( \text{IPH} \) can be used only after the continuations \( \text{FC} \) and \( \text{IMS} \) have been provided.

- The conclusion (Step1G) of Corollary 3.2 states that the final result should be a finite list \( l \) of indices. However, since we are using the weak existence, the program \( \text{FS} \) (Finite Sequence) would not directly return the final answer, but instead would feed the currently computed list of length \( n \) to its continuation parameter, named \( P \) in Figure 1b).

- The final program \( \text{UPH} \) is the algorithm we expect to extract: given a sequence \( f \), a number \( r \) of colors and the length \( n \), it returns a finite list \( l \) of indices at which the sequence \( f \) has the same color. Note that the specification corresponds to the strong existence formula \( \text{UPH-\exists} \).

We will make use of the following two abbreviations:

\[
(h[n])(m) := h(n \sqcup m) \quad (h][n)(m) := n \sqcup h(m).
\]
$\text{IPH}(r, f, FC, IMS) := \text{if } r = 0 \text{ then } FC(0, \Box) \text{ else } IMS(r - 1, IS)$

$IS(n, SE) := \text{IPH}(r - 1, f[n, FC_n, IMS_n])$

$FC_n(n', \bot) := FC(n \sqcup n', \text{if } f(n \sqcup n') \neq r - 1 \text{ then } \bot \text{ else } SE(n \sqcup n'))$

$IMS_n(q, IS') := IMS(q, IS'_n)$

$IS'_n(k, SE) := IS'(k, SE|n)$

a) A-Translation program for IPH

$\text{UPH}(r, f, n) := \text{IPH}(r, f, FC, IMS)$

$FC(n, \bot) := \text{if } f(n) < r \text{ then } \bot \text{ else } \Box, \text{ IMS}(q, IS) := FS(n, \lambda l)$

$FS(n, P) := \text{if } n = 0 \text{ then } \text{nil} \text{ else }$

$\text{if } n = 1 \text{ then } IS(0, \lambda k. P(k)) \text{ else } FS(n - 1, P')$

$P'(k :: l) := IS(k + 1, \lambda k'. P(k' :: k :: l))$

b) A-Translation programs for IPH and UPH

Figure 1: A-Translation programs for IPH and UPH

The program scheme IPH given in Figure 1a) follows the proof structure of IPH very closely by recursing on $r$. For the base case, the use of $efq$ in the proof is reflected by calling $FC$ with a dummy candidate ($\Box$). The recursive case calls $IMS$ in an attempt to build a monochromatic sequence of color $r - 1$, for which the program $IS$ is expected to calculate an index after $n$ of this color and to pass it to its parameter $SE$. Following the proof, $IS$ assumes that $r - 1$ does not appear after $n$ and attempts to obtain the final result by a recursive call to IPH, with the parameters $f$, $FC$ and $IMS$ changed to $f[n, FC_n]$ and $IMS_n$ respectively, thus disregarding indices smaller than $n$. $FC_n$ deserves special attention: as noted above, it obtains an index $n'$ and a result $\bot$, which will be correct if ($*\text{)}$) $(f[n])(n') < r - 1$. When we know for a fact that this color differs from $r - 1$, ($*\text{)}$) is equivalent to $f(n \sqcup n') < r$, which allows us to pass $\bot$ to $FC$ in this case. If the color happens to be $r - 1$, this contradicts the assumption under which IPH was called recursively; however, we have found exactly an index that $SE$ expects.

We now examine the main program given in Figure 1b). Since the proof is obtained by a cut, UPH is defined by a call to IPH with specially constructed parameters. The program $FC$, being extracted from the Cases Lemma 2.6, acts as a correctness guard: returns $\bot$ on $f(n) < r$ and a dummy value $\Box$ otherwise. $FS$ corresponds to Step 1 of Corollary 3.2 and thus recursively extracts the values of constant color. The role of the function $SE$ is played by $P$, which receives the next index in the infinite sequence and accumulates it in a list.

