A functional interpretation with state

Thomas Powell
Department of Mathematics
Technische Universität Darmstadt
powell@mathematik.tu-darmstadt.de

Abstract

We present a new variant of Gödel's functional interpretation in which extracted programs, rather than being pure terms of system T, interact with a global state. The purpose of the state is to store relevant information about the underlying mathematical environment. Because the validity of extracted programs can depend on the validity of the state, this offers us an alternative way of dealing with the contraction problem. Furthermore, this new formulation of the functional interpretation gives us a clear *semantic* insight into the computational content of proofs, and provides us with a way of improving the efficiency of extracted programs.

Keywords Functional interpretation, program extraction, state monad

1 Introduction

Proof interpretations are well known techniques for extracting programs from proofs. Not only are they central to the proof mining program [9], but they have been used as a means of synthesising verified programs, to which end specialised proof assistants such as Minlog [1] have been developed.

Traditionally, proof interpretations take as input some logical theory such as Peano arithmetic, and translate proofs in this system to realizing terms written in some lambda calculus, typically a variant of Gödel's system T. As such, when viewed as *programs*, it is natural to think of these as being implemented in a functional language.

However, more recently, researchers in this area have been leaning towards extracting programs in languages with *imperative* features such as stacks and continuations, which is exemplified, for instance, in Krivine's classical realizability [11]. Not only do imperative structures lend themselves very well towards expressing the computational content of classical reasoning, but writing extracted programs in languages of this kind brings them much closer to programming paradigms used in practise, which is particularly relevant if one aims to use proof interpretations for program synthesis.

The purpose of this article is to present a variant of Gödel's famous functional (or 'Dialectica') interpretation, which extracts programs that, rather than being purely functional, can access and change a global state. The idea is that our state will represent a 'state of knowledge' which contains computational information about our mathematical environment. For example, if we are sorting some data structure, the state might contain a list of comparisons which have already been made; if we are computing over some sequence of rationals in the unit interval, the state might contain information about where certain elements of the sequence are located. However, we work in a general framework that allows considerable flexibility in characterising both the state and how our programs interact with it.

Our work is motivated by a number of separate factors, each of which we explore in a more detail towards the end of the article.

First of all, we believe that our interpretation is of theoretical interest in its own right, as is constitutes a broad generalisation of the original functional interpretation whose realizers can now potentially take as an additional input a sequence of *conditions* which are forced to be true, and in turn add further conditions to the state as they evaluate. On the most simple level, we can define these conditions to be lists of atomic formulas, and as such our realizing terms can be viewed as programs which traverse a Herbrand tree. However, much more intricate characterisations of the state are possible, and we hope to explore potential applications of our interpretation in pure logic in the future.

A different motivation for introducing a state to the functional interpretation was to connect it to the range of elegant computational interpretations of classical logic which are based on 'backtracking' or 'learning' (such as [2, 4, 7]). These interpretations are oriented towards giving a *semantic* meaning to classical reasoning, characterising realizing terms as programs which build successively better approximations to ideal objects. This is in stark contrast to the functional interpretation, for which the corresponding semantics lies hidden in the structure the extracted terms via the case-distinction functions. The main results of this paper address this by using the state to capture how programs extracted by the functional interpretation interact with the mathematical environment.

Finally, we have a number of *practical* objectives in mind, and hope that our functional interpretation with state will take a step further towards successfully applying proof interpretations for program synthesis. Firstly, we can improve the efficiency of extracted programs by using the state to avoid unnecessary computations, something which has already been studied in e.g. [8, 14]. Secondly, by producing terms which interact with a state we end up with programs closer in style to those a normal programmer would write, something which has already been explored in the context of modified realizability in [3].

The article is structured as follows: In Sections 2 and 3 respectively we define our main logical systems (which include a variant of System T extended with a state type), and outline the usual functional interpretation. Then, in Section 4, we construct a monadic interpretation of System T based on the state monad, and characterise how the resulting state-sensitive terms relate to their pure counterparts via a logical relation. We then give our new formulation of the functional interpretation in Section 5 and present the main soundness theorem. This forms our central contributions, and a detailed study of the aforementioned applications are beyond the scope of the paper. We nevertheless conclude, in Section 6, by presenting several concrete examples of how our state could be implemented, and pose a number of specific open questions which we hope can be addressed in the future.

, ,
.

1.1 A preliminary example

Before we begin, we consider a very simple example which will be treated in more detail in Section 6.1. This is a variant of the so-called drinkers paradox:

$$\exists x^0 \forall y^0 (P(y) \to P(x)). \tag{1}$$

In general, we cannot effectively compute a witness for x. However, it is possible to give a computational interpretation of the following reformulation of the drinkers paradox, which is essentially its functional interpretation:

$$\forall f^{0\to 0} \exists x^0 (P(fx) \to P(x)). \tag{2}$$

Assuming that P is decidable, we can define a realizer $\Phi_{\text{Gödel}} : (0 \to 0) \to 0$ for (2) by

$$\Phi_{\text{Gödel}} f := \begin{cases} 0 & \text{if } \neg P(f0) \\ f0 & \text{otherwise,} \end{cases} \tag{3}$$

where $\Phi_{\text{Gödel}} f$ serves as an exact witness for $\exists x$ in (2). Alternatively, we can produce a realizer for (2) which is independent of P (and so works even when P is not decidable), which instead returns a finite sequence of potential witnesses for $\exists x$, in this case $\Phi_{\text{Herbrand}} : (0 \to 0) \to 0^*$ defined by

$$\Phi_{\text{Herbrand}} f := [0, f0]\,.$$

These two choices give rise to two distinct flavours of the functional interpretation: The original interpretation, which relies on case distinctions, and the many Herbrand style interpretations (which include e.g. the Diller-Nahm variant), which work for theories whose atomic formulas are undecidable, such as nonstandard arithmetic [15].

In this article, we consider a third possibility, in which we produce both an exact witness but at the same time do not necessarily rely on the decidability of P. Instead, we make *assumptions* about P and collect these in a global state. The validity of our realizer will then depend on the validity of the end state. More specifically, let S be some set of possible states, where $\pi \in S$ consists of a sequence of predicates. We can define two terms $\Phi^i_{\text{State}} : (0 \to 0) \to S \to 0 \times S$ for $i = 1, 2$, which take an arbitrary input state π and return as output a state-realizer pair:

$$\Phi^1_{\text{State}} f \pi := \langle 0, \pi :: \neg P(f0) \rangle \quad \text{and} \quad \Phi^2_{\text{State}} f \pi := \langle f0, \pi :: P(f0) \rangle.$$

where $\pi :: P(f0)$ denotes the extension of the state π with $P(f0)$. Each of these realizers is valid provided the corresponding state is, and can be characterised as traversing a branch in the tree which underlies the corresponding Herbrand normal form of (1).

This extremely simplistic example does not illustrate the full potential of our state, which we present on a much more abstract level. However, we hope that it at least gives the reader some insight into the basic mechanism of the interpretation, whose formal construction involves a monadic translation in all finite types and a rather intricate set of logical relations.

2 Formal systems

Three formal systems will play a role in this paper: The logical theory HA^+ of Heyting arithmetic with predicate parameters (together with its classical counterpart PA^+), which will serve as our input theory, the standard theory $\mathsf{E\text{-}HA}^\omega$ of Heyting arithmetic in all finite types which allows us to express basic extracted programs, and finally an extension $\mathsf{E\text{-}HA}^\omega_S$ of this theory with a state type, which will allow us to write state-sensitive programs. It is the case that $\mathsf{HA}^+ \subset \mathsf{E\text{-}HA}^\omega \subset \mathsf{E\text{-}HA}^\omega_S$ i.e. each theory can be embedded, in an obvious way, in the next.

2.1 Heyting arithmetic with predicate parameters (HA^+)

Our input theory HA^+ (resp. PA^+) will be the standard theory of Heyting (resp. Peano) arithmetic, extended with a countable collection of predicate parameters $P, Q, \ldots$ where each of these has some fixed arity. The purpose of these parameters will be to represent our 'mathematical environment'. This can be chosen according to the context, and the parameters will represent distinguished formulas, information about which will be stored in the state. We can freely add to HA^+ universal axioms which characterise these predicates, since the functional interpretation is insensitive to these.

Note that for each predicate P (of arity r_P, say) we associate a characteristic function $\chi_P : 0^{r_P} \to 0$ which satisfies

$$\forall x_1, \ldots, x_r (\chi_P(x_1, \ldots, x_r) \leq 1).$$

In fact, since we are working in arithmetic it is simpler to take the characteristic function as the primitive notion and *define*

$$P(x_1, \ldots, x_r) :\equiv \chi_P(x_1, \ldots, x_r) = 0.$$

However, it will still be useful to make the distinction between P and χ_P, as the former typically plays a role in a proof, and the latter in the resulting program. Note that while we officially treat the χ_P as being genuine parameters (i.e. function symbols) whose interpretation will depend on the model, for certain applications these will be nothing more than labels for closed primitive recursive functions, in which case HA^+ will be just a definitional extension of HA with new symbols denoting relevant primitive recursive predicates.

We could alternatively just consider the weakly extensional variant $\mathsf{WE\text{-}HA}^\omega$ of $\mathsf{E\text{-}HA}^\omega$ as being our input theory, as this has a functional interpretation in $\mathsf{E\text{-}HA}^\omega$ and comes already equipped with function symbols which can act as our parameters, although strictly speaking these would have to be distinguished from ordinary function *variables*, as they should not be quantified over. Moreover, there are a few additional details at higher type that we prefer to avoid for now, which is why we simply work in HA^+.

2.2 Heyting arithmetic in all finite types ($\mathsf{E\text{-}HA}^\omega$)

By $\mathsf{E\text{-}HA}^\omega$ we denote the standard extension of Heyting arithmetic in all finite types, as outlined in e.g. [9]. The only difference here is that we choose a variant with product types, and among our function symbols of type level 1 we isolate a countable collection $\chi_P, \chi_Q, \ldots$ which correspond to the parameters of HA^+, and are assumed not to be quantified over. Formally, the types of $\mathsf{E\text{-}HA}^\omega$ are generated by the following grammar:

$$\rho, \tau ::= 0 \mid \rho \times \tau \mid \rho \to \tau$$

We write either $x : \rho$ or x^ρ to denote that x is of type ρ. Occasionally we abbreviate $\rho \to \tau$ as $\tau\rho$. Terms of $\mathsf{E\text{-}HA}^\omega$ are constructed as usual, and include as constants the zero, successor, projection and pairing, and recursor. For each term t it will be helpful to assign a formal typing $\Gamma \vdash t$, which we outline in Figure 1, where Γ is some context i.e. a list of distinct free variables. For each typed term $\Gamma \vdash t$, the free variables of t are contained in Γ. Note that because the χ_P are treated as parameters, we do not include them as variables in our typing: This will also play a role when we set up our monadic

translation on terms of E-HA$^\omega$, in which the χ_P will be interpreted by some fixed term.

