Classical program extraction in the calculus of constructions

Alexandre Miquel

PPS & Université Paris 7
Case 7014, 2 Place Jussieu
75251 PARIS Cedex 05 – France.
alexandre.miquel@pps.jussieu.fr

Abstract. We show how to extract classical programs expressed in Krivine λ_c-calculus from proof-terms built in a proof-irrelevant and classical version of the calculus of constructions with universes. For that, we extend Krivine's realisability model of classical second-order arithmetic to the calculus of constructions with universes using a structure of Π-set which is reminiscent of ω-sets, and show that our realisability model validates Peirce's law and proof-irrelevance. Finally, we extend the extraction scheme to a primitive data-type of natural numbers in a way which preserves the whole compatibility with the classical realisability interpretation of second-order arithmetic.

Introduction

Program extraction has been a major concern from the early development of the calculus of constructions (CC) [3] to its more recent extensions [13, 17] implemented in proof assistants such as Coq [18], LEGO or Plastic. The first extraction scheme implemented in Coq [16] was based on the dependency erasing translation from CC to $F\omega$ [4], with a facility allowing to distinguish computationally relevant parts of the proof from the purely logical parts. (This facility relied on a distinction between two impredicative sorts Prop and Set.) However, as the system grew up, the initial mechanism became obsolete, so that in 2002 the extraction mechanism of Coq was completely redesigned [11]. The currently implemented mechanism now extracts the constructive skeleton of terms (corresponding to the parts built in the predicative universes Type_i) while removing their purely logical part (corresponding to the parts built in Prop).

In this paper we present another extraction mechanism, that extends the extraction mechanism used in system AF2 [6] to the whole type system of Coq. Moreover, this mechanism is able to extract programs from classical proofs (using the control operator call/cc), and it is actually compatible with Krivine's recent results about realising different forms of the axiom of choice [9, 10].

The richness of the type system of CC naturally raises difficulties which do not exist when program extraction is performed in second-order arithmetic only. The first difficulty comes from the fact that in CC, types (and propositions) may

depend on proofs. The traditional way to solve this problem is to add an axiom of *proof-irrelevance* (or to modify the conversion rule) in order to make all the proofs of a given proposition equal. For a long time it has been believed that proof-irrelevance was incompatible with an extraction *à la* AF2, where programs are extracted from purely logical proofs (i.e. built in Prop).[1] As we shall see, this is not the case. Proof-irrelevance is not only compatible with classical program extraction, but it also removes the need of introducing an equational theory for each constant implementing classical reasoning, simply because such constants become transparent in the conversion rule.

From ω-sets to Π-sets The classical extraction presented here is based on a realisability model constructed with Π-sets, a structure which is directly inspired from ω-sets [5, 12], D-sets [19] and saturated λ-sets [2, 14]. Historically, ω-sets (and their generalisation to all partial combinatory algebras: D-sets) have been used to define a realisability model of CC that provides a non-trivial interpretation of the impredicative sort Prop. (Such a model is extended to ECC in [13].) The next improvement came with Altenkirch [2], who noticed that by adding saturation conditions to λ-sets, the λ-set model of CC could be turned into a strong normalisation model. (The structure of saturated λ-set was later reused to extend this construction to a larger class of systems [14].)

However, recent works about normalisation stressed on the importance of definitions by orthogonality in the design of reducibility candidates. Since classical realisability [8–10] also deeply relies on definitions by orthogonality, it is natural to shift the point of view of realisability from the player (the λ-term) to the opponent (the stack it is applied to). In this move, the realisability relation becomes an orthogonality relation, and the structure of λ-set is turned into a new structure: the structure of Π-set which is described in section 3.

Which target (classical) λ-calculus? Since the beginning of the 90's, many λ-calculi have been designed to extend the Curry-Howard correspondence to classical logic. Although we are convinced that most (if not all) of them could be used successfully as the target calculus of classical extraction, we focus here to the λ_c-calculus for several reasons.

