Realizability of the axiom of choice in HOL. (An analysis of Krivine's work)

Christophe Raffalli, Frédéric Ruyer

Realizability of the axiom of choice in HOL.
(An analysis of Krivine's work)

Christophe Raffalli[*]
Frédéric Ruyer [†‡]

March 14, 2007

Abstract

This paper is an introduction to recent works in realizability, mainly Krivine's work to realize the dependent choice axiom. We also show how to improve programs extracted from classical proofs by distinguishing formulas with and without algorithmic contents.

1 Introduction

This work gives a way of realizing the axiom of choice (AC) in higher order logic (HOL). It follows from an idea of J-L Krivine developed in [7], consisting in introducing a new constant in the λC-calculus which could be interpreted with a "clock".

However, the theorem AC_{clock} which is realized by this clock is not the usual axiom of choice. Nevertheless, it implies the dependant axiom of choice AC_D (and in fact the non extensional axiom of choice AC_{NE}) and we will give and study the behaviour of the program extracted from the proof of AC_{NE} using the AC_{clock} axiom realized by a clock instruction.

To get more readable terms, we also use a part of the formalism of system ST developed in [13, 15, 16] inspired by the work of Michel Parigot [11], Christine Paulin [12], Catherine Parent [10], Stephano Berardi [1] and Philippe Curmin [4] which allows to remove computationally useless parts of a proof. Our first goal was to "prune" the proof given in the original paper [7], for example by removing parts concerning comparison of integers, or equality tests. But in the paper to appear [8] the author gives almost the same result as us. What is different in our approach is that we address more strictly the gap between synctactic typability ($\vdash t : A$) and semantic realizability ($t \Vdash A$), using propositions without algorithmic contents in the type syntax thanks to the special arrow $\rightharpoonup$.

---

[*]email: christophe.raffalli@univ-savoie.fr

[†]email: frederic.ruyer@univ-savoie.fr

[‡]address: Laboratoire de Mathématique, Université de Savoie, 73376 Le Bourget-du-Lac cedex (FRANCE)

For example, it allows us to avoid the uncomfortable notations like $\cap_{m<n}(\{\underline{m}\} \rightarrow \ldots$ which are replaced by $\forall m(m < n \rightharpoonup Nm \rightarrow \ldots$ We also can give a type to the term $Comp$ (see fact 3), whereas in [8] it is used only on the semantic side.

Remark: All the terms given in this article have been produced by the PhoX proof assistant developed by C.Raffalli [14].

2 λC-calculus

We use the same formalism as in [7], stacks allowing to handle easily lists of arguments, although it adds some steps of reduction.

2.1   Terms and stacks

We suppose given a denumerable set $\mathcal{V}$ of variables, and one constant : C. The sets Λ_c and Π are defined by mutual induction as follows:

- $\Lambda_c = \mathcal{V} \bigcup \{C\} \bigcup \{\lambda x.t; t \in \Lambda_c\} \bigcup \{(tu); t, u \in \Lambda_c\} \bigcup \{k_\pi; \pi \in \Pi\}$

- $\Pi = \{\epsilon\} \bigcup \{t.\pi; t \in \Lambda_c, \pi \in \Pi\}$

2.2   Execution of processes and weak head reduction

The set of *processes* is defined as $\mathbb{P} = \Lambda_c \times \Pi$; we write $t \star \pi$ for (t, π). There are four rules of reduction:

1. (push) $(tu) \star \pi \succ t \star u.\pi$

2. (pop) $\lambda x.t \star u.\pi \succ t[x := u] \star \pi$

3. (store) $C \star t.\pi \succ t \star k_\pi.\pi$

4. (restore) $k_\pi \star t.\pi' \succ t \star \pi$

This formalism has one drawback: it increases the number of steps. Effectively, we have usually $(\lambda x.u\, v)v_1 \cdots v_n \succ_{w.h.r} u[x := v]v_1 \cdots v_n$, and it is here simulated by $(\lambda x.u\, v)v_1 \cdots v_n \star \pi \succ^{n+2} u[x := v] \star v_1 \cdots v_n.\pi$. But we will see soon that it gives an elegant way of defining the value of a formula.

2.3   Evaluable processes

We say that a set of process $P \subset \mathbb{P}$ is *saturated* if it is closed by anti-reduction:

$$\forall p \in P, q \succ p \text{ implies } q \in P$$

Suppose we have defined a set $\perp\!\!\!\perp$ of *evaluable processes*, which is saturated. If $S \subset \Pi$, we note $S^\perp = \{t \in \Lambda_c; \forall \pi \in S, t \star \pi \in \perp\!\!\!\perp\}$. Every $L \subset \Lambda_c$ such that there is an $S \subset \Pi$ such that $L = S^\perp$ is called a *truth value*. This means we call

truth value the orthogonal of a set of stacks. Sets of stacks themselves could be considered as counter proofs or *falsity values*.

It is clear that $.^\perp$ is decreasing, that is:

$$S \subset S' \implies S'^\perp \subset S^\perp$$

Therefore, $\Pi^\perp$ is the least truth value, and $\emptyset^\perp = \Lambda_c$ the largest one.

The semantics we will define for HOL will always be (implicitly) parametrised by the choice of a saturated set $\perp\!\!\!\perp$ of *evaluable processes*. Moreover, some specific results may depend upon specific properties of this set.

We now end this section by one last definition (which is used for the interpretation of implication): if Φ is a set of λ-terms and S is a set of stacks then:

$$\Phi.S = \{t.\pi; t \in \Phi \text{ and } \pi \in S\}$$

3 HOL with propositions and types

Here we define HOL [2], with two sorts of formulas:

- Propositions: which are formulas without algorithmic content. Their interpretation will be either true or false.

- Types: which are formulas with algorithmic content. They will be interpreted by set of programs of the form $S^\perp$ in order to be able to realize Peirce's law.

In fact this will be two copies of the same logic (except that we will extract programs only for one of them) which interact only through one special implication written $\rightharpoonup$ with a proposition on the left and a type on the right.

The interpretation of $P \rightharpoonup A$ will be the interpretation of A when P is true and the largest possible interpretation $\Lambda_c = \emptyset^\perp$ when P is false. This connective is similar to parigots $A \upharpoonright P$ in [11] which is a conjunction instead of an implication.

3.1 Sorts, terms and formulas

The set of sorts $\mathfrak{S}$ is defined by the grammar:

$$\mathfrak{S} = \mathfrak{V} | \mathfrak{C} | \mathfrak{S} \rightarrow \mathfrak{S}$$

$\mathfrak{V}$ is a denumerable set of *variables of sorts*, and $\mathfrak{C}$ is a set of *sorts constants*. In the rest of the paper, we will suppose that $\{\tau; o; \omega\} \subset \mathfrak{C}$, τ denoting the sort of *types*, o denoting the sort of *propositions* and ω denoting the sort of *natural numbers*.