The language of E-HA$^\omega$ includes quantifiers for all types, together with an equality symbol $=_0$ at base type (equality $=_\rho$ for arbitrary types is defined inductively in terms of $=_0$). The axioms and rules of E-HA$^\omega$ include those of intuitionistic logic in all finite types, axioms which govern the terms, along with induction, equality and extensionality axioms. Note that because E-HA$^\omega$ is an equational theory, axioms for the constants are equations rather than rewrite rules.

$$\Gamma \cup \{x:\rho\} \vdash x:\rho \quad \Gamma \vdash 0:0 \quad \Gamma \vdash \mathrm{sc}:0\to 0$$
$$\Gamma \vdash p_i : \rho_i(\rho_0\times\rho_1) \quad \text{for } i=0,1 \qquad \Gamma \vdash \langle\,,\rangle : (\rho_0\times\rho_1)\rho_1\rho_0$$
$$\frac{\Gamma\cup\{x:\rho\}\vdash t:\tau}{\Gamma\vdash \lambda x.t:\rho\to\tau} \qquad \frac{\Gamma\vdash s:\rho \;\; \Delta\vdash t:\rho\to\tau}{\Gamma\cup\Delta\vdash ts:\tau}$$
$$\Gamma \vdash R_\rho : \rho \to (0\to\rho\to\rho)\to 0\to\rho$$
$$\Gamma \vdash \chi_P : 0^{r_P}\to 0$$

Figure 1. Terms of E-HA$^\omega$

2.3 Heyting arithmetic with state (E-HA$^\omega_S$)

Our extension E-HA$^\omega_S$ of E-HA$^\omega$ with a state type S is not something that we will specify entirely, as we want to work in the most general framework possible. Rather, we provide a series of conditions that it must satisfy in order for what follows to make sense. Note that all of our examples in Section 6 can be properly formalised if we consider E-HA$^\omega_S$ to be the definitional extension E-HA$^\omega_*$ of E-HA$^\omega$ which includes finite sequence types ρ^*. Then states will be encodable as objects of type 0^*. Therefore if the reader prefers, they can just take E-HA$^\omega_S$ to be E-HA$^\omega_*$, although in principle Section 4-5 apply in more general setting.

The types of E-HA$^\omega_S$ consist of the base type 0 and must allow the construction of product and function types. In addition, they include a special type S of *states*, which could be either a basic type or defined in terms of the others (as $S := 0^*$, for example). The terms of E-HA$^\omega_S$ include at the very least those built from the constants and rules given in Figure 1. Note that we now include variables of state type, which we typically denote $\pi^S, \sigma^S, \ldots$, and so in particular allow lambda-abstraction, primitive recursion etc. over state types. Similarly, the language of E-HA$^\omega_S$ includes in addition quantifiers for types built from the state, together with an equality symbol $=_S$ between states, and the usual axioms and rules of E-HA$^\omega$ extended to encompass the state type.

In addition, we make use of two new predicates which act on states: A unary 'truth' predicate T together with a binary 'extension' predicate $\sqsubseteq$. These are required to satisfy the following conditions:

(A) $\sqsubseteq$ is reflexive and transitive;
(B) $\forall\pi,\sigma(\pi\sqsubseteq\sigma\wedge T(\sigma)\to T(\pi))$.

Officially, these will be parameters of our interpretation, but in practice they will be instantiated by something concrete (see Section 6). For now, the reader should have in mind the following informal interpretations:

- States π are some abstract data structures which encode information or assumptions about our parameters P;
- $T(\pi)$ is a predicate which denotes the conjunction of all assumptions encoded by our state;
- $\pi\sqsubseteq\sigma$ indicates that σ contains more information than π.

The condition (B) simply says that if all assumptions encoded by σ are true, and σ contains more information than π, then all assumptions encoded by π must be true. We remark that E-HA$^\omega$ can always be embedded in E-HA$^\omega_S$, and we refer to (the embedding of) types and terms of E-HA$^\omega$ as 'pure' types and terms respectively.

3 Gödel's functional interpretation

We now outline the functional interpretation. This section is essentially standard material, and a full exposition can be found in e.g. [9]. However, for our purposes, we will need a slightly more explicit presentation, in which interpreted formulas are given as substitution instances of some quantifier-free skeleton. A similar decomposition is used in [6].

The functional interpretation maps each formula A in HA$^+$ to a quantifier-free formula $|A|^{\vec{x}}_{\vec{y}}$ in E-HA$^\omega$ (where $\vec{x},\vec{y}$ are possibly empty tuples of variables), inductively over the structure of A as given in Figure 2, where $P\vee_i Q$ denotes $(i=0\to P)\wedge(i\neq 0\to Q)$.

$$|A| :\equiv A \text{ for } A \text{ atomic} \quad |A\wedge B|^{\vec{x},\vec{u}}_{\vec{y},\vec{v}} :\equiv |A|^{\vec{x}}_{\vec{y}} \wedge |B|^{\vec{u}}_{\vec{v}}$$
$$|A\vee B|^{i,\vec{x},\vec{u}}_{\vec{y},\vec{v}} :\equiv |A|^{\vec{x}}_{\vec{y}} \vee_i |B|^{\vec{u}}_{\vec{v}}$$
$$|A\to B|^{\vec{U},\vec{Y}}_{\vec{x},\vec{v}} :\equiv |A|^{\vec{x}}_{\vec{Y}\vec{x}\vec{v}} \to |B|^{\vec{U}\vec{x}}_{\vec{v}}$$
$$|\exists z A(z)|^{x,\vec{u}}_{\vec{v}} :\equiv |A(x)|^{\vec{u}}_{\vec{v}} \quad |\forall z A(z)|^{\vec{U}}_{x,\vec{v}} :\equiv |A(x)|^{\vec{U}x}_{\vec{v}}$$

Figure 2. The functional interpretation

We break down the functional interpretation into a substitution and a skeleton component. This decomposition will play a role in Section 5. To each formula A of HA$^+$ we assign a skeleton formula $A_s(\vec{a};\vec{b})$ which is *also* a formula of HA$^+$ (strictly speaking its embedding in E-HA$^\omega$). The variables $\vec{b}$ represent the free variables of the original formula A, while $\vec{a}$ are those which arise through the interpretation. Associated with A_s is a closed term $r_A:\vec{\rho}\to\vec{\tau}\to 0^n$ (where $\vec{\rho}\to 0$ is shorthand for $\rho_1\to\ldots\to\rho_k\to 0$), where $\vec{\rho},\vec{\tau}$ are the types of $\vec{x},\vec{y}$ in $|A|^{\vec{x}}_{\vec{y}}$ and n is the length of $\vec{a}$ (in the case $n=0$ we leave r_A undefined). The full definition is given in Figure 3[1]. Here $\vec{b}\cup\vec{d}$ denotes the union of both sequences of variables (so identical variables are only counted once).

A simple induction over the structure of $A(\vec{b})$ proves that

$$\left|A(\vec{b})\right|^{\vec{x}}_{\vec{y}} \equiv A_s(r_A\vec{x}\vec{y}/\vec{a};\vec{b}),$$

where $\langle t_1,\ldots,t_n\rangle/\vec{a}$ denotes the substitution $t_1/a_1,\ldots,t_n/a_n$.

Theorem 3.1. *Suppose that $A(\vec{b})$ is a formula of HA$^+$ with only $\vec{b}$ free, and that HA$^+\vdash A(\vec{b})$. Then*

$$\textit{E-HA}^\omega \vdash \forall\vec{y}\left|A(\vec{b})\right|^{\vec{s}\vec{b}}_{\vec{y}}$$

[1] Note that in the definition of r_A, the induction doesn't begin with the atomic formulas, but rather the quantifiers or disjunction. In cases where r_A is undefined because e.g. A is atomic, $r_{A\wedge B}$ and so on are defined in the obvious way.

$$A_s(\,;\vec{b}) :\equiv A \text{ for } A \text{ atomic, where } \vec{b} = FV(A)$$
$$(A \wedge B)_s(\vec{a},\vec{c};\vec{b} \cup \vec{d}) :\equiv A_s(\vec{a};\vec{b}) \wedge A_s(\vec{c};\vec{d})$$
$$(A \vee B)_s(e,\vec{a},\vec{c};\vec{b} \cup \vec{d}) :\equiv A_s(\vec{a};\vec{b}) \vee_e B_s(\vec{c};\vec{d})$$
$$(A \to B)_s(\vec{a},\vec{c};\vec{b} \cup \vec{d}) :\equiv A_s(\vec{a};\vec{b}) \to B_s(\vec{c};\vec{d})$$
$$(\exists z A(z))_s(e,\vec{a};\vec{b}) \equiv (\forall z A(z))_s(e,\vec{a};\vec{b}) :\equiv A(e)_s(\vec{a};\vec{b} \cup e)$$

$$r_{A\wedge B}\vec{x}\vec{u}\vec{y}\vec{v} := \langle r_A\vec{x}\vec{y}, r_B\vec{u}\vec{v}\rangle \quad r_{A\vee B}i\vec{x}\vec{u}\vec{y}\vec{v} := \langle i, r_A\vec{x}\vec{y}, r_B\vec{u}\vec{v}\rangle$$
$$r_{A\to B}\vec{U}\vec{Y}\vec{x}\vec{v} := \langle r_A\vec{x}(\vec{Y}\vec{x}\vec{v}), r_B(\vec{U}\vec{x})\vec{v}\rangle$$
$$r_{\exists z A(z)}x\vec{u}\vec{v} := \langle x, r_{A(x)}\vec{u}\vec{v}\rangle \quad r_{\forall z A(z)}\vec{U}x\vec{v} := \langle x, r_{A(x)}(\vec{U}x)\vec{v}\rangle$$

Figure 3. A_s and r_A

where $\vec{s}$ is a sequence of closed terms of E-HA$^\omega$ (which could nevertheless contain function parameters χ_P), which can be extracted from the proof of $A(\vec{b})$. By precomposing with the negative translation, an analogous result holds for PA$^+$.