The first reason, which is pedagogical, is that it illustrates the combination of two formalisms that are technically very different: on one side the calculus of constructions, whose conversion rule is based on strong evaluation and whose meta-theory deeply relies on the Church-Rosser property; on the other side the λ_c-calculus, that only performs weak head evaluation and for which the notion of normal form and the notion of confluence are simply irrelevant.

The second reason, and actually the main reason, is that λ_c can be extended with extra instructions allowing several forms of the axiom of choice to be realised—and in particular the axiom of dependent choice [9, 10].[2] By taking λ_c as the target calculus—and provided we ensure that the full realisability

[1] Notice that since the currently extraction of Coq erases all proof-terms (in Prop), it is *de facto* compatible with proof-irrelevance.

[2] These results have not been ported yet to other classical calculi.

model of CC is compatible with Krivine's realisability model of second-order arithmetic—we thus allow the extraction mechanism to deal with axioms such as the axiom of the dependent choice (now expressed in CC) and more generally to benefit from the most advanced results of classical realisability.

1 A proof-irrelevant calculus of constructions

1.1 Syntax

The *proof-irrelevant calculus of constructions with universes* ($\mathrm{CC}_\omega^{\mathrm{irr}}$) is built from the same syntax as the calculus of constructions with universes (CC_ω):

$$\begin{array}{lrcl} \textbf{Sorts} & s \in \mathscr{S} & ::= & \mathsf{Prop} \mid \mathsf{Type}_i \qquad (i \geq 1) \\ \textbf{Terms} & M, N, T, U & ::= & x \mid s \mid \Pi x{:}T\,.\,U \mid \lambda x{:}T\,.\,M \mid MN \end{array}$$

Here, Prop denotes the sort of *propositions* (seen as the types of their proofs) whereas Type_i ($i \geq 1$) denotes the ith *predicative universe*.

The set $\mathscr{S}$ of sorts is equipped with a set of axioms $\mathscr{A} \subset \mathscr{S}^2$ and with a set of rules $\mathscr{R} \subset \mathscr{S}^3$ defined by

$$\begin{array}{rcl} \mathscr{A} & = & \{(\mathsf{Prop} : \mathsf{Type}_1);\ (\mathsf{Type}_i : \mathsf{Type}_{i+1}) \mid i \geq 1\} \\ \mathscr{R} & = & \{(\mathsf{Prop}, \mathsf{Prop}, \mathsf{Prop});\ (\mathsf{Type}_i, \mathsf{Prop}, \mathsf{Prop}); \\ & & \ (\mathsf{Prop}, \mathsf{Type}_i, \mathsf{Type}_i);\ (\mathsf{Type}_i, \mathsf{Type}_i, \mathsf{Type}_i) \mid i \geq 1\} \end{array}$$

as well as a (total) order $s_1 \leq s_2$ (the *cumulative order*) which is generated from the relations $\mathsf{Prop} \leq \mathsf{Type}_1$ and $\mathsf{Type}_i \leq \mathsf{Type}_{i+1}$ ($i \geq 1$).

In both constructions $\lambda x{:}T\,.\,M$ and $\Pi x{:}T\,.\,U$, the symbols λ and Π are binders, which bind all the free occurrences of the variable x in M and U, but no occurrence of x in T. The set of free variables of M is written $FV(M)$. As usual we write $T \to U \equiv \Pi x{:}T\,.\,U$ (when $x \notin FV(U)$) the non-dependent product. The external operation of substitution, written $M\{x := N\}$, is defined as expected (taking care of renaming bound variables to avoid variable capture). In what follows, we work with terms up to α-conversion.

Terms of $\mathrm{CC}_\omega^{\mathrm{irr}}$ come with an untyped notion of β-reduction (defined as expected) which is confluent and enjoys Church and Rosser's property. However, we will not consider the untyped notion of β-reduction of $\mathrm{CC}_\omega^{\mathrm{irr}}$ in what follows, since we will identify β-equivalent terms (and more) in the conversion/subsumption rule using a typed equality judgement $\Gamma \vdash M = M' : T$ à la Martin-Löf.