We associate to each sort $\mathfrak{s}$ a denumerable set of variables $V_\mathfrak{s}$, whose elements are noted $x^\mathfrak{s}$ and define the set of terms T as the set of simply-typed λ-terms using the set of sorts $\mathfrak{S}$ as simple types and using at least the following constants with their given sorts:

$$\left\{ \begin{array}{lllll} 0 & : & \omega; & s & : & \omega \to \omega; \\ \to & : & \tau \to \tau \to \tau; & \to & : & o \to o \to o; \\ \forall^{\mathfrak{s}} & : & (\mathfrak{s} \to \tau) \to \tau; & \forall^{\mathfrak{s}} & : & (\mathfrak{s} \to o) \to o \\ & & \multicolumn{3}{c}{\rightharpoonup: o \to \tau \to \tau} \end{array} \right\}$$

There are two arrows that we write in the same way but which have distinct sorts. There are also two polymorphic universal quantifiers. The context and the following convention will avoid confusion:

We will use the capital letters A, B, C for types (that is expressions of sort τ) and the letter P, Q, R for propositions that is expressions of sort o. Therefore, if we write $A \to B$ we know that the arrow here is of sort $\tau \to \tau \to \tau$.

We note as usual $\forall x\, A$ instead of $\forall(\lambda x.A)$, $A \to B$ for $(\to A\, B)$ and $\forall x{:}A\, B$ for $\forall x(A(x) \to B)$. We use these notations both for types and propositions.

3.2 Deduction rules

A context Γ is a finite set of propositions of sort o and pairs (x, A), x being a λ-variable and A a type of sort τ. We note $x : A$ instead of (x, A). We write a context:

$$\Gamma = P_1, \cdots P_n, x_1 : A_1, \cdots x_m : A_m$$

In proofs, we will manipulate two kinds of sequents: $\Gamma \vdash t : A$ when we prove a type A (here t is a λ-term) and $\Gamma \vdash P$ when P is a proposition.

The rules for HOL with propositions and types are given in figure 1.

Lemma 1 *If σ is a substitution on variables, we note $A[\sigma]$ (resp. $\Gamma[\sigma]$) the result of the substitution on the formula A(resp. the context Γ).*
If $\Gamma \vdash A$ then $\Gamma[\sigma] \vdash A[\sigma]$

Proof: Immediate induction on the size of the derivation. $\qquad\square$

Remark: The semantics of the new connective $\rightharpoonup$ is reflected by its rules, because there is nothing to prove when P is false in the introduction rule. However, one could consider a rule saying $\Gamma \vdash \neg P$ implies $\Gamma \vdash t : P \rightharpoonup A$ for any term t. This rule is not derivable because in a derivation the free variables of a term are always declared in the context.

4 Values and realizability

4.1 Valuation in a model

A *model* $\mathcal{M}$ is a triple $(|\mathcal{M}|, \mathfrak{s} \mapsto \bar{\mathfrak{s}}, \bot\!\!\!\bot)$ such that

- $|\mathcal{M}|$ is a set of sets with the following properties : if $m \in |\mathcal{M}|$ and $n \in |\mathcal{M}|$ then $n \to m \in |\mathcal{M}|$ (here $n \to m$ is the set of all functions from n to m).

- The function $\mathfrak{s} \to \bar{\mathfrak{s}}$ has the following properties:

Axiom : $$\overline{\Gamma, x : A \vdash x : A}$$	Axiom' : $$\overline{\Gamma, P \vdash P}$$
$\rightarrow$ - Intro : $$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \rightarrow B}$$	$\rightarrow'$ - Intro : $$\frac{\Gamma, P \vdash Q}{\Gamma \vdash P \rightarrow Q}$$
$\rightarrow$ - Elim : $$\frac{\Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash (tu) : B}$$	$\rightarrow'$ - Elim : $$\frac{\Gamma \vdash P \rightarrow Q \quad \Gamma \vdash P}{\Gamma \vdash Q}$$
$\forall$ - Intro : $$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall x A} \text{ If } x \text{ not free in } \Gamma.$$	$\forall'$ - Intro : $$\frac{\Gamma \vdash P}{\Gamma \vdash \forall x P} \text{ If } x \text{ not free in } \Gamma.$$
$\forall$ - Elim $$\frac{\Gamma \vdash t : \forall x^{\mathfrak{s}} A}{\Gamma \vdash t : A[x := v]} \text{ For every } v \text{ of sort } \mathfrak{s}.$$	$\forall'$ - Elim : $$\frac{\Gamma \vdash \forall x^{\mathfrak{s}} P}{\Gamma \vdash P[x := v]} \text{ For every } v \text{ of sort } \mathfrak{s}.$$
$\rightharpoonup$- Intro : $$\frac{\Gamma, P \vdash t : A}{\Gamma \vdash t : P \rightharpoonup A}$$	$\rightharpoonup$- Elim : $$\frac{\Gamma \vdash t : P \rightharpoonup A \quad \Gamma \vdash P}{\Gamma \vdash t : A}$$
Peirce law : $$\overline{\Gamma \vdash C : (((A \rightarrow B) \rightarrow A) \rightarrow A)}$$	Peirce law' : $$\overline{\Gamma \vdash (((P \rightarrow Q) \rightarrow P) \rightarrow P)}$$

Figure 1: HOL rules with propositions and types

- for every $\mathfrak{s} \in \mathfrak{V} \cup \mathfrak{C}$, $\bar{\mathfrak{s}} \in \mathcal{M}$.
- $\overline{\mathfrak{s} \to \mathfrak{s}'} = \bar{\mathfrak{s}} \to \bar{\mathfrak{s}'}$. Remark: we should allow for *non full model* to allow completeness result even for realizability (see [5, 5]), but this is not needed important here.

We will assume that $\bar{\omega}$ is the set of natural numbers $\mathbb{N}$, $\bar{\tau} = \mathcal{P}(\Pi)$ and $\bar{o} = \{0; 1\}$.

- $\perp\!\!\!\perp$ is a saturated set of processes.

Once a *model* $\mathcal{M}$ is defined, we can build the set of terms with *parameters* in $\mathcal{M}$: we just need to consider that $\bigcup_{\mathfrak{s} \in \mathfrak{S}} \{\underline{a}; a \in \bar{\mathfrak{s}}\}$ is a subset of C (the set of constants), and that each of these constants has the intended sort (or sorts, because the same element may appear in the interpretation of more that one sort) : $a \in \bar{\mathfrak{s}}$, then $\underline{a} : \mathfrak{s}$.

Now, we define the notion of interpretation (or value) of a closed term with parameters in the model. We write it $\|.\|^{\mathcal{M}}$ and it is defined by induction as follows:

- for every a in $|\mathcal{M}|$, $\|\underline{a}\| = a$

- $\|0^{\omega}\| = 0$ and $\|s^{\omega \to \omega}\| = x \mapsto x + 1$.

- $\| \to^{\tau \to \tau \to \tau} \| = A \mapsto B \mapsto (A^{\perp}.B)$.

- $\| \to^{o \to o \to o} \| = P \mapsto Q \mapsto$ if $P = 1$ then Q else 0.

- $\| \rightharpoonup \| = P \mapsto A \mapsto$ if $P = 1$ then A else $\emptyset$.

- $\|\forall^{(\mathfrak{s} \to \tau) \to \tau}\| = A \mapsto \bigcup_{x \in \bar{\mathfrak{s}}} A(x)$.