In order to understand the operational semantics of the obtained program, we should note that both recursions on $r$ and $n$ unfold immediately and the actual computation is carried out during the folding process, from the base case up. This has the effect that for every color $q < r$ a program $FS$ is started, each of them calculating a list of indices of the corresponding color. The program $FS$ performs a “successful step” only when $SE$ receives some index $k$ of color $q$; in this case $IS$ is invoked, asking for the index after $k + 1$. The process ends
when some list reaches length \( n \), and it is returned as the final result. However, there is one important pitfall: the program \( IS \), called after each successful step, restarts \( IPH \) from the base case. This invokes fresh programs \( FS \) for all colors below \( q \), while the partially accumulated lists for these colors are lost. As a result, the program in Figure 1 need not necessarily find the first \( n \) occurrences of constant color; it returns a list of the smallest possible indices of a color \( q \), between which no color larger than \( q \) appears.

### 3.3 Results with Dialectica

Following the Soundness Theorem 2.13 for the Dialectica interpretation we can directly extract positive and negative realizers for each component of the classical proof of Corollary 3.2. Since we will not need \( \bot \) to be a predicate variable, we can safely substitute it with the arithmetical falsity \( F \). This choice makes the double negation in the assumption \( \forall n(\neg \neg f(n) < r) \) redundant, but we nevertheless choose to keep it for the sake of honest comparison.

As before, we first describe and explain the computational types of the subformulas for the Dialectica interpretation (Table 2). Recall that we distinguish between positive and negative types: the programs with negative specifications compute challenges for the corresponding formula, whereas the positive specifications provide solutions. More precisely,

- the negative specification \( FC^- \) (Finitely Colored) provides an index, at which the sequence has a color \( \geq r \);
- \( IS^- \) (Infinite Sequence), also negative, provides an index \( n \) after which the color \( q \) is not expected to appear in the sequence anymore;
- \( IS^+ \) is a positive specification that extends a sequence colored by \( q \) unboundedly, i.e., for every index \( n \) provides a next index \( k \);
- the positive specification \( IMS^+ \) (Infinite Monochromatic Sequence) provides a color \( q \) and an extending function \( IS^+ \) for this color

<table>
<thead>
<tr>
<th>Formula</th>
<th>Specification</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( IPH )</td>
<td>( IPH^+ )</td>
<td>( r, f, IMS^- )</td>
<td>( IMS^+, FC^- )</td>
</tr>
<tr>
<td>( \forall n f(n) &lt; r )</td>
<td>( FC^- )</td>
<td>( n )</td>
<td></td>
</tr>
<tr>
<td>( \exists q \forall n \exists k. n \leq k \land f(k) = q )</td>
<td>( IMS )</td>
<td>( IMS^+ )</td>
<td>( IS )</td>
</tr>
<tr>
<td>( IMS^)</td>
<td>( q, IS^+ )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \forall n \exists k. n \leq k \land f(k) = q )</td>
<td>( IS )</td>
<td>( n )</td>
<td>( k )</td>
</tr>
<tr>
<td>( IS^- )</td>
<td>( n )</td>
<td>( k )</td>
<td></td>
</tr>
<tr>
<td>( \forall q. (Step1H) \Rightarrow (Step1G) )</td>
<td>( FS^+ )</td>
<td>( f, IMS^+, n )</td>
<td>( l, IS^- )</td>
</tr>
<tr>
<td>( UPH )</td>
<td>( UPH^+ )</td>
<td>( r, f, n )</td>
<td>( l )</td>
</tr>
</tbody>
</table>

Table 2: Dialectica computational types

Formulas of higher complexity have, correspondingly, specifications of higher order, taking other programs as parameters. Thus, the use of such formulas is intertwined:
- IMS− takes as input a candidate for a monochromatic sequence IMS+ and produces a challenge IS− that the sequence cannot be infinitely extended;

- the specification IPH+, corresponding to the proof of IPH, receives a “correctness test” IMS− for the output solution IMS+, in addition to the expected parameters r and f. Note that a further output is produced here: a challenge FC− that f does not satisfy its input specification. As we will see later, this challenge is important for backtracking, in the case when a false solution is provided on some intermediate step;

- the specification FS+ (Finite Subsequence) corresponds to the Step 1 of the proof of Corollary 3.2. Given a monochromatic subsequence IMS+ of a colored sequence f and a number n, FS+ collects n indices from IMS+ in a list l. Since we are under the assumption (Step1H) that an infinite monochromatic sequence exists, there is also a candidate counterexample IS− in case it is false;

- finally, the specification UPH+ for UPH returns the list of monochromatic indices, given r, f and n.