1.2 Typing

A *typing context* (for short: a *context*) is a finite list of declarations of the form

$$\Gamma \ \equiv\ x_1 : T_1, \ldots, x_n : T_n$$

where $x_1, \ldots, x_n$ are pairwise distinct variables and where $T_1, \ldots, T_n$ are arbitrary terms. The *domain* of a context $\Gamma = x_1 : T_1, \ldots, x_n : T_n$ is written

$\mathrm{dom}(\Gamma)$ and defined by $\mathrm{dom}(\Gamma) = \{x_1; \ldots; x_n\}$. Given two contexts Γ and Γ', we write $\Gamma \subseteq \Gamma'$ when all the declarations that appear in Γ also appear in Γ', not necessarily in the same order.

The type system of $\mathrm{CC}^{\mathrm{irr}}_{\omega}$ is defined from four forms of judgements, namely:

$\vdash \Gamma$ **ctx**	'Γ is a well-formed context'
$\Gamma \vdash M : T$	'in the context Γ, the term M has type T'
$\Gamma \vdash T_1 \leq T_2$	'in the context Γ, T_1 is a subtype of T_2
$\Gamma \vdash M_1 = M_2 : T$	'in Γ, M_1 and M_2 are equal terms of type T'

The corresponding rules of inference are given in Fig. 1.

The main syntactic properties of this type system are the following (writing J any of $M : T$ or $T_1 \leq T_2$ or $M_1 = M_2 : T$). We do not indicate the proofs, that all proceed by induction on the suitable derivation.

Lemma 1 (Context well-formedness). *From any derivation of $\Gamma \vdash J$, one can extract a sub-derivation of $\vdash \Gamma$ **ctx**.*

Lemma 2 (Weakening). *If $\Gamma \vdash J$ and $\Gamma \subseteq \Gamma'$ and $\vdash \Gamma'$ **ctx**, then $\Gamma' \vdash J$.*

Lemma 3 (Substitutivity). *If $\Gamma, x : T, \Delta \vdash J$ and $\Gamma \vdash N : T$, then $\Gamma, \Delta\{x := N\} \vdash J\{x := N\}$.*

Lemma 4 (Type of types).

- *If $\Gamma \vdash M : T$ or $\Gamma \vdash M_1 = M_2 : T$, then $\Gamma \vdash T : s$ for some $s \in \mathscr{S}$.*
- *If $\Gamma \vdash T \leq T'$, then $\Gamma \vdash T : s$ and $\Gamma \vdash T' : s'$ for some $s, s' \in \mathscr{S}$.*

2 The language of realisers

2.1 Terms, stacks and processes

Terms of λ_c [7, 10] are simply the pure λ-terms enriched with infinitely many constants taken in a denumerable set $\mathcal{C}$:

$$\textbf{Terms} \qquad t, u \quad ::= \quad x \quad | \quad \lambda x\,.\,t \quad | \quad tu \quad | \quad c \qquad (c \in \mathcal{C})$$

The notion of free and bound variable is defined as usual, as well as the external operation of substitution. In what follows, we will only consider closed terms, and write Λ for the set of all closed terms.

Stacks are built from a denumerable set $\mathcal{B}$ of *stack constants* (a.k.a. *stack bottoms*). Formally, stacks are defined as lists of closed terms terminated by a stack constant:

$$\textbf{Stacks} \qquad \pi \quad ::= \quad b \quad | \quad t \cdot \pi \qquad (b \in \mathcal{B},\, t \in \Lambda)$$

(writing $t \cdot \pi$ the 'consing' operation). The set of stacks is written Π.