- $\|\forall^{(\mathfrak{s} \to o) \to o}\| = P \mapsto$ if (for all x in $\bar{\mathfrak{s}}$, $P(x) = 1$) then 1 else 0.

- $\|(t)u\| = \|t\|(\|u\|)$.

- $\|\lambda x.t\| = a \mapsto \|t[x := \underline{a}]\|$.

- for every other constant of sort s, $c^{\mathfrak{s}}$ in C, $\|c^{\mathfrak{s}}\| \in \bar{\mathfrak{s}}$ should be given with the definition of the model.

We will need a little lemma on substitutions:

Lemma 2 *For all terms A and v, if x is a F.V. of A, then $\|A[x := v]\| = \|A[x := \underline{\|v\|}]\|$.*

Proof: It is an easy proof on the structure of the term A. $\qquad\square$

When the context is clear, to lighten the writing, we will often allow us not to underline a parameter and not to write the model as exponent.

The *truth value* of a type A (of sort τ), written $|A|$ is $\|A\|^{\perp}$.

<u>Remark</u> : $|\forall X\, X|$ is the least truth value.

We now give some comments about the choices in the way the semantics is presented:

- We choose not to interpret the terms with free variables. In fact there are two equivalent alternatives: use terms with parameters in the model and substitute all variables by terms before computing the interpretation, or parametrise the valuation with a function giving the value of the variables.

- For a type, as we said, $\|A\|$ is not the truth value but a set of counter proofs. This explains why the definition are the dual of the natural definition for truth value. The truth value of a predicate is $|A| = \|A\|^{\perp}$. This is because we choose $\overline{\tau} = \mathcal{P}(\Pi)$.

  It would have been equivalent to take for $\overline{\tau}$ the set of sets of terms which are the dual of a set of stacks. This would require to change a bit the interpretation of the connectives (for instance $\forall$ becomes an intersection). This alternative gives a definition which is the same as the one used for intuitionist system F or AF2 [9], but requires a few easy lemmas in the classical case to prove that the interpretation of the connectives preserves the property of being a dual of a set of stacks.

  These, notions are also related to game semantics, as explained in the work of Coquand [3]: intuitively, a stack might be seen as the environment, and a term as a program that will give an answer to the environment, i.e. "sending" the environment in $\bot\!\!\!\bot$. For the reader, things might become clearer after reading the interpretation of the term that computes the minimum of a predicate (see theorem 2 below), the semantic interpretation of its type (together with the behaviour of the program) allowing to describe the play between $\exists$loïse and $\forall$belard (see also the work of Hayashi [6] for a more detailed presentation of the subject).

4.2 Adequation lemma

We say that a term t *realizes* a type A in a model $\mathcal{M}$ if $t \in |A|^{\mathcal{M}}$, and we write $\mathcal{M} \vDash t \Vdash A$ (or $t \Vdash A$ if the context is clear enough).

We say that a formula without algorithmic content P *is satisfied* in $\mathcal{M}$, and we write $\mathcal{M} \vDash P$ if $\|P\| = 1$.

We say that a substitution $\sigma = [x_1 := t_1, \dots, x_m = t_m]$ realizes a context $\Gamma = P_1 \cdots P_n, x_1 : A_1 \cdots x_m : A_m$ in a model $\mathcal{M}$ if and only if:

- for all $1 \leq i \leq m$ and $\mathcal{M} \vDash t_i \Vdash A_i$

- and for all $1 \leq i \leq n$, $\mathcal{M} \vDash P_i$

We write this $\mathcal{M} \vDash \sigma \Vdash \Gamma$ (or $\sigma \Vdash \Gamma$ if the context is clear enough).

Our aim is to show the following theorem:

Theorem 1 *(adequation lemma) Let us fix a model $\mathcal{M}$. For all context Γ, if $\sigma \Vdash \Gamma$ and $\Gamma \vdash t : A$ then $t\sigma \Vdash A$.*

We first show a lemma about propositions:

Lemma 3 *For all contexts Γ, if $\sigma \Vdash \Gamma$ and $\Gamma \vdash P$ then $\mathcal{M} \vDash P$.*

Proof: We make the proof by induction on the length of the deduction.

Axiom'
$$\Gamma = P_1 \cdots P_n, x_1 : A_1 \cdots x_m : A_m \vdash P_i$$

trivial because $\mathcal{M} \vDash P_j$ for all $1 \leq j \leq n$.

$\rightarrow'$ - Intro
$$\frac{\Gamma, P \vdash Q}{\Gamma \vdash P \rightarrow Q}$$

We suppose that $\mathcal{M} \vDash P_i$ for all $1 \leq i \leq n$. We have $\|P \rightarrow Q\| = 1$ if and only if $\|P\| = 0$ of $\|Q\| = 1$. Therefore, we assume $\|P\| = 1$, and by induction hypothesis, we have $\mathcal{M} \vDash Q$ which means $\|Q\| = 1$.

$\rightarrow'$ - Elim
$$\frac{\Gamma \vdash P \rightarrow Q \quad \Gamma \vdash P}{\Gamma \vdash B'}$$

We suppose that $\mathcal{M} \vDash P_i$ for all P_i in Γ. By I.H., $\|P\| = 1$ and $\|P \rightarrow Q\| = 1$. Thus, $\|Q\| = 1$.

$\forall'$ - Intro
$$\frac{\Gamma \vdash P}{\Gamma \vdash \forall x^{\mathfrak{s}} P}$$

with x not free in Γ. We need lemma 1. It is here easy to conclude with I.H. and the interpretation of $\forall$.

$\forall'$ - Elim
$$\frac{\Gamma \vdash \forall x^{\mathfrak{s}} P}{\Gamma \vdash P[x := u]}$$

For every u of sort $\mathfrak{s}$. Immediate with the interpretation of $\forall$.

$\square$

Proof: We can now prove the adequation lemma by induction on the length of the deduction.

Axiom
$$\Gamma = P_1 \cdots P_n, x_1 : A_1 \cdots x_m : A_m \vdash x_i : A_i$$

trivial because $t\sigma = t_i$.

→ - Intro

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}$$

We have to show that $(\lambda x.t)\sigma \in |A \to B|$. We have $(\lambda x.t)\sigma = \lambda x.(t\sigma)$ because we can rename the bound variable x to avoid capture. Take π in $\|A \to B\|$, by the interpretation of $\to$, there is t_A in $|A|$ an π_B in $\|B\|$ such that $\pi = t_A.\pi_B$. We have:

$$(\lambda x.t)\sigma \star \pi \succ t\sigma[x := t_A] \star \pi_B$$

By I.H. , $t\sigma[x := t_A]$ belongs to $|B|$ (because $\sigma[x := t_A] \Vdash \Gamma, x : A$). Therefore, $t\sigma[x := t_A] \star \pi_B \in \bot\!\!\!\bot$, so $(\lambda x.t)\sigma \star \pi$ too by saturation.

→ - Elim

$$\frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash (tu) : B}$$

We have $(tu)\sigma = (t\sigma\, u\sigma)$. Take π in $\|B\|$.

$$(tu)\sigma \star \pi \succ t\sigma \star u\sigma.\pi$$

By I.H., we have $u\sigma \in |A|$ and $t\sigma \in |A \to B|$, then it is clear that $t\sigma \star u\sigma.\pi \in \bot\!\!\!\bot$ by definition of the interpretation of $\to$, and $(t\sigma\, u\sigma) \star \pi$ too by saturation.

