Relating classical realizability and negative translation for existential witness extraction

Alexandre Miquel

Université Paris 7 & LIP (ENS Lyon)

alexandre.miquel@ens-lyon.fr

Abstract. Friedman showed how to turn a classical proof of a Σ^0_1 formula into an intuitionistic proof of the same formula, thus giving an effective method to extract witnesses from classical proofs of such formulae. In this paper we show how to achieve the same goal efficiently using Krivine realizability with primitive numerals, and prove that the corresponding program is but the direct-style equivalent (using call-cc) of the CPS-style program underlying Friedman's method.

1 Introduction

Classical realizability is a powerful framework introduced by Krivine [4, 7] to study the proofs-as-programs paradigm in classical logic. Its main feature is that the computational interpretation of proofs is not described via a negative translation, but instead expressed in direct style using a λ-calculus extended with the control operator call/cc. Although classical realizability is traditionally presented in second-order classical arithmetic, it can be extended to much more expressive logical frameworks such as Zermelo-Fraenkel set theory [5] or the calculus of constructions with universes [8]. And with the help of extra instructions, it can even provide realizers for several forms of the axiom of choice [5].

The purpose of this paper is twofold. First, it aims at presenting the method that naturally comes with classical realizability in order to extract a witness from a classical proof of a Σ^0_1-formula. Second, it aims to relate this extraction method with the traditional method introduced by Friedman [2] that combines a negative translation with an intuitionistic realizability interpretation, and to show that through this translation, both extraction methods are basically the same (up to the details in the definition of the negative translation).

For that, we first present Krivine's framework for classical realizability as well as the corresponding witness extraction method, introducing a primitive (and we believe, more efficient) representation of natural numbers in the language of realizers—instead of using Church numerals. We then define an intuitionistic realizability model for second-order arithmetic as well as a negative translation in the spirit of [9], but extended to primitive numerals. We finally analyze the witness extraction method of classical realizability through the negative translation, and show that it corresponds to the transformation used by Friedman to prove the conservativity of Peano arithmetic over Heyting arithmetic for Σ^0_1 (and actually Π^0_2) formulæ.

2 Classical realizability in second-order logic

2.1 The language of second-order logic

The language of second-order logic is parameterized by a first-order language of *expressions* (a.k.a. first-order terms) to represent the individuals. In this paper, we shall only consider *arithmetic expressions* (notation: e, e', etc.) that are formed from first-order variables (notation: x, y, z, etc.) and the constant symbol 0 ('zero') using function symbols for all primitive recursive definitions of functions (notation: f, g, h, etc.), including a unary function symbol s for the successor function, binary function symbols $+$ and $\times$ for addition and multiplication, and a unary function symbol 'pred' for the predecessor function.

Formulæ of the (minimal) language of second-order logic are formed from second-order variables (notation: X, Y, Z, etc.) of all arities using implication and first- and second-order universal quantification:

$$\textbf{Formulæ} \qquad A, B \quad ::= \quad X(e_1, \ldots, e_k) \quad | \quad A \Rightarrow B \quad | \quad \forall x\, A \quad | \quad \forall X\, A\,.$$

The set of all free (first- and second-order) variables of a formula A is written $FV(A)$. The notions of first- and second-order substitution in a formula are defined as usual, and written $A\{x := e\}$ and $A\{X(x_1, \ldots, x_k) := B\}$ respectively.

In what follows we shall consider the following (standard) second-order encodings for connectives and first- and second-order existential quantifications as well as for Leibniz equality:

$$\begin{array}{rcl}
\bot & \equiv & \forall Z\, Z \\
\neg A & \equiv & A \Rightarrow \bot \\[4pt]
A \wedge B & \equiv & \forall Z\, ((A \Rightarrow B \Rightarrow Z) \Rightarrow Z) \\
A \vee B & \equiv & \forall Z\, ((A \Rightarrow Z) \Rightarrow (B \Rightarrow Z) \Rightarrow Z) \\[4pt]
\exists x\, A(x) & \equiv & \forall Z\, (\forall x\, (A(x) \Rightarrow Z) \ \Rightarrow \ Z) \\
\exists X\, A(X) & \equiv & \forall Z\, (\forall X\, (A(X) \Rightarrow Z) \ \Rightarrow \ Z) \\[4pt]
e = e' & \equiv & \forall Z\, (Z(e) \Rightarrow Z(e'))
\end{array}$$

(where Z is a fresh variable).

2.2 A type system for second-order logic

We now define a type system for classical second-order logic, based on a judgment of the form $\Gamma \vdash_{\mathrm{NK}} t : A$, where Γ is a *typing context*, t a *proof-term* and A a formula of the language defined above. Here, proof-terms (notation: t, u, etc.) are simply the pure λ-terms enriched with a special constant written $\mathbf{c\!c}$ ('call/cc'), to prove Peirce's law. Typing contexts (notation: Γ, Γ', etc.) are finite functions from proof-variables to formulæ.

The inference rules of this system are given in Fig. 1. These rules contain an axiom rule, introduction and elimination rules for implication and first- and second-order universal quantification, plus a typing rule for $\mathbf{c\!c}$ (Peirce's axiom).

The semantics of this system is given by the classical realizability model we are going to define now.

$$\frac{}{\Gamma \vdash_{\mathrm{NK}} x : A}\ {\scriptstyle (x:A)\in\Gamma} \qquad \frac{}{\Gamma \vdash_{\mathrm{NK}} \text{cc} : ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}$$

$$\frac{\Gamma, x : A \vdash_{\mathrm{NK}} t : B}{\Gamma \vdash_{\mathrm{NK}} \lambda x\,.\,t : A \Rightarrow B} \qquad \frac{\Gamma \vdash_{\mathrm{NK}} t : A \Rightarrow B \qquad \Gamma \vdash_{\mathrm{NK}} t : A}{\Gamma \vdash_{\mathrm{NK}} tu : B}$$

$$\frac{\Gamma \vdash_{\mathrm{NK}} t : A}{\Gamma \vdash_{\mathrm{NK}} t : \forall x\, A}\ {\scriptstyle x \notin FV(\Gamma)} \qquad \frac{\Gamma \vdash_{\mathrm{NK}} t : \forall x\, A}{\Gamma \vdash_{\mathrm{NK}} t : A\{x := e\}}$$

$$\frac{\Gamma \vdash_{\mathrm{NK}} t : A}{\Gamma \vdash_{\mathrm{NK}} t : \forall X\, A}\ {\scriptstyle X \notin FV(\Gamma)} \qquad \frac{\Gamma \vdash_{\mathrm{NK}} t : \forall X\, A}{\Gamma \vdash_{\mathrm{NK}} t : A\{X(x_1, \ldots, x_k) := B\}}$$

Fig. 1. Typing rules for classical second-order logic

2.3 A calculus of realizers

Krivine's classical realizability model [7] is based on a much larger larger calculus than the calculus of proof-terms described in 2.2. Instead, the language λ_c distinguishes three distinct syntactic categories: *terms*, *stacks* and *processes*.

