A Calculus of Realizers for $\text{EM}_1$ Arithmetic

Stefano Berardi, Ugo de'Liguoro

Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, 10149 Torino, Italy
stefano@di.unito.it deliguoro@di.unito.it

Abstract. We propose a realizability interpretation of a system for quantifier free arithmetic which is equivalent to the fragment of classical arithmetic without nested quantifiers, which we call $\text{EM}_1$-arithmetic. We interpret classical proofs as interactive learning strategies, namely as processes going through several stages of knowledge and learning by interacting with the “environment” and with each other. With respect to known constructive interpretations of classical arithmetic, the present one differs under many respects: for instance, the interpretation is compositional in a strict sense; in particular the interpretation of (the analogous of) the cut rule is the plain composition of functionals. As an additional remark, any two quantifier-free formulas provably equivalent in classical arithmetic have the same realizer.

1 Introduction

We propose a new notion of realizability (see e.g. [15] vol. I p. 195 for an introduction), in which the classical principle $\text{EM}_1$ (which is Excluded Middle restricted to $\Sigma^0_1$ formulas, see [1]) is treated by means of realizers that depend on certain growing pieces of knowledge, obtained by trial and error. In our approach the witness hidden in a proof of a $\Sigma^0_1$ statement (with parameters) can be computed in the limit (see [5, 3, 7]), and in this sense it is “learnt”. The essential difference with Gold’s idea is that the realizer embodies a learning strategy which is the actual content of the proof, and which is often an ingenuous method. Because of the limitation to $\text{EM}_1$, as in [7], the realizers we obtain are actually effective. With respect to known constructive interpretations of non-constructive arithmetic, the advantage of the present proposal lies in the fact that functionals realizing the classical proofs of our system closely mirror the structure of the original derivations, while the effect of learning is limited to a few crucial non-constructive steps, in which the searching and testing mechanism becomes essential.

The first step of the construction is the introduction of an oracle $\chi_P$ (a predicate symbol) and of a Skolem function $\varphi_P$ relative to each primitive recursive predicate $P$, in such a way that $\exists y. P(x, y) \iff \chi_P(x) \iff P(x, \varphi_P(x))$. The existence of these oracles intuitionistically implies $\text{EM}_1$, and also that atomic formulas of our system represent 1-quantifier formulas of arithmetic. Then we introduce a set $S$ of the states of knowledge, which are finite sequences recording finitely many values of $\chi_P$ and $\varphi_P$, and ordered by prefix (by “increasing knowledge”). A formula containing such $\chi_P$ and $\varphi_P$ is not decidable in general, however it can be evaluated w.r.t. a finite state $s$ of knowledge, by assigning dummy values to all values of $\chi_P$ and $\varphi_P$ which are unknown. A formula is valid if for any state $s$ of knowledge we can effectively find some $s' \geq s$ in which the formula is true. The soundness property we expect from the semantic interpretation states that all derivable formulas are valid.

More precisely, since the meaning of a predicate or a term might depend on a state of knowledge $s \in S$, numbers and truth values are lifted to numbers and truth values indexed over $S$. We see ordinary numbers and truth values as limits, and we ask that an indexed object is convergent in every sequence $s_0 \leq s_1 \leq s_2 \leq \ldots \leq s_n \leq \ldots$ weakly increasing w.r.t. prefix. Using the state we provide an effective semantics for all terms of a simply typed $\lambda$-calculus closed under primitive recursion, and extended by the non-effective maps $\chi_P$ and $\varphi_P$. This $\lambda$-calculus represents the terms and the atomic formulas of our fragment of arithmetic.
A tentative definition for a realizer of an atomic formula is that of a mapping sending any state $s$ into some extension $s' \geq s$ in which the predicate takes the meaning “true” for all $s'' \geq s'$. The key problem here is that the realizer has to be effective, while there are no uniform and effective means to decide when the value of a term or the truth value of a formula has become stable.

The next idea is that the correct state can be learned. The outcome of the realizer is not a state, rather a process such that, as soon as the realizer becomes aware that something has gone wrong, so that the predicate is not true any more, it is able to extend the present state of knowledge looking for a larger one, where the predicate becomes true anew. Since we classically know that the truth value of the predicate eventually stabilizes, it will be eventually true forever, even if we shall never be able to say when and where.

There is still something missing here: we want a compositional interpretation of proofs, validating logical laws like the modus ponens and the cut rule. This rises the issue of the interaction of different parts of a proof, namely of how to compose two or more realizers. To solve this problem realizers use continuations: the process extending the states of knowledge in order to make some predicate true is passed along in the composition of realizers, and used by them as the last step in the computation of the new state. In this way, at the price of having realizers of type two, the combination of the realizers of several premises of an inference rule is (pointwise) composition, which is unsensible to the order: this does not mean, of course, that say $F \circ G$ and $G \circ F$ compute the same value, rather that any choice will be a realizer of the conclusion.

The paper is organized as follows. In §2 we introduce a version of EM$_1$-Arithmetic without nested quantifiers. Terms are expressed in a simply typed λ-calculus with primitive recursion of level 1. All formulas are quantifier-free, while formulas with non-nested quantifiers are represented through oracles $\chi_P(x)$. EM$_1$ (Excluded Middle for $\Sigma^0_1$-formulas) is the following axiom schema:

\[(\text{EM}_1) \quad \forall x(\exists y P(x, y) \lor \forall y \neg P(x, y))\]

where $P(x, y)$ is a primitive recursive predicate. It is represented in our formalism without quantifiers through the oracle constants $\chi_P$. In §3 we introduce a motivating example, the Minimum Principle, a typical non-constructive arithmetical proof which can be formalized using just the classical principle EM$_1$. For a discussion of the relation between EM$_1$ and full Classical Arithmetic we refer to [1]. In section §4 we lift the standard interpretation of numbers, boolean and functions to indexed numbers, booleans and functions introducing the notion of synchronous and convergent functional. §5 is the core of the paper: we introduce a new notion of realizers for classical proofs, representing the construction hidden in the classical principle EM$_1$. All other constructions are represented by terms of the system. In §6 we test our notion of realizer against the example of the Minimum Principle.

**Related Works.** A primary source of the present research is Coquand’s semantics of evidence for classical arithmetic [2], where the role of realizers is taken by the strategies, the state of knowledge is the state of a play, and computation is the interaction of strategies through a dialogue.

The idea of lifting to truth values and numbers depending from a state and converging (in the sense of stabilization) w.r.t. this state comes from Gold’s recursiveness in the limit [5] and Hayashi’s Limit Computable Mathematics [7]. Our main contribution is to frame these ideas in the longstanding tradition of realizability interpretation of constructive logic.

The investigation of the computational content of classical proofs via continuations is well known and widely documented in the literature. It is impossible to provide a reasonably complete list of the numerous contributions to this topic; see e.g. [4, 9] for some basic ideas behind the use of continuations in the interpretation of classical principles; continuations and CPS translation have been used to extend the formula as types paradigm to classical logic in [6, 12]; these ideas are found also in the μ-calculus of [13] and in related systems. Our improvement is the compositional property of our approach in the strict sense of functional composition of the realizers, which allows for a clean reading of the use of continuations as mappings that “force” the stabilization of predicates and terms.
2 EM₁ Arithmetic of Primitive Recursive Functions

Let Type be the set of simple types with atoms Nat and Bool and the arrow as type constructor. As usual external parentheses will be omitted and the arrow associates to the right: $T₀ ⇒ T₁ ⇒ T₂$ reads as $T₀ ⇒ (T₁ ⇒ T₂)$; we also write $T^k ⇒ T'$ for $T ⇒ ⋯ ⇒ T ⇒ T'$ with $k$ occurrences of $T$ to the left of $T'$. The symbol $≡$ is used for syntactical identity.

Definition 1 (Term Languages $L₀$ and $L₁$). Let $L₀$ be the language of simply typed $λ$-calculus with types in Type, whose constants are:

- zero, successor: $0 : \text{Nat}$, succ : $\text{Nat} ⇒ \text{Nat}$, equality: eq : $\text{Nat}^2 ⇒ \text{Bool}$, booleans: true, false : Bool
- if-then-else: if$_f$ : $\text{Bool} ⇒ T ⇒ T ⇒ T$, where either $T ≡ \text{Nat}$ or $T ≡ \text{Bool}$
- primitive recursion: $\text{PR} : \text{Nat} ⇒ (\text{Nat}^2 ⇒ \text{Nat}) ⇒ \text{Nat} ⇒ \text{Nat}$

The language $L₁$ is obtained by adding to $L₀$ a pair of constants $φ_P : \text{Nat}^k ⇒ \text{Nat}$ and $χ_P : \text{Nat}^k ⇒ \text{Bool}$ for each closed term $P : \text{Nat}^{k+1} ⇒ \text{Bool}$ of $L₀$, and then closing under term formation rules.

Term application associates to the left: $M N P$ reads as $(M N)P$; the abstraction $λxM$ will be written $λx. M$ when $T$ is clear from the context. We abbreviate $n ≡ \text{succ}^n 0$ ($n$-times applications of succ to 0), which is the numeral for $n ∈ \mathbb{N}$. We write $M[x₁, ⋯, xₙ]$ to mean that $FV(M) ⊆ \{x₁, ⋯, xₙ\}$, and $M[N₁, ⋯, Nₙ]$ for $M[N₁/x₁, ⋯, Nₙ/xₙ]$, that is the result of the simultaneous substitution of $x_i$ by $N_i$ for all $i$ (which are supposed to be of the same type), avoiding variable clashes.