⇀- Intro

$$\frac{\Gamma, A \vdash t : B}{\Gamma \vdash t : A \rightharpoonup B}$$

Suppose that $\mathcal{M} \vDash A$. By the interpretation of $\rightharpoonup$, we have $\|A \rightharpoonup B\| = \|B\|$ and by I.H. , we have $t\sigma \in |B|$. It is OK.
Else, we have $\|A \rightharpoonup B\| = \emptyset$ and $|A \rightharpoonup B| = \Lambda_c$ and we don't need any I.H. to conclude.

⇀- Elim

$$\frac{\Gamma \vdash t : A \rightharpoonup B \quad \Gamma \vdash A}{\Gamma \vdash t : B}$$

By the previous lemma 3, we have $\mathcal{M} \vDash A$, and thus $\|A \rightharpoonup B\| = \|B\|$. By I.H., $t\sigma \in |A \rightharpoonup B|$, and so $t\sigma \in |B|$.

Peirce law

$$\Gamma \vdash C : \forall A \forall B(((A \to B) \to A) \to A)$$

Take A and B in the model, $\pi \in \|A\|$ and $t \in |((A \to B) \to A)|$. We have

$$Ct \star \pi \succ t \star k_\pi.\pi$$

We just have to check that $k_\pi \in |A \to B|$. Take $\pi' \in \|A \to B\|$; by the interpretation of $\to$, there is t_A in $|A|$ an π_B in $\|B\|$ such that $\pi' = t_A.\pi_B$.

$$k_\pi \star \pi' = k_\pi \star t_A.\pi_B \succ t_A \star \pi$$

The conclusion is immediate.

∀ - Intro

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall x A}$$

with x not free in Γ. Take $\pi \in \|\forall x A\| = \|\forall \lambda x^{\mathfrak{s}}.A\|$. There is an a in $\overline{\mathfrak{s}}$ such that $\pi \in \|\lambda x^{\mathfrak{s}}.A\|(a) = \|A[x := \underline{a}]\|$. By lemma 1, and the I.H. , we have $t\sigma \in \|A[x := \underline{a}]\|$.

∀ - Elim

$$\frac{\Gamma \vdash t : \forall x^{\mathfrak{s}} A}{\Gamma \vdash t : A[x := v]}$$

Take v of sort $\mathfrak{s}$. We have to check that $t\sigma \in A[x := v]$. It is easy because $\|A[x := v]\| = \|A[x := \underline{\|v\|}]\| = \|\lambda x.A\|(\underline{\|v\|})$, which is a subset of $\|\forall x^{\mathfrak{s}} A\|$. Therefore, $|\forall x^{\mathfrak{s}} A|$ is a subset of $|A[x := v]|$, and the conclusion is immediate from the I.H.

$\square$

5 Conjunction and existential

In this section, to ease reading, we really start to use notation introduced at the end of section 3.1.

We define the type $(A \times B) = \forall K ((A \to B \to K) \to K)$ and the three terms $\mathtt{pair}[a, b] = \lambda x\ (x\ a\ b)$, $\mathtt{pi1}[c] = (c\ \lambda a\ \lambda b\ a)$, and $\mathtt{pi2}[c] = (c\ \lambda a\ \lambda b\ b)$.

Fact 1 *We extract the following constructor and destructors for pairs:*

- *If $\Gamma \vdash a : A$ and $\Gamma \vdash b : B$ then $\Gamma \vdash \boldsymbol{pair}[a, b] : A \times B$.*

- *If $\Gamma \vdash c : A \times B$ then $\Gamma \vdash \boldsymbol{pi1}[c] : A$ and $\Gamma \vdash \boldsymbol{pi2}[c] : B$*

We define the type $\exists x\, A(x) = \forall K (\forall x{:}A\ K \to K)$ and the λ-term $\mathtt{box}[a] = \lambda x\ (x\ a)$.

Fact 2 *We extract the following constructor and destructor for existential:*

- *If $\Gamma \vdash a : A(x)$ then $\Gamma \vdash \boldsymbol{box}[a] : \exists x\, A(x)$.*

- *If $\Gamma \vdash b : \exists x\, A(x)$ and $\Gamma \vdash f : \forall x{:}A\ B$ then $\Gamma \vdash b\ f : B$.*

We now have a conjunction $\times$ and an existential quantifier ... however, very often, to simplify the extracted program and to deal with conjunction mixing types and propositions, we will use specific coding. For instance, if we want to write an existential on two type Ax, Bx and a proposition Px, we will use the formula:

$$\forall K (\forall x{:}A\ (B(x) \to P(x) \rightharpoonup K) \to K).$$

5.1 Leibniz equality and extensionality

The equality can be defined both as a type or a proposition, using Leibniz definition. One remarkable point, is that the interpretation of the equality type can only be two values (as for propositions). This fact will be essential to realize axioms involving equality like $\forall x(0 \neq sx)$.

The type and proposition $a = b$ are defined by $\forall X(Xa \rightarrow Xb)$ (this is two distinct definitions, but we only write one, since we use the same notation for both implication and quantification).

Lemma 4 *When $a = b$ is a proposition, $\|a = b\| = 1$ if and only if $\|a\| = \|b\|$.*

Proof: Recall that $\|Xa\| = \|X\|(\|a\|)$ and choose for $\|X\|$ the characteristic function of $\{\|a\|\}$ when $\|a\| \neq \|b\|$. □

Corollary 1 *The following rules preserve the adequation lemma (and will simplify extracted λ-terms):*

Equality	Equality'
$\dfrac{\Gamma \vdash t : A[x := a] \quad \Gamma \vdash a = b}{\Gamma \vdash t : A[x := b]}$	$\dfrac{\Gamma \vdash P[x := a] \quad \Gamma \vdash a = b}{\Gamma \vdash P[x := b]}$
Extensionality $\dfrac{\Gamma \vdash P \leftrightarrow Q}{\Gamma \vdash P = Q}$	

where the formula $P \leftrightarrow Q := (P \rightarrow Q) \wedge (Q \rightarrow P)$.

Proof: Immediate from the previous lemma. One just have to notice that $\|P[x := a]\| = \|P[x := \underline{\|a\|}]\|$ and $\mathcal{M} \vDash P \leftrightarrow Q$ implies $\|P\| = \|Q\|$. □

Corollary 2 *For any model and any terms $t_1, \ldots, t_p$ (possibly with x free in t_i or $p = 0$), we have $\lambda x (x\, t_1\, \ldots\, t_p) \Vdash \forall n(0 \neq sn)$ and $\lambda x\, x \Vdash \forall n, m(sn = sm \rightarrow n = m)$ (here $x \neq y := x = y \rightarrow \forall X\, X$). This means that the axioms $\forall n(0 \neq sn)$ $\forall n, m(sn = sm \rightarrow n = m)$ may be used both at the level of propositions and type while preserving the adequation lemma.*

Proof: Let us choose a model $\mathcal{M}$. We use that fact that in $\mathcal{M}$ the sort ω is interpreted by natural numbers in the standard way. Therefore, we have $\|0\| \neq \|s\underline{a}\|$ and $\|s\underline{a}\| = \|s\underline{b}\|$ implies $a = b$.