Terms (notation: t, u, etc.) and stacks (notation: π, π', etc.) are defined by mutual induction as follows:

Terms	$t, u \ ::= \ x \ \mid \ \lambda x\,.\,t \ \mid \ tu \ \mid \ \kappa \ \mid \ \mathsf{k}_\pi$		
Stacks	$\pi \ ::= \ \alpha \ \mid \ t \cdot \pi$		(t closed)

Terms are the pure λ-terms enriched with constants for every *instruction* (notation: κ, κ', etc.) of a fixed instruction set $\mathcal{K}$ that contains (at least) the instruction cc, plus a *continuation constant* for every stack π. Stacks are finite lists of *closed* terms terminated by a *stack constant*[1] (notation: α, β, etc.) Note that unlike terms (that may be open or closed), stacks only contain closed terms and are thus closed objects—so that the continuation constant k_π associated to every stack π is really a constant.

Finally, a *process* (notation: p, q, etc.) is a pair written $t \star \pi$ and formed by a closed term t and a stack π:

Processes $\qquad p, q \ ::= \ t \star \pi \qquad$ (t closed)

The set of closed terms (resp. the set of stacks) is written Λ_c (resp. Π), and the set of processes is written $\Lambda_c \star \Pi$.

[1] Since the witness extraction method we present in this paper (in section 4) does not make use of the constant at the bottom of the stack, it is safe to assume here that there is only a single stack constant 'nil'. However, the presence of several stack constants is sometimes desirable when working with extra instructions that make use of them, such as the 'clock' instruction [5, 7].

2.4 Evaluation

The set of processes is equipped with a binary relation $p \succ p'$ of *evaluation* that satisfies (at least) the following axioms

Grab	$\lambda x\,.\,t \;\star\; u\cdot\pi$	$\succ$	$t\{x:=u\} \;\star\; \pi$
Push	$tu \;\star\; \pi$	$\succ$	$t \;\star\; u\cdot\pi$
Call/cc	$\mathbf{c\!\!c} \;\star\; t\cdot\pi$	$\succ$	$t \;\star\; \mathsf{k}_\pi\cdot\pi$
Resume	$\mathsf{k}_\pi \;\star\; t\cdot\pi'$	$\succ$	$t \;\star\; \pi$

for all $t,u \in \Lambda_c$ and $\pi,\pi' \in \Pi$. Note that only processes are subject to evaluation: there is no notion of reduction for either terms or stacks in λ_c.

2.5 The realizability interpretation

The construction of the classical realizability model is parameterized by a set of processes $\perp\!\!\!\perp \subseteq \Lambda_c \star \Pi$ (the 'pole') which we assume to be *saturated* (or *closed under anti-evaluation*) in the sense that conditions $p \succ p'$ and $p' \in \perp\!\!\!\perp$ together imply $p \in \perp\!\!\!\perp$ for all $p,p' \in \Lambda_c \star \Pi$.

We call a *falsity value* any set of stacks $S \subseteq \Pi$. By orthogonality, every falsity value $S \subseteq \Pi$ induces a *truth value* $S^{\perp\!\!\!\perp} \subseteq \Lambda_c$ defined as:

$$S^{\perp\!\!\!\perp} \;=\; \{t \in \Lambda_c \;:\; \forall \pi \in S \;\; t\star\pi \in \perp\!\!\!\perp\}\,.$$

A *valuation* is a function ρ that maps every first-order variable x to a natural number $\rho(x) \in \mathbb{N}$, and every second-order variable X of arity k to a falsity value function $\rho(X) : \mathbb{N}^k \to \mathfrak{P}(\Pi)$. A *parametric expression* (resp. a *parametric formula*) is simply an expression e (resp. a formula A) equipped with a valuation ρ, that we write $e[\rho]$ (resp. $A[\rho]$). Parametric contexts are defined similarly.

For every parametric expression $e[\rho]$ we write $\mathbf{Val}(e[\rho]) \in \mathbb{N}$ the *value* of $e[\rho]$, interpreting variables by their images in ρ while giving to the primitive recursive function symbols in e their standard interpretation.

Every parametric formula $A[\rho]$ is interpreted as two sets, namely: a *falsity value* $\|A[\rho]\| \subseteq \Pi$ and a *truth value* $|A[\rho]| \subseteq \Lambda_c$. Both sets are defined by induction on A as follows:

$$\begin{aligned}
\|X(e_1,\ldots,e_k)[\rho]\| &= \rho(X)(\mathbf{Val}(e_1[\rho]),\ldots,\mathbf{Val}(e_k[\rho])) \\
\|(A \Rightarrow B)[\rho]\| &= |A[\rho]|\cdot\|B[\rho]\| \;=\; \{t\cdot\pi \;:\; t \in |A[\rho]|,\; \pi \in \|B[\rho]\|\} \\
\|(\forall x\, A)[\rho]\| &= \bigcup_{n\in\mathbb{N}} \|A[\rho; x \leftarrow n]\| \\
\|(\forall X\, A)[\rho]\| &= \bigcup_{F:\mathbb{N}^k\to\mathfrak{P}(\Pi)} \|A[\rho; x \leftarrow F]\| \\
|A[\rho]| &= \|A\|^{\perp\!\!\!\perp} \;=\; \{t \in \Lambda_c \;:\; \forall \pi \in \|A[\rho]\| \;\; t\star\pi \in \perp\!\!\!\perp\}
\end{aligned}$$

Since the truth value $|A[\rho]|$ and the falsity value $\|A[\rho]\|$ actually depend on the parameter $\perp\!\!\!\perp$, we shall sometimes use the notations $|A[\rho]|_{\perp\!\!\!\perp}$ and $\|A[\rho]\|_{\perp\!\!\!\perp}$ to indicate this dependency explicitly. In what follows, we shall write

- $t \Vdash_{\mathrm{NK}} A[\rho]$ ('t realizes $A[\rho]$') when $t \in |A[\rho]|_{\perp\!\!\!\perp}$ (keeping in mind that this notion depends on the choice of the pole $\perp\!\!\!\perp$);
- $t \Vvdash_{\mathrm{NK}} A[\rho]$ ('t universally realizes $A[\rho]$') when $t \in |A[\rho]|_{\perp\!\!\!\perp}$ for all saturated sets $\perp\!\!\!\perp \subseteq \Lambda_c \star \Pi$.

2.6 Adequacy

We call a *substitution* any finite function from proof-variables to closed λ_c-terms, and write $t[\sigma]$ the closed term obtained by applying a substitution σ to a term t. Given a substitution σ and a parametric context $\Gamma[\rho]$, we write $\sigma \Vdash_{\mathrm{NK}} \Gamma[\rho]$ when $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\sigma)$ and $\sigma(x) \Vdash_{\mathrm{NK}} A[\rho]$ for all $(x : A) \in \Gamma$. We say that:

- A judgment $\Gamma \vdash_{\mathrm{NK}} t : A$ is *sound* (w.r.t. the pole $\perp\!\!\!\perp$) when for all valuations ρ and for all substitutions σ such that $\sigma \Vdash_{\mathrm{NK}} \Gamma[\rho]$, we have $t[\sigma] \Vdash_{\mathrm{NK}} A[\rho]$.
- An inference rule $\frac{P_1 \cdots P_n}{C}$ (where $P_1, \ldots, P_n$ and C are typing judgments) is *sound* (w.r.t. the pole $\perp\!\!\!\perp$) when the soundness of its premises $P_1, \ldots, P_n$ (in the above sense) implies the soundness of its conclusion C.