Definition 2 (Equational Theory for $L₀$). The theory $T₀$ is the equational theory of terms in $L₀$ whose formulas are typed equations $M = N : T$ with both $M$ and $N$ of type $T$. Axioms and inference rules of $T₀$ are the axioms of equality, $β$ and $η$ from the $λ$-calculus, plus:

- $\text{eq } 0 0 = \text{true}$, $\text{eq } (\text{succ } x) 0 = \text{false}$, $\text{eq } (\text{succ } x) (\text{succ } y) = \text{eq } x y : \text{Bool}$,
- $\text{if }_T \text{true } M N = M : T$, $\text{if }_T \text{false } M N = N : T$
- $\text{PR } M N 0 = M : \text{Nat}$, $\text{PR } M N (\text{succ } x) = N x (\text{PR } M N x) : \text{Nat}$

By $T₀ ⊢ M = N : T$ we mean that the equation $M = N : T$ is derivable in $T₀$.

We explicitly exclude the function symbols $φ_P, χ_P$ from $T₀$: they will denote non-computable maps. The primitive recursor to define $k + 1$-ary functions: $\text{PR}_k : (\text{Nat}^k ⇒ \text{Nat}) ⇒ (\text{Nat}^{k+2} ⇒ \text{Nat}) ⇒ \text{Nat}^{k+1} ⇒ \text{Nat}$ is definable from the unary $\text{PR}$ by: $\text{PR}_k ≡ λg h x₁ ⋯ xₖ \text{PR}(g x₁ ⋯ xₖ)(hx₁ ⋯ xₖ)$.

Although $T₀$ is an equational theory, it is the theory of the convertibility relation associated to a notion of reduction which is confluent and strongly normalizing. In particular it is decidable whether $T₀ ⊢ M = N : T$. Indeed $T₀$ is a fragment of Gödel system $T$, where the essential limitation consists in the restriction of the recursor $\text{PR}$ whose functional arguments are of type one. By this the presence of abstraction of variables at any type has no effect w.r.t. function definability. A $k$-ary function over natural numbers $f$ is definable in $T₀$ if there exists a closed term (a combinator) $f : \text{Nat}^k ⇒ \text{Nat} ∈ L₀$ such that for all $n₁, ⋯, nₖ, m ∈ \mathbb{N}$, $T₀ ⊢ f n₁ ⋯ nₖ = m : \text{Nat}$ if and only if $f(n₁, ⋯, nₖ) = m$.

Proposition 1. The number theoretic functions definable in $T₀$ are exactly the primitive recursive functions (in particular, are computable).
We find useful having in $T_0$ an operator for (weighted) total recursion. Let $ifz$ be the primitive recursive function such that $ifz(0,n) = n$ and $ifz(m+1,n) = 0$. For any $w : \mathbb{N} \rightarrow \mathbb{N}$, let us abbreviate by $x \prec_w y$ the (primitive recursive) function giving 0 if $w(x) < w(y)$, 1 otherwise. We say that the function $f : \mathbb{N} \rightarrow \mathbb{N}$ is defined by *weighted well-founded recursion* in terms of the functions $w : \mathbb{N} \rightarrow \mathbb{N}$ (weight), $g : \mathbb{N}^2 \rightarrow \mathbb{N}$, $h_1 : \mathbb{N} \rightarrow \mathbb{N}$ if:

$$f(n) = g(n, ifz(h_1(n) \prec_w n, f(h_1(n))))$$

This definition schema generalizes to any list of maps $h_1, \ldots, h_k : \mathbb{N} \rightarrow \mathbb{N}$, and to any list $m = m_1, \ldots, m_h$ of parameters. Weighted total recursion can be expressed in $T_0$. The proof of this result, which we omit, relies on the fact that weighted well-founded recursion can be defined as primitive recursion with a bound which depends on the weight function, and on the definability result of Proposition 1. We will now express this result. Let $\text{Nat}^2 \Rightarrow \text{Bool}$ be the (primitive recursive) predicate such that $T_0 \vdash \text{lt} n m = \text{true}$ if $n < m$, $T_0 \vdash \text{lt} n m = \text{false}$ else. Define two terms by $ifz \equiv \lambda x y. \text{if} x y 0 : \text{Bool} \Rightarrow \text{Nat} \Rightarrow \text{Nat}$ and $x \prec_W y \equiv ifz(\text{lt}(W x) (W y))$. We can now state:

**Proposition 2. (Definability of Weighted Total Recursion)** If $f$ is defined by weighted well-founded recursion in terms of primitive recursive $w, g, h_1, \ldots, h_k$, then it is *itself* primitive recursive. Moreover there exists a combinator $\text{WR} : (\text{Nat} \Rightarrow \text{Nat}) \Rightarrow (\text{Nat}^2 \Rightarrow \text{Nat}) \Rightarrow (\text{Nat} \Rightarrow \text{Nat}) \Rightarrow (\text{Nat} \Rightarrow \text{Nat})$ such that, if $F \equiv \text{WR} W G H$ then: $T_0 \vdash F x = G x (ifz(H x \prec_W x)(F(H x)))$.

This result generalizes to any $k$-ple of terms $H_1, \ldots, H_n : \text{Nat} \Rightarrow \text{Nat}$.

We will now define a formal theory $\text{PRA-3}$ of arithmetic in the formalism of $L_1$. We rephrase some basic notions of logic. Terms, ranged over by $t, r, \ldots$ (possibly with primes and indexes) are terms of type $\text{Nat}$ in $L_1$ with free variables of type $\text{Nat}$, and formulas, ranged over by $P, Q, R, \ldots$ (possibly with primes and indexes) are terms in $L_1$ of type $\text{Bool}$, with free variables of type $\text{Nat}$. The atomic formulas of the shape $\text{eq } t r$ and $\text{lt } t r$ are written $t = r$ and $t < r$ respectively. Connectives are definable in $L_1$, and we write them in the usual way: in particular we use the ordinary infix notation for binary connectives.

A *propositional formula* is a term $E[z_1, \ldots, z_n]$ of type $\text{Bool}$ built out of variables $z$ of type $\text{Bool}$ and connectives (hence it is in $L_0$); a propositional formula $E$ is *tautological consequence* of $E_1, \ldots, E_k$ if any instantiation of the boolean variables by true or false in the implication $(E_1 \wedge \cdots \wedge E_k) \implies E$ is provably equal to true in $T_0$.

The theory $\text{PRA-3}$ defined below is formally quantifier free: as a matter of fact the meaning of $\chi_P$ and $\varphi_P$ induced by the axioms ($\chi$) and ($\varphi$) is that of the oracle and of a Skolem function for the predicate $\exists y. P[x, y]$ where $P$ is in $L_0$ (hence primitive recursive). We stress that we cannot have Skolem functions in $P$: this limitation accounts for the fact that $\text{PRA-3}$ is 1-quantifier arithmetic, and not the entire arithmetic.

**Definition 3.** $\text{PRA-3}$ is the theory whose theorems are the formulas derivable by the following axioms and rules:

- **Post rules:**

  $\frac{P_1 \quad \cdots \quad P_k}{Q} \quad \text{Post}$

  consisting of the axioms of equality; an axiom for each equation $t = r : \text{Nat}$ derivable in $T_0$; all rules with $E[z_1, \ldots, z_n]$ tautological consequence of $E_1[z_1, \ldots, z_n], \ldots, E_k[z_1, \ldots, z_n]$:

  $\frac{E_1[z_1, \ldots, z_n], \ldots, E_k[z_1, \ldots, z_n]}{E[z_1, \ldots, z_n]} \quad \text{Taut}$

- **Skolem Axioms:** for each formula $P[x, y]$ in $L_0$, the axioms:

  \[
  \frac{P[x, y] \implies \chi_P x}{\chi_P x} \quad \frac{\chi_P x \implies P[x, \varphi_P x]}{\varphi_P x}
  \]
- Well Founded Induction:

\[
\frac{\text{if} (t_1 x \prec_w x) P[z, t_1 x] \land \cdots \land \text{if} (t_k x \prec_w x) P[z, t_k x] \implies P[z, x]}{P[z, r]} \quad \text{WF Ind}
\]

where \( x \) is not free in the conclusion, and \( w, t_1, \ldots, t_k : \text{Nat} \Rightarrow \text{Nat}, r : \text{Nat} \) and \( t_1 x \prec_w x \equiv w(t_1 x) < w x \).

The theory PRA-3, when restricted to the language \( L_0 \) is a variant of system PRA in [15] for primitive recursive arithmetic. The quantifier free version of induction rule:

\[
\frac{P[z, 0] \quad P[z, x] \implies P[z, \text{succ} x]}{P[z, t]} \quad \text{Ind}
\]

is admissible by using some instance of WF Ind with weight function \( w = \text{id} \), so that \( \prec_w = \prec \) in \( T_0 \):

\[
\frac{\text{if} z(x - 1 < x) P[z, x - 1] \implies P[z, x]}{P[z, t]}
\]

The map \(-1\) is the natural predecessor, so that \( \text{if} z(0 - 1 < 0) \) is false and \( \text{if} z(0 - 1 < 0) P[z, 0 - 1] \) is true and \( \text{if} z(0 - 1 < 0) P[z, 0 - 1] \implies P[z, 0] \) is equivalent to \( P[z, 0] \), the base case of induction rule.