For the first result, we take $n \in \omega$ and $u \in |(0 = sn)|$ and $\pi \in \|\forall X\, X\| = \Pi$ and we have to show that $\lambda x (x\, t_1\, \ldots\, t_p) \star u.\pi \in \bot\!\!\!\bot$. We have $\lambda x (x\, t_1\, \ldots\, t_p) \star u.\pi \succ u \star t'_1\, \ldots\, t'_p.\pi$ and $t'_1\, \ldots\, t'_p.\pi \in \|\forall n(0 = sn)\|$ by the previous lemma. This gives the wanted result because $u \in |\forall n(0 = sn)|$.

The second result is immediate because the previous lemma and the choice of models implies $\|s\underline{a} = s\underline{b}\| = \|\underline{a} = \underline{b}\|$. □

Important remark: the previous rules allow to deduce $\Gamma \vdash \lambda x\, x : a = b$ from $\Gamma \vdash a = b$. This means that the equality as a proposition implies the equality as a type. The converse is not true, because sometimes, an equality proof may contain a k_π constant that will arrive in head position. This means an equality proof may *raise an exception* and in this case we can not derive the corresponding propositional equality.

Remark: all the previous results are even simpler (like in recent Krivine's work [8]) if one consider a new type predicate $x \neq y$ with the interpretation $\|\underline{a} \neq \underline{b}\| = \Pi$ when $a \neq b$ and $\|\underline{a} \neq \underline{b}\| = \emptyset$ when $a = b$. Then, $t \Vdash \forall n(0 \neq sn)$ for any term and we can realize the equivalence between Leibniz equality and the negation of our new type predicate $a \neq b$.

6 Some results on integers

In this section, we give some more or less well-known results on integers. We note, for $n \in \mathbb{N}$, n^C the Church integer $\lambda x.\lambda f.f^n x$.

6.1 Recursion principle and induction principle

We define the formulas $N(x) = \forall X\, (X(0) \to \forall y{:}X\, X(\mathrm{S}(y)) \to X(x))$, we add a new propositional predicate $a < b$ with the sort $\omega \to \omega \to o$ interpreted by the usual ordering on natural numbers (we could also define $a < b = \forall X\, (X(\mathtt{Succ}[a]) \to \forall n{:}X\, X(\mathtt{Succ}[n]) \to X(b))$, but it makes no difference to use a new constant with axioms or a definition, at the level of propositions, for extracted programs).

We also define the following λ-terms which are the two constructors on natural numbers, the *recursor* and a term $\mathtt{comp}$ that compares two integers and distinguish three cases (equal, less or greater):

- $\mathtt{Zero} = \lambda x\, \lambda f\, x$,

- $\mathtt{Succ}[n] = \lambda x\, \lambda f\, (f\ (n\ \ x\ \ f))$,

- $\mathtt{NRec}[t_0,\ t_S,\ n] =$

    $\mathtt{pi2}[n\ \mathtt{pair}[\mathtt{Zero}, t_0]\ \ \lambda p\, \mathtt{pair}[\mathtt{Succ}[\mathtt{pi1}[p]], t_S\ \mathtt{pi1}[p]\ \ \mathtt{pi2}[p]]]$

- $\mathtt{comp} = \lambda n\, \mathtt{NRec}[\mathtt{comp1}, \mathtt{comp2}, n]$, with

    - $\mathtt{comp1} = \lambda p\ (p\ \ \lambda a\, \lambda b\, \lambda c\, a\ \ \lambda x\, \lambda a\, \lambda b\, \lambda c\, b)$ and
    - $\mathtt{comp2} = \lambda n\, \lambda r\, \lambda p\, \mathtt{NRec}[\lambda a\, \lambda b\, \lambda c\, c,\ \lambda p'\, \lambda r'\ (r\ \ p'),\ p]$.

Fact 3 *The previous programs were extracted from the following proofs:*

- *For all $n \in \mathbb{N}$, $\vdash n^C : N(\underline{n})$, and especially $\vdash \mathtt{Zero} : N(0)$.*

- $\vdash \lambda n\, \mathbf{\mathit{Succ}}[n] : \forall n(Nn \to Nsn)$.

- If $\vdash t_0 : X(0)$, $u : N(y), v : X(y) \vdash tS(u,v) : X(S(y))$ and $\vdash n : N(p)$ then $\vdash \textit{NRec}[t_0,\ t_S,\ n] : X(p)$.

- $\vdash \textit{comp} : \forall X \, \forall n, p{:}N \left(\begin{array}{l} (n = p \rightharpoonup X) \rightarrow ((n < p) \rightharpoonup X) \rightarrow \\ ((p < n) \rightharpoonup X) \rightarrow X \end{array} \right)$

Proof: The first three items are very usual.

For the last one let us explain the behaviour of the term: if we suppose that $n \Vdash Nn'$ and $p \Vdash Np'$ then $\texttt{comp}\ \ n\ \ p\ \ a\ \ b\ \ c$ reduces to a if $n = p$, b if $n < p$ and c if $p < n$.

If we did not use the special arrow $\rightharpoonup$ in the type of $\texttt{comp}$, then a, b and c would receive as argument the proof of the relation between n and p. We here see the purpose of mixing propositions and types.

Now, for the proof itself, it is a double recursion on n and p, using the extensionnality in the last case to be able to replace $Sn = Sp$ by $n = p$. $\qquad \square$

Let Y be a fixpoint operator i.e. such that for all π and f: $Y \star f.\pi \succ f \star (Yf).\pi$. We will see that the fact that $<$ is well founded is associated directly to a fixpoint combinator. We have the following fact:

Fact 4 $Y \Vdash \forall X \, (\forall a \, (\forall b{:}N \, ((b < a) \rightharpoonup X(b)) \rightarrow N(a) \rightarrow X(a)) \rightarrow \forall a{:}N \, X(a))$. *This means we can use this as axiom and keep the adequation lemma.*

Proof: Take X, and $f \Vdash \forall a \, (\forall b{:}N \, ((b < a) \rightharpoonup X(b)) \rightarrow N(a) \rightarrow X(a))$. We want to prove that for all $n \in \mathbb{N}$, if $t \Vdash N\underline{n}$ and $\pi \in \|X\underline{n}\|$, then $Y \star f.t.\pi \in \bot\!\!\!\bot$. We have:

$$Y \star f.t.\pi \succ f \star (Yf).t.\pi$$

We can assume as induction hypothesis that for all $m < n$, $s \Vdash N\underline{m}$ and $\pi' \in \|X\underline{m}\|$ we have $(Yf) \star s.\pi' \succ Y \star f.s.\pi' \in \bot\!\!\!\bot$. So, $(Yf) \Vdash (N\underline{m} \rightarrow \underline{m} < \underline{n} \mapsto X\underline{m})$, that is $(Yf) \Vdash \forall b(Nb \rightarrow b < \underline{n} \mapsto Xb)$. So, with our hypothesis on f, we have $f \star (Yf).t.\pi \in \bot\!\!\!\bot$. $\qquad \square$