Proof. This is standard, and follows from the usual soundness proofs found in e.g. [9]. To be absolutely formal regarding our parameters χ_P, we can regard a proof of the formula $A(\vec{b})$ as a proof of the formula $A(\vec{\chi},\vec{b})$ in the weakly extensional variant WE-HA$^\omega$ of Heyting arithmetic in finite types, where $\vec{\chi}$ is a sequence of parameters used in the proof, which are now treated as function symbols in WE-HA$^\omega$. By the usual functional interpretation of WE-HA$^\omega$ we can extract a sequence of *genuinely* closed terms $\vec{s}$ such that $\mathsf{E\text{-}HA}^\omega \vdash \forall\vec{y}\left|A(\vec{\chi},\vec{b})\right|^{\vec{s}\vec{\chi}\vec{b}}_{\vec{y}}$. Then the sequence $\vec{s}\vec{\chi}$, which is closed in our sense, does the job. □

4 A monadic translation from E-HA$^\omega$ to E-HA$^\omega_S$

We now outline the first key steps in constructing our functional interpretation with state: A translation from pure terms of E-HA$^\omega$ to state sensitive terms of E-HA$^\omega_S$. To do this we will make use of the state monad: a mapping M on types defined by

$$M\rho := S \to \rho \times S.$$

Monads are a well known technique for modelling imperative structures within a pure functional calculus [12]. A key aspect of our translation will be to replace the characteristic functions $\chi_P : 0^{r_P} \to 0$ for each predicate parameter by some new functions $\varphi_P : 0^{r_P} \to S \to 0 \times S$ which are *monotone*, in the sense that

$$\forall x, \pi(\pi \sqsubseteq \varphi_P x \pi_1),$$

and which satisfy the following relation:

$$\chi_P \approx \varphi_P :\equiv \forall x, \pi(T(\varphi_P x\pi_1) \to \chi_P x = \varphi_P x \pi_0)$$

where $\varphi_P x\pi_i$ for $i = 0,1$ is short for $p_i(\varphi_P x\pi)$. The φ_P act as 'approximate' characteristic functions for the parameters P, in that they are only required to be accurate relative to whatever is in the current state. This allows us to e.g. *assume* the truth of $P(x)$ by simply adding it to our state. The monotonicity condition intuitively means that the φ_P can only add information to the state, and cannot erase existing information. The reader may prefer to take a quick look ahead at Section 6 to get an idea of what this all means in a concrete setting.

4.1 The translation $t \mapsto [t]$

To each type ρ in E-HA$^\omega$ (which corresponds also to a pure type in E-HA$^\omega_S$) we assign a type $[\rho]$ of E-HA$^\omega_S$ inductively as

$$[0] := 0 \quad [\rho \times \tau] := [\rho] \times [\tau] \quad [\rho \to \tau] := [\rho] \to M[\tau].$$

In defining the translation on terms we will make use of the usual *unit* and *bind* operations for this monad. More specifically, we define $\eta : \rho \to M\rho$ by

$$\eta x := \lambda\pi.\langle x, \pi\rangle.$$

and $\circ : (\rho \to M\tau) \to M\rho \to M\tau$ by

$$f \circ x := \lambda\pi.f(x\pi_0)(x\pi_1).$$

We will primarily use the bind operation in the form of the associated operation $* : M(\rho \to M\tau) \to M\rho \to M\tau$ given by

$$g * x := \lambda\pi.(g\pi_0 \circ x)(g\pi_1).$$

Finally, if $x : [\rho]$ and $t : M[\tau]$ we define 'neutral' lambda abstraction $\lambda^* x.t : M[\rho \to \tau]$ as

$$\lambda^* x.t := \lambda\pi.\langle \lambda x.t, \pi\rangle.$$

Entirely analogously, for $x_1 : [\rho_1], \ldots, x_n : [\rho_n]$ and $t : M[\tau]$ we define $\lambda^* x_1, \ldots, x_n.t : M[\vec{\rho} \to \tau]$ as shorthand for repeated neutral abstraction i.e.

$$\lambda^* x_1, \ldots, x_n.t := \lambda\pi.\langle \lambda x_1 \lambda^* x_2, \ldots, x_n.t, \pi\rangle.$$

Now, formally our translation will take some typed term $\Gamma \vdash t : \rho$ of E-HA$^\omega$, and will translate it to some typed term $[\Gamma \vdash t : \rho] := [\Gamma] \vdash [t] : M[\rho]$ of E-HA$^\omega_S$, where if $\Gamma := x_1 : \rho_1, \ldots, x_n : \rho_n$ then $[\Gamma] := y_1 : [\rho_1], \ldots, y_n : [\rho_n]$ is a new context, which assigns to each $x_i : \rho_i$ a fresh variable $y_i : [\rho_i]$. Officially, the translation is parametrised by some $\varphi_P : 0^{r_P} \to M0$ and should be written as $[t]_{\varphi_P}$, although in practice it can be safely omitted from the notation. The full translation is given below:

- $[\Gamma \cup \{x : \rho\} \vdash x : \rho] := [\Gamma] \cup \{y : [\rho]\} \vdash \eta y : M[\rho]$;
- $[\Gamma \vdash 0 : 0] := [\Gamma] \vdash \eta 0 : M[0]$ and $[\Gamma \vdash \mathrm{sc} : 00] := [\Gamma] \vdash \lambda^* x.\eta(\mathrm{sc}(x)) : M[00]$;
- $[\Gamma \vdash p_i : \rho_i(\rho_0 \times \rho_1)] := [\Gamma] \vdash \lambda^* x.\eta(p_i x) : M[\rho_i(\rho_0 \times \rho_1)]$ for $i = 0,1$ (where on the r.h.s. p_i denotes the projection of type $[\rho_i]([\rho_0] \times [\rho_1])$);
- $[\Gamma \vdash \langle\,,\rangle : (\rho \times \tau)\tau\rho] := [\Gamma] \vdash \lambda^* x, y.\eta\langle x, y\rangle : M[(\rho \times \tau)\tau\rho]$
- If $[\Gamma \cup \{x : \rho\} \vdash t : \tau] = [\Gamma] \cup \{y : [\rho]\} \vdash [t] : M[\tau]$ then
$$[\Gamma \vdash \lambda x.t : \rho \to \tau] := [\Gamma] \vdash \lambda^* y.[t] : M[\rho \to \tau]$$
- If $[\Gamma \vdash s : \rho] = [\Gamma] \vdash [s] : M[\rho]$ and $[\Delta \vdash t : \rho \to \tau] = [\Delta] \vdash [t] : M[\rho \to \tau]$ then
$$[\Gamma \cup \Delta \vdash ts : \tau] := [\Gamma] \cup [\Delta] \vdash [t] * [s] : M[\tau]$$
- $[\Gamma \vdash R_\rho : \rho 0(\rho\rho 0)\rho] := [\Gamma] \vdash \lambda^* b, g, x.R^*_\rho bgx : M[\rho 0(\rho\rho 0)\rho]$ where $R^*_\rho : [\rho] \to [\rho\rho 0] \to 0 \to M[\rho]$ is defined using $R_{M[\rho]}$ as
$$R^*_\rho bg0 = \eta b$$
$$R^*_\rho bg(\mathrm{sc}(x)) = gx * R^*_\rho bgx$$
- $[\Gamma \vdash \chi_P : 0^{r_P} \to 0] := [\Gamma] \vdash \eta(\varphi_P) : M[0^{r_P} \to 0]$.

This is a fairly straightforward call-by-value monadic translation in all finite types which simply replaces the characteristic functions χ_P with some new φ_P. For example, for the interpretation of function application, we take as our input state π and apply it first to $[t]$, so that $[t]\pi = \langle u, \pi'\rangle$ for some $u : [\rho \to \tau]$ and intermediate

state π'. We then apply π' to $[s]$, so that $[s]\pi' = \langle v, \pi''\rangle$ for some $v : [\rho]$. Finally, we provide u with v and the current state π'' as arguments, so that $uv\pi'' = \langle w, \pi'''\rangle$ for some $w : [\tau]$ and final state π'''. Unwinding the definition of $[ts]$ yields $[ts]\pi = \langle w, \pi'''\rangle$.

Having established our monadification of terms of E-HA$^\omega$, we need to characterise how they are related to the original pure terms.

4.2 The relation $t \sim^M_\rho [t]$

The function φ_P has two crucial properties: that it is monotone, and that it is an approximation of χ_P. We will generalise both of these properties so that they apply to arbitrary terms, starting with monotonicity.

For each pure type ρ (i.e. of E-HA$^\omega$), we define two predicates mon_ρ and mon^M_ρ on objects of type $[\rho]$ and $M[\rho]$ respectively, by induction as follows (recall that $z\pi_i = p_i(z\pi)$):

$$\begin{aligned}
\mathrm{mon}_0(y) &:\equiv 0 =_0 0\\
\mathrm{mon}_{\rho\times\tau}(y) &:\equiv \mathrm{mon}_\rho(p_0 y) \wedge \mathrm{mon}_\tau(p_1 y)\\
\mathrm{mon}_{\rho\to\tau}(g) &:\equiv \forall y^{[\rho]}(\mathrm{mon}_\rho(y) \to \mathrm{mon}^M_\tau(gy))\\
\mathrm{mon}^M_\rho(z) &:\equiv \forall \pi(\pi \sqsubseteq z\pi_1 \wedge \mathrm{mon}_\rho(z\pi_0)).
\end{aligned}$$

We now generalise the approximation condition by defining, for each pure type ρ, a pair of relations $x^\rho \approx_\rho y^{[\rho]}$ and $x^\rho \approx^M_\rho z^{M[\rho]}$ by induction as follows:

$$\begin{aligned}
x^0 \approx_0 y^0 &:\equiv (x =_0 y)\\
x^{\rho\times\tau} \approx_{\rho\times\tau} y^{[\rho]\times[\tau]} &:\equiv p_0 x \approx_\rho p_0 y \wedge p_1 x \approx_\tau p_1 y\\
f^{\rho\to\tau} \approx_{\rho\to\tau} g^{[\rho\to\tau]} &:\equiv \forall x^\rho, y^{[\rho]}(x \approx_\rho y \wedge \mathrm{mon}_\rho(y) \to fx \approx^M_\tau gy)\\
x^\rho \approx^M_\rho z^{M[\rho]} &:\equiv \forall \pi^S(T(z\pi_1) \to x \approx_\rho z\pi_0).
\end{aligned}$$