As with the refined A-translation, we skip the formal extraction and display the final programs in a readable form, with separate definitions of common sub-terms. In addition, for the sake of readability we have η-permuted λ-abstractions and conditional statements over pairs, wherever possible.9 We will use a special notation (e.g. IPH+(r, f, IMS−) △ IMS+) to select between multiple program outputs, as named in Table 2.

\[
\begin{align*}
\text{IPH}^+(r, f, \text{IMS}^-) & := \text{if } r = 0 \text{ then } \Box \text{ else} \\
& \quad \text{if } m \leq \text{FC}^- \land f(\text{FC}^-) = r - 1 \text{ then } \langle r - 1, g \rangle, \text{FC}^- \\
& \quad \text{else } \langle q, h \rangle m, \text{FC}^- \\
& \quad g(n) := n \sqcup \text{IPH}^+(r - 1, f[n, \text{IMS}^-_n]) \triangleright \text{FC}^- \\
& \quad m := \text{IMS}^-(r - 1, g) \\
& \quad \text{FC}^- := g(m) \\
& \quad \langle q, h \rangle := \text{IPH}^+(r - 1, f[m, \text{IMS}^-_m]) \triangleright \text{IMS}^+ \\
& \quad \text{IMS}^-_n (q, h) := \text{IMS}^-(q, h][n)
\end{align*}
\]

**Figure 2:** Dialectica program for IPH

In Figure 2 we present the program extracted from the proof of IPH. As before, we use the abbreviations (10) and take □ to represent an arbitrary output for IPH+, coming from the use of efq in the base step of the proof. The operational semantics of the program is not easy to grasp, so we will examine it in parallel with the proof of IPH.

9In principle, it could be argued that such transformations are an attempt for “manual” optimization of the automatically obtained program. However, in our case we have verified that the asymptotic complexity of the programs is not modified.
The program reflects the classical (i.e., undecidable) case distinction on (Inf), i.e., whether \( r \) appears infinitely often in given the sequence. Assuming that the answer is “yes”, IPH\( ^+ \) \( \triangleright \) FC\( ^- \) should provide an index where the color is not less than \( r - 1 \). In addition, if the assumption (StepH) is also true for this index, then its color should be exactly equal to \( r - 1 \). For this reason, we use the output FC\( ^- \) to construct \( g \), a sequence-extending function that we output together with the color \( r - 1 \). If, on the contrary, the assumption (StepH) is false, then we provide FC\( ^- \) as a counterexample for that.

However, if the answer is “no”, then at some point our strategy for getting the next index via \( g \) will fail. This will be detected by the conditional statement in IPH\( ^+ \). Note that the correctness condition is checked at the index \( m \) where, according to the challenge parameter IMS\( ^- \), \( g \) is supposed to fail. In this case the proof uses the induction hypothesis, because after \( m \) we know that the color \( r - 1 \) does not appear anymore. Correspondingly, we obtain in the program a solution IMS\( ^+ \) by a recursive call on the subsequence which is “cut” from \( f \) from the index \( m \) onward.

\[
FS^+(f, q, g, n) := \begin{cases} 
\langle \text{nil}, 0 \rangle & \text{if } n = 0 \\
\langle g(0), 0 \rangle & \text{if } n = 1 \\
\langle \langle k :: l, k_2 \rangle, \langle l', k_1 \rangle \rangle & \text{if } k_1 \leq g(k_1) \land f(g(k_1)) = q \\
\langle \langle l', k_1 \rangle \rangle & \text{else }
\end{cases}
\]

\[
k_1 := Sk \\
l' := (g(k_1) :: k :: l)
\]

\[
UPH^+(r, f, n) := FS^+(f, IPH^+(r, f, IMS^-_n) \triangleright IMS^+, n) \triangleright l
\]

\[
IMS^-_n(q, g) := FS^+(f, q, g, n) \triangleright IS^-
\]

Figure 3: Dialectica program for UPH

Figure 3 depicts the program extracted from the Corollary 1.3. FS\( ^+ \) accurately reflects Step 1, with the base cases \( n \leq 1 \) directly borrowed from the proof. For the step case, we recursively obtain a non-empty list of length \( n - 1 \), starting with an element \( k \). FS uses its parameter \( g \) to calculate the next index after \( k + 1 \) of color \( q \). The negative output IS\( ^- \) is needed if the assumption “\( q \) and \( g \) provide a monochromatic sequence” fails. The case distinction ensures that we always output the last index at which \( g \) failed — either the current one \( (k_1) \), or the previous one \( (k_2) \). It is easy to see that if \( g \) does not fail until we construct the whole list, then the index 0 is returned. This will be an invalid challenge, which is to be detected by the program that invokes FS\( ^+ \).