6.2 Minimum principle

The formula No_lt expressing that no natural number less than n satisfies the predicate Q) is defined as:

$$\text{No\_lt}(n,\ Q) = \forall p{:}N \, (Q(p) \rightarrow (p < n) \rightharpoonup \forall X X)$$

We define three terms to prove the minimum principle:

- $\texttt{Min}_{\texttt{unique}} = \begin{array}{l} \lambda n \, \lambda n' \, \lambda q \, \lambda q' \, \lambda m \, \lambda m' \, \lambda e \\ \quad \left(\texttt{comp}\ \ n\ \ n'\ \ e\ \ \left(m'\ \ n\ \ q \right)\ \ \left(m\ \ n'\ \ q' \right) \right) \end{array}$

- $\texttt{Min}_{\texttt{fix}}[k] = (\text{Y}\ \ \lambda r \, \lambda a \, \lambda q \, \lambda f \, (f\ \ a\ \ q\ \ \lambda b \, \lambda q' \, (k\ \ (r\ \ b\ \ q'))))$

- $\texttt{Min}_{\texttt{exists}} = \lambda n \, \lambda q \, (C\ \ \lambda k \, (\texttt{Min}_{\texttt{fix}}[k]\ \ n\ \ q))$

Fact 5 *(Unicity of the minimum)*

$$\vdash Min_{unique} : \forall Q \, \forall n,p \begin{pmatrix} N(n) \rightarrow N(p) \rightarrow Q(n) \rightarrow Q(p) \rightarrow No\_lt(n, \ Q) \rightarrow \\ No\_lt(p, \ Q) \rightarrow \forall X \, ((n = p \rightharpoonup X) \rightarrow X) \end{pmatrix}$$

Proof: We apply the hypothesis $No\_lt(n, \ Q)$ and $No\_lt(p, \ Q)$ with respectively $N(n)$, $Q(n)$ and $N(p)$, $Q(p)$ to get then the premises of *comp* in Fact 3 ! $\square$

The behaviour of Min_{unique} is easy to understand with the help of comp.

Fact 6 *(existence of the minimum) Let us define the type that expresses that the minimum of the predicate Q exists:*

$$\exists_{Min} \ Q = \forall K \, (\forall n{:}N \, (Q(n) \rightarrow No\_lt(n, \ Q) \rightarrow K) \rightarrow K)$$

We have

$$\vdash Min_{exists} : \forall Q \, \forall n{:}N \, (Q(n) \rightarrow \exists_{Min} \ Q)$$

Proof: Choose an integer n with $n' : N(n)$, $q : Q(n)$. We use the Peirce law with $k : (\exists_{Min} \ Q) \rightarrow \forall X \, X$, and we have to deduce that $Min_{fix}[k] \ \ m \ \ q : \exists_{Min} \ Q$. Using fact 4, what we have to prove is that

$$\lambda r \, \lambda a \, \lambda q_0 \, \lambda f \, \big(f \ \ a \ \ q_0 \ \ \lambda b \, \lambda q' \, (k \ \ (r \ \ b \ \ q'))\big) :$$
$$\forall a \, (\forall b{:}N \, ((b < a) \rightharpoonup Q(b) \rightarrow \exists_{Min} \ Q) \rightarrow N(a) \rightarrow Q(a) \rightarrow \exists_{Min} \ Q)$$

Now, we have the hypothesis $r : \forall b{:}N \, ((b < a) \rightharpoonup Q(b) \rightarrow \exists_{Min} \ Q)$, $A : N(a)$ and $q' : Q(a)$, and we have to show:

$$\lambda f \, (f \ \ A \ \ q' \ \ \lambda b \, \lambda q_0' \, (k \ \ (r \ \ b \ \ q_0'))) : \exists_{Min} \ Q$$

We assume $f : \forall n_0{:}N \, (Q(n_0) \rightarrow No\_lt(n_0, \ Q) \rightarrow K)$

and we use it with a. We have to prove $\lambda b \, \lambda q_0' \, (k \ \ (r \ \ b \ \ q_0')) : No\_lt(a, \ Q)$ and that leads to deduce $k \ \ (r \ \ B \ \ q'') : \forall X X$ from $B : N(b)$ and $q'' : Q(b)$, and it is immediate using $k : (\exists_{Min} \ Q) \rightarrow \forall X X$ and $r : \forall b_0{:}N \, ((b_0 < a) \rightharpoonup Q(b_0) \rightarrow \exists_{Min} \ Q)$. $\square$

To understand how Min_{exists} works, we run the program:

$$Min_{exists} \star a.q.f.\pi \ \succ \ Min_{fix}[k_{f.\pi}] \star a.q.f.\pi$$
$$\succ \ f \star a.q.M.\pi \text{ with } M = \lambda b \, \lambda q' \, (Min_{fix}[k_{f.\pi}] \ \ b \ \ q')$$

If the above was typed, we have

$$a \in |N(n)|, \, q \in |Q(n)|, \, M \in |No\_lt(n, \ Q)| \, ,$$
$$f \in |\forall n{:}N \, (Q(n) \rightarrow No\_lt(n, \ Q) \rightarrow K)| \text{ and } \pi \in \|K\|.$$

Having f in head position with the stack $a.q.M.\pi$ means that the program *pretends* that n is the minimum.

To get a more precise understanding of how the program behaves, we introduce a new constant κ wich will play the role of the opponent (you may read [8]

for the relation between games theory and realizability). We suppose given a term s for the successor (e.g $\lambda n \texttt{Succ}[n]$), and $0 = \texttt{Zero}$. Assume Q is a decidable predicate (which makes κ possibly recursive), and the constant has the following behaviour:

$$\kappa \star s^n 0.q.\xi.\pi \succ \xi \star s^p 0.q'.\pi'$$

for a $p < n$ and a term $q' \Vdash Q(p)$ if there exists one. Else,

$$\kappa \star s^n 0.q.\xi.\pi \succ s^n 0 \star \epsilon$$

If we take a storage operator T as defined in [8], we have the following:

Theorem 2 *For every integer n and any program $\vdash q : Q(n)$, using a constant κ associated to Q as above, every reduction of $((\texttt{Min}_{\texttt{exists}})s^n 0)q \star T\kappa.\pi$ ends on the Church integer $\overline{p} \star \epsilon$ with p the least integer such that $Q(p)$ is inhabited.*

In fact, the proof does not depend on the specific behaviour of $\texttt{Min}_{\texttt{exists}}$, but only on its specification.

Proof: We take $\bot\!\!\!\bot = \{p$ such that every reduction of p ends on $k_n \star \epsilon\}$. We have to show that $((\texttt{Min}_{\texttt{exists}})s^n 0)q \star T\kappa.\pi \in \bot\!\!\!\bot$. By the adequation lemma, we have:

$$((\texttt{Min}_{\texttt{exists}})s^n 0)q \Vdash \forall x(Nx \to \text{No\_lt}(x, Q) \to \bot) \to \bot$$

By definition of $\Vdash$, we have to show that $T\kappa \Vdash \forall x(N(x) \to Q(x) \to \text{No\_lt}(x, Q) \to \bot)$. Take $n \in \mathbb{N}$; we must have:

$$T\kappa \Vdash N(n) \to Q(n) \to \text{No\_lt}(n, Q) \to \bot$$

By theorem 11 of [8] concerning the behaviour of T, it is sufficient to show that for every π, for every $q \Vdash Q(n)$, and $\xi \Vdash \text{No\_lt}(n, Q)$, $\kappa \star s^n 0.q.\xi.\pi \in \bot\!\!\!\bot$. Here, two things can happen:

- either n is the least integer satisfying Q, in this case the execution stops at the good state.