Finally, we define the relations $x^\rho \sim_\rho y^{[\rho]}$ and $x^\rho \sim^M_\rho z^{M[\rho]}$ by

$$x \sim_\rho y :\equiv x \approx_\rho y \wedge \mathrm{mon}_\rho(y) \quad \text{and} \quad x \sim^M_\rho z :\equiv x \approx^M_\rho z \wedge \mathrm{mon}^M_\rho(z).$$

Lemma 4.1. *The relations $\mathrm{mon}_\rho, \mathrm{mon}^M_\rho, \approx_\rho, \approx^M_\rho, \sim_\rho$ and $\sim^M_\rho$ are compatible with equality (i.e. extensional), provably in E-HA$^\omega_S$.*

Proof. A simple induction on the types. □

Lemma 4.2. *E-HA$^\omega_S$ proves that for any ρ, τ:*

$$\mathrm{mon}^M_\rho(z) \wedge \mathrm{mon}^M_{\rho\to\tau}(h) \to \mathrm{mon}^M_\tau(h * z).$$

Proof. We first show that

$$\mathrm{mon}^M_\rho(z) \wedge \mathrm{mon}_{\rho\to\tau}(g) \to \mathrm{mon}^M_\tau(g \circ z). \tag{4}$$

For some initial state π, from $\mathrm{mon}^M_\rho(z)$ we have $\pi \sqsubseteq z\pi_1$ and $\mathrm{mon}_\rho(z\pi_0)$, and hence from $\mathrm{mon}_{\rho\to\tau}(g)$ we obtain $\mathrm{mon}^M_\tau(g(z\pi_0))$ which in turn implies $z\pi_1 \sqsubseteq g(z\pi_0)(z\pi_1)_1$ and $\mathrm{mon}_\tau(g(z\pi_0)(z\pi_1)_0)$. But since by transitivity of $\sqsubseteq$ we have $\pi \sqsubseteq g(z\pi_0)(z\pi_1)_1$ it follows that $\mathrm{mon}^M_\tau(g \circ z)$.

Now, suppose that $\mathrm{mon}^M_{\rho\to\tau}(h)$. Then for arbitrary π we have $\pi \sqsubseteq h\pi_1$ and $\mathrm{mon}_{\rho\to\tau}(h\pi_0)$, and so setting $g := h\pi_0$ in (4) we obtain $\mathrm{mon}^M_\tau(h\pi_0 \circ z)$, from which $\mathrm{mon}^M_\tau(h * z)$ follows, again using transitivity of $\sqsubseteq$. □

Lemma 4.2 confirms that the monotonicity predicates are well behaved with respect to substitution, which allows us to establish monotonicity of complex terms by induction over the term structure. This is the next result:

Lemma 4.3. *For any term $\Gamma \vdash t : \rho$ of E-HA$^\omega$ (viewed as a term of E-HA$^\omega_S$), E-HA$^\omega_S$ proves that*

$$\mathrm{mon}([\Gamma]) \to \mathrm{mon}^M_\rho([t])$$

where $\mathrm{mon}(y_1 : [\rho_1], \ldots, y_n : [\rho_n])$ denotes $\mathrm{mon}_{\rho_1}(y_1) \wedge \ldots \wedge \mathrm{mon}_{\rho_n}(y_n)$.

Proof. A routine induction on the structure of t, using Lemmas 4.1 and 4.2. □

Having dealt with monotonicity, we now prove a corresponding substitution lemma for $\sim^M_\rho$.

Lemma 4.4. *E-HA$^\omega_S$ proves that for any ρ, τ:*

$$x \sim^M_\rho z \wedge f \sim^M_{\rho\to\tau} h \to fx \sim^M_\tau h * z.$$

Proof. Analogously to the proof of Lemma 4.2, we first claim that

$$x \sim^M_\rho z \wedge f \sim_{\rho\to\tau} g \to fx \sim^M_\tau g \circ z.$$

Note that $\mathrm{mon}^M_\tau(g \circ z)$ follows as in Lemma 4.2, so it remains to prove $fx \approx^M_\tau g \circ z$. Now, from $x \approx^M_\rho z$ we obtain

$$T(z\pi_1) \to x \approx_\rho z\pi_0$$

and from $f \approx_{\rho\to\tau} g$ we obtain (using also $\mathrm{mon}_\rho(z\pi_0)$)

$$T(z\pi_1) \to fx \approx^M_\tau g(z\pi_0).$$

and substituting in the state $z\pi_1$ yields

$$T(z\pi_1) \to (T(g(z\pi_0)(z\pi_1)_1) \to fx \approx_\tau g(z\pi_0)(z\pi_1)_0).$$

Now using the fact that $z\pi_1 \sqsubseteq g(z\pi_0)(z\pi_1)_1$ (by monotonicity of $g(z\pi_0)$) we have, using the state axiom,

$$T(g(z\pi_0)(z\pi_1)_1) \to T(z\pi_1)$$

and therefore putting the last two equations together we end up with

$$T(g(z\pi_0)(z\pi_1)_1) \to fx \approx_\tau g(z\pi_0)(z\pi_1)_0,$$

which, noting that $g(z\pi_0)(z\pi_1) = (g \circ z)\pi$, is just $fx \approx^M_\tau g \circ z$, which proves our claim.

For the main result, we know that $\mathrm{mon}^M_\tau(h * z)$ must hold by Lemma 4.2, so it remains to show $fx \approx^M_\tau h * z$. Using $f \sim^M_{\rho\to\tau} h$ we have

$$T(h\pi_1) \to f \sim_{\rho\to\tau} h\pi_0$$

and using the claim we end up with

$$T(h\pi_1) \to fx \sim^M_\tau h\pi_0 \circ z.$$

Substituting in the current state $h\pi_1$ yields

$$T(h\pi_1) \to (T((h\pi_0 \circ z)(h\pi_1)_1) \to fx \approx_\tau (h\pi_0 \circ z)(h\pi_1)_0)$$

and by using $h\pi_1 \sqsubseteq (h\pi_0 \circ z)(h\pi_1)_1$ (by monotonicity of $(h\pi_0 \circ z)$) and noting that $(h\pi_0 \circ z)(h\pi_1) = (h * z)\pi$ we obtain

$$T((h * z)\pi_1) \to fx \approx_\tau (h * z)\pi_0,$$

which is just $fx \approx^M_\tau h * z$, and so we're done. □

We now use this substitution Lemma to establish the main result of the section:

Theorem 4.5. *For any term $\Gamma \vdash t : \rho$ of E-HA$^\omega$ (viewed as a term of E-HA$^\omega_S$), E-HA$^\omega_S$ proves that*

$$\Gamma \sim [\Gamma] \to t \sim^M_\rho [t]$$

where $x_1 : \rho_1, \ldots, x_n : \rho_n \sim y_1 : [\rho_1], \ldots, y_n : [\rho_n]$ is shorthand for $x_1 \sim_{\rho_1} y_1 \wedge \ldots \wedge x_n \sim_{\rho_n} y_n$.

Proof. Induction using Lemmas 4.1, 4.2 and 4.4. □

5 The functional interpretation with state

We now come to the main result of the paper. In the previous section we defined a translation which takes pure terms $t : \rho$ and transforms them to state-sensitive terms $[t] : M[\rho]$. The purpose of this section is to reformulate the functional interpretation so that formulas A of HA^+ are, instead of being mapped to a formula $|A|^{\vec{x}}_{\vec{y}}$ of E-HA^ω for $\vec{x} : \vec{\rho}$ and $\vec{y} : \vec{\tau}$, mapped to a formula $\{A\}^{\vec{u}}_{\vec{v}}$ of E-HA^ω_S, where now $\vec{u} : M[\vec{\rho}]$ and $\vec{v} : [\vec{\tau}]$ (here $M[\rho_1, \dots, \rho_n]$ is shorthand for $M[\rho_1], \dots, M[\rho_n]$). Note that $\vec{v}$ is a tuple of free input variables, which is why they have type $[\vec{\tau}]$ instead of $M[\vec{\tau}]$, in line with our monadic translation.

In addition to this, our functional interpretation with state will include an explicit state component $\|A\|^{\vec{u}}_{\vec{v}}$, which takes as an argument an input state π and returns the truth value of an output state π', which in turn is the result of each of the realizing terms evaluating from the initial state π.

We now make all of this precise. Suppose that r_A is as in Section 3: Note that r_A is not defined for all formulas A, only for those whose functional interpretation does not consist of tuples of empty variables. For such formulas, suppose that $r_A : \vec{\rho} \to \vec{\tau} \to 0^n$ (where $n > 0$). Then we define $r^M_A : M[\vec{\rho}] \to M[\vec{\tau}] \to M0^n$ by

$$r^M_A \vec{u} \vec{v} :=_{M0^n} [r_A] * u_1 * \ldots * u_i * v_1 * \ldots * v_j$$

where $g * u * v$ denotes $(g * u) * v$. Note that because r_A does not contain any predicate parameters χ_P, its interpretation $[r_A]$ is independent of the parameter φ.

Lemma 5.1. *Suppose that $\vec{t} : \vec{\rho}$ and $\vec{s} : \vec{\tau}$ are terms of E-HA^ω. Then we have (provably in E-HA^ω_S)*

$$r^M_A[\vec{t}][\vec{s}] =_{M0^n} [r_A \vec{t} \vec{s}]$$

where $[t_1, \dots, t_i]$ denotes $[t_1], \dots, [t_i]$.

Proof. Direct from unwinding the definition of $[r_A \vec{t} \vec{s}]$. □

We are now ready to define our new functional interpretation. To each formula A of HA^+ and we assign a pair of formulas $\|A\|^{\vec{u}}_{\vec{v}} \pi$, $\{A\}^{\vec{u}}_{\vec{v}} \pi$ (which now contain the input state π as an additional free variable) as follows:

$$\|A\|^{\vec{u}}_{\vec{v}} \pi :\equiv T(r^M_A \vec{u} \eta(\vec{v}) \pi_1) \quad \text{or just } T(\pi) \text{ if } r_A \text{ is not defined}$$
$$\{A\}^{\vec{u}}_{\vec{v}} \pi :\equiv A_s(r^M_A \vec{u} \eta(\vec{v}) \pi_0 / \vec{a}; \vec{b}),$$

where $\eta(v_1, \dots, v_i) := \eta v_1, \dots, \eta v_i$. Note that these are well typed: In the case that $\vec{a}$ has length $n > 0$ we have $r^M_A \vec{u} \eta(\vec{v}) \pi : 0^n \times S$, and otherwise we simply have $\|A\| \pi \equiv T(\pi)$ and $\{A\} \pi \equiv A_s(\,; \vec{b})$. Finally, for each pure type ρ we define, in E-HA^ω_S, a class of functionals Δ_ρ of type $[\rho]$ by

$$\Delta_\rho(z^{[\rho]}) := \exists y^\rho (y \sim_\rho z)$$

and write $z \in \Delta_\rho$ as shorthand for $\Delta_\rho(z)$. The class Δ^M_ρ of functional of type $M[\rho]$ is defined analogously with $\sim^M_\rho$.