We finish this section by stipulating some notational conventions. We shall often use the uncurried notation of combinators: if \( M : T_1 \Rightarrow \cdots \Rightarrow T_k \Rightarrow T \) we shall write \( M(N_1, \ldots, N_k) : T_{k+1} \Rightarrow \cdots \Rightarrow T_k \Rightarrow T \) for \( 0 \leq h \leq k \), in place of \( MN_1 \cdots N_k \), and even \( MN_1 \cdots N_k(N_{h+1}, \ldots, N_k) \) for \( MN_1 \cdots N_k \). The more familiar if \( M \) then \( N \) else \( R \) will be used instead of if \( M N R \). When both italics notation \( P, \ldots \) occur in the same context, the latter is the formal counterpart of the (informal) object \( P \). However, when no confusion arises, and this improves readability, we shall prefer the informal notation.

3 The minimum principle

The minimum principle MP for recursive functions states that for any total \( f : \mathbb{N} \rightarrow \mathbb{N} \) there exists \( n \in \mathbb{N} \) such that for all \( m \in \mathbb{N} \), \( f(n) \leq f(m) \), that is: \( \exists x \forall y. f(x) \leq f(y) \). We call \( n \) a minimum point of \( f \). MP is not provable in HA, since it is equivalent in HA to EM1. A classical proof is as follows: first reformulate the principle claiming that given any \( n \), we can find a minimum point of \( f \). Then the proof proceeds by total induction over \( n \) with weight \( f \), by using the instance of the Excluded Middle:

\[
\forall y. f(n) \leq f(y) \lor \exists y. f(n) > f(y).
\]

In case \( \forall y. f(n) \leq f(y) \) holds, \( n \) itself is a minimum point. Otherwise \( \exists y. f(n) > f(y) \) is true: choose \( n' \in \mathbb{N} \) such that \( f(n) > f(n') \). Then \( n' \) has weight less than \( n \), and the induction hypothesis applies to \( n' \), so that we can find a minimum point.

PRA-3 proves MP for primitive recursive functions, which are representable by terms in \( L_0 \). The proof in PRA-3 relies on a well founded induction w.r.t. the ordering: \( n \prec f m \equiv f(n) < f(m) \). Let \( P(x, y) \equiv f(x) > f(y) \); under the hypothesis that \( f \) is primitive recursive \( P \) is definable by a predicate \( P : \text{Nat}^2 \rightarrow \text{Bool} \in L_0 \). Therefore, writing simply \( \chi, \varphi \) for \( \chi_P \) and \( \varphi_P \) respectively, we have the axioms:

\[
f(n) > f(x) \implies \chi(x), \quad \chi(x) \implies f(n) > f(\varphi(n)).
\]

\( \chi(x) \) is equivalent to \( \exists y. f(x) > f(y) \), therefore we can express “\( n \) is a minimum point of \( f \)” by \( \neg \chi(n) \). We cannot express the existence of a minimum point using primitive recursive predicates and \( \chi, \varphi \), because
it requires two nested quantifiers. However, we can define from \( \varphi, \chi \) and by well-founded recursion w.r.t. \( \prec_f \) a (non-computable) term \( t(x) \), denoting some minimum point of \( f \) for any \( x \). The term \( t(x) \) is defined by cases, according to the cases in which \( x \) is a minimum point of \( f \) or not. This fact cannot be decided recursively in \( f \), but it can be “decided” using the oracle \( \chi \):

\[
t(x) = \text{if } \neg \chi(x) \text{ then } x \text{ else ifz}(\varphi(x) \prec x)t(\varphi(x))
\]

that is \( t = \text{WR}(f, g, \varphi) \) where \( g(x, y) = \text{if } \neg \chi(x) \text{ then } x \text{ else } y \). To express that \( t(0) \) is the minimum we consider the formula \( Q(0) \) where \( Q(x) \equiv \neg \chi(t(x)) \) says that \( t(x) \) is a minimum point of \( f \). Let \( R(x) \equiv \text{ifz}(\varphi(x) \prec_f x)Q(\varphi(x)) \implies Q(x) \) be the assumption of the rule \( \text{WF Ind} \) of conclusion \( Q(0) \). This assumption can be proved as a tautological consequence of \( \neg \chi(x) \implies R(x) \) and \( \chi(x) \implies R(x) \). The formal proof of \( Q(0) \) in \( \text{PRA-}3 \) ends by:

\[
\frac{\neg \chi(x) \implies R(x) \quad \chi(x) \implies R(x)}{R(x)} \quad \text{Taut}
\]

It remains to prove by Post rules the two premises of rule Taut above. To derive \( \neg \chi(x) \implies R(x) \), note that \( \neg \chi(x) \implies t(x) = x \) by definition of \( t \), so that \( \neg \chi(x) \implies \neg \chi(t(x)) \) that is \( \neg \chi(x) \implies Q(x) \), and by Taut it follows that \( \neg \chi(x) \implies R(x) \).

To derive \( \chi(x) \implies R(x) \) we use the \( \varphi \) axiom \( \chi(x) \implies f(n) \geq f(\varphi(n)) \), that is (a) \( \chi(x) \implies \varphi(x) \prec_f x \); hence, by definition of \( t \), \( \chi(x) \implies t(x) = t(\varphi(x)) \). Again using the same axiom we have (b) \( \chi(x) \implies [\text{ifz}(\varphi(x) \prec_f x)Q(\varphi(x)) \implies Q(x)] \); but since by (a) we know that \( \chi(x) \implies Q(x) = Q(\varphi(x)) \), we conclude from (b) that \( \chi(x) \implies [\text{ifz}(\varphi(x) \prec_f x)Q(\varphi(x)) \implies Q(x)] \), namely \( \chi(x) \implies R(x) \).

4 States of Knowledge, Synchronous and Convergent Functionals

In this section we introduce the notion of state of knowledge, then we define hereditary synchronous functionals, whose computation all takes place in the same state of knowledge, then hereditary convergent functionals. These latter will be used to interpret terms of \( \mathcal{L}_1 \).

Let \( \bot \) denote a divergent computation, and define \( A_{\bot} = A \cup \{ \bot \} \) for any set of values \( A \). An element \( a \in A_{\bot} \) is total if \( a \neq \bot \); a map \( \tau : A_{\bot} \to B_{\bot} \) is total if \( \tau \) sends total elements into total ones. If \( f : A^n \to B \), we extend \( f \) to \( f_{\bot} : A^n_{\bot} \to B_{\bot} \) by \( f_{\bot}(a) = \bot \) if \( a_i = \bot \) for some \( i \), and \( f_{\bot}(a) = f(a) \) otherwise.

**Definition 4 (States of Knowledge).** A state of knowledge, shortly a state, is a finite list of triples \( \langle P, n, m \rangle \) (with possibly different \( P, n \) and \( m \)) such that \( P : \mathbb{Nat}^{k+1} \to \mathbb{Bool} \) is a predicate of \( \mathcal{L}_0 \) and \( n = n_1, \ldots, n_k \in \mathbb{N}, m \in \mathbb{N}, \) and \( T_0 \vdash P[n, m] = \text{true} \). We call the empty list \( \langle \rangle \) the initial state. Let \( \mathbb{S} \) denote the set of states, partially ordered by the prefix ordering \( \leq \).

A triple \( \langle P, n, m \rangle \) stays for the equations \( \chi_P(n) = \text{true} \) and \( \varphi_P(n) = m \). A state represents a finite set of such equations, the initial state is the empty set. We now define synchronous maps \( F \) as mapping of indexed objects such that both the argument and the value of \( F \) are evaluated at the same state \( s \). Let us write \( \lambda_{\bot}a \) for the function constantly equal to \( a \).

**Definition 5 (Synchronous Functions).** Let \( A \) and \( B \) be any sets:

1. \( F : (\mathbb{S}_{\bot} \to A) \to (\mathbb{S}_{\bot} \to B) \) is synchronous if \( F(\tau, s) = F(\lambda_{\bot}\tau(s), s) \) for all \( \tau \) and \( s \);
2. given \( f : A \to B \) the synchronous extension \( f^\uparrow : (\mathbb{S}_{\bot} \to A) \to (\mathbb{S}_{\bot} \to B) \) of \( f \) is defined by \( f^\uparrow(\tau, s) = f(\tau(s)) \).
For any \( \tau \) and \( s \) we have \( f^i(\tau, s) = f(\tau(s)) = f^i(\lambda_\tau \tau(s), s) \), hence \( f^i \) is synchronous. We will now interpret terms and formulas of \( L_0 \) as synchronous and convergent functions. To this aim, we have to extend the notion of synchronicity to higher types.

We recall that an embedding-projection pair \((\epsilon, \pi) : X < Y \) (e-p pair for short) of \( X \) into \( Y \) is a pair of mappings \( \epsilon : X \rightarrow Y \) and \( \pi : Y \rightarrow X \) s.t. \( \pi \circ \epsilon = \text{id}_X \), where composition will be also written as \( \pi \). The composition, \( \epsilon \pi : Y \rightarrow Y \), called retraction, is idempotent so that \( \epsilon(X) \) (the image of \( X \) under \( \epsilon \)) is the set of fixed points of \( \epsilon \pi \), and \( \pi : \epsilon(X) \rightarrow X \) is bijective.