From these definitions, it is clear that the conclusion of any typing derivation formed with only sound inference rules is sound.

Proposition 1 (Adequacy). — *The typing rules of Fig. 1 are sound w.r.t. all poles $\perp\!\!\!\perp \subseteq \Lambda_c \times \Pi$.*

A consequence of this proposition is that closed proof-terms given by the type system of Fig. 1 provide universal realizers of the corresponding formulæ. (But not all realizers can be detected via typing [7].)

3 From second-order logic to second-order arithmetic

3.1 Extending the language of formulæ

We enrich the language of formulæ with a unary predicate constant null(e) (whose name is self-explanatory) plus a syntactic construct $\{e\} \Rightarrow B$ (where e is an expression and B a formula) whose semantics will be given in 3.3[2]:

Formulæ $\qquad A, B \quad ::= \quad \cdots \quad | \quad \mathrm{null}(e) \quad | \quad \{e\} \Rightarrow B$

This extension of the language is accompanied with the shorthands:

$$\begin{aligned} \top \;\equiv\; \mathrm{null}(0) \qquad\qquad \mathrm{nat}(e) \;&\equiv\; \forall Z\,((\{e\} \Rightarrow Z) \Rightarrow Z) \\ \forall^{\mathbb{N}} x\, A(x) \;&\equiv\; \forall x\,(\{x\} \Rightarrow A(x)) \\ \exists^{\mathbb{N}} x\, A(x) \;&\equiv\; \forall Z\,(\forall x\,(\{x\} \Rightarrow A(x) \Rightarrow Z) \Rightarrow Z) \end{aligned}$$

[2] Intuitively: $\{e\} \Rightarrow B$ is the type of functions producing a proof of B when applied to the value of e, using the primitive representation of numerals described in 3.2

We also introduce two congruences over expressions and formulæ, written $e \cong e'$ and $A \cong A'$. The congruence $e \cong e'$ over the class of expressions is the congruence generated by the equational theory of the primitive recursive function symbols expressions are made of. The congruence $A \cong A'$ over the class of formulæ is then defined as the least congruence containing the contextual closure of the congruence $e \cong e'$ across atomic formulæ, and satisfying the extra equation $\mathrm{null}(s(e)) \cong \bot$ (writing $\bot \equiv \forall Z\, Z$). Note that from the definition of the propositional constant $\top$, the equation $\mathrm{null}(0) \cong \top$ comes for free.

3.2 Adding primitive numerals to λ_c

The instruction set $\mathcal{K}$ is enriched with the following instructions:

- For every $n \in \mathbb{N}$, a (pseudo-)instruction $\hat{n} \in \mathcal{K}$ representing the numeral n as a pure datum. We call the constant $\hat{n}$ a *pseudo-instruction* since it comes with no evaluation rule (i.e. of the form $\hat{n} \star \pi \succ \cdots$), thus expressing that the constant $\hat{n}$ is meaningless in function position.
- Two constants s and rec with the reduction rules

$$\begin{array}{rclcrcl} \mathsf{s} & \star & \hat{n} \cdot u \cdot \pi & \succ & u & \star & \widehat{n+1} \cdot \pi \\ \mathsf{rec} & \star & u_0 \cdot u_1 \cdot \hat{0} \cdot \pi & \succ & u_0 & \star & \pi \\ \mathsf{rec} & \star & u_0 \cdot u_1 \cdot \widehat{n+1} \cdot \pi & \succ & u_1 & \star & \hat{n} \cdot (\mathsf{rec}\, u_0\, u_1\, \hat{n}) \cdot \pi \end{array}$$

  for all $u, u_0, u_1 \in \Lambda_c$, $n \in \mathbb{N}$ and $\pi \in \Pi$.

With these instructions, it is possible to implement every primitive recursive function f of arity k as a term $\check{f}$ with the reduction rule

$$\check{f} \star \hat{n}_1 \cdots \hat{n}_k \cdot u \cdot \pi \quad \succ^* \quad u \star \widehat{m} \cdot \pi\,,$$

writing m the image of $(n_1, \ldots, n_k)$ by f. To improve efficiency, we can also introduce the $\check{f}$s (or some of them) as primitive instructions.

Apart from the representation of numerals as pure data, every natural number $n \in \mathbb{N}$ can be also represented as a *program* $\check{n}$ defined by $\check{n} \equiv \lambda x\,.\,x\hat{n}$. (This program will receive the type $\mathrm{nat}(n)$ from the type system defined in 3.4.) More generally we call a *lazy numeral* any closed term t such that $tu \succ u\hat{n}$ for some $n \in \mathbb{N}$ (that may depend on u) for all $u \in \Lambda_c$.

3.3 The extended realizability interpretation

The realizability interpretation defined in 2.5 is extended to the new syntactic constructs by letting:

$$\|\mathrm{null}(e)[\rho]\| = \begin{cases} \varnothing & \text{if } \mathbf{Val}(e[\rho]) = 0 \\ \Pi & \text{otherwise} \end{cases}$$

$$\|(\{e\} \Rightarrow B)[\rho]\| = \{\hat{n} \cdot \pi \ :\ n = \mathbf{Val}(e[\rho]),\ \pi \in \|B[\rho]\|\}$$

From the interpretation of the predicate $\mathrm{null}(e)$ and from the definitions of the congruences $e \cong e'$ and $A \cong A'$, we immediately get:

Proposition 2 (Denotations of congruent expressions/formulæ).

1. *If $e \cong e'$, then $\mathbf{Val}(e[\rho]) = \mathbf{Val}(e'[\rho])$ for all valuations ρ;*
2. *If $A \cong A'$, then $\|A[\rho]\| = \|A'[\rho]\|$ and $|A[\rho]| = |A'[\rho]|$ for all valuations ρ.*

3.4 The extended type system and its adequacy

We first extend the notion of typing context by allowing a second form of declaration $x : \{e\}$, where x is a proof-variable and e an arithmetic expression. Given a substitution σ and a parametric context $\Gamma[\rho]$ (according to the extended definition of contexts), the notation $\sigma \Vdash_{\mathrm{NK}} \Gamma[\rho]$ now means that:

- $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\sigma)$;
- $\sigma(x) \Vdash_{\mathrm{NK}} A[\rho]$ for all $(x : A) \in \Gamma$;
- $\sigma(x) \equiv \hat{n}$ where $n = \mathbf{Val}(e[\rho])$, for all $(x : \{e\}) \in \Gamma$.