Context formation rules

$$\frac{}{\vdash [] \ \mathbf{ctx}} \qquad \frac{\Gamma \vdash T : s \qquad x \notin \mathrm{dom}(\Gamma)}{\vdash \Gamma, x : T \ \mathbf{ctx}}$$

Typing rules

$$\frac{\vdash \Gamma \ \mathbf{ctx}}{\Gamma \vdash x : T}\ {\scriptstyle (x:T)\in\Gamma} \qquad \frac{\vdash \Gamma \ \mathbf{ctx}}{\Gamma \vdash s_1 : s_2}\ {\scriptstyle (s_1,s_2)\in\mathscr{A}} \qquad \frac{\Gamma \vdash M : T \qquad \Gamma \vdash T \leq T'}{\Gamma \vdash M : T'}$$

$$\frac{\Gamma \vdash T : s_1 \qquad \Gamma, x : T \vdash U : s_2}{\Gamma \vdash \Pi x : T \,.\, M : s_3}\ {\scriptstyle (s_1,s_2,s_3)\in\mathscr{R}}$$

$$\frac{\Gamma \vdash \Pi x : T \,.\, U : s \qquad \Gamma, x : T \vdash M : U}{\Gamma \vdash \lambda x : T \,.\, M : \Pi x : T \,.\, U} \qquad \frac{\Gamma \vdash M : \Pi x : T \,.\, U \qquad \Gamma \vdash N : T}{\Gamma \vdash MN : U\{x := N\}}$$

Subtyping rules

$$\frac{\Gamma \vdash T = T' : s}{\Gamma \vdash T \leq T'} \qquad \frac{\Gamma \vdash T \leq T' \qquad \Gamma \vdash T' \leq T''}{\Gamma \vdash T \leq T''}$$

$$\frac{\vdash \Gamma \ \mathbf{ctx} \qquad s_1 \leq s_2}{\Gamma \vdash s_1 \leq s_2} \qquad \frac{\Gamma \vdash T = T' : s \qquad \Gamma \vdash U \leq U'}{\Gamma \vdash \Pi x : T \,.\, U \leq \Pi x : T' \,.\, U'}$$

Equality rules

$$\frac{\Gamma \vdash M : T}{\Gamma \vdash M = M : T} \qquad \frac{\Gamma \vdash M = M' : T \qquad \Gamma \vdash M' = M'' : T}{\Gamma \vdash M = M'' : T}$$

$$\frac{\Gamma \vdash M = M' : T}{\Gamma \vdash M' = M : T} \qquad \frac{\Gamma \vdash M = M' : T \qquad \Gamma \vdash T \leq T'}{\Gamma \vdash M = M' : T'}$$

$$\frac{\Gamma \vdash T = T' : s_1 \qquad \Gamma, x : T \vdash U = U' : s_2 \qquad (s_1, s_2, s_3) \in \mathscr{R}}{\Gamma \vdash \Pi x : T \,.\, U = \Pi x : T' \,.\, U' : s_3}$$

$$\frac{\Gamma \vdash T = T' : s \qquad \Gamma, x : T \vdash M = M' : U}{\Gamma \vdash \lambda x : T \,.\, M = \lambda x : T' \,.\, M' : \Pi x : T \,.\, U}$$

$$\frac{\Gamma \vdash M = M' : \Pi x : T \,.\, U \qquad \Gamma \vdash N = N' : T}{\Gamma \vdash MN = M'N' : U\{x := N\}}$$

$$\frac{\Gamma \vdash \Pi x : T \,.\, U : s \qquad \Gamma, x : T \vdash M : U \qquad \Gamma \vdash N : T}{\Gamma \vdash (\lambda x : T \,.\, M)N = M\{x := N\} : U\{x := N\}}$$

$$\frac{\Gamma \vdash T : \mathsf{Prop} \qquad \Gamma \vdash M : T \qquad \Gamma \vdash M' : T}{\Gamma \vdash M = M' : T}$$

Fig. 1. Typing rules of CC_ω

To each stack $\pi \in \Pi$ we associate a constant $\mathsf{k}_\pi \in \mathcal{C}$ in such a way that (1) the correspondence $\pi \mapsto \mathsf{k}_\pi$ is injective, and (2) the set of all $c \neq \mathsf{k}_\pi$ (for all $\pi \in \Pi$) is still a denumerably infinite subset of $\mathcal{C}$.