Theorem 5.2. *Suppose that $A(\vec{b})$ is a formula of HA^+ with only $\vec{b}$ free, and that $\vec{s} : \vec{0} \to \vec{\rho}$ is a sequence of closed terms of E-HA^ω satisfying*

$$\textit{E-HA}^\omega \vdash \forall \vec{y}^{\vec{\tau}} \left| A(\vec{b}) \right|^{\vec{s}\vec{b}}_{\vec{y}}$$

Define $\vec{t} : \vec{0} \to M[\vec{\rho}]$ by^2 $\lambda \vec{b}.[\vec{s}] * \eta(\vec{b})$. Then $\vec{t}\vec{b} \in \Delta^M_{\vec{\rho}}$ and

$$\textit{E-HA}^\omega_S \vdash \forall \vec{v} \in \Delta_{\vec{\tau}}, \pi \left(\left\| A(\vec{b}) \right\|^{\vec{t}\vec{b}}_{\vec{v}} \pi \to \left\{ A(\vec{b}) \right\}^{\vec{t}\vec{b}}_{\vec{v}} \pi \right).$$

Theorem 5.3 (Main soundness theorem)**.** *Suppose that $HA^+ \vdash A(\vec{b})$. Then for any collection of approximations φ_P satisfying $\chi_P \approx_{0 \to 0} \varphi_P$, there is a corresponding sequence of state-sensitive terms $\vec{t}$ satisfying*

$$\textit{E-HA}^\omega_S \vdash \forall \vec{v} \in \Delta_{\vec{\tau}}, \pi \left(\left\| A(\vec{b}) \right\|^{\vec{t}\vec{b}}_{\vec{v}} \pi \to \left\{ A(\vec{b}) \right\}^{\vec{t}\vec{b}}_{\vec{v}} \pi \right)$$

which can be formally extracted from the proof of $A(\vec{b})$.

Proof. By the usual soundness theorem 3.1 we can extract some $\vec{s}$ satisfying E-$\mathsf{HA}^\omega \vdash \forall \vec{y} \left| A(\vec{b}) \right|^{\vec{s}\vec{b}}_{\vec{y}}$. The approximations φ_P then induce a corresponding monadic interpretation $[\cdot]_\varphi$, and the result follow from Theorem 5.2. □

Proof of Theorem 5.2. First, for the case that A is non-computational i.e. is mapped to some $A_s(\,; \vec{b}) \equiv A$, then the result trivially follows from $T(\pi) \to A$, so we assume that A is computational and therefore r_A is properly defined.

Take some arbitrary $\vec{v} \in \Delta_{\vec{\tau}}$ and let $\vec{y} : \vec{\tau}$ be such that $\vec{y} \sim_{\vec{\tau}} \vec{v}$. By our assumption we have

$$\text{E-}\mathsf{HA}^\omega \vdash A_s(r_A(\vec{s}\vec{b})\vec{y}/\vec{a}; \vec{b}). \tag{5}$$

Define the context $\Gamma := \vec{b} : \vec{0}, \vec{y} : \vec{\tau}$. Then $\Gamma \vdash r_A(\vec{s}\vec{b})\vec{y}$. Setting $[\Gamma] := \vec{b} : \vec{0}, \vec{v} : [\vec{\tau}]$ then by assumption we have $\Gamma \sim [\Gamma]$, and so by Theorem 4.5 it follows that

$$\text{E-}\mathsf{HA}^\omega_S \vdash r_A(\vec{s}\vec{b})\vec{y} \sim^M_{0^n} [r_A(\vec{s}\vec{b})\vec{y}]. \tag{6}$$

Here of course, $[r_A(\vec{s}\vec{b})\vec{y}]$ is defined relative to $[\Gamma]$, and in particular, by Lemma 5.1, we have

$$[r_A(\vec{s}\vec{b})\vec{y}] = r^M_A[\vec{s}\vec{b}][\vec{y}] = r^M_A(\vec{t}\vec{b})\eta(\vec{v}) \tag{7}$$

where for the second equality we have $[\vec{s}\vec{b}] = [\vec{s}] * \eta(\vec{b}) = \vec{t}\vec{b}$. Also note that, again by Theorem 4.5, we have $\vec{s}\vec{b} \sim^M_{\vec{\rho}} [\vec{s}\vec{b}] = \vec{t}\vec{b}$ and therefore $\vec{t}\vec{b} \in \Delta^M_{\vec{\rho}}$.

Now, substituting (7) into (6) and expanding the definition of $\sim^M_{0^n}$ we have

$$\text{E-}\mathsf{HA}^\omega_S \vdash T(r^M_A(\vec{t}\vec{b})\eta(\vec{v})\pi_1) \to r_A(\vec{s}\vec{b})\vec{y} =_{0^n} r^M_A(\vec{t}\vec{b})\eta(\vec{v})\pi_0 \tag{8}$$

But by extensionality we have

$$\begin{aligned}\text{E-}\mathsf{HA}^\omega_S \vdash & r_A(\vec{s}\vec{b})\vec{y} =_{0^n} r^M_A(\vec{t}\vec{b})\eta(\vec{v})\pi_0 \\ & \to (A_s(r_A(\vec{s}\vec{b})\vec{y}/\vec{a}; \vec{b}) \to A_s(r^M_A(\vec{t}\vec{b})\eta(\vec{v})\pi_0/\vec{a}; \vec{b}))\end{aligned}$$

and combining this with (5) yields

$$\text{E-}\mathsf{HA}^\omega_S \vdash r_A(\vec{s}\vec{b})\vec{y} =_{0^n} r^M_A(\vec{t}\vec{b})\eta(\vec{v})\pi_0 \to A_s(r^M_A(\vec{t}\vec{b})\eta(\vec{v})\pi_0/\vec{a}; \vec{b}).$$

But putting this together with (8) we obtain

$$\text{E-}\mathsf{HA}^\omega_S \vdash \underbrace{T(r^M_A(\vec{t}\vec{b})\eta(\vec{v})\pi_1)}_{\|A\|^{\vec{t}\vec{b}}_{\vec{v}} \pi} \to \underbrace{A_s(r^M_A(\vec{t}\vec{b})\eta(\vec{v})\pi_0/\vec{a}; \vec{b})}_{\{A\}^{\vec{t}\vec{b}}_{\vec{v}} \pi}$$

Quantifying over π and $\vec{v}$ yields the result. □

[2] where $[\vec{s}] * \eta(\vec{b})$ is shorthand for $[s_1] * \eta b_1 * \ldots * \eta b_i, \ldots, [s_n] * \eta b_1 * \ldots * \eta b_i$

The fact that we restrict our variables $\vec{v} : [\vec{\rho}]$ to inhabiting the class $\Delta_{\vec{\tau}}$ is an essential part of result. Technically speaking, it allows us to take a detour through the usual functional interpretation by viewing $\vec{v}$ as the monotone approximation of some normal variables $\vec{y} : \vec{\rho}$. However, while this condition may seem artificial at first glance, in reality it forms a natural and reasonable restriction. If we have $v \in \Delta_\rho$, it tells us two key things: Firstly that v is monotone, and so according to our intuitive reading does not destroy anything in the state. The second, that it satisfies $y \approx_\rho v$, is more subtle, but essentially characterizes v as being 'consistent' with respect to the state. We will see a concrete example of the role both conditions play in Example 6.2.

We now give, as special cases, the (classical) functional interpretation for Π^0_2 and Σ^0_2 formulas.

Example 5.4. Suppose that $\mathsf{PA}^+ \vdash \forall x \exists y Q(x,y)$ where $Q(x,y)$ is quantifier-free and contains only x, y as free variables. By the soundness of the negative translation, we have $\mathsf{HA}^+ \vdash A :\equiv \forall x \neg\neg \exists y Q(x,y)$. But the functional interpretation $|A|^f_x$ is equivalent to $Q(x, fx)$ which can be decomposed as $A_s(a_1, a_2;\ ) \leftrightarrow Q(a_1, a_2)$ and $r_A = \lambda f^{0\to 0}, x.\langle x, fx\rangle$. It is not too difficult to show that

$$[r_A] := \lambda^* g^{[0\to 0]}, x^0 \lambda\pi.\langle x, gx\pi_0, gx\pi_1\rangle$$

and in particular for $u : M[0 \to 0]$ we have

$$r^M_A u(\eta x)\pi = \langle x, (u * (\eta x))\pi_0, (u * (\eta x))\pi_1\rangle.$$

Therefore in this case, the term $t : M[0 \to 0]$ of Theorem 5.2 would satisfy

$$\forall x, \pi\ (T((t * (\eta x))\pi_1) \to Q(x, (t * (\eta x))\pi_0))$$

or simply setting $t'x := (t * (\eta x))$ we would have

$$\forall x, \pi\ (T(t'x\pi_1) \to Q(x, t'x\pi_0))\ .$$

The intuition here is quite straightforward: The monadic interpretation realizes $\forall x \exists y Q(x,y)$ via some program $t' : 0 \to S \to 0 \times S$ which takes as input some argument x and initial state π and returns some output $y := t'x\pi_0$ together with an end state $\sigma := t'x\pi_1$ such that $Q(x,y)$ holds relative to the validity of the state σ.

Example 5.5. Suppose that $\mathsf{PA}^+ \vdash \exists x \forall y Q(x,y)$ for $Q(x,y)$ quantifier-free and containing only x, y as free variables. By the negative translation $\mathsf{HA}^+ \vdash B :\equiv \neg\neg \exists x \forall y Q(x,y)$, and the functional interpretation $|B|^F_f$ is equivalent to $Q(Ff, f(Ff))$ which can be decomposed as $B_s(a_1, a_2;\ ) \leftrightarrow Q(a_1, a_2)$ and $r_B = \lambda F^{(0\to 0)\to 0}, f^{0\to 0}.\langle Ff, f(Ff)\rangle$. We can show that $[r_B]$ is given by

$$\lambda^* G^{[(0\to 0)\to 0]}, g^{[0\to 0]}, \pi.\langle Gg\pi_0, (g \circ Gg)\pi_0, (g \circ Gg)\pi_1\rangle$$

and a further calculation yields

$$r^M_B \Phi^{M[(0\to 0)\to 0]}(\eta g)\pi = \langle (\Phi * (\eta g))\pi_0, (g \circ (\Phi * (\eta g)))\pi_0, (g \circ (\Phi * (\eta g)))\pi_1\rangle.$$

In this case, the term $t : M[(0 \to 0) \to 0]$ of Theorem 5.2 would satisfy

$$\forall g \in \Delta_{0\to 0}, \pi\ (T((g \circ t'g)\pi_1) \to Q(t'g\pi_0, (g \circ t'g)\pi_0))$$

using the abbreviation $t'g := t * (\eta g)$.