The type system defined in Fig. 1 is extended with the inference rules of Fig. 2. These rules comprise a conversion rule (in the spirit of type theory and

$$\frac{\Gamma \vdash_{\mathrm{NK}} t : A}{\Gamma \vdash_{\mathrm{NK}} t : A'}\ {\scriptstyle A\cong A'} \qquad \frac{}{\Gamma \vdash_{\mathrm{NK}} t : \top} \qquad \frac{}{\Gamma \vdash_{\mathrm{NK}} \mathsf{s} : \forall^{\mathbb{N}} x\ \mathrm{nat}(s(x))}$$

$$\frac{}{\Gamma \vdash_{\mathrm{NK}} \mathsf{rec} : \forall X\, (X(0) \Rightarrow \forall^{\mathbb{N}} x\, (X(x) \Rightarrow X(s(x))) \Rightarrow \forall^{\mathbb{N}} x\, X(x))}$$

$$\frac{\Gamma, x : \{e\} \vdash_{\mathrm{NK}} t : B}{\Gamma \vdash_{\mathrm{NK}} \lambda x\,.\,t : \{e\} \Rightarrow B} \qquad \frac{\Gamma \vdash_{\mathrm{NK}} t : \{e\} \Rightarrow B}{\Gamma \vdash_{\mathrm{NK}} tx : B}\ {\scriptstyle (x:\{e\})\in\Gamma} \qquad \frac{\Gamma \vdash_{\mathrm{NK}} t : \{s^n 0\} \Rightarrow B}{\Gamma \vdash_{\mathrm{NK}} t\hat{n} : B}$$

Fig. 2. Typing rules for classical second-order arithmetic

deduction modulo), typing rules for the instructions s (2nd Peano axiom) and rec (induction principle), plus typing rules for the construct $\{e\} \Rightarrow B$.

Proposition 3 (Adequacy). — *The typing rules of Fig. 2 are sound w.r.t. all poles $\perp\!\!\!\perp \subseteq \Lambda_c \times \Pi$.*

Using these rules, one can derive for instance that $\check{n} \equiv \lambda x\,.\,x\hat{n}$ has type $\mathrm{nat}(s^n 0)$, and more generally build proof-terms for the axioms of arithmetic:

Fact 1 — *The following judgments are derivable:*

1. $\vdash_{\mathrm{NK}} \check{0} : \mathrm{nat}(0)$ *(1st Peano axiom)*
2. $\vdash_{\mathrm{NK}} \mathsf{s} : \forall^{\mathbb{N}} x\ \mathrm{nat}(s(x))$ *(2nd Peano axiom)*
3. $\vdash_{\mathrm{NK}} \lambda z\,.\,z : \forall x\, (s(x) = s(y) \Rightarrow x = y)$ *(3rd Peano axiom)*
4. $\vdash_{\mathrm{NK}} \lambda z\,.\,z\ (\lambda w\,.\,w) : \forall x\ \neg(0 = s(x))$ *(4th Peano axiom)*
5. $\vdash_{\mathrm{NK}} \mathsf{rec} : A(0) \Rightarrow \forall^{\mathbb{N}} x\, (A(x) \Rightarrow A(s(x))) \Rightarrow \forall^{\mathbb{N}} x A(x)$ *(5th Peano axiom)*

4 Witness extraction in classical realizability

A fundamental difference between classical realizability and intuitionistic realizability is that in classical realizability we do not evaluate terms but processes, that are objects formed by combining a proof (the current term) and a counter-proof (the current stack) of the same formula. From a logical point of view, evaluation thus takes place in an inconsistent world, where a proof and a counter-proof can coexist and interact with each other.

It is well-known that in classical realizability, the truth value $|A|$ of any (parametric) formula A is always inhabited provided the pole $\perp\!\!\!\perp$ is not empty.[3] On the other hand, the realizability model induced by the empty pole $\perp\!\!\!\perp = \varnothing$ simply mimics the (full) standard model of PA2, since $|A|_{(\perp\!\!\!\perp=\varnothing)} = \Lambda_c$ iff A is true (in the standard model), and $|A|_{(\perp\!\!\!\perp=\varnothing)} = \varnothing$ otherwise [7].

A consequence of the 'local inconsistency' of classical realizability is that when we get a classical realizer t of an existential formula $\exists^{\mathbb{N}}x\, A(x)$ from which we extract a number $n \in \mathbb{N}$ and a realizer $t_n \Vdash_{\mathrm{NK}} A(n)$, we can never trust the certificate t_n that '$A(n)$ holds'. Instead, we have to test the proposed (and potentially false) witness to check whether $A(n)$ holds or not—which requires a decision procedure for the predicate $A(x)$—and repudiate the current witness (to get another witness) as long as the test fails.

Here is how to proceed formally.

We assume given a unary primitive recursive symbol f with a universal realizer $t_0 \Vvdash_{\mathrm{NK}} \exists^{\mathbb{N}}x\, \mathrm{null}(f(x))$, for instance a universal realizer that comes from a proof in the system described in Fig. 1 and 2. (Any Σ^0_1 formula can be given this form.) Let $\check{f}$ be a term that computes f in the sense of 3.2, that is: a term such that $\check{f} \star \hat{n} \cdot u \cdot \pi \succ^* u \star \widehat{f(n)} \cdot \pi$ for all $n \in \mathbb{N}$, $u \in \Lambda_c$ and $\pi \in \Pi$. (Such a term is also a universal realizer of $\forall^{\mathbb{N}}x\ \mathrm{nat}(f(x))$.)

From the term $\check{f}$ let us define $d_f \equiv \lambda nxy\,.\,\check{f}\,n\,(\lambda p\,.\,\mathsf{rec}\,x\,(\lambda\_\_\,.\,y)\,p)$. By definition, the term d_f decides whether $f(n) = 0$ or not, in the sense that

$$d_f \star \hat{n} \cdot u_0 \cdot u_1 \cdot \pi \quad \succ \quad \begin{cases} u_0 \cdot \pi & \text{if } f(n) = 0 \\ u_1 \cdot \pi & \text{if } f(n) \neq 0 \end{cases}$$

for all $n \in \mathbb{N}$, $u_0, u_1 \in \Lambda_c$ and $\pi \in \Pi$. Let us now form the term

$$t_0' \equiv t_0\,(\lambda xy\,.\,d_f\,x\,(\mathsf{stop}\,x)\,y)$$

where 'stop' is an instruction with no evaluation rule.