In what follows, we call a *quasi-proof* any closed term containing none of the constants k_π ($\pi \in \Pi$). Finally, we take a fresh constant $\mathsf{cc} \in \mathcal{C}$ ('call/cc')[3] such that $\mathsf{cc} \neq \mathsf{k}_\pi$ for all $\pi \in \Pi$, which we reserve to realise Peirce's law.

Processes are then defined as pairs formed by a closed term and a stack:

Processes $\quad p, q \quad ::= \quad t \star \pi \qquad\qquad (t \in \Lambda,\ \pi \in \Pi)$

2.2 Evaluation and realisability

Processes are equipped with a binary relation of one step evaluation, written $p \succ p'$, which is defined by the following rules:

$$\begin{array}{rcl@{\qquad\qquad}rcl} \lambda\xi\,.\,t \star u \cdot \pi & \succ & t\{\xi := u\} \star \pi & \mathsf{cc} \star t \cdot \pi & \succ & t \star \mathsf{k}_\pi \cdot \pi \\ tu \star \pi & \succ & t \star u \cdot \pi & \mathsf{k}_\pi \star t \cdot \pi' & \succ & t \star \pi \end{array}$$

The definition of a realisability model based on the language λ_c (for second-order arithmetic or for CC_ω) is parameterised by a fixed set of processes $\perp\!\!\!\perp$ that we assume to be *saturated*, in the sense that:

$$p \succ p' \quad \text{and} \quad p' \in \perp\!\!\!\perp \quad \text{imply} \quad p \in \perp\!\!\!\perp \qquad\qquad (\text{for all } p, p')$$

Intuitively, $\perp\!\!\!\perp$ represents a set of accepting processes (w.r.t. some correctness criterion), and the condition of saturation expresses that each processes that evaluates to an accepting process is itself an accepting process. A typical candidate for $\perp\!\!\!\perp$ is the set $\perp\!\!\!\perp_0$ of all terminating processes defined by:

$$\perp\!\!\!\perp_0 \ = \ \{p \mid \exists p' \ (p \succ^* p' \ \wedge \ p' \not\succ)\}\,.$$

In classical realisability, sets of stacks are used as *falsity values* (that is, as sets of potential refutations). Each falsity value $S \subset \Pi$ defines by orthogonality a *truth value* written $S^{\perp\!\!\!\perp}$ and defined by

$$S^{\perp\!\!\!\perp} \ = \ \{t \in \Lambda \mid \forall \pi \in S \ \ t \star \pi \in \perp\!\!\!\perp\}\,.$$

In section 4, we will construct a model where all the objects are annotated with falsity values, using a structure of Π-set.

3 The Π-set structure

3.1 Definition

Definition 1 (Π-set). *A* Π-set *is a pair* $X = \langle |X|, \perp_X \rangle$ *formed by a set* $|X|$ *(called the* carrier *of* X*) equipped with a binary relation* $\perp_X \subset |X| \times \Pi$ *(called the* local orthogonality relation *of* X*).*

[3] From the Scheme operator 'call-cc' (call with current continuation)

Intuitively, the binary relation $x \perp_X \pi$ expresses that the stack π is an attempt to refute (or to attack, or to falsify) the denotation $x \in |X|$. Given an element $x \in |X|$, we write $x^{\perp_X} = \{\pi \mid x \perp_X \pi\}$ the orthogonal of x w.r.t. X. From this we define the *local realisability relation* $t \Vdash_X x$ by setting

$$\begin{array}{lll} t \Vdash_X x & \text{iff} & \forall \pi\ (x \perp_X \pi \Rightarrow t \star \pi \in \perp\!\!\!\perp) \\ & \text{iff} & t \in (x^{\perp_X})^{\perp\!\!\!\perp} \end{array}$$

for all $t \in \Lambda$ and $x \in |X|$.