**Definition 6** (Synchronous Retraction). For any sets \( A \) and \( B \) we define the mappings:

\[
\epsilon_{A, B} : (S_L \rightarrow (A \rightarrow B)) \rightarrow ((S_L \rightarrow A) \rightarrow (S_L \rightarrow B)) \quad \pi_{A, B} : ((S_L \rightarrow A) \rightarrow (S_L \rightarrow B)) \rightarrow (S_L \rightarrow (A \rightarrow B))
\]

by

\[
\epsilon_{A, B}(\alpha, \tau) = \alpha(s, \tau(s)) \quad \text{and} \quad \pi_{A, B}(F, s) = F(\alpha, a, s)
\]

where \( \alpha : S_L \rightarrow (A \rightarrow B), \tau : S_L \rightarrow A, \) \( s \in S_L, F : (S_L \rightarrow A) \rightarrow (S_L \rightarrow B) \) and \( a \in A \).

The name of synchronous retraction is justified by the fact that the fixed points of the retraction are maps \((S_L \rightarrow A) \rightarrow (S_L \rightarrow B) \) bijective to the elements of \( S_L \rightarrow (A \rightarrow B) \), therefore are maps depending on a single state. They coincide with the synchronous maps:

**Lemma 1.** If \( \epsilon_{A, B} \) and \( \pi_{A, B} \) are as in Definition 6 then:

1. \((\epsilon_{A, B}, \pi_{A, B})\) is an e-p pair;
2. \( F : (S_L \rightarrow A) \rightarrow (S_L \rightarrow B) \) is synchronous if and only if \( F \in \epsilon_{A, B}(S_L \rightarrow (A \rightarrow B)) \).
3. If \( f : A \rightarrow B \), the synchronous extension \( f^1 : (S_L \rightarrow A) \rightarrow (S_L \rightarrow B) \) of \( f \) is equal to \( \epsilon_{A, B}(\lambda_\tau f, \pi) \).

The next step is to state some well known properties of embedding-projection-pairs, and can be rephrased by stating that they are closed under composition and indeed they do form a category; moreover since the category of sets is cartesian closed, then the category of embedding-projection pairs over sets is such. Observe that retraction are covariant w.r.t. the arrow, which is also known as their characteristic property.

**Lemma 2.**

1. If \((\epsilon, \pi) : A < B \) and \((\epsilon', \pi') : B < C \) are e-p pairs, then: \((\epsilon', \pi \pi') : A < C \) is such where \((\epsilon', \pi_1) \rightarrow (\epsilon_2, \pi_2) : A_1 \rightarrow A_2 < B_2 \) is such where \((\epsilon_1, \pi_1) \rightarrow (\epsilon_2, \pi_2) \) is the pair \((\lambda f, \epsilon_2 \circ f \circ \pi_1, \lambda g \circ \pi_2 \circ g \circ \epsilon_1) \).

Let \( \text{St} \) be a new ground type; then we define the interpretation \([T] \) of the simple type \( T \) of the extended type language set theoretically by: \([\text{Nat}] = N_i, [\text{Bool}] = B_i, [\text{St}] = S_i, \) and \([T \Rightarrow T'] = [T] \rightarrow [T'] \), namely the full function space. By \( T^{\text{St}} \) we denote the result of replacing in \( T \) each occurrence of \( \text{Nat} \) by \( \text{St} \Rightarrow \text{Nat} \) and of \( \text{Bool} \) by \( \text{St} \Rightarrow \text{Bool} \) respectively.

**Lemma 3.** For each type \( T \) there is an e-p pair \((\epsilon_T, \pi_T) : S_L \rightarrow [T] < [T^{\text{St}}] \).

**Proof.** By induction over \( T \). If \( T \) is ground then \( T^{\text{St}} = (\text{St} \Rightarrow T) \) and \( \epsilon_T = \pi_T = \text{id}_{S_L \rightarrow [T]} \). Otherwise let \( T = T_1 \Rightarrow T_2 \). By induction there exist the e-p pair \((\epsilon_{T_i}, \pi_{T_i}) : S_L \rightarrow [T_i] < [T_i^{\text{St}}] \) for \( i = 1, 2 \). By Lemma 1 \((\epsilon_{[T_i]}[T_1], \pi_{[T_1]}[T_2]) : S_L \rightarrow ([T_1] \rightarrow [T_2]) < (S_L \rightarrow [T_1]) \rightarrow (S_L \rightarrow [T_2]) \) is an e-p pair. Therefore by Lemma 2 we have the e-p pair:

\[
(\epsilon_{T_1 \rightarrow T_2}, \pi_{T_1 \rightarrow T_2}) = ((\epsilon_{T_1}, \pi_{T_1}) \rightarrow (\epsilon_{T_2}, \pi_{T_2})) \circ (\epsilon_{[T_1]}, \pi_{[T_1]})[T_2]
\]

We are now in place to characterize those functionals of \( [T^{\text{St}}] \) depending on a single state, i.e. in the image of \( S_L \rightarrow [T] \), as fixed points of the retraction \((\epsilon_T, \pi_T) : S_L \rightarrow [T] < [T^{\text{St}}] \). The elements \( a \in [T] \) can be raised to elements of \( [T^{\text{St}}] \) by applying \( \epsilon_T \) to \( \lambda \_a \).
Definition 7 (Hereditarily Synchronous Functionals). Let $T$ be any type:

1. $f \in [T^{S_1}]$ is hereditarily synchronous, or h. sync. for short, if $f \in \epsilon_T(S_+ \to [T])$;
2. the canonical injection $\omega^o : [T] \to [T^{S_1}]$ is defined: $f^o = \epsilon_T(\lambda_-. f)$. We also set $[T]^0 = \{f^o \mid f \in [T]\}$.

H. sync. functionals are closed under application. Indeed, if $f : S_+ \to [T_1 \Rightarrow T_2]$ then $\epsilon_{T_1 \Rightarrow T_2}(f) = \epsilon_{T_2}(\omega^o f) \circ \tau_{T_1}$, hence for any $a \in [T^{S_1}]$ we have $\epsilon_{T_1 \Rightarrow T_2}(f)(a) = b$ for some $b \in \epsilon_{T_2}(S_+ \to [T_2]) \subseteq [T_2]$. It follows that, if $g \in [(T_1 \Rightarrow T_2)^{S_1}]$ is hereditarily synchronous, then $g(a)$ is such for any $a \in [T^{S_1}]$.

If $f \in [T]$ then $f^o \in [T^{S_1}]$ is a map ignoring its input state, but forcing its argument to use $s$ as the only state. Indeed, by unraveling definitions we obtain:

$$f^o(\tau_1, \ldots, \tau_n, s) = f(\tau_1(s), \ldots, \tau_n(s)).$$

We can now describe h.sync. functionals by an equation. Let $T = T_1, \ldots, T_n \Rightarrow o$ for either $o = \mathbb{N}$ or $o = \mathbb{B}$, and consider $F \in [T^{S_1}]$ and $\tau_i \in [T_i^{S_1}]$ for $i = 1, \ldots, n$. By unraveling definitions, $F$ is hereditarily synchronous if and only if for all $s \in S_+$

$$F(\tau_1, \ldots, \tau_n, s) = F((\tau_1(s))^o, \ldots, (\tau_n(s))^o, s).$$

The operator $\omega^o$ applied to the arguments of $F$ forces them to reject their input state, and to use only the same input state $s$ of $F$. This implies that the behavior of any functional over $[T^{S_1}]$ is fully determined by its behavior over $[T]^o$.

Proposition 3. Let $F, G \in [(T_1, \ldots, T_n \Rightarrow U)^{S_1}]$, then $F = G$ if and only if $F(a_1^o, \ldots, a_n^o) = G(a_1^o, \ldots, a_n^o)$ for all $a_1 \in [T_1], \ldots, a_n \in [T_n]$.

In the sequel we denote by $\{s_i\}_{i<\omega}^{S_+}$ a weakly increasing chain of states $s_0 \leq s_1 \leq s_2 \leq \ldots$.

Definition 8 (Convergence). Let $A$ be either $\mathbb{B}$ or $\mathbb{N}$. A function $\tau : S_+ \to A_+$ converges, written $\tau \downarrow$, if

$$\forall \{s_i\}_{i<\omega}^{S_+} \exists j \forall k \geq j. \tau(s_j) = \tau(s_k) \neq \bot.$$
Lemma 5. The mapping ς is functorial w.r.t. the quotient of \([T^S]\) under \(\sim_T\): \(\text{Id}_T^\circ \sim \text{Id}_{T^S}\); moreover \(f(a_1, \ldots, a_n)^\circ = f^\circ(a_1^\circ, \ldots, a_n^\circ)\), and \((f \circ g)^\circ = f^\circ \circ g^\circ\).

We can now interpret each term \(M : T\) of \(\mathcal{L}_1\) into our non standard model by an element of \([T^S]\).

Definition 10 (Term Interpretation for \(\mathcal{L}_1\)).