Intuitively, the argument that is passed to the realizer t_0 is a function that extracts a potential witness $x \in \mathbb{N}$ with a certificate $y \Vdash_{\mathrm{NK}} A(x)$, and that decides whether $f(x) = 0$ or not using d_f. When the test succeeds, the (correct) witness x is returned via the return instruction stop. When the test fails, the (wrong) certificate y—a realizer of $\mathrm{null}(f(x))$, that is, a realizer of $\perp$—is given

[3] Given an arbitrary process $t_0 \star \pi_0 \in \perp\!\!\!\perp$, it is easy to check that $\mathsf{k}_{\pi_0} t_0 \in |A|$ for every parametric formula A.

the control. In practice, such a realizer of $\bot$ can do nothing but backtrack using a formerly saved stack. In this way we implement a retroaction loop where the successive witnesses proposed by the realizer t_0 are tested and repudiated as long as the test fails, until a correct witness is found and then returned.[4]

Putting these intuitions into symbols, we get the following:

Proposition 4 (Decidable witness extraction). — *For all $\pi \in \Pi$, the process $t'_0 \star \pi$ evaluates to $\mathsf{stop} \star \hat{n} \cdot \pi$ for some $n \in \mathbb{N}$ such that $f(n) = 0$.*

Proof. Fix $\pi \in \Pi$, and consider the pole $\perp\!\!\!\perp$ formed by all the processes p such that $p \succ^* \mathsf{stop} \star \hat{n} \cdot \pi$ for some $n \in \mathbb{N}$ such that $f(n) = 0$. Since t_0 is a universal realizer of the formula $\exists^{\mathbb{N}} x\, \mathrm{null}(f(x))$, it is also a realizer w.r.t. the pole $\perp\!\!\!\perp$ defined above. Taking a valuation ρ such that $\rho(Z) = \{\pi\}$, we immediately check that $\mathsf{stop} \Vdash_{\mathrm{NK}} (\{n\} \Rightarrow Z)[\rho]$ for all $n \in \mathbb{N}$ such that $f(n) = 0$ (by definition of $\perp\!\!\!\perp$). Distinguishing the cases where $f(n) = 0$ and $f(n) \neq 0$, we then prove that $\lambda xy\,.\, d_f\, x\, (\mathsf{stop}\, x)\, y \Vdash_{\mathrm{NK}} (\{n\} \Rightarrow \mathrm{null}(f(n)) \Rightarrow Z)[\rho]$ for all $n \in \mathbb{N}$, hence the same term realizes $\forall x\, (\{x\} \Rightarrow \mathrm{null}(f(x)) \Rightarrow Z)[\rho]$. Consequently, we have $t'_0 \star \pi \succ^* t_0 \star (\lambda xy\,.\, d_f\, x\, (\mathsf{stop}\, x)\, y) \cdot \pi \in \perp\!\!\!\perp$, hence the desired result. □

In section 7 we shall reinterpret this witness extraction method through a well-suited negative translation.

5 Intuitionistic realizability for second-order arithmetic

We now define an intuitionistic type system accompanied with its realizability model whose definition follows the global pattern of the Brouwer-Heyting-Kolmogorov interpretation. As in [9], we introduce a primitive form of conjunction (as a Cartesian product) and primitive forms of first- and second-order existential quantification (as infinitary unions).

5.1 The language of formulæ

Taking the same language of arithmetic expressions as before (cf 2.1) with its congruence $e \cong e'$ (cf 3.1) we now consider the following language of formulæ:

Formulæ
$$\begin{array}{rcl} A, B & ::= & \mathrm{null}(e) \quad | \quad \mathrm{nat}(e) \quad | \quad X(e_1, \ldots, e_k) \\ & | & A \Rightarrow B \quad | \quad \forall x\, A \quad | \quad \forall X\, A \\ & | & A \wedge B \quad | \quad \exists x\, A \quad | \quad \exists X\, A \end{array}$$

Compared with the language for classical logic described in 2.1 and 3.1, the language above replaces the construct $\{e\} \Rightarrow B$ by a (more standard) primitive predicate $\mathrm{nat}(e)$. We also consider primitive constructions for conjunction and first- and second-order existential quantifications. In this setting, numeric quantifications are defined as

$$\forall^{\mathbb{N}} x\, A(x) \;\equiv\; \forall x\, (\mathrm{nat}(x) \Rightarrow A(x)) \quad \text{and} \quad \exists^{\mathbb{N}} x\, A(x) \;\equiv\; \exists x\, (\mathrm{nat}(x) \wedge A(x))$$

[4] This witness extraction method is actually implemented in the module for classical program extraction currently developped by the author for the Coq assistant [10].

The congruence $A \cong A'$ over the class of formulæ is defined from the congruence induced by $e \cong e'$ (across atomic formulæ) by adding the equations

$$\mathrm{null}(e) \;\cong\; \bot \qquad \text{and} \qquad (\exists v\, A(v)) \Rightarrow B \;\cong\; \forall v\,(A(v) \Rightarrow B)$$

where v is any first- or second-order variable that does not occur free in B. (This second equation will be crucial to establish the result of Prop. 9.) As before, we write $\top \equiv \mathrm{null}(0)$.

5.2 A type system for intuitionistic second-order arithmetic

We introduce an intuitionistic (and more traditional) proof system based on a judgment of the form $\Gamma \vdash_{\mathrm{NJ}} t : A$, where the proof-term t is now formed in the pure λ-calculus enriched with the following constants: pair (pairing), fst (first projection), snd (second projection), $\mathsf{0}$ (zero), s (successor) and rec (recursor). In what follows we shall write $\langle t; u\rangle$ for the application $\mathsf{pair}\, t\, u$, and denote by Λ the set of all closed proof-terms. Typing contexts are simply defined here as finite functions from proof-variables to formulæ.

The class of derivable judgments $\Gamma \vdash_{\mathrm{NJ}} t : A$ is inductively defined from the rules of inference of Fig. 3, using the abbreviation $\forall^{\mathbb{N}} x\, A(x)$ for the numeric quantification $\forall x\,(\mathrm{nat}(x) \Rightarrow A(x))$ such as defined in 5.1. (Note that there is no elimination rule for the primitive existential quantifier, since the desired elimination can be performed using the conversion $\forall v\,(A(v) \Rightarrow B) \cong (\exists v\, A(v)) \Rightarrow B$.)

This system is expressive enough to provide typable proof-terms for all the theorems of intuitionistic second-order arithmetic.

5.3 Weak reduction

Proof-terms are equipped with a binary relation of one-step *weak reduction* written $t \succ_w t'$ and defined by the rules

$$\frac{}{(\lambda x\,.\,t)u \succ_w t\{x := u\}} \qquad \frac{}{\mathsf{rec}\, u_0\, u_1\, \mathsf{0} \succ_w u_0} \qquad \frac{}{\mathsf{rec}\, u_0\, u_1\, (\mathsf{s}\, t) \succ_w u_1\, t\, (\mathsf{rec}\, u_0\, u_1\, t)}$$

$$\frac{}{\mathsf{fst}\,\langle t_1; t_2\rangle \succ_w t_1} \qquad \frac{}{\mathsf{snd}\,\langle t_1; t_2\rangle \succ_w t_2} \qquad \frac{t \succ_w t'}{tu \succ_w t'u} \qquad \frac{u \succ_w u'}{tu \succ_w tu'}$$

Note that weak reduction is allowed both in the left- and right hand-side of applications, but not below λ-abstraction (i.e. we disable the ξ-rule of λ-calculus). We write $\succ_w^*$ the reflexive-transitive closure of one step weak reduction.