Remark. Unlike ω-sets, we do not require that each element of X is realised by at least a quasi-proof—we do not even require that each element of X has a realiser. However, all the elements of the carrier have realisers as soon as the set $\perp\!\!\!\perp$ is inhabited: given a fixed process $t_0 \star \pi_0 \in \perp\!\!\!\perp$, it is easy to check that the term $\mathsf{k}_{\pi_0} t_0$ is orthogonal to any stack (w.r.t. $\perp\!\!\!\perp$), and thus realises any denotation.

Coarse Π-sets We say that a Π-set X is *coarse* when $\perp_X = \emptyset$ (i.e. when the orthogonality relation on X is empty). By duality, we get $t \Vdash_X x$ for all $t \in \Lambda$ and $x \in |X|$, which means that any term realises any element of the carrier of X.

Notice that any set X can be given the structure of a coarse Π-set simply by taking $|X| = X$ and $\vdash_X = \emptyset$.

Pointed Π-sets In many situations, it is desirable to exclude the empty Π-set and to work only with Π-sets whose carrier is inhabited. To avoid the cost of introducing the axiom of choice in the meta-theory (typically to ensure that the Cartesian product of a family of inhabited Π-sets is inhabited), we will only consider Π-sets with a distinguished element of the carrier, that is: *pointed Π-sets*. Formally, a pointed Π-set is a triple $X = \langle |X|, \perp_X, \varepsilon_X \rangle$ where $\langle |X|, \perp_X \rangle$ is a Π-set and where $\varepsilon_X \in |X|$.

3.2 Cartesian product of a family of Π-sets

Let $(Y_x)_{x \in |X|}$ be a family of Π-sets indexed by the carrier of a Π-set X. The *Cartesian product* of the family $(Y_x)_{x \in |X|}$ is the Π-set written $\Pi x : X \,.\, Y_x$ and defined by:

$$|\Pi x : X \,.\, Y_x| = \prod_{x \in |X|} |Y_x| \quad \text{and} \quad f^{\perp_{\Pi x : X \,.\, Y_x}} = \bigcup_{x \in |X|} \Big((x^{\perp_X})^{\perp\!\!\!\perp} \cdot (f(x)^{\perp_{Y_x}}) \Big)$$

for all $f \in |\Pi x : X \,.\, Y_x|$. Moreover if Y_x is a pointed Π-set for all $x \in |X|$, then the product $\Pi x : X \,.\, Y_x$ can be given the structure of a pointed Π-set by setting

$$\varepsilon_{\Pi x : X \,.\, Y_x} \;=\; (x \in |X| \mapsto \varepsilon_{Y_x}) \,.$$

Fact 1 *If Y_x is a coarse Π-set (resp. a coarse pointed Π-set) for all $x \in |X|$, then $\Pi x : X \,.\, Y_x$ is a coarse Π-set (resp. a coarse pointed Π-set).*

In section 4 we will interpret the sorts Prop, Type_i by coarse pointed Π-sets; hence the fact above will automatically imply that more generally, all types of predicates will be interpreted by coarse pointed Π-sets.

3.3 Degenerated Π-sets

A Π-set is said to be *degenerated* when its carrier is a singleton: $|X| = \{\varepsilon_X\}$. (In this case we can always consider X as a pointed Π-set by taking ε_X as the unique element of its carrier.) A degenerated Π-set X is characterised by its unique element ε_X and by the set of stacks $\varepsilon_X^{\perp_X}$ that are orthogonal to this element, which set of stacks will be written $X^\perp$ $(= \varepsilon_X^{\perp_X})$.

Fact 2 (Product of degenerated Π-sets) *If Y_x is a degenerated Π-set for all $x \in |X|$, then the Π-set $\Pi x : X \,.\, Y_x$ is degenerated too, and we have*

$$(\Pi x : X \,.\, Y_x)^\perp \quad = \quad \bigcup_{x \in |X|} \Big((x^{\perp_X})^{\perp\!\!\!\perp} \cdot Y_x^\perp \Big)$$

Moreover, if the Π-set X is degenerated, then the Cartesian product $\Pi x : X \,.\, Y_x$ is non-dependent and $(\Pi x : X \,.\, Y_x)^\perp \;=\; (X \to Y)^\perp \;=\; (X^\perp)^{\perp\!\!\!\perp} \cdot Y^\perp$.