- either there is a $p < n$ and a $q' \Vdash Q(p)$. In this case, we have: $\kappa \star s^n 0.q.\xi.\pi \succ \xi \star s^p 0.q'.\pi'$ Now he have to show that $\xi \star s^p 0.q'.\pi' \in \bot\!\!\!\bot$. By the specification of ξ, we just have to check that $\pi' \in (p < n) \rightharpoonup \forall X X$ which is immediate by the semantics.

$$\square$$

We shall notice that the previous remark concerning the behaviour of $\texttt{Min}_{\texttt{exists}}$ shows us that the term keeps $T\kappa$ in the head of its stored continuation, which is coherent with the intuitive meaning of the program given thanks to the game-semantics style interpretation we gave of its type.

7 Application : realizing AC

The idea is to realize the following shape of the axiom of choice; it is divided into three parts describing the properties we attend of the choice constructor C:

- The first part of the specification expresses that if $F(x, X)$ is inhabited, so is $C(F, x, X)$.

- The second part expresses that C is functional, that is it gives only one candidate X for each choice of x (the formal definition of $Func$ will be given later).

- The third part expresses that if X is chosen by $C(F, x)$, then we have $F(x, X)$.

Thus, our formulation of NAC (the name is explain below) is the following:

$$\forall F \exists C \left(\begin{array}{c} \forall x \forall X (F(x, X) \rightarrow \exists X C(F, x, X)) \times Func(C(F)) \times \\ \forall x \forall X (C(F, x, X) \rightarrow F(x, X)) \end{array} \right)$$

Remark: the above formalisation is polymorphic. The variables x and X could have any sort s and s' and therefore, F could have any sort of the shape $s \rightarrow s' \rightarrow o$.

However, this axiom is not the standard axiom of choice. It is the non extensional axiom of choice (hence it's name NAC). Nevertheless, when the sort of x is ω this is really the countable axiom of choice, because C will really be a choice function with argument in ω.

The reason it is not the usual axiom of choice for sorts more complex that ω is the following: if the sort of x is $\omega \rightarrow o$, then one would also expect the following to hold for C:

$$\forall F, x, x', X, X'(\forall y, y'(x(y) \leftrightarrow x'(y')) \wedge C(F, x, X) \wedge C(F, x', X') \rightarrow X = X')$$

This formula means that C is extensional (in general, this formula is defined by induction on the sort of C), and we did not state this in our formulation, but only that C was functional which is much weaker.

7.1 A new constant

To realize the axiom of choice, we need to add a new constant to the λ-calculus and show that it realizes a specific axiom.

Let $F(x, X)$ be a formula with parameter F, x, X in a model $\mathcal{M}$, and ϕ a λ-term. Using the axiom of choice, there is a function Ψ such that

$$\phi \Vdash F(x, \Psi(F, x, \phi)) \text{ if and only if } \exists X \in \mathcal{M} \ \phi \Vdash F(x, X).$$

Ψ just needs to choose such an X. The problem is that Ψ is not in the model, since its third argument is a λ-term.

To solve this problem, we consider a surjection $n \mapsto t_n$ from $\mathbb{N}$ to Λ and one inverse of this surjection $t \mapsto n_t$. This means we choose a (non unique) index n_t for each λ-term t. We will use n_t as a natural number, but also as a Church numeral when using it in a λ-term. This means we can write $n_t \Vdash N(n_t)$.

Using this surjection, we define a function $\Phi(F, x, n) = \Psi(F, x, t_n)$. Now, Φ is in the model and

$$\phi \Vdash F(x, \Phi(F, x, n_\phi)) \text{ if and only if } \exists X \in \mathcal{M} \ \phi \Vdash F(x, X).$$

We define a new constant added to Λ, χ, with the following reduction rule.

$$\chi \star t.\phi.\pi \succ t \star n_\phi.\phi.\pi$$

Fact 7 *The following axiom preserves the adequation lemma.*

$$\vdash \chi : \forall K \ \forall F \ \forall x \ (\forall n{:}N \ (F(x, \Phi(F,x,n)) \to K) \to \forall X \ (F(x,X) \to K)).$$

Proof: Take K, F, x and x in the model and $u : \forall n{:}N \ (F(x, \Phi(F,x,n)) \to K)$ and $\phi : F(x, X)$. We just have to show $n_\phi.\phi.\Pi \in \|\forall n{:}N \ (F(x, \Phi(F,x,n)) \to K)\|$ which is immediate using the definition of Φ. $\square$

7.2 Realization of NAC

We will proceed with three lemmas:

We want to be able to choose X such that $F(x, X)$ if there exists one. To do so, we can take $\Phi(F,x,n)$ where n is the smallest integer satisfying this property.

We define the following predicate:

$$Z_1(F,x,n) = \text{No\_lt}(n, \ \lambda p F(x, \Phi(F,x,p)))$$

However, Z_1 describes an integer n, and we want to choose the second argument of F (i.e. $\Phi(F, x, n)$). For this, we define:

$$Z(F,x,X) = \forall K \left(\forall n{:}N \begin{pmatrix} F(x, \Phi(F,x,n)) \to \\ Z_1(F,x,n) \to \\ X = \Phi(F,x,n) \rightharpoonup K \end{pmatrix} \to K \right)$$

$Z(F,x,X)$ means that $X = \Phi(F,x,n)$ where n is the smallest integer such that $F(x, \Phi(F,x,n))$ if there is one. This definition is a coding of existential and conjunction specific to our need (with a propositional equality) to make the extracted term as simple as possible.

Now, our first lemma will show that Z does choose an X such that $F(x, X)$ when there exists one. To do so, we need the following terms:

- $\text{T}'_1 = \lambda n \, \lambda f \, \lambda g \, (\text{Min}_{\text{exists}} \ n \ f \ \lambda a \, \lambda q \, \lambda m \, (g \ \lambda x \, (x \ a \ q \ m)))$

- $\mathtt{T}_1 = \chi \ \mathtt{T}'_1$

Lemma 5 *Using our new constant χ, we can prove:*

$$T_1 \ : \ \forall F \, \forall x \, \forall X' (F(x,X') \to \exists X \, Z(F,x,X))$$

Proof: Using the axiom for the constant χ, it is enough to prove

$$\vdash \mathtt{T}'_1 \ : \ \forall F \, \forall x \, \forall n{:}\mathrm{N} \ (F(x,\Phi(F,x,n)) \to \exists X \, Z(F,x,X))$$