An environment is a map \(\rho\) sending any variable \(x : T\) into an element \(\rho(x) \in [T^S]\). The term interpretation map \([M]_\rho\) for \(M \in \mathcal{L}_1\) is defined:

- \([x]_\rho = \rho(x); [c]_\rho = c_\rho^\circ\) where \(c\) is any constant among \(0, \text{succ, eq, true, false, if, PR}\) and \(c\) is its standard interpretation in \([T]\): \([M]_\rho = [M]_\rho[N]_\rho; [\lambda x.T].M]_\rho = \lambda a \in [T^S].[M]_\rho\), where \(\rho(x) = a, \rho(y) = \rho(y)\) if \(y \neq x\);
- \([\chi_p]_\rho : (S_1 \rightarrow N_\perp)^k \rightarrow (S_1 \rightarrow B_\perp), \) where \(P : \text{Nat}^{k+1} \Rightarrow \text{Bool}\), is such that for all \(\tau \in (S_1 \rightarrow N_\perp)^k\) and \(s \in S_1, [\chi_p]_\rho(\tau, s) = \perp\) if there exists \(m\) such that \((\perp, \tau(s), m) \in s\) and \(\perp = \perp\); otherwise;
- \([\varphi_p]_\rho : (S_1 \rightarrow N_\perp)^k \rightarrow (S_1 \rightarrow N_\perp), \) where \(P : \text{Nat}^{k+1} \Rightarrow \text{Bool}\), is such that for all \(\tau \in (S_1 \rightarrow N_\perp)^k\) and \(s \in S_1, [\varphi_p]_\rho(\tau, s) = \perp\) if there exists \(m\) such that \((\perp, \tau(s), m) \in s\) and \(\perp = \perp\); otherwise;
- \([P]_\rho(\tau, s) = m\) where \(P, s) = m\) is the first triple in \(s\) whose first entries are \(P\) and \(\tau(s)\), if any; \(m = 0\) else.

The definition above has, as a consequence, the following interpretation for the derived combinator \(WR\). If \(f = [WR]_\rho(g, w, h) : (S_1 \rightarrow N_\perp) \rightarrow (S_1 \rightarrow N_\perp),\) then \(f(\tau)\) is (using some terms of \(\mathcal{L}_0\) in place of their interpretations for brevity):

\[g(\tau,[ifz][[H](w(h(\tau)), w(\tau)), f(h(\tau))])\]

An environment \(\rho\) is convergent, written \(\rho \downarrow\), if \(\rho(x) \downarrow^T\) for all \(x : T\). We then summarize the main properties of the construction developed so far.

Theorem 1 (Soundness of the Interpretation). Let \(M, N : T\) be any terms of \(\mathcal{L}_1\):

1. \([M]_\rho \in [T^S]\) for any \(\rho\), and if \(\rho \downarrow\) then \([M]_\rho \downarrow^T;\)
2. if \(\rho \in \mathcal{L}_0\) then \([M]_\rho \in [T]^\circ\) for any \(\rho\) such that \(\rho(x) \in [T]^\circ\) for all \(x : T\); furthermore \([M]_\rho\) is computable;
3. \([M]_\rho\) is hereditarily convergent if all \(\rho(x)\) are such;
4. if \(\mathcal{T} \vdash M = N\) (hence both \(M, N \in \mathcal{L}_0\), then \([M]_\rho \sim [N]_\rho;\)
5. if \(\vdash P\) is derivable in the theory \(\text{PRA} - \exists\) then \([P]_\rho \sim_{\text{Bool}} t\rho\) for any convergent \(\rho\).

5 The Realizability Interpretation

If \(P[t] \in \mathcal{L}_1\) is a closed predicate, then \([P[t]]_\rho\) is a boolean depending on a state, i.e., a map : \(S_1 \rightarrow B_\perp\). If \(P[x] \in \mathcal{L}_0\) (i.e., if \(P[x]\) is a primitive recursive predicate) then \([P[t][s]_\rho = \text{true value of } P[x] \text{ on } [H](s) = N\). Our goal is to extract from a proof \(H\) of a primitive recursive property \(P[t] \in \mathcal{L}_0\) for a term \(t \in \mathcal{L}_1\), some state \(s \in S_1\) such that \(= [H](s) = N\) satisfies \(P[x]\): in other words, we want to extract
from the proof of $P[t]$ some witness $n$ for $P[x]$. This is by no means immediate: even in the case in which $P[t]$ is provable, we cannot guarantee that for all $s \in S_\perp$ we have $[P[t]](s) = tt$. We will show that we can turn any proof $\Pi_1$ of $P[t]$ into a realizer picking, given any $s \in S_\perp$, some $s' \geq s$ (some extension of the state $s$ in the prefix ordering) such that $[P[t]](s') = tt$ (i.e., such that $P[n']$ for $n' = [t](s')$). The realizer is the part of the constructive content of the classical proof $\Pi_1$ which is not included in the term $t$.

As a first approximation, the realizer of $P[t]$ associated to $\Pi_1$ could be some map $\kappa_1 : S_\perp \rightarrow S_\perp$ such that for all $s \in S_\perp$, if $s' = \kappa_1(s)$ then $s' \geq s$ and $[P[t]](s') = tt$. $\kappa_1$, however, is not enough when $\Pi_1$ is in included in some proof-context $\Pi_2[\Pi_1]$ and when $\Pi_2[\_ \text{]}$ corresponds to some other construction $\kappa_2 : S_\perp \rightarrow S_\perp$. In this case the construction associated to the whole proof $\Pi_2[\Pi_1]$ would be $s'' = \kappa_2(\kappa_1(s)) \geq s$, and since $s''$ is not a value of $\kappa_1$ we cannot guarantee that $[P[t]](s'') = tt$. We overcome this problem by requiring that the realizer associated to $\Pi_1$ is some $F$ taking a state $s$ and a map $\kappa_2$ associated to some proof context $\Pi_2[\_ \text{]}$ including $\Pi_1$. We ask that $F$ extends $s$ to some $F(\kappa_2, s) \geq s$ in which the conclusions of both $\Pi_1$ and $\Pi_2[\_ \text{]}$ are true. The map $\kappa_2$ is used by $F$ as a continuation, that is, as a function representing the part of the program to be executed after $F$. Two further constraints on realizers (see the definition below) are: $\kappa_2$ is applied as the last step of the computation of $F$ (i.e., $F(\kappa_2, s) = \kappa_2(s''')$ for some $s''' \geq s$), and $\kappa_2$ is applied by $F$ only to extensions of the original state $s$.

We interpret a realizer $F : (S_\perp \rightarrow S_\perp) \rightarrow (S_\perp \rightarrow S_\perp)$ as a “process”. We think of any $\kappa : S_\perp \rightarrow S_\perp$ such that $\kappa(s) \geq s$ as the combined action of all processes outside $F$. Remark that the type of a set of processes is a subtype of the type of processes: in this way we can represent, within the simply typed lambda calculus, the fact that a process can interact with a set of processes. $id = id_{S_\perp}$ represents the empty action made by the empty set of processes. $F(id(\_ \text{]))$ is the canonical evaluation of a process, w.r.t. an empty set of other processes and the initial state. If $\kappa$ represents a set $\{F_1, \ldots, F_n\}$ of processes, and $F$ is a process, then $F(\kappa)$ represents the set of processes $\{F, F_1, \ldots, F_n\}$. The compound process whose components are $\{F_1, \ldots, F_n\}$ is $F \circ \ldots \circ F_n$. We think of the composition of realizers as an arbitrary sequentialization of the parallel composition of “processes”.

**Definition 11 (Realizer sets and Realizers).** Let us abbreviate $St = (S_\perp \rightarrow S_\perp) \rightarrow (S_\perp \rightarrow S_\perp)$.

1. A realizer set is a total function $\kappa : S_\perp \rightarrow S_\perp$ such that for all $s \in S$, $s \leq \kappa(s)$;
2. a realizer is any $F \in St$ such that
   (a) $F(\kappa) = \kappa \circ F'(\kappa)$ for some $F' \in St$ sending realizer sets into realizer sets and
   (b) if $s \neq \perp$ and $\kappa_1(s') = \kappa_2(s')$ for all $s' \geq s$, then $F(\kappa_1, s) = F(\kappa_2, s)$;
3. a realizing map is a function $\Phi : (S_\perp \rightarrow N_\perp)^k \rightarrow St$ mapping indexed numbers realizers.
4. The $k$-ary pointwise identity is the realizing map $I_k : (S_\perp \rightarrow N_\perp)^k \rightarrow St$ defined by $I_k(\tau) = \tau : S_\perp$ for all $\tau \in (S_\perp \rightarrow N_\perp)^k$.

We call $Id = id_{S_\perp \rightarrow S_\perp}$ the trivial realizing and $I_k$ the trivial realizing map.

We list below a few basic properties of realizers. For instance (Lemma 6.4) the composition of $n$ realizers $F_1, \ldots, F_n$ is a realizer whose range is included in the range of all $F_1, \ldots, F_n$. This is not true for generic functions, but it depends on the fact that the realizer set received as input is used as a continuation, and it will be crucial in order to prove the correctness of the realization interpretation.