Complementarily to the notion of weak reduction, we also define a relation of *inner reduction* written $t \succ_i t'$ from the rules:

$$\frac{t \succ_w t'}{\lambda x\,.\,t \succ_i \lambda x\,.\,t'} \qquad \frac{t \succ_i t'}{tu \succ_i t'u} \qquad \frac{u \succ_i u'}{tu \succ_i tu'} \qquad \frac{t \succ_i t'}{\lambda x\,.\,t \succ_i \lambda x\,.\,t'}$$

The reflexive-transitive closure of the relation of inner reduction is written $\succ_i^*$ while its reflexive-symmetric-transitive closure is written $=_i$.

The union of both relations $\succ_w$ and $\succ_i$ is the ordinary relation of one step reduction, written $\succ$. By the standard method of parallel reductions we get:

$$\frac{}{\Gamma \vdash_{\mathrm{NJ}} x : A}\ {\scriptstyle (x:A)\in\Gamma} \qquad \frac{}{\Gamma \vdash_{\mathrm{NJ}} t : \top} \qquad \frac{\Gamma \vdash_{\mathrm{NJ}} t : A}{\Gamma \vdash_{\mathrm{NJ}} t : A'}\ {\scriptstyle A\simeq A'}$$

$$\frac{}{\Gamma \vdash_{\mathrm{NJ}} \mathsf{pair} : A \Rightarrow B \Rightarrow A \wedge B}$$

$$\frac{}{\Gamma \vdash_{\mathrm{NJ}} \mathsf{fst} : A \wedge B \Rightarrow A} \qquad \frac{}{\Gamma \vdash_{\mathrm{NJ}} \mathsf{snd} : A \wedge B \Rightarrow B}$$

$$\frac{}{\Gamma \vdash_{\mathrm{NJ}} \mathsf{0} : \mathrm{nat}(0)} \qquad \frac{}{\Gamma \vdash_{\mathrm{NJ}} \mathsf{s} : \forall^{\mathbb{N}} x\ \mathrm{nat}(s(x))}$$

$$\frac{}{\Gamma \vdash_{\mathrm{NJ}} \mathsf{rec} : \forall X\,(X(0) \Rightarrow \forall^{\mathbb{N}} x\,(X(x) \Rightarrow X(s(x))) \Rightarrow \forall^{\mathbb{N}} x\, X(x))}$$

$$\frac{\Gamma, x : A \vdash_{\mathrm{NJ}} t : B}{\Gamma \vdash_{\mathrm{NJ}} \lambda x\,.\,t : A \Rightarrow B} \qquad \frac{\Gamma \vdash_{\mathrm{NJ}} t : A \Rightarrow B \qquad \Gamma \vdash_{\mathrm{NJ}} t : A}{\Gamma \vdash_{\mathrm{NJ}} tu : B}$$

$$\frac{\Gamma \vdash_{\mathrm{NJ}} t : A}{\Gamma \vdash_{\mathrm{NJ}} t : \forall x\, A}\ {\scriptstyle x\notin FV(\Gamma)} \qquad \frac{\Gamma \vdash_{\mathrm{NJ}} t : \forall x\, A}{\Gamma \vdash_{\mathrm{NJ}} t : A\{x := e\}}$$

$$\frac{\Gamma \vdash_{\mathrm{NJ}} t : A}{\Gamma \vdash_{\mathrm{NJ}} t : \forall X\, A}\ {\scriptstyle X\notin FV(\Gamma)} \qquad \frac{\Gamma \vdash_{\mathrm{NJ}} t : \forall X\, A}{\Gamma \vdash_{\mathrm{NJ}} t : A\{X(x_1,\dots,x_k) := B\}}$$

$$\frac{\Gamma \vdash_{\mathrm{NJ}} t : A\{x := e\}}{\Gamma \vdash_{\mathrm{NJ}} t : \exists x\, A} \qquad \frac{\Gamma \vdash_{\mathrm{NJ}} t : A\{X(x_1,\dots,x_k) := B\}}{\Gamma \vdash_{\mathrm{NJ}} t : \exists X\, A}$$

Fig. 3. Typing rules for intuitionistic second-order arithmetic

Proposition 5. — *The relation $\succ$ is confluent.*

Moreover, we easily check that inner reduction can always be postponed:

Proposition 6. — *If $t \succ_i t' \succ_w t''$, then $t \succ_w t'_1 \succ_i^* t''$ for some t'_1.*

From this proposition and the confluence of $\succ$ we get:

Proposition 7 (Confluence of $\succ_w$ modulo $=_i$). — *It $t \succ_w t_1$ and $t \succ_w t_2$, then there are terms t'_1 and t'_2 such that $t_1 \succ_w t'_1$, $t_2 \succ_w t'_2$ and $t'_1 =_i t'_2$.*

5.4 The intuitionistic realizability model

We now build a simple realizability model for the type system defined above, in which formulæ are interpreted as *saturated sets* of terms, that is, as sets of closed proof-terms $S \subseteq \Lambda$ such that both conditions $t \succ_w t'$ and $t' \in S$ imply $t \in S$. The set of all saturated sets is written **SAT**.

Here, a *valuation* is a function ρ that maps every first-order variable x to a natural number $\rho(x) \in \mathbb{N}$, and every second-order variable X of arity k to a function $\rho(X) : \mathbb{N}^k \to \mathbf{SAT}$. Parametric expressions, formulæ and contexts are defined as before. Every parametric formula $A[\rho]$ is interpreted as a saturated set $[\![A[\rho]]\!] \in \mathbf{SAT}$ that is defined by the standard equations

$$\begin{aligned}
[\![X(e_1,\ldots,e_k)[\rho]]\!] &= \rho(X)(\mathbf{Val}(e_1[\rho]),\ldots,\mathbf{Val}(e_k[\rho])) \\
[\![\mathrm{null}(e)[\rho]]\!] &= \begin{cases} \Lambda & \text{if } \mathbf{Val}(e[\rho]) = 0 \\ \varnothing & \text{otherwise} \end{cases} \\
[\![\mathrm{nat}(e)[\rho]]\!] &= \{t \in \Lambda \ : \ t \succ_w^* \mathsf{s}^n 0, \ \text{where } n = \mathbf{Val}(e[\rho])\} \\
[\![(A \Rightarrow B)[\rho]]\!] &= \{t \in \Lambda \ : \ \forall u \in [\![A[\rho]]\!] \ \ tu \in [\![B[\rho]]\!]\} \\
[\![(A \land B)[\rho]]\!] &= \{t \in \Lambda \ : \ \exists u_1 \in [\![A[\rho]]\!] \ \exists u_2 \in [\![B[\rho]]\!] \ \ t \succ_w^* \langle t; u\rangle\}
\end{aligned}$$

$$[\![(\forall x\, A)[\rho]]\!] = \bigcap_{n\in\mathbb{N}} [\![A[\rho; x \leftarrow n]]\!] \qquad [\![(\forall X A)[\rho]]\!] = \bigcap_{F:\mathbb{N}^k\to\mathbf{SAT}} [\![A[\rho; X \leftarrow F]]\!]$$

$$[\![(\exists x\, A)[\rho]]\!] = \bigcup_{n\in\mathbb{N}} [\![A[\rho; x \leftarrow n]]\!] \qquad [\![(\exists X A)[\rho]]\!] = \bigcup_{F:\mathbb{N}^k\to\mathbf{SAT}} [\![A[\rho; X \leftarrow F]]\!]$$

In what follows, we shall write $t \Vdash_{\mathrm{NJ}} A[\rho]$ for $t \in [\![A[\rho]]\!]$.