Assume $m : \mathrm{N}(n), f : F(x,\Phi(F,x,n))$ and $g : \forall X \, (Z(F,x,X) \to K)$. We must show $\mathtt{Min_{exists}} \ m \ f \ \lambda a \, \lambda q \, \lambda m_0 \ (g \ \lambda x_0 \ (x_0 \ a \ q \ m_0)) : K$, using fact 6, what is left to prove is:

$$\lambda a \, \lambda q \, \lambda m_0 \ (g \ \lambda x_0 \ (x_0 \ a \ q \ m_0)) :$$
$$\forall n_0{:}\mathrm{N} \ (F(x,\Phi(F,x,n_0)) \to \mathrm{No\_lt}(n_0, \ \lambda n_0 F(x,\Phi(F,x,n_0))) \to K)$$

It means we want $(g \ \lambda x_0 \ (x_0 \ A \ q \ m)) : K$ with $A : \mathrm{N}(a)$, $q : F(x,\Phi(F,x,a))$ and $m : \mathrm{No\_lt}(a, \ \lambda n_0 F(x,\Phi(F,x,n_0)))$. This is immediate using the definition of Z. $\qquad\square$

Then next lemma will show that Z defines a function. To define when a predicate is a function, we use the following definition:

$$\mathrm{Func}(Q) = \forall x \, \forall X,X' \ (Q(x,X) \to \ Q(x,X') \to (X = \ X')).$$

Remark: we can not use propositional equality here. Indeed, the algorithmic content of the next lemma, may trigger some backtracking as we will explain later with more details. Therefore, in the next lemma, Func and = are used as type.

Lemma 6 *We define the term*

$$T_2 = \lambda z \, \lambda z' \ (z \ (z' \ \lambda n \, \lambda f \, \lambda m \, \lambda n' \, \lambda f' \, \lambda m' \ (\mathtt{Min_{unique}} \ n \ n' \ f \ f' \ m \ m' \ \lambda x \, x)))$$

and we have $\vdash T_2 : \forall F \, \mathrm{Func}(Z(F))$;

Proof: This is an immediate application of the fact 5. $\qquad\square$

The last lemma shows that Z always chooses a well suited X:

Lemma 7 *We define the term*

$$T_3 = \lambda z \, (z \ \lambda a \, \lambda b \, \lambda c \ b)$$

and we have $\vdash T_3 : \forall F \, \forall x \, \forall X{:}Z(F,x) \ F(x,X)$.

Proof: Immediate from the definition of Z. $\qquad\square$

Finally, if we put together the previous three lemmas, and if we quantify over Z (with an existential), we realize the following theorem:

$$\forall F \, \exists C \ \begin{pmatrix} \forall x \, \forall X \ (F(x,X) \to \exists X \, C(F,x,X)) \ \times \ \mathrm{Func}(C(F)) \ \times \\ \forall x \, \forall X \ (C(F,x,X) \to F(x,X)) \end{pmatrix}$$

Now, we will encode the conjunction and existential as usual to simplify the term, and we get the following theorem:

Theorem 3 *(a realizer of NAC) We define the term $T_{NAC} = \lambda x\,(x\ \ T_1\ \ T_2\ \ T_3)$ and we get*

$$\vdash T_{NAC} : \forall F\,\forall K\,\left(\forall C\begin{pmatrix}\forall x\,\forall X\,(F(x,X) \rightarrow \exists X\,C(F,x,X))\ \rightarrow\\ Func(C(F))\ \rightarrow\\ \forall x\,\forall X\,(C(F,x,X) \rightarrow F(x,X))\ \rightarrow\ K\end{pmatrix} \rightarrow K\right)$$

Proof: Immediate from the three previous lemmas. $\qquad\square$

We can briefly and intuitively deduce from the behaviour of the program for the minimum principle the behaviour of T_{NAC}: To be able to choose an X such that $F(x,X)$, you have to give a witness. As in the minimum principle, T_{NAC} will always choose this witness! However, when you use the fact that the choice is functional, you will have access to the two natural numbers build by the χ constant for two previous choices. Then the term will backtrack to the choice with the smallest natural number to make the two choices identical.

In fact we could change the behaviour of the term χ with

$$\chi \star t.\phi.\pi \succ t \star \mathtt{clock}.\phi.\pi$$

where clock is the current time. In this case, when T_{NAC} detects that two choices should have been identical, then, it will backtrack to the latest choice, which seems the most efficient behaviour. You may read [8] for a proof that χ realizes the same formula using the clock in specific models called *generic models*.

Remark: the previous theorem shows that the axiom of choice is realized. We proved it using a new constant χ in the λ-calculus, a new constant Z in the logic and a new axiom relating both. This shows that the derivation used in the previous theorem preserve the adequation lemma in all models.

However, as previously mentioned, we only realize the *non extensional* axiom of choice. Nevertheless, this non extensional axiom of choice is sufficient to prove the countable axiom of choice or the dependant axiom of choice, which suffices for analysis in general.

The next challenge for realisability is therefore the extensional axiom of choice, or at least the extensional choice on real numbers. . .

References

[1] Stefano Berardi. Pruning simply typed λ-terms. *Journal of Logic and Computation*, 6:663–681, 1996.

[2] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–80, 1940.

[3] Thierry Coquand. A semantics of evidence for classical arithmetic. *Journal of Symbolic Logic*, 60:325–337, 1995.

[4] Philippe Curmin. *Marquage des preuves et extraction de programmes*. PhD thesis, Équipe de logique Université Paris VII, 1999.

[5] Samir Farkh and Karim Nour. Complete types in an extension of the system AF2. *Journal of Applied Non-Classical Logics*, 13:73–85, 2003.

[6] Susumu Hayashi. Can Poofs be Animated by Games? In *TLCA*, pages 11–22, 2005.

[7] Jean-Louis Krivine. Dependent choice, 'quote' and the clock. *Th. Comp. Sc.*, 308:259–276, 2003.

[8] Jean-Louis Krivine. Realizability in classical logic. Phd Course, To appear in Panoramas et synthèses, Société Mathématique de France., 2004.

[9] Jean-Louis Krivine and Michel Parigot. Programming with Proofs. *Inf. Process. Cybern.*, EIK 26(3):149–167, 1990.

[10] Catherine Parent. Synthesizing proofs from programs in the Calculus of Inductive Constructions. In *Mathematics for Programs Constructions*, volume 947 of *Lecture Notes in Computer Science*. Springer Verlag, 1995.

[11] Michel Parigot. Recursive programming with proofs. *Theoretical Computer Science*, 94:335–356, 1992.

[12] C. Paulin-Mohring and B. Werner. Synthesis of ML programs in the system Coq. *Journal of Symbolic Computation*, 15:607–640, 1993.

[13] Christophe Raffalli. System ST, Toward A Type System for Extraction AND Proof of Programs. *Archive for Pure and Applied Logic*, 122:107–130, 2002.

[14] Christophe Raffalli. "the phox proof assistant version 0.8". software available on the Internet: `http://www.lama.univ-savoie.fr/~RAFFALLI/phox.html`, 2002.

[15] Christophe Raffalli. System ST, β-reduction and completeness. In *Logic In Computer Science*, pages 21–32, Toronto, 2003. "IEEE".

[16] Frederic Ruyer. *Types, sous-types et preuves*. PhD thesis, Université de Savoie, 2006.