In this case, the pure functional interpretation would interpret $\exists x \forall y Q(x,y)$ with some program $s : (0 \to 0) \to 0$, which takes as input some state *insensitive* counter function $f : 0 \to 0$ and returns some output $x := sf$ such that $Q(x,y)$ holds for $y := fx$. The monadic interpretation instead builds a program $t' : (0 \to S \to 0 \times S) \to S \to 0 \times S$ which takes as input some state *sensitive* counter function g and some initial state π, and returns some output $x := t'g\pi_0$ together with an *intermediate* state $\pi' := t'g\pi_1$. We now feed x and π' into g to return some counterexample $y := gx\pi'_0$ together with an end state $\sigma := gx\pi'_1$ such that $Q(x,y)$ holds relative to σ. This is where the restriction $g \in \Delta_{0\to 0}$ plays a role: In particular, it must not delete any conditions imposed by the realizer t'.

6 Examples and applications

In this section we give some insight into how the state can be used, and outline a number of open problems. What follows should be seen as a series of small illustrative examples, together with broad sketches of applications which should be properly developed in future work - in each case there are many details and subtleties that are not fully treated here.

First of all, it is important to stress that our monadic translation is only sensitive to the special state-predicates encoded via our the function parameters $\chi_P, \chi_Q, \ldots$. Therefore it is natural to ask how these should be chosen and used in a formal program extraction. This is something we have left deliberately open, since it will depend very much on the application in question. At one extreme, we could assign a state predicate as a label for each primitive recursive predicate, and use the state to record, for example, every case distinction which arises from an instance of contraction in the formal proof (provided the proof is formalised using these labels).

On the other hand, we may be interested in extracting a term from a specific mathematical theorem which quantifies over some concrete objects, such as a colouring of the natural numbers or a well quasi-ordering. In this case, we could introduce a small number of state-relevant function parameters in our formalisation which represent these objects, thereby narrowing our attention to interactions with the 'relevant' mathematical environment and ignoring other bureaucratic case distinctions made by the program. Of course, in this situation we would require the *user* to make a choice as to which predicates are relevant and to formalise the proof using these predicates in an appropriate way.

6.1 The state as a list of conditions

For our first example, suppose that we have a countable collection of predicate parameters, together with a bijective encoding $P \leftrightarrow i_P$ between these parameters and objects of type 0. We define $\mathsf{E\text{-}HA}^\omega_S$ as $\mathsf{E\text{-}HA}^\omega_*$, that is the usual $\mathsf{E\text{-}HA}^\omega$ but now with finite sequence types ρ^* and a type ι of booleans, and consider S to consist of finite sequences of tuples of the form $\langle i_P, \vec{x}^{r_P}, b^\iota\rangle$. The idea is that $\langle i_P, \vec{x}, 0\rangle$ encodes the atomic formula $P(\vec{x})$, while $\langle i_P, \vec{x}, 1\rangle$ encodes $\neg P(\vec{x})$, and from now on we express elements π as sequences $[Q_1(\vec{x}_1), Q_2(\vec{x}_2), \ldots, Q_k(\vec{x}_k)]$ where $Q_i(\vec{x}_i)$ is a *condition* of the form $P(\vec{x}_i)$ or $\neg P(\vec{x}_i)$ for some parameter P. We then define

$$T([Q_1(\vec{x}_1), \ldots, Q_k(\vec{x}_k)]) = Q_1(\vec{x}_1) \wedge \ldots \wedge Q_k(\vec{x}_k).$$

Note that we make this formal by setting $T(\pi) :\equiv \chi^*(\pi) = 0$, where χ^* is given in terms of the characteristic functions χ_P as

$$\chi^*([\,]) = 0 \quad \chi^*(\pi :: \langle i_P, \vec{x}, 0\rangle) = \chi^*(\pi) + \chi_P(\vec{x})$$
$$\chi^*(\pi :: \langle i_P, \vec{x}, 1\rangle) = \chi^*(\pi) + \bar{\chi}_P(\vec{x})$$

We say that $\pi \sqsubseteq \sigma$ if π is a prefix of σ: this relation is clearly reflexive and transitive, and satisfies the condition (B).

Let us now define a family of approximations to our characteristic functions χ_P, which induces a monadic functional interpretation. Suppose that we have, for each P, some *arbitrary* decision function $d_{r_P} : 0^{r_P} \to S \to \{0,1\}$. We define $\varphi_P : 0^{r_P} \to S \to 0 \times S$ by

$$\varphi_P \vec{x} \pi := \begin{cases} \langle 0, \pi \rangle & \text{if } \langle i_P, \vec{x}, 0 \rangle \in \pi \\ \langle 1, \pi \rangle & \text{if } \langle i_P, \vec{x}, 1 \rangle \in \pi \\ \langle d_P \vec{x} \pi, \pi :: \langle i_P, \vec{x}, d_P \vec{x} \pi \rangle \rangle & \text{otherwise.} \end{cases}$$

The idea behind $\varphi_P \vec{x} \pi$ is as follows: First of all, we check whether or not one of $P(\vec{x})$ or $\neg P(\vec{x})$ are already included in our state. If so, then we return the corresponding truth value and leave our output state unchanged. Otherwise, we make an assumption about $P(\vec{x})$ via the function d_P, and add a record of this assumption to the state.

Lemma 6.1. *For any term $d_P : 0^{r_P} \to S \to \{0,1\}$ of E-HA$^\omega_S$, we have $\chi_P \approx_{0^{r_P} \to 0} \varphi_P$, and hence any choice of the d_Ps for each P induces a monadic translation $[\cdot]$.*

Proof. Straightforward. □

Example 6.2 (Drinker's paradox)**.** Let's now revisit our initial example from Section 1.1 in a more formal setting. We know that $\mathsf{PA}^+ \vdash \exists x \forall y (P(y) \to P(x))$, and therefore following Example 5.5 we obtain a term $t : [0 \to 0] \to S \to 0 \times S$ such that

$$T((g \circ tg)\pi_1) \to (P((g \circ tg)\pi_0) \to P(tg\pi_0)) .$$

Under the above translation, the state sensitive program corresponding to (3) would be

$$tg\pi := \begin{cases} \langle x, \pi' \rangle & \text{if } P(x) \in \pi' \\ \langle 0, \pi' \rangle & \text{if } \neg P(x) \in \pi' \\ \langle x, \pi' :: P(x) \rangle & \text{if } d_P x \pi' = 0 \\ \langle 0, \pi' :: \neg P(x) \rangle & \text{if } d_P x \pi' = 1 \end{cases}$$

where $\langle x, \pi' \rangle := g0\pi$. A direct verification of this realizer (the validity of which is anyway guaranteed by the soundness theorem) is nevertheless illuminating as it reveals the role that the condition $g \in \Delta_{0 \to 0}$ plays. Let us consider the most complex case where $tg\pi = \langle 0, \pi' :: \neg P(x) \rangle$. We now evaluate $g0\pi''$ where $\pi'' = \pi' :: \neg P(x)$, so let's write $\langle y, \sigma \rangle := g0\pi''$. We need to verify that

$$T(\sigma) \to (P(y) \to P(0)). \tag{9}$$

By monotonicity of g we have that $\pi'' \sqsubseteq \sigma$ and so $T(\sigma) \to \neg P(x)$. Now, we know that there is some $f^{0 \to 0}$ such that $f \approx_{0 \to 0} g$, and so in particular we have

$$T(\pi') \to x = f0 \quad \text{and} \quad T(\sigma) \to y = f0.$$

Using monotonicity again we get $T(\sigma) \to x = y$, and hence putting these together $T(\sigma) \to \neg P(y)$. But then (9) holds. It is instructive to observe that even for such a simple example as the Drinker's paradox, matters regarding state are by no means trivial.

6.2 The relationship to Herbrand's theorem

With our state defined as in Section 6.1, realizers $t : S \to 0 \times S$ of simple existential statements $\exists x A(x)$ become terms which take as input a sequence π of *conditions* which are forced. Furthermore, the terms are able to circumvent querying the truth of subsequent atomic predicates by simply adding them as to the state as additional conditions. We illustrate this in the table below.

Interpretation	Realizer of $\exists x A(x)$	Decidability?
Gödel's variant	$A(t)$	yes
Herbrand variant	$\exists x \in \{t_1, \dots, t_n\} A(x)$	no
State variant	$\underbrace{Q_1 \wedge \dots \wedge Q_n}_{:= t\pi_1} \to A(t\pi_0)$	no

In this way, we can replace the characteristic function χ_P by some *arbitrary* term d_P, which rather than finding an exact witness, chooses (1) a branch in a Herbrand tree, as represented by end state $t\pi_1$, and (2) and a witness which corresponds to this branch, as represented by $t\pi_0$.

Intuitively speaking, this would also give us a way of computing a full Herbrand disjunction. Running our program on an initial empty state would take us along a default branch in the Herbrand tree given by the d_Ps, and would return as the output state the finite set of assumptions made during the evaluation. We could then systematically alter these assumptions, either by changing the input state or the approximations d_P, repeatedly rerunning the program in order to traverse the whole Herbrand tree. This process would be analogous to Lemma 10 of [6], although here we would not analyse terms in normal form, since this is dealt with implicitly by the monad.

We also conjecture that in the context of Herbrand's theorem, there is a connection between the monotonicity properly of our state and the cumulative herbrandized functional interpretation of [5], and it would be interesting to carry out some further case studies to understand this connection more clearly.

However, everything mentioned above needs to be made precise:

Problem 1. Adapt the functional interpretation with state to pure predicate logic, clarify its relationship with [5, 6], and give a new proof of Herbrand's theorem.

6.3 Learning semantics and the functional interpretation

A great deal of work has been done on giving a computational semantics to classical reasoning based on the idea of *learning*. This includes the game semantics of Coquand [4], the limit computable mathematics of Hayashi [7], and the Interactive Realizability of Aschieri and Berardi (e.g [2]). We conjecture that our functional interpretation, with a suitable instantiation of state, constitutes a computational interpretation of a similar kind.