Fact 2 — *If $A \cong A'$, then $[\![A[\rho]]\!] = [\![A'[\rho]]\!]$ for all valuations ρ.*

5.5 Adequacy

Given a substitution σ and a parametric context $\Gamma[\rho]$, we write $\sigma \Vdash_{\mathrm{NJ}} \Gamma[\rho]$ when $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\sigma)$ and $\sigma(x) \Vdash_{\mathrm{NJ}} A[\rho]$ for all $(x : A) \in \Gamma$. We say that:

– A judgment $\Gamma \vdash_{\mathrm{NJ}} t : A$ is *sound* when for all valuations ρ and for all substitutions σ such that $\sigma \Vdash_{\mathrm{NJ}} \Gamma[\rho]$, we have $t[\sigma] \Vdash_{\mathrm{NJ}} A[\rho]$.
– An inference rule $\frac{P_1 \cdots P_n}{C}$ (where $P_1,\ldots,P_n$ and C are typing judgments) is *sound* when the soundness of its premises $P_1,\ldots,P_n$ (in the above sense) implies the soundness of its conclusion C.

Proposition 8 (Adequacy). — *The typing rules of Fig. 3 are sound.*

From this result combined with the realizability interpretation of existential quantification and conjunction, we immediately get:

Fact 3 (Witness property) — *If $\vdash_{\mathrm{NJ}} t : \exists^{\mathbb{N}} x\, A(x) \equiv \exists x\,(\mathrm{nat}(x) \land A(x))$, then $t \succ_w^* \langle \mathsf{s}^n 0; t'\rangle$ for some $n \in \mathbb{N}$ and for some realizer $t' \Vdash_{\mathrm{NJ}} A(n)$.*

A consequence of this is that every proof of $\exists^{\mathbb{N}} x\, \mathrm{null}(f(x))$ weakly reduces to a pair of the form $\langle \mathsf{s}^n 0; t'\rangle$, where n is such that $f(n) = 0$.

6 The negative translation

6.1 Translating formulæ

We now define a negative translation of classical formulæ (such as defined in subsections 2.1 and 3.1) into intuitionistic formulæ (in the sense of subsection 5.1), in the spirit of [9]. As usual, this translation is parameterized by a fixed intuitionistic formula R—the formula that will represent the pole $\perp\!\!\!\perp$. In what follows, we write $\neg_R A$ for $A \Rightarrow R$.

Every classical formula A is translated as two intuitionistic formulæ written $A^{\neg\neg}$ and $A^{\perp}$. The formula $A^{\neg\neg}$ is simply defined as a shorthand for $A^{\neg\neg} \equiv \neg_R A^{\perp}$ whereas the formula $A^{\perp}$ is defined by induction on A as follows:

$$\begin{array}{rcl@{\qquad\qquad}rcl}
(X(e_1,\ldots,e_k))^{\perp} & \equiv & X(e_1,\ldots,e_k) & (\mathrm{null}(e))^{\perp} & \equiv & \mathrm{null}(h(e)) \\
(A \Rightarrow B)^{\perp} & \equiv & A^{\neg\neg} \wedge B^{\perp} & (\forall x\, A)^{\perp} & \equiv & \exists x\, A^{\perp} \\
(\{e\} \Rightarrow B)^{\perp} & \equiv & \mathrm{nat}(e) \wedge B^{\perp} & (\forall X\, A)^{\perp} & \equiv & \exists X\, A^{\perp}
\end{array}$$

where the (unary) primitive recursive function symbol h is defined by the equations $h(0) = 1$ and $h(s(x)) = 0$. It is a simple exercise to check that:

Fact 4 — *If $A \cong A'$, then $A^{\perp} \cong A'^{\perp}$ and $A^{\neg\neg} \cong A'^{\neg\neg}$.*

Note also that by definition, we have

$$(\forall v\, A(v))^{\neg\neg} \;\equiv\; \exists v\, A(v)^{\perp} \Rightarrow R \;\cong\; \forall v\,(A(v)^{\perp} \Rightarrow R) \;\equiv\; \forall v\,(A(v)^{\neg\neg})$$

6.2 CPS-translating terms and stacks

We now define two translations $t \mapsto t^*$ and $\pi \mapsto \pi^*$ (that are defined by mutual induction on t and π) from the terms and stacks of the λ_c-calculus to the proof-terms defined in 5.2. These translations are parameterized by a fixed mapping $\alpha \mapsto \alpha^*$ associating a proof-term α^* to every stack constant α of λ_c.

Stacks are translated in the obvious way, as finite lists:

$$(\alpha)^* \;\equiv\; \alpha^* \qquad\qquad (t\cdot\pi)^* \;\equiv\; \langle t^*;\ \pi^*\rangle$$

Variable, abstraction and application are translated as expected

$$\begin{array}{rcl}
x^* & \equiv & x \\
(tu)^* & \equiv & \lambda k\,.\,t^*\,\langle u^*;k\rangle \\
(\lambda x\,.\,t)^* & \equiv & \lambda k\,.\,(\lambda x\,.\,t^*)\,(\mathsf{fst}\,k)\,(\mathsf{snd}\,k)
\end{array}$$

whereas continuation constants and call/cc are translated as

$$\begin{array}{rcl}
(\mathsf{k}_\pi)^* & \equiv & \lambda k\,.\,\mathsf{fst}\,k\,\pi^* \\
(\mathsf{cc})^* & \equiv & \lambda k\,.\,\mathsf{fst}\,k\,\langle(\lambda z k'\,.\,\mathsf{fst}\,k'\,z)\,(\mathsf{snd}\,k);\ \mathsf{snd}\,k\rangle
\end{array}$$