**Lemma 6.** 1. $id$ is a realizer set. If $\kappa_1, \kappa_2$ are realizer sets, then $\kappa_1 \circ \kappa_2$ is a realizer set.
2. If $\kappa$ is a realizer set and $F$ a realizer, then $F(\kappa)$ is a realizer set.
3. $id$ is a realizer. $I_k$ is a realizing map. If $F,G$ are realizers, then $F \circ G$ is such.
4. If $F_1, \ldots, F_n$ are realizers, then for all realizer set $\kappa$ and all state $s$ if $s' = (F_1 \circ \ldots \circ F_n)(\kappa, s)$ then for some realizer sets $\kappa_1, \ldots, \kappa_n$ and states $s_1, \ldots, s_n$ we have $s' = F_1(\kappa_1, s_1) = \ldots = F_n(\kappa_n, s_n)$.

Proof.

1. $id(s) = s \geq s$. If $\kappa_1, \kappa_2$ are realizer sets, then $\kappa_1 \circ \kappa_2$ is total, and $(\kappa_1 \circ \kappa_2)(s) = \kappa_1(\kappa_2(s)) \geq \kappa_2(s) \geq s$. 


2. \(F(\kappa) = \kappa \circ F'(\kappa)\) is a realizer set because it is a composition of realizer sets (point 1).
3. Assume \(\kappa\) is a realizer set. We have \(\text{Id}(\kappa) = \kappa = \kappa \circ \text{id}\). If \(\kappa_1(s') = \kappa_2(s')\) for all \(s' \geq s\), then \(\text{Id}(\kappa_1)(s) = \kappa_1(s) = \kappa_2(s) = \text{Id}(\kappa_2)(s)\). Thus, \(\text{Id}\) is a realizer, and \(I_k\) a realizer map. Now assume \(F, G\) are realizers. We have \((\circ \circ \circ)(\kappa) = F(G(\kappa)) = \kappa \circ F'(G(\kappa))\), and \(F'(G(\kappa))\) is a realizer set because \(G(\kappa)\) is such by point 2. If \(\kappa_1(s') = \kappa_2(s')\) for all \(s' \geq s\), then \(\kappa_1(s'') = \kappa_2(s'')\) for all \(s'' \geq s' \geq s\). We deduce \(G(\kappa_1)(s') = G(\kappa_2)(s')\) for all \(s' \geq s\), therefore \(F(G(\kappa_1)), s) = F(G(\kappa_2), s)\). Thus, \(F \circ G\) is a realizer.
4. It is enough to prove the case \(n = 2\), the general case follows by induction on \(n\). By definition of realizer applied to \(F_1\) we have that, for any realizer set \(\kappa\) and state \(s\) there is some \(F'_1\) such that

\[
(F_1 \circ F_2)(\kappa, s) = F_1(F_2, s) = (F_2(\kappa) \circ F'_1(\kappa))(s) = F_2(\kappa, F'_1(\kappa, s)),
\]

Since \(\kappa\) is a realizer set, by point 2 above \(F_2(\kappa)\) is a realizer set. By choosing \(\kappa_1 = F_2(\kappa)\), \(s_1 = s\) and \(\kappa_2 = \kappa, s_2 = F'_1(\kappa, s)\) we conclude.

The central notion of this paper is that of a realization relation \(F \models \tau\), where \(F\) is a realizer and \(\tau: S_\perp \rightarrow B_\perp\) is an indexed truth value. The intended meaning is that \(F\) realizes \(\tau\) if for any realizer set \(\kappa, F(\kappa)\) sends any state \(s\) to some extension \(s' = F(\kappa, s) \geq s\) in which \(\tau\) is true (i.e. \(\tau(s') = tt\)).

**Definition 12.** Let \(F: \text{St}\) be a realizer and \(\tau: S_\perp \rightarrow B_\perp\) a convergent map.

1. \(F \models \tau\) if for all realizer sets \(\kappa\) and all \(s \in \mathbb{S}\) if \(s' = F(\kappa, s)\) then \(\tau(s') = tt\).
2. If \(P\) is a closed predicate of \(L_1\) and \(F\) a realizer, we say that \(F \models P\) if \(F \models [P]_\perp\).
3. If \(P\) is a predicate of \(L_1\) with free variables \(x_1, \ldots, x_k\) all of type \(\text{Nat}\), and \(\Phi: (S_\perp \rightarrow N_\perp)^k \rightarrow \text{St}\) a realizer map, then \(\Phi \models P\) if for all vectors \(\tau: (S_\perp \rightarrow N_\perp)^k\) of convergent maps \(\Phi(\tau) \models [P]_\perp/\tau\).

We will now prove that each rule of our system can be interpreted by an operation sending realizers of the assumptions into realizers of the conclusion. Eventually we will define, by induction over the proofs, a map \(R(.)\) sending a proof \(P\) of \(A\) into a realizer \(R(A)\) of \(A\). The realizer will express a part of the construction hidden in \(A\), the part which is not expressed by the subterms of \(A\).

Let \(\Phi, \psi: (S_\perp \rightarrow N_\perp)^k \rightarrow \text{St}\), then the pointwise composition \(\Phi \bullet \psi: (S_\perp \rightarrow N_\perp)^k \rightarrow \text{St}\) is defined as \((\Phi \bullet \psi)(\tau) = \Phi(\tau) \circ \psi(\tau)\). We will now check that the pointwise composition of the realizers of the assumptions of a Post rule is the realizer of the conclusion of the same rule. The proof relies on the fact that the realizer of the first assumption of the rule uses as continuation the set of all realizers of the remaining assumptions of the rule.

**Lemma 7.** Let \(P_1, \ldots, P_k\) and \(Q\) be the premises and the conclusion of an instance of the Post scheme: for any \(s \in S_\perp\), if \([P_1]_\rho(s) = \cdots = [P_k]_\rho(s) = tt\), then \([Q]_\rho(s) = tt\), for any \(\rho\).

**Proposition 4.** Suppose that in system \(\text{PRA-}\exists\) there is a derivation ending by an instance of the Post rule schema:

\[
\begin{array}{c}
P_1 & \cdots & P_k \\
\hline
Q
\end{array}
\]

and let \(\Phi_1 \models P_1, \ldots, \Phi_k \models P_k\): then \(\Phi_1 \bullet \cdots \bullet \Phi_k \models Q\).

**Proof.** It suffices to prove the statement when \(\Phi_1 = F_1, \ldots, \Phi_n = F_k\) are just realizers (so that in particular \(F_1 \circ \cdots \circ F_k = F_1 \circ \cdots \circ F_k\)). By Lemma 6.4, for all realizer set \(\kappa\) and state \(s\) if \(s' = (F_1 \circ \cdots \circ F_k)(\kappa, s)\) then for some realizer sets \(\kappa_1, \ldots, \kappa_k\) and some states \(s_1, \ldots, s_k\) we have \(s' = F(\kappa_1, s_1) = \cdots = F(\kappa_k, s_k)\). By assumption \([P_k](F(\kappa_k, s_k)) = \cdots = [P_k](F(\kappa_k, s_k)) = tt\) for all realizer sets \(\kappa_1, \ldots, \kappa_k\) and states
\[ P_1(s') \ldots P_k(s') = tt, \] and by Lemma 7 that \([Q](s') = tt\). By definition of realization we conclude \(F_1 \circ \ldots \circ F_k \models Q\).

Note that we can replace, in the proof above, the composition \(F_1 \circ \ldots \circ F_k\) by any permutation of it, and we would obtain some (in general, different) realization of the same conclusion \(Q\) (see the end of §6). Our interpretation of this fact is that composition is an arbitrary sequentialization of a parallel composition between "processes" \(F_1 \circ \ldots \circ F_k\). Note also that if the Post Rule is unary, then any realization of the only assumption is a realization of the conclusion.

For each instance of the \(\varphi\)-axiom and of the \(\chi\)-axiom we can define two realization maps. The \(k\)-ary pointwise identity (i.e., the trivial realization) realizes the \(\varphi\)-axiom with \(k\) free variables. In fact, the \(\varphi\)-axiom hides no construction on states, because it is true in all states by the very interpretation of \(\chi\) and \(\varphi\).

**Proposition 5.** Fix any instance \(Q \equiv \chi\varphi(x) \implies P[x, \varphi\varphi(x)]\) of the \(\varphi\)-axiom, with \(x = x_1, \ldots, x_n : \mathbb{N}\). Then \(I_k \models Q\).

The crucial step is defining a realization \(r_{P, k+1}\) of the \(\chi\)-axiom for the \(k + 1\)-ary primitive recursive predicate \(P[x, y] \in L_0\) instantiated in some \(n, m : \mathbb{N}\). The interpretation of such an instance might be false in some state \(s\). Indeed, we might have \([x\varphi](n, m)(s) = ff\) while \([P]_{n, m, x, y}(s) = tt\). In this case we extend \(s\) to some state \(s' > s\) in which the given instance of the \(\chi\)-axiom is true. The realization \(r_{P, k+1}\) defines \(s'\) by first adding the triple \((P, n, m)\) to \(s\), then by applying the realization set variable \(\kappa\) and obtaining some \(s'' = \kappa(s') \geq s'\). By definition we have \([x\varphi](n, m)(s'') = tt\), therefore the \(\chi\)-axiom for \(P\) and \(n, m : \mathbb{N}\) is true in \(s''\). We interpret this step as an atomic "learning" step: the realization learns one point in the graph of the map \(\chi\) (and of the map \(\varphi\), since they are closely related).