The main program UPH\( ^+ \) invokes FS\( ^+ \), giving as a parameter a monotone sequence obtained from IPH\( ^+ \); this was to be expected, since Step 2 is proved by a cut. It is notable that the cut is reflected in the negative direction as well. In particular, the challenge parameter IMS\( ^- \) for IPH\( ^+ \), which is expected to falsify a given monochromatic sequence, is produced by the challenge output IS\( ^- \) of FS\( ^+ \). This is exactly a backtracking effect; it is used by the program FS\( ^+ \) to inform its caller that it has been provided with invalid parameters and to show where they fail.
4 Analysis

4.1 Experiments

Before we comment on the effects of the two methods with respect to the extracted programs, we compare the runs of programs on uniformly distributed random sequences. Selected case studies are presented in the two tables below:

<table>
<thead>
<tr>
<th>n</th>
<th>r</th>
<th>last index</th>
<th>time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2</td>
<td>18</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>184</td>
<td>48</td>
</tr>
<tr>
<td>200</td>
<td>2</td>
<td>390</td>
<td>133</td>
</tr>
<tr>
<td>500</td>
<td>2</td>
<td>988</td>
<td>835</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>33</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>45</td>
<td>6</td>
</tr>
<tr>
<td>50</td>
<td>4</td>
<td>205</td>
<td>289</td>
</tr>
</tbody>
</table>

(a) A-Translation (UPH)

<table>
<thead>
<tr>
<th>n</th>
<th>r</th>
<th>last index</th>
<th>time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2</td>
<td>85</td>
<td>9</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>376</td>
<td>88</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>2453</td>
<td>2563</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>9899</td>
<td>40389</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>849</td>
<td>520</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>8302</td>
<td>54990</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>7487</td>
<td>27879</td>
</tr>
</tbody>
</table>

(b) Dialectica (UPH+)

Figure 4: Average runs of the programs obtained with methods (a) and (b)

In the next subsection we carry out a thorough complexity analysis of the two programs, which should shed some light on the results summarized in the above tables. Subsection 4.3 attempts at a deeper parallel between the two methods, in view of the experiments depicted in the tables.

4.2 Complexity

Worst time complexity of the program extracted by A-Translation (Figure 1). As we have already discussed in Section 3.2, the program starts by preparing \( r \) unfolded recursions of depth \( n \), which are waiting to fold. Let us denote by \( p(q) \) an upper bound on the number of recursion folds for the color \( q \). Since the recursions of lower color are restarted for every recursion fold of higher index, we have

\[
p(r) = 1 \quad p(q) = p(q + 1) \cdot n \quad \text{for } q < r
\]

Therefore, we obtain worst time complexity \( O(n^r) \).

The complexity of the program closely follows the largest index in the returned list, because on every next call we compute an index, which by the “\( \lceil \)" operation is guaranteed to be greater or equal than the current index. Having this in mind, we clearly reach the exponential upper bound. Indeed, consider the following family of initial segments:

\[
l^{n,0} = \text{nil} \quad l^{n,q+1} = \underbrace{l^{n,q}, q, l^{n,q}, q, \ldots, l^{n,q}}_{n-1}
\]

It is easy to see that \( |l^{n,r}| = n^r - 1 \). As noted before, the program in Figure 1 always returns a list of length \( n \) of the smallest indices of some color \( q \), such that no higher color appears between the first and the last index. Let us call such monochromatic subsequences undisturbed. The lists \( l^{n,r} \) have the property that they contain no undisturbed subsequence of length \( n \) of any color. Note also.
that appending even a single element to $l^{n,q}$ will break this property, thus such lists are also maximal. Therefore, when run on any infinite sequence starting with $l^{n,r}$, the program $UPH$ executes in $n^r$ steps and the last returned index is $n^r - 1$.