Interestingly, the pure datum $\hat{n}$ is translated as

$$(\hat{n})^* \;\equiv\; \mathsf{s}^n\,0\,.$$

Here, the translation does not start with a continuation abstraction $\lambda k \ldots$, since the construct $\hat{n}$ is not intended to appear in head position. Finally, the instructions s and rec are translated as:

$$\begin{aligned}
(\mathsf{s})^* &\equiv \lambda k\,.\,\mathsf{fst}\,(\mathsf{snd}\,k)\,\langle \mathsf{s}\,(\mathsf{fst}\,k);\ \mathsf{snd}\,(\mathsf{snd}\,k)\rangle \\
(\mathsf{rec})^* &\equiv \lambda k\,.\,\mathsf{rec}\ (\mathsf{fst}_1\,k)\ (\lambda pyk'\,.\,\mathsf{fst}\,(\mathsf{snd}\,k)\,\langle p; \langle y; k'\rangle\rangle) \\
&\qquad\qquad (\mathsf{fst}\,(\mathsf{snd}\,(\mathsf{snd}\,k)))\ (\mathsf{snd}\,(\mathsf{snd}\,(\mathsf{snd}\,k)))
\end{aligned}$$

Proposition 9 (Correctness w.r.t. typing). — *If $\Gamma \vdash_{\mathrm{NK}} t : A$ (Fig. 1–2), then $\Gamma^{\neg\neg} \vdash_{\mathrm{NJ}} t^* : A^{\neg\neg}$ (Fig. 3).*

6.3 Simulation of evaluation by weak reduction

The expected property would be that each evaluation step $t_1 \star \pi_2 \succ t_2 \star \pi_2$ in λ_c corresponds to one or several weak reduction steps $t_1^*\,\pi_1^* \succ_w^+ t_2^*\,\pi_2^*$ through the CPS-translation. Although this works for almost all the evaluation rules—application, abstraction, call/cc, continuation and successor—the property does not hold for the evaluation of rec so that we need to refine a little bit more.

Proposition 10 (One step simulation). — *If $t_1 \star \pi_1 \succ t_2 \star \pi_2$ (one step evaluation in λ_c), then $t_1^*\,\pi_1^* \succ_w^+ t_2^*\,u$ (weak reduction) for some term $u =_i \pi_2^*$.*

Corollary 1 (Grand simulation). — *If $t_1 \star \pi_1 \succ^* t_2 \star \pi_2$ (evaluation in λ_c), then $t_1^*\,\pi_1^* \succ_w^* u$ (weak reduction) for some term $u =_i t_2^*\,\pi_2^*$.*

7 The negative interpretation of witness extraction

Let us now reinterpret the witness extraction method described in section 4 through the negative translation defined in section 6.

For that, consider a λ_c-term t_0 such that $\vdash_{\mathrm{NK}} t_0 : \exists^{\mathbb{N}} x\, \mathrm{null}(f(x))$, where $\exists^{\mathbb{N}} x\, \mathrm{null}(f(x))$ is a shorthand for $\forall Z\,(\forall x\,(\{x\} \Rightarrow \mathrm{null}(f(x)) \Rightarrow Z) \Rightarrow Z)$. Let d_f be the decision function for the predicate $\mathrm{null}(f(x))$ introduced in section 4, and write $u_f \equiv \lambda xy\,.\,d_f\,x\,(\mathsf{stop}\,x)\,y$ (where 'stop' is an instruction with no evaluation rule) and $p_0 \equiv t_0 \star u_f \cdot \alpha$ (where α is a stack constant). From the discussion of section 4, we know that the process p_0 evaluates to $\mathsf{stop} \star \hat{n} \cdot \alpha$ for some $n \in \mathbb{N}$ such that $f(n) = 0$.

Via the negative translation we have by Prop. 9:

$$\vdash_{\mathrm{NJ}}\ t_0^*\ :\ \forall Z\left(\forall x\left(\mathrm{nat}(x) \wedge (\mathrm{null}(h(f(x))) \Rightarrow R) \wedge Z \Rightarrow R\right)\ \wedge\ Z\ \Rightarrow\ R\right)$$

The crucial point is the following:

Proposition 11. — *In the intuitionistic realizability model:*

$$d_f^*\ \Vdash_{\mathrm{NJ}}\ \forall x\left(\mathrm{nat}(x) \wedge \left(\mathrm{null}(f(x)) \Rightarrow R\right) \wedge \left(\mathrm{null}(h(f(x))) \Rightarrow R\right) \wedge \top \Rightarrow R\right)$$

(independently from the choice of R).

To type-check the term $u_f \equiv \lambda xy\,.\,d_f\,x\,(\mathsf{stop}\,x)\,y$ through the negative translation, let us now fix the pole by setting $R \equiv \exists x\,(\mathrm{nat}(x) \wedge \mathrm{null}(f(x)))$, while defining the translation of the instruction stop as $(\mathsf{stop})^* \equiv \lambda z\,.\,z$. With this implementation of stop^* we clearly have

$$\vdash_{\mathrm{NJ}} \ \mathsf{stop}^* \ : \ \forall x\,\big(\mathrm{nat}(x) \wedge \mathrm{null}(f(x)) \Rightarrow R\big)$$

hence (combining the latter with Prop. 11 using adequacy)

$$u_f^* \ \Vdash_{\mathrm{NJ}} \ \forall x\,\Big(\mathrm{nat}(x) \wedge \big(\mathrm{null}(h(f(x))) \Rightarrow R\big) \wedge \top \Rightarrow R\Big)\,,$$

from which we deduce that

$$p_0^* \ \Vdash_{\mathrm{NJ}} \ R \ \equiv \ \exists x\,\big(\mathrm{nat}(x) \wedge \mathrm{null}(f(x))\big)\,.$$

We have thus shown that through the negative translation, the transformation of the classical proof t_0 into the process p_0 is nothing but the transformation of a classical proof of a Σ_1^0-formula into an intuitionistic realizer of the same formula, thus giving a constructive explanation why the procedure described in section 4 successfully extracts a reliable witness in finite time.

Of course, the point here is that through the negative interpretation, the transformation of the classical proof t_0 into the process p_0 exactly follows the well-known method (due to Friedman [2]) to transform a classical proof of a Σ_1^0-formula into an intuitionistic proof of the same formula.

References

1. H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and The Foundations of Mathematics*. North-Holland, 1984.
2. H. Friedman. Classically and intuitionistically provably recursive functions. *Higher Set Theory*, 669:21–28, 1978.
3. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.
4. J.-L. Krivine. A general storage theorem for integers in call-by-name lambda-calculus. *Th. Comp. Sc.*, 129:79–94, 1994.
5. J.-L. Krivine. Typed lambda-calculus in classical Zermelo-Fraenkel set theory. *Arch. Math. Log.*, 40(3):189–205, 2001.
6. J.-L. Krivine. Dependent choice, ‘quote’ and the clock. *Th. Comp. Sc.*, 308:259–276, 2003.
7. J.-L. Krivine. Realizability in classical logic. Unpublished lecture notes (available on the author’s web page), 2005.
8. A. Miquel. Classical program extraction in the calculus of constructions. In J. Duparc and T. A. Henzinger, editors, *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 2007.
9. P. Oliva and T. Streicher. On Krivine’s realizability interpretation of classical second-order arithmetic. *Fundam. Inform.*, 84(2):207–220, 2008.
10. The Coq Development Team (LogiCal Project). The Coq Proof Assistant Reference Manual – Version 8.1. Technical report, INRIA, 2006.