In the learning realizability of [2] one considers proofs which involve the law of excluded middle for Σ^0_1-formulas i.e. instances of the following axiom

$$\forall n \, (\forall y \neg P(n,y) \vee \exists x P(n,x)) \tag{10}$$

where the P is some primitive recursive predicate. Associated to this axiom are a pair of noncomputable choice functionals W and Φ which satisfy

$$\Phi n, Wn = \begin{cases} \langle 0, x \rangle & \text{for some } x \text{ such that } P(n,x) \text{ holds} \\ \langle 1, 0 \rangle & \text{if no such } x \text{ exists.} \end{cases}$$

The idea behind learning realizability is that whenever (10) is used as a lemma in the proof of some existential statement, only a finite *approximation* to the non-computable functionals W and Φ is necessary to compute a witness to this statement.

This motivates their form of realizability: To replace these functional by approximations $w[\pi]$ and $\phi[\pi]$ which are based on some finite amount of knowledge about the $P(n,x)$ encoded in some state π. Either these approximations are sufficiently good to yield a valid

witness to the existential statement, or they fail, in which case we learn some piece of constructive information from their failure, and update the state π to a new one π' which represents a better approximation. This process iterates until a realizer has been found.

By making explicit the way in which the functional interpretation interacts with the mathematical environment via characteristic functions χ_P, our state allows us to give the functional interpretation a similarly elegant semantic reading in terms of learning.

To see this, let us define a monotone variant of the state from Section 6.1 as follows: Elements of S will consist of finite *partial functions* π of the form $\pi(n) = \langle b, x\rangle$ or $\pi(n)$ undefined (these can be encoded in E-HA$^\omega_*$ as finite sequences). This time, we define $T(\emptyset)$ as being valid (where $\emptyset$ is the empty partial function) and

$$T([n_1 \mapsto \langle b_1, x_1\rangle, \ldots, n_k \mapsto \langle b_k, x_k\rangle]) :\equiv \bigwedge_{i=1,\ldots,k} \theta(n_i, b_i, x_i)$$

where

$$\theta(n,0,x) :\equiv P_n(x) \wedge \forall y < x \neg P_n(y) \quad \theta(n,1,x) :\equiv \forall y < x \neg P_n(y).$$

We then define $\pi \sqsubseteq \sigma$ to hold when σ has at least as much information as π i.e. if $\pi(n) = \langle 1, x\rangle$ then $\sigma(n) = \langle b, x'\rangle$ with $x \leq x'$, or if $\pi(n) = \langle 0, x\rangle$ then $\sigma(n) = \langle 0, x\rangle$. It is not difficult to see that the condition (B) is satisfied in this case.

To each state, we can associate an approximation to the functionals W and Φ as follows:

$$w[\pi], \phi[\pi] := \begin{cases} \langle 0, x\rangle & \text{if } \pi(n) = \langle 0, x\rangle \\ \langle 1, 0\rangle & \text{otherwise.} \end{cases}$$

Note that out states π contain not only potential evidence about the validity of $\exists x P(n,x)$, but even information about $P(n,y)$ in cases where we have not found a realizer: Namely that $\neg P(n,y)$ holds for all y up to a certain point.

In order to determine our monadic translation, we need to explain how we interpret the characteristic functions χ_P. In contrast to Section 6.1, we continue to use the characteristic function to interact with our environment (rather than replacing it with some arbitrary function d_P), but now we add what it learns to the state.

We make a slight modification and assume that the predicate parameters of HA$^+$ are *monotone* - more precisely are defined to consist solely of the primitive recursive predicate $P'(n,x) :\equiv \exists y < x P(n,y)$, whose associated characteristic function is denoted χ'_P. This restriction is essential given how we have defined states, since these only include information about the *least* witness when it exists.[3] We now define $\varphi_P : 0^2 \to S \to 0 \times S$ by

$$\varphi_P(n,x)\pi := \begin{cases} \langle 0, \pi\rangle & \text{if } \pi(n) = \langle 0, x'\rangle \wedge x > x' \\ \langle 1, \pi\rangle & \text{if } \pi(n) = \langle b, x'\rangle \wedge x \leq x' \\ \langle 0, \pi[n \mapsto \langle 0, y\rangle]\rangle & \text{for least } y < x \text{ with } P(n,y) \\ \langle 1, \pi[n \mapsto \langle 1, x\rangle]\rangle & \text{if } \forall y < x \neg P(n,y) \end{cases}$$

It can be shown, by checking all cases, that $\varphi_P(n,x)\pi_1 \sqsupseteq \pi$, and moreover that if $T(\pi)$ holds then $\varphi_P(n,x)\pi_0 = \chi'_P(n,x)$. Moreover, in contrast to the previous sections we only update the state with information which is *valid*, so if we start with the empty state $\emptyset$ our end state will always satisfy $T(\pi)$.

Now, suppose that we have a proof of $\forall x \exists y Q(x,y)$ which involves the law of excluded middle for the formulas $\exists x P'(n,x)$ (i.e. is formalized using the characteristic functions χ'_P). The usual extracted program $t : 0 \to 0$ would simply return a witness tx for $\exists y$. In contrast, our extracted program with state $t : 0 \to S \to 0 \times S$ gives us insight into how the characteristic function $\chi_{P'}$ is used to test hypothesis about the mathematical environment, and moreover ensures that these tests are carried out by interacting with a global state which encodes our current knowledge. As a result, realizers have the following informal interpretation:

- t takes an argument x and a state π representing an approximation to W, Φ;
- t returns a realizer y together with a final state $\sigma \sqsupseteq \pi$ representing a better approximation to W, Φ, containing what $tx\pi$ has learned as it evaluates.

This gives us a characterisation of how realizers for Π^0_2 statements extracted by the functional interpretation carry out 'learning'. In our setting the learning is carried out internally as the program evaluates, whereas in [2], the realizer is evaluated relative to a fixed state, and only afterwards is this updated. We stress that we have not given a precise connection between the functional interpretation and learning realizability - and in fact to do so would involve extending our interpretation so that it can deal with comprehension functions - but our extension of the functional interpretation with a state is a first step in this direction.

Problem 2: Establish more clearly the relationship between existing variants of realizability based on learning on the one hand, and the functional interpretation on the other.

Another direction for future work would be to carry out some concrete case studies in mathematics, an area in which the functional interpretation excels. It would be particularly interesting to extend our monadic translation to encompass variants of bar recursion, which would allow us to give a learning-based computational interpretation Ramsey's theorem or Higman's lemma [10, 13]. In the former case, our state would interact with some colouring $c : \mathbb{N}^{(2)} \to \{0,1\}$, learning finite pieces of information about it and terminating once it has found an approximation to an infinite monochromatic subset $X \subseteq \mathbb{N}$, while in the latter it would interact with some infinite sequence of words $w_1, w_2, \ldots$ in some well quasi-order, until it finds a pair of indices $i < j$ such that w_i is embedded in w_j. In both cases, understanding how our program interacts with the environment via the state would give a much deeper insight into the computational meaning of these theorems.

Problem 3: Extend our interpretation to full mathematical analysis, and carry out concrete case studies to better understand the computational role of non-constructive methods such as choice.

6.4 Using the state to improve program efficiency

So far we have focused on applications which are oriented towards foundational or semantic issues. However, the final application we propose has a practical emphasis, namely to use the state to improve the behaviour of extracted programs.

In both of the previous sections, our state-sensitive realizer first checks whether or not we can already infer the truth or falsity of our formula in question by looking at the state, and only in the latter case does it update the state with new information. Therefore at the most basic level our state sensitive realizers can be seen as

[3] A similar restriction is present in learning realizability, where states can only update in a consistent way and include a single witness for $\exists x P(n,x)$.

improvements of the original terms, in the sense that they do not need to repeatedly evaluate $P(n,x)$ when it has already been added to the state. However, we can also interact with our state in a more intelligent way, as the following simple example illustrates:

Example 6.3. Suppose that $f : 0 \to 0$ represents some primitive recursive function. Then $\mathsf{PA} \vdash \exists x \forall y (f(x) \leq f(y))$. To interpret this, the functional interpretation demands that for any $g : 0 \to 0$ we produce some x satisfying $f(x) \leq f(g(x))$. Usually this would be computed by setting $tg := sg0$ where:

$$sgx := \begin{cases} x & \text{if } f(x) \leq f(g(x)) \\ sg(g(x)) & \text{otherwise.} \end{cases} \tag{11}$$

Now, suppose that we introduce a state sensitive label $P(x,y) :\equiv f(x) \leq f(y)$, and define our mathematical environment to consist simply of P. Similarly to Section 6.1 we define elements of S to consist of finite sequences of the form $\pi := [\langle x_1, y_1\rangle, \dots, \langle x_k, y_k\rangle]$ and then

$$T([\langle x_1, y_1\rangle, \dots, \langle x_k, y_k\rangle]) \leftrightarrow \bigwedge (f(x_i) \leq f(y_i)).$$

If at any point during our computation we want to check $f(x) \leq f(y)$ relative to our current state π, even if $\langle x,y\rangle \notin \pi$ we could still infer it if $\langle x,z\rangle, \langle z,y\rangle \in \pi$ for some z. Therefore let's set e.g.

$$\pi \rhd \langle x,y\rangle :\equiv (x = y \vee \langle x,y\rangle \in \pi \vee \exists z(\langle x,z\rangle, \langle z,y\rangle \in \pi))$$

and define $\varphi_P(x,y)\pi$ by

$$\varphi_P(x,y)\pi := \begin{cases} \langle 0, \pi\rangle & \text{if } \pi \rhd \langle x,y\rangle \\ \langle 1, \pi\rangle & \text{if } \pi \rhd \langle y,x\rangle \\ \langle 0, \pi :: \langle x,y\rangle\rangle & \text{if } f(x) \leq f(y) \\ \langle 1, \pi :: \langle y,x\rangle\rangle & \text{if } f(y) < f(x) \end{cases}$$

Then it is easy to show that $\varphi_P \approx_{0^2 \to 0} \chi_P$, but in cases where evaluating f is particularly costly, we improve our efficiency by checking whether or not we can deduce $f(x) \leq f(y)$ without evaluating $f(x), f(y)$ again. In particular, a state sensitive variant of the program (11) is given by

$$sgx\pi := \begin{cases} \langle x, \pi\rangle & \text{if } \pi \rhd \langle x, gx\rangle \\ sg(gx)\pi & \text{if } \pi \rhd \langle gx, x\rangle \\ \langle x, \pi :: \langle x, gx\rangle\rangle & \text{if } f(x) \leq f(gx) \\ sg(gx)(\pi :: \langle gx, x\rangle) & \text{if } f(gx) < f(x) \end{cases} \tag{12}$$

and it would not be difficult to come up with concrete examples where (an implementation of) the program (12) carries out strictly fewer comparisons than (11). However, it is not the program itself that is interesting, rather that such a program could be extracted automatically via our monadic translation - assuming of course that the underlying proof is formalised using P in the correct way.