Sometimes one step of learning is not enough to validate the \(\chi\)-axiom. We therefore define an operator \(\Omega\), raising \(r_{P, k+1}\) to some realization \(R_{P, k+1}\) of the \(\chi\)-axiom instantiated over \(\varphi, \tau : [\mathbb{N}]^{k+1}\), a vector of indexed integers. \(\Omega\) repeatedly applies \(r_{P, k+1}\) to \(\varphi, \tau\) in order to validate the given instance of the \(\chi\)-axiom, because the values \(\varphi, \tau\) might change with the state (see the example at the end of §5). In the rest of the paper, when we write \textbf{let} \((x = u)\) in \((t)\) we mean \((\lambda x.t)(u)\).

**Definition 13 (Realizer of the \(\chi\)-axiom).** Assume \(n : \mathbb{N}^k, m : \mathbb{N}\) and \(\kappa : S_\bot \rightarrow S_\bot\) and \(s : S_\bot\). Let \(P \in L_0\) be a primitive recursive predicate interpreting it. Then we define \(r_{P, k+1} : S_\bot \rightarrow [\mathbb{N}]^{k+1} \rightarrow \text{St}\) and \(R_{P, k+1} : (S_\bot \rightarrow [\mathbb{N}]^{k+1}) \rightarrow \text{St}\) by:

1. \(r_{P, k+1}(n, m, \kappa, s) \equiv (\neg [x\varphi](n, s) \wedge [P]_{n, m, x, y}(s), \kappa(s \circ (P, n, m)), \kappa(s)) : S_\bot\)
2. Assume \(\Phi : \mathbb{N}^k \rightarrow \text{St}\) is a family of realizers indexed over \([\mathbb{N}]^k\), and \(\tau : (S_\bot \rightarrow [\mathbb{N}])^k\) and \(\kappa : (S_\bot \rightarrow S_\bot)\). Then the realization map \(\Omega(\Phi) : (S_\bot \rightarrow [\mathbb{N}])^k \rightarrow \text{St}\) is defined by

\[
\Omega(\Phi, \tau)(\kappa, s) = \text{let} \ (s' = \Phi(\tau(s))(\kappa, s)) \ 	ext{in} \ (if \ \tau(s) = \tau(s') \ then \ s' \ else \ \Omega(\Phi, \tau)(\kappa, s')) : S_\bot
\]
3. \(R_{P, k+1} \equiv \Omega(r_{P, k+1}) : \text{St}\)

The computation of \(\Omega(\Phi, \tau)(\kappa, s)\) produces a weakly increasing sequence of states \(s = s_0 \leq s_1 \leq s_2 \leq \cdots\) such that \(s_{n+1} = \Phi(\tau(s_n))(\kappa, s_n)\) for all \(n\). If \(\tau\) is convergent, then \(\tau(s_{n+1}) = \tau(s_n)\) for some \(n\), and the computation terminates with output \(s_{n+1}\). \(\Omega\) defines in this way a realization map for the \(\chi\)-axiom.

**Proposition 6 (Realizer of the \(\chi\)-axiom).** Fix any instance \(Q \equiv (P[x, y] \implies \chi\varphi(x))\) of the \(\chi\)-axiom, for the \(k + 1\)-ary primitive recursive predicate \(P[x, y] \in L_0\). Let \(R[y] \in L_1\).

1. \(R_{P, k+1}(n, n) \models [Q]_{n, m, x, y}, \) for all \(n \in \mathbb{N}^k\) and \(n \in \mathbb{N}\).
2. If \(\Phi(m) \models [R]_{m, x, y}\) for all \(m \in \mathbb{N}^k\), then \(\Omega(\Phi) \models R\).
3. \(r_{P, k+1} \models Q\).
We can also define a realizer $WF$ mapping a realizer map $\Phi$ for the assumption of an induction rule with parameters $t, t$ into a realizer map $WF(\Phi, [t], [t])$ for the conclusion of the induction rule. Apart from a single detail we will precise in a moment, $WF$ is the usual realizer of the rule of well-founded induction.

**Definition 14.** Let $C : S_\bot \rightarrow B_\bot$. For any realizer $F$ we define a conditional realizer, which is $F$ when $C$ holds, and $\text{Id}$ otherwise: $\text{if}(C, F)(\kappa, s) = \text{if}(C(s), F(\kappa, s), \kappa(s))$.

**Definition 15.** Assume $w, f_1, \ldots, f_k : [\text{Nat} \Rightarrow \text{Nat}]$. Let $C_1 : S_\bot \rightarrow B_\bot$ denote the condition $w(f_i(\tau)) < w(\tau)$, for $i = 1, \ldots, k$. If $C : S_\bot \rightarrow B_\bot$, then $C \wedge^C C_1$ is by definition $\lambda s. C(s) \land C_1(s)$. Let $\Phi : (S_\bot \rightarrow N_\bot) \rightarrow St$. Then

1. $WF_C$ is recursively defined as follows: abbreviate $WF_C(\Phi)(\tau) \equiv WF_C(\Phi, w, f)(\tau)$, we set $WF_C(\Phi)(\tau) = \text{if}(C, \Phi(\tau) \circ WF_{C \land C_1}(\Phi)(f_1(\tau)) \ldots \circ WF_{C \land C_k}(\Phi)(f_k(\tau)))$

2. Assume $\Phi : (S_\bot \rightarrow N_\bot)^{k+1} \rightarrow St$ and $w, f_1, \ldots, f_k : [\text{Nat}^{k+1} \rightarrow \text{Nat}]$. Then we define $WF(\Phi, w, f)(\tau) = WF_T(\Phi(\tau), w(\tau), f(\tau))$, where the index $T$ is the always true condition.

Note the “guard” $C(\tau)$ in front of the definition $WF_C = \ldots$. By definition unfolding, this means that whenever $C(s)$ is false in a state $s$, the realizer $WF_C$ together with all its recursive calls trivialize to the identity. The reason for having these “guards” is that the clauses $C_i = w(f_i(\tau)) < w(\tau)$ on which the recursive calls depend may change their truth value from true to false, and whenever this happens the recursive call must disappear.

We can now state that $WF$ produces a realizer map for the conclusion of the induction rule.

**Lemma 8.** Assume $P \in L_1$ is a predicate and $t, t_1, \ldots, t_k : \text{Nat} \rightarrow \text{Nat} \in L_1$ a, with $x = x_1, \ldots, x_n$ and $\text{FV}(P) = x, x$ and $\text{FV}(t, t) = x$. Assume $\Phi : (S_\bot \rightarrow N_\bot)^{k+1} \rightarrow St$ be a realizer map. Abbreviate $[t] = \lambda x. \Phi(t)\Phi(t)$ and $[t] = \lambda x. \Phi(t)\Phi(t)\Phi(t)$. If $\Phi$ realizes the assumption of the induction rule for $P$: $\Phi \Rightarrow \text{ifz}(t_1(x) \prec x)P[t_1(x)/x] \wedge \cdots \wedge \text{ifz}(t_k(x) \prec x)P[t_k(x)/x] \Rightarrow P$

then $WF(\Phi, [t], [t]) \Rightarrow P$.

By putting all together, we define by induction on $\Pi$ a map $R(\Pi)$ from proofs to realizer maps.

**Definition 16 (The realizer map $R(\Pi)$).** Assume $\Pi$ is a proof with free variables $x = x_1, \ldots, x_k$.

1. if $\Pi$ ends with a $\varphi$-axiom, then $R(\Pi) = I$ (the $k$-ary pointwise equality).

2. if $\Pi$ ends with a $\chi$-axiom, then $R(\Pi) = R_{\varphi,k}$

3. if $\Pi$ ends with a Post rule with immediate subproofs $\Pi_1, \ldots, \Pi_k$, then $R(\Pi) = R(\Pi_1) \bullet \cdots \bullet R(\Pi_1)$

4. If $\Pi$ ends with an induction rule with parameters $t, t$ and immediate subproof $\Pi_1$, then $R(\Pi) = WF(R(\Pi_1), [t], [t])$.

We claim that $R(\Pi)$ realizes the conclusion $P$ of $\Pi$. By this we mean that, given any initial knowledge state $s \in S_\bot$, and any “continuation” $\kappa$ (representing the action of other realizers taking place at the end of the computation of $R(\Pi)$), $R(\Pi)(\kappa, s)$ returns a state in which the interpretation of $P$ is true.

**Theorem 2 (Realizability).** Let $\Pi : x_1, \ldots, x_k \vdash P$ be a proof in system PRA-$\exists$, then $R(\Pi) \models P$, that is for all convergent $\tau \in [\text{Nat}^{\exists}]$, realizer set $\kappa$ and state $s \in S_\bot$,

$$[P]_{\tau/x}(R(\Pi)(\tau, \kappa, s)) = tt.$$
Corollary 1 (Trivial Realizability). Assume \( P \in \mathcal{L} \), has free variables \( x = x_1, \ldots, x_k \). Let \( \Pi \) be a proof of \( P \) including no instance of the \( \chi \)-axiom. Denote by \( R(\Pi) \) the realizor of \( P \) computed out of \( \Pi \), and by \( I_k \) the \( k \)-ary realization map constantly equal to \( \text{Id} \).