**Worst time complexity of the program extracted with Dialectica** (Figure 3). We analyze the time complexity of $UPH^+$ under a strict evaluation strategy. We assume that there is no redundant reevaluation of syntactically equal subterms. In order to find the time complexity of $UPH^+$, we need to compute an upper bound $p$ for the number of times the parameter $IMS^-$ is being invoked in $IPH^+$. However, we need to take into account that $IMS^-$ is used to construct the functional outputs $g$ and $h$, which can be invoked outside $IPH^+$ multiple times. Let us introduce a parameter $s_e$ bounding the number of such external calls. Furthermore, the function $g$ is also used internally in order to compute the index $m$. This is done by possibly multiple invocations of $g$ performed by $IMS^-$ and we introduce another bounding parameter $s_i$ for that. We have one more invocation of $g$ on $m$ to compute $FC^-$. Note that both parameters $s_e$ and $s_i$ do not change on a recursive call, because $h$ is not being invoked inside the program itself and $IMS_n^-$ has exactly the same number of invocations of its functional parameter as $IMS^-$. Thus, we obtain the following equations:

$$p(0) = 0, \quad p(r + 1) = (2s_e + s_i + 1) \cdot p(r) + 1.$$  

Finally, by looking at $UPH^+$ we see that both $s_e$ and $s_i$ correspond to the number of times the parameter $g$ of $FS^+$ is being called, which is exactly $n$. With this, we obtain that the complexity of the whole program $UPH^+$ is also $O(n^r)$.

**Average time complexity of the program extracted with A-Translation.** On a first sight it seems that the programs extracted by both methods should be equally exponentially slow. However, experiments with uniformly distributed random sequences, presented in Table 4, show that $UPH$ is much more efficient than $UPH^+$ in the average case. The case studies also display an asymmetry of the programs: it is more likely that the returned color is large. Indeed, $UPH$ will return only undisturbed subsequences of color $q$ and there are more such subsequences for higher colors. In fact, all monochromatic subsequences of the highest color $r - 1$ are undisturbed and such a sequence will be returned if everything else fails. In an uniformly distributed sequence, every color should appear with rate $1/r$, i.e., every $r$ indices. Therefore, a monochromatic subsequence of color $r - 1$ and length $n$ would be found in time $r \cdot n$ in the average case. An undisturbed sequence of a lower color would be found earlier than one of higher color, so we can expect that the average time complexity of $UPH$ is $O(r \cdot n)$. It is important to note that this is the same as the worst (and the average) time complexity of the direct naive algorithm, which goes linearly on the sequence and constructs $r$ lists until one of them reaches length $n$.

**Average time complexity of the program extracted with Dialectica.** As the experiments from Table 4 show, the program $UPH^+$ performs exponentially even in the average case. The reason is that the index returned by an extending function $g$ for the color $q$ depends on the last index at which the extending function for color $q - 1$ failed. This failure index is returned by the
program $FS^+ o IS^-$ and will be found on average at the $[n-1/r]$-th step of $FS^+$ in the case of a uniform distribution. Therefore, the average value of the distance between the starting and the failure index for an extending function $q$ and a color $q < r - 1$ is $O(n^{q+1})$. As with the program $UPH$, a subsequence of a color $q$ is found only when the attempts for lower colors have failed. Again, the strategy for obtaining a monochromatic sequence of the highest color $r - 1$ should never fail, because all other possibilities would have been already exhausted. Therefore, by a similar argument as above, the average time complexity for finding a monochromatic subsequence of color $r - 1$ and length $n$ should be $O(n^r)$; a monochromatic subsequence of a lower color would be found even earlier.

### 4.3 Comparison

A common feature of the algorithms extracted by the two methods is that both of them reflect the structure of the proof in a specific way. Most importantly, the programs are based in both cases on an interaction between two components — (1) a higher order program (or a program scheme) corresponding to the proof of IPH, used by (2) the main program, corresponding to the proof of the corollary $UPH$. Moreover, both algorithms implement some form of backtracking, reflecting the fact that the proof that we interpret by the two methods is non-constructive.