A number of refinements of the functional interpretation have been developed (e.g. [8, 14]), most of which deal in one way or another with the 'contraction problem', namely that crudely extracted programs can be hugely inefficient with respect to how many unnecessary case distinctions they make. Our interpretation achieves a similar goal. However, we don't see it as competing with the aforementioned works, but as something complementary which could be combined with these ideas and in particular *formalised*.

Problem 4. Implement our functional interpretation in a proof assistant and use it as a tool for synthesising efficient programs from proofs.

Finally, we propose that our interpretation could also be used to understand how extracted programs behave imperatively, and to *characterise* the algorithm they implement, along the lines of [3]. There, a version of the well-know quicksort algorithm is formally extracted from a proof into a functional language, and is then transformed using the state monad to a imperative version which can be seen as the 'real' quicksort. It would be interesting to see whether we can automatically extract any well-known algorithms, by implementing our state in a certain way. A more ambitious goal would be to extend HA^+ with some axioms which reason *directly* about the state, which allow us to extract finely tuned imperative programs via the functional interpretation.

Problem 5. Extend our functional interpretation to a fully fledged imperative interpretation, specifically designed for extracting state sensitive programs from proofs.

Acknowledgements. I am grateful to the anonymous referees for their detailed reviews, which led to a much improved version of the paper.

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] U. Berger, M. Seisenberger, and G. Woods. Extracting imperative programs from proofs: In-place quicksort. In *Proceedings of TYPES 2013*, volume 26 of *LIPIcs*, pages 84–106, 2014.
[4] T. Coquand. A semantics of evidence for classical arithmetic. *Journal of Symbolic Logic*, 60:325–337, 1995.
[5] F. Ferreira and G. Ferreira. A hebrandized functional intepretation of classical first-order logic. *Archive for Mathematical Logic*, 56(5–6):523–539, 2017.
[6] P. Gerhardy and U. Kohlenbach. Extracting Herbrand disjunctions by functional interpretation. *Archive for Mathematical Logic*, 44:633–644, 2005.
[7] S. Hayashi. Mathematics based on incremental learning - excluded middle and inductive inference. *Theoretical Computer Science*, 350:125–139, 2006.
[8] M.-D. Hernest. Light functional interpretation. An optimization of Gödel's technique towards the extraction of (more) efficient programs from (classical) proofs. In *Computer Science Logic (CSL'05)*, volume 3634 of *LNCS*, pages 477–492, 2005.
[9] U. Kohlenbach. *Applied Proof Theory: Proof Interpretations and their Use in Mathematics*. Monographs in Mathematics. Springer, 2008.
[10] A. Kreuzer. *Proof mining and Combinatorics : Program Extraction for Ramsey's Theorem for Pairs*. PhD thesis, TU Darmstadt, 2012.
[11] J.-L. Krivine. Realizability in classical logic. In *Interactive Models of Computation and Program Behaviour*, volume 27 of *Panoramas et Synthèses*, pages 197–229. Société Mathématique de France, 2009.
[12] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.
[13] T. Powell. Well quasi-orders and the functional interpretation. To appear in: Schuster, P., Seisenberger, M. and Weiermann, A. editors, *Well Quasi-Orders in Computation, Logic, Language and Reasoning*, Trends in Logic, Springer.
[14] T. Trifonov. *Analysis of Methods for Extraction of Programs from Non-Constructive Proofs*. PhD thesis, Ludwig-Maximilians-Universität Munich, 2011.
[15] B. van den Berg, B. Briseid, and P. Safarik. A functional interpretation for nonstandard arithmetic. *Annals of Pure and Applied Logic*, 163(12):1962–1994, 2012.

A Appendix

Proof of Lemma 4.3. For the arithmetical constants, products and pairing this is straightforward, using reflexivity of $\sqsubseteq$.

For lambda abstraction, suppose that we have proven

$$\mathrm{mon}([\Gamma]) \wedge \mathrm{mon}_\rho(y) \to \mathrm{mon}^M_\tau([t]).$$

Renaming the variable y and applying the β-rule we obtain

$$\mathrm{mon}([\Gamma]) \to \mathrm{mon}_\rho(y') \to \mathrm{mon}^M_\tau((\lambda y.[t])(y'))$$

then quantifying over y' and applying reflexivity of $\sqsubseteq$ yields

$$\mathrm{mon}([\Gamma]) \to \mathrm{mon}^M_{\rho\to\tau}(\lambda^* y.[t]),$$

which by definition is just

$$\mathrm{mon}([\Gamma]) \to \mathrm{mon}^M_{\rho\to\tau}([\lambda x.t]).$$

For application, the induction hypothesis yields

$$\mathrm{mon}([\Gamma \cup \Delta]) \to \mathrm{mon}^M_\rho([s]) \wedge \mathrm{mon}^M_{\rho\to\tau}([t])$$

and so an application of Lemma 4.2 gives

$$\mathrm{mon}([\Gamma \cup \Delta]) \to \mathrm{mon}^M_{\rho\to\tau}([t] * [s])$$

which is just

$$\mathrm{mon}([\Gamma \cup \Delta]) \to \mathrm{mon}^M_{\rho\to\tau}([ts])$$

For the recursor, $\mathrm{mon}^M_{\rho 0(\rho\rho 0)\rho}([R_\rho])$ follows directly from

$$\mathrm{mon}_\rho(b) \wedge \mathrm{mon}_{0\to\rho\to\rho\to\rho}(g) \to \forall x^0 \mathrm{mon}^M_\rho(R^*_\rho bgx).$$

Assuming the premise, we prove the conclusion by induction on x. For $x = 0$ we have

$$\mathrm{mon}^M_\rho(R^*_\rho bg0) \leftrightarrow \mathrm{mon}^M_\rho((\eta b))$$

and the latter follows trivially from $\mathrm{mon}_\rho(b)$. Similarly

$$\mathrm{mon}^M_\rho(R^*_\rho bg(\mathrm{sc}(x))) \leftrightarrow \mathrm{mon}^M_\rho(gx * R^*_\rho bgx),$$

and the latter follows using Lemma 4.2, $\mathrm{mon}^M_{\rho\to\rho}(gx)$ and the induction hypothesis $\mathrm{mon}^M_\rho(R^*_\rho bgx)$.

Finally, $\mathrm{mon}^M_{0^{r_n}\to 0}([\chi_P])$ is a direct consequence of the assumed property $\forall x, \pi(\pi \sqsubseteq \varphi_P x \pi_1)$. □

Proof of Theorem 4.5. Note that at each step it suffices to prove the weaker statement

$$\Gamma \sim [\Gamma] \to t \approx^M_\rho [t].$$

and then apply Lemma 4.3, since $\Gamma \sim [\Gamma]$ implies $\mathrm{mon}([\Gamma])$ in particular.

For the arithmetic constants, products and pairing this is again straightforward. For example, the verification of pairing reduces to showing that

$$x_1 \sim_\rho y_1 \wedge x_2 \sim_\tau y_2 \to \langle x_1, x_2\rangle \approx_{\rho\times\tau} \langle y_1, y_2\rangle$$

which follows directly from the definitions.

For lambda abstraction, we suppose that

$$\Gamma \sim [\Gamma] \wedge x \sim_\rho y \to t \approx^M_\tau [t]$$

Renaming the variables x and y and applying the β rule we obtain

$$\Gamma \sim [\Gamma] \to (x' \sim_\rho y' \to (\lambda x.t)(x') \approx^M_\tau (\lambda y.[t])(y'))$$

Quantifying over x' and y' yields

$$\Gamma \sim [\Gamma] \to \lambda x.t \approx_{\rho\to\tau} \lambda y.[t]$$

from which it follows easily that

$$\Gamma \sim [\Gamma] \to \lambda x.t \approx^M_{\rho\to\tau} \lambda^* y.[t]$$

and the conclusion is just $\lambda x.t \approx^M_{\rho\to\tau} [\lambda x.t]$.

For application, from the induction hypothesis we have that

$$\Gamma \cup \Delta \sim [\Gamma \cup \Delta] \to t \sim^M_{\rho\to\tau} [t] \wedge s \sim^M_\rho [s]$$

and so by Lemma 4.4 we have

$$\Gamma \cup \Delta \sim [\Gamma \cup \Delta] \to ts \sim^M_\tau [t] * [s]$$

and the conclusion is just $ts \sim^T_\tau M[ts]$.

For the recursor, $R_\rho \sim^M_{\rho 0(\rho\rho 0)\rho} [R_\rho]$ follows directly from

$$a \sim_\rho b \wedge f \sim_{0\to\rho\to\rho\to\rho} g \to x =_0 y \to R_\rho afx \approx_\rho R^*_\rho bgy,$$

which in turn follows from

$$a \sim_\rho b \wedge f \sim_{0\to\rho\to\rho\to\rho} g \to \forall x(R_\rho afx \approx_\rho R^*_\rho bgx),$$

which we prove by induction. Suppose that the premise of the above holds. For $x = 0$ we have

$$R_\rho af0 \approx_\rho R^*_\rho bg0 \leftrightarrow a \approx_\rho b$$

and the right hand side follow directly from $a \sim_\rho b$. Similarly,

$$R_\rho af(\mathrm{sc}(x)) \approx_\rho R^*_\rho bg(\mathrm{sc}(x)) \leftrightarrow fx(R_\rho afx) \approx_\rho gx * R^*_\rho bgx$$

Now since $f \sim_{0\to\rho\to\rho} g$ implies $fx \approx^M_{\rho\to\rho} gx$ and hence also $fx \sim^M_{\rho\to\rho} gx$ (since we know that gx is also monotone), and moreover $R_\rho afx \approx^M_\rho R^*_\rho bgy$ and hence also $R_\rho afx \sim^M_\rho R^*_\rho bgy$ (since we know that $R^*_\rho bgy$ is also monotone), then the right hand side follows from Lemma 4.4. This completes the induction.

Finally $\chi_P \approx^M_{0^{r_P}\to 0} [\chi_P]$ is a direct consequence of the assumed property that $\chi_P \approx_{0^{r_P}\to 0} \varphi_P$. □