Then \( [P] \) is always true, i.e., for all convergent \( \tau \in [\text{Nat}^5]^k \) and all \( s \in S \) we have \( [P]_{\tau / x}(s) = \text{tt} \) and \( I_k \models P \) and \( R(\Pi) = I_k \).

6 An Example of Realizer

Let \( P(x, y) \equiv f(x) > f(y) \). The proof of \( Q(0) \equiv \neg \chi_P(t(0)) \) in section 2 does not involve the \( \chi \)-axiom, which is the only one that corresponds to some non-trivial realizor. By Lemma 1, if \( \Pi \) is a proof of \( Q(0) \) in \( \text{PRA} \), then \( R(\Pi) = I_k \) (the \( k \)-ary pointwise identity). The fact that \( R(\Pi) \) is trivial depends on the fact that in the proof of “\( t(0) \) is a minimum point of \( f \)” we do not make any essential use of the “states of knowledge”, namely of the outcomes of the use of the oracle \( \chi_P \) against particular points.

Things are different if we use the proof \( \Pi \) to derive consequences of \( Q(0) \), this time using the \( \chi \)-axiom in the remaining parts of the proof. Abbreviate \( e = t(0) \); then \( e \) is a minimum point of \( f \) by our proof of MP. By \( \chi \)-axiom and contraposition from \( \neg \chi_P(e) \) we deduce \( f(e) \leq f(x) \) (\( x \) free variable). This is nothing else than the original definition of “\( e \) is a minimum point of \( f \)”.

Assume \( g, h : \mathbb{N} \to \mathbb{N} \) are primitive recursive functions. If we instantiate \( x \) to \( g(e) \) and to \( h(e) \) we deduce \( f(e) \leq f(g(e)) \vee f(e) \leq f(h(e)) \). In other words, the minimum principle implies that the inequations \( f(x) \leq f(g(x)) \wedge f(x) \leq f(h(x)) \) have a solution for \( x = e \). The classical proof of the existence of such a solution implicitly defines a non-trivial way of finding a solution, which is caught in the realization semantics. We stress that the solution that comes out is not necessarily the minimum point \( e \) itself, because \( e \) in general is not computable in \( f, g, h \), but some computable \( x = e(s) \in \mathbb{N} \) “approximating” the property of being a minimum point enough to validate \( f(x) \leq f(g(x)) \wedge f(x) \leq f(h(x)) \). A sketch of the proof-tree is:

\[
\frac{\chi_P(e) \quad \chi_P(e) \quad \chi_P(e) 
\text{Taut}}{
\chi_P(e) \quad \chi_P(e) \quad \chi_P(e) 
\text{Taut}}
\]

In our semantics, \( f, g, h \) are interpreted by \( f^\circ, g^\circ, h^\circ \), with \( f^\circ(\tau)(s) = f(\tau(s)), \ldots \). Let \( \eta = [e] \) be the interpretation of \( e \). Then the realizers of the left and right instances of the \( \chi \)-axioms in the proof-tree above are some \( G = \text{R}_P(\eta, g^\circ(\eta)) \) and \( H = \text{R}_P(\eta, h^\circ(\eta)) \). Since Post rules are interpreted by composition, and \( \text{Id} \) realizes \( \neg \chi_P(e) \), the realizor of the whole proof is \( \text{Id} \circ G \circ \text{Id} \circ H = \text{Id} \circ H \).

In order to analyze \( G \circ H \), we will analyze in this order the behavior of \( \eta(s) \), then the behavior of the realizers \( \text{R}_P, \text{R}_P = \Omega(\text{R}_P) \), \( G, H \) and \( G \circ H \), all applied to the initial state \( s_0 = \emptyset \) and to the continuation \( \text{Id} \), corresponding to the empty realizor set. By unfolding the definition of \( \eta, \eta(s) \) computes a sequence \( 0 = x_0, x_1, x_2, \ldots \) with \( x_{i+1} = [\varphi_P](x_i^\circ, s) \) for all \( i \), as long as \( [\chi_P](x_i^\circ, s) \) is true. Eventually \( \eta(s) \) outputs \( x_i \) for the first \( i \) such that \( [\chi_P](x_i^\circ, s) \) is false. \( [\chi_P](x_i^\circ, s) \) is true if there is some triple \( (P, x_i, m) \) in \( s \), and in this case \( x_{i+1} = m \) for the first such triple. The more points \( s \) includes of the graph of \( \varphi_P \), the more \( \eta(s) \) is close to be some minimum point of \( f \), but usually \( \eta(s) \) does not reach a minimum point. Consider the application \( \text{R}_P(\eta, h^\circ(\eta), \text{Id}, s_0) \), associated to the rightmost instance of the \( \chi \)-axiom in the proof. \( \text{R}_P \) checks whether \( f^\circ(\eta)(s_0) > f^\circ(h^\circ(\eta))(s_0) \) (i.e., \( f(\eta(s_0)) > f(h(\eta(s_0))) \)) is true and \( [\chi_P](\eta)(s) \) is false. In the initial state \( s_0 = \emptyset \), \( [\chi_P](\eta)(s_0) \) is false and \( \eta(s_0) = 0 \). Therefore \( f(\eta(s_0)) > f(h(\eta(s_0))) \) is equivalent to \( f(0) > f(h(0)) \). If this is true, then the realizor \( \text{R}_P \) learns that \( [\chi_P](\eta) \) should be true, and returns some extended state \( s_1 = s_0 @ (P, 0, h(0)) > s_0 \), in which this is indeed the case. \( \eta(s_1) \) is equal to \( h^\circ(\eta)(s_0) = h(\eta(s_0)) = h(0) \). By definition unfolding, the role of \( \Omega \) in \( \text{R}_P = \Omega(\text{R}_P) \) is to cyclically apply \( \text{R}_P(\eta, h^\circ(\eta)) \) as long as the value of \( \eta(s_1) \) continues to change.
Therefore $H = \Phi(\eta, h^\omega(\eta))$, when applied to $\mathbf{id}, s_0$, repeatedly checks whether $f^\omega(\eta)(s_i) > f^\omega(h^\omega(\eta))(s_i)$ (i.e., $f(\eta(s_i)) > f(h(\eta(s_i)))$) is true and $\chi_F(\eta)(s_i)$ is false. If this is the case, then the realizer $\tau_F$ learns that $\chi_F(\eta)(s_i)$ should be true, and returns some extended state $s_i+1 = s_i \circ \langle \mathcal{P}, \eta(s_i), h(\eta(s_i)) \rangle > s_i$, in which this is indeed the case. Thus, $\eta(s_i+1)$ is equal to $h^\omega(\eta)(s_i) = h(\eta(s_i))$. By induction on $i$ we deduce $\eta(s_i) = h^i(0)$ for all $i$. This cycle stops in the first $i$ such that $f(\eta(s_i)) \leq f(h(\eta(s_i)))$, i.e., such that $f(h^i(0)) \leq f(h^{i+1}(0))$.

By a similar reasoning, $G(\mathbf{id}, s_0)$ stops in the first $i_1$ such that $f(g^{i_1}(0)) \leq f(g^{i_1+1}(0))$. Now, what does the compound realizer $G \circ H$ do? That is, what happens if we compute $(G \circ H)(\mathbf{id}, s_0)$? This is the realizer $G$ using $H(\mathbf{id})$ as continuation. This means that $G \circ H$ finds the first $i_1$ such that $f(g^{i_1}(0)) \leq f(g^{i_1+1}(0))$, then it calls the continuation $H(\mathbf{id})$ on the state $s_{i_1}$, and it finds the first $i_2$ such that $f(h^{i_2}(g^{i_1}(0))) \leq f(h^{i_2+1}(g^{i_1}(0)))$. We produce in this way a sequence $x_0, x_1, x_2, \ldots$ with $x_{i+1} h^{i+1}(0))$). These operations are repeated as long as the value of $\eta(s_j)$ continues to change. In the next step, $(G \circ H)(\mathbf{id}, s_0)$ finds the first $i_3$ such that $f(g^{i_3}(x_{i_1+i_2})) \leq f(g^{i_3+1}(x_{i_1+i_2}))$, then it calls the continuation $H(\mathbf{id})$ on the state $s_{i_1+i_2+i_3}$, and it finds the first $i_4$ such that $f(h^{i_4}(g^{i_3}(x_{i_1+i_2+i_3}))) \leq f(h^{i_4+1}(g^{i_3}(x_{i_1+i_2+i_3})))$. And so on. When the value of $\eta(s_j)$ stops changing, for instance, if $i_4 = 0$, we have found some $j = i_1 + i_2 + i_3$ and some $x_j = g^{i_3}(h^{i_4}(g^{i_3}(0)))$ such that both $f(x_j) \leq f(g(x_j))$ and $f(x_j) \leq f(h(x_j))$. This is a solution to the original problem, by unwinding a non-trivial algorithm hidden in the classical proof.

Remark that, by computing $(H \circ G)(\mathbf{id}, s_0)$ instead, we would reverse the role of $g, h$, obtaining an algorithm from the same proof, equally correct, but returning in general a different result. Not all algorithms implicit in the proof can be obtained in this way. For instance, our semantics does not allow an execution alternating one step of $G$ and one step of $H$. Yet this is a correct algorithm, which is implicit in the proof, and returning a result which is in general different from all previous ones. To put otherwise, the parallel composition of $G, H$ is not expressible in our semantics in its full generality.

References