The most important difference between the two methods lies in the way in which the computational contents of IPH and $UPH$ are combined. In Dialectica we have two separate programs for the two proofs, interacting only by means of their outputs. On the other hand, in the refined A-Translation the use of an abstract type prevents extraction of an independent program for IPH, its computational content being determined by the corollary. However, this allows for a much tighter connection between the two components residing in the use of continuations, whose return type is left abstract in IPH and specialized in $UPH$. Thus, we have an immediate transfer of control flow between the two programs, whenever backtracking is needed, i.e., when the witness being provided fails. This can be clearly seen in the program $FC_n$ in Figure 1b): whenever the color $r - 1$ does not appear as expected, we do not invoke the continuation $SE$, but instead go back one step by calling $FC$, provided by the recursive invocation. By contrast, whenever witnesses for a color $q$ and a sequence extending function $h$ are suggested by the Dialectica program $IPH^+ o IMS^+$ in Figure 2, the parameter $IMS^-$ is invoked, which leads to a call of $FS^+$ in Figure 3. Then the whole list is constructed only to determine a counterexample index $k_i$, which is examined by $IPH^+$ to decide whether the program should proceed forward or step back.

As discussed in Section 4.2, both algorithms have the same exponential worst time complexity. However, the immediate backtracking of the $UPH$ program brings down its average time complexity to polynomial. On the other hand, the ineffectiveness of the Dialectica program $UPH^+$, which constructs a whole list of length $n$ only to determine the last failure index is reflected in its average time complexity, which is again exponential.

It might seem tempting to optimize the program $FS^+$ to return a result as soon as the first failure index is found. In the average case this would happen at the $[1+1/r]$-th step of $FS^+$ and then by the argument in Section 4.2 the average time complexity of $UPH^+$ should be $O(r \cdot n)$. However, the Dialectica interpretation of induction as presented in Section 2.3 gives no means
to terminate the recursion earlier and therefore such an optimized program could not have been extracted.

5 Related Work

The Infinite Pigeonhole Principle and special variants of it have been previously addressed in the literature in connection with program extraction from proofs. The simplest and most commonly treated case is the existence of two equally colored indices in a two-colored sequence and has been attributed to Stolzenberg. In [Barbanera et al.(1997)] and [Urban(2000)] a classical proof of this example is treated with normalization by cut-elimination in order to retrieve its computational content. The papers argue that in the process of reduction, one has the freedom in which direction to propagate the cuts and such a non-deterministic choice gives rise to distinct programs, corresponding to the same proof. The case study has been followed up by [Makarov(2006)], where a program with control operators is extracted, and later simplified to eliminate the use of continuations.

[Seisenberger(2003)] takes the example a step further, by searching for a finite number of equally colored elements in a two-colored sequence. The non-constructive proof in consideration involves the axiom of dependent choice, as the paper investigates the interaction between the refined A-Translation and external realizers, in this case computed by the bar recursion.

The novelty of the present paper with respect to the aforementioned results lies on one hand in introducing one more degree of freedom by allowing an arbitrary (finite) number of colors. In this way both the classical and the constructive parts of the proof are proved by induction, which allows a complexity analysis not only of the final extracted programs, but also of the contribution of the non-constructive argument. In this light we are able to compare the behavior of two extraction methods, refined A-translation and Dialectica, which, to the authors’ knowledge, has not been done in the literature before.

6 Conclusion and future work

In the present paper we have investigated the computational meaning of the non-constructive Infinite Pigeonhole Principle by means of two proof interpretations: modified realizability combined with refined A-Translation and Gödel’s Dialectica interpretation. Since the corresponding extracted programs are of higher order, we chose to analyze the way in which they extract the computational content out of the classical proof of the $\Pi^0_2$-corollary UPH. It turned out that the non-decidable case distinction in IPH corresponds to a backtracking scheme, implemented in different ways with the two methods considered. For the refined A-Translation we had a polymorphic program IPH, which uses a continuations passing style (CPS) to control the forward and backward steps in search of a solution. The instantiation of the abstract type parameter with the type of lists was determined by the main program UPH. By the Dialectica interpretation, we have obtained a standalone backtracking program $\text{IPH}^+$, which exchanges counterexamples with the main program $\text{UPH}^+$, to accept or deny a candidate solution. The more loose connection between the two programs
leads to exponential average time complexity in this case, compared with the polynomial complexity of the refined A-Translation program UPH.

IPH together with the Axiom of Dependent Choice (DC) constitute the two major uses of classical logic in the proof of the Infinite Ramsey’s Theorem. It has been known from Spector that the computational meaning of DC is bar recursion. Later, there have been many proposed modifications of bar recursion (see e.g. [Berger and Oliva(2006)]), which optimize its computational effectiveness. The authors believe that these results combined with the present research open the way towards extraction from Finite Ramsey’s Theorem, proved as a corollary of its non-constructive infinite variant.

References


22