In what follows, degenerated Π-sets will be used to interpret propositions.

3.4 Subtyping

Given two Π-sets X and X', we write $X \leq X'$ (X is a *subtype* of X') when

$$|X| \subseteq |X'| \qquad \text{and} \qquad x^{\perp_X} \supseteq x^{\perp_{X'}} \quad \text{for all } x \in |X| \,.$$

(Notice that the reverse inclusion above is necessary to ensure that $t \Vdash_X x$ implies $t \Vdash_{X'} x$ for all $t \in \Lambda$ and $x \in |X|$.) When both X and X' are pointed Π-sets, we also require that: $\varepsilon_{X'} = \varepsilon_X$.

4 Constructing the model

In what follows, we work in ZF set theory extended with an axiom expressing the existence of infinitely many inaccessible cardinals to interpret the hierarchy of predicative universes.

4.1 An alternative encoding of functions

To achieve proof-irrelevance in the model, we will interpret all proof-objects by a single value written $\bullet$ and all propositions by degenerated Π-sets based on the singleton $\{\bullet\}$. Since we want to keep the property of closure under Cartesian products (Fact 2), it is necessary to identify all constant functions $(x \in D \mapsto \bullet)$ for the proof-object $\bullet$ itself. For that, we adopt a set-theoretic encoding of functions (proposed by [1] and inspired from the notion of trace in domain theory) in which functions are represented not as set of pairs $\langle x, y \rangle$ such that $y = f(x)$, but as set of pairs $\langle x, z \rangle$ such that $z \in f(x)$.

Formally, we introduce the following abbreviations:

$$\begin{array}{rcl} f \text{ function} & \equiv & \forall p \in f\ \exists x\ \exists y\ \ p = \langle x, y\rangle \\ (x \in D \mapsto E_x) & = & \{\langle x, z\rangle \mid x \in D\ \wedge\ z \in E_x\} \\ f(x) & = & \{z \mid \exists x\ \langle x, z\rangle \in f\} \end{array}$$

This encoding is sound in the sense that given a function $f = (x \in D \mapsto E_x)$, we have $f(d) = E_d$ for all $d \in D$. However, the domain information is partially lost since the encoding keeps no track of elements of the domain mapped to the empty set. We can only define the *support* of a function

$$\mathrm{supp}(f) \ = \ \{x \mid \exists z\ \ \langle x, z\rangle \in f\}\,.$$

Apart from this (minor) difference, this alternative encoding of functions can be used the same way as the traditional encoding. From now on we consider that functions in the model are represented in this way, and we take $\bullet = \emptyset$ so that the equality $(x \in D \mapsto \bullet) = \bullet$ now holds for all D.

4.2 Interpreting sorts

Let $(\lambda_i)_{i\geq 1}$ be an increasing sequence of inaccessible cardinals and set:

$$\begin{array}{l} \mathcal{U}_0 = \{\{\bullet\}\} \times \mathfrak{P}(\{\bullet\} \times \Pi) \times \{\bullet\} \\ \mathcal{U}_i = \displaystyle\bigcup_{\substack{S \in V_{\lambda_i} \\ s_0 \in S}} \{S\} \times \mathfrak{P}(S \times \Pi) \times \{s_0\} \qquad (\subset V_{\lambda_i}) \end{array}$$

By definition, $\mathcal{U}_0$ is the set of all degenerated pointed Π-sets based on the singleton $\{\bullet\}$ whereas $\mathcal{U}_i$ $(i \geq 1)$ is the set of all pointed Π-sets whose (nonempty) carrier belongs to V_{λ_i} (i.e. the ith ZF-universe). Each set of Π-sets $\mathcal{U}_i$ $(i \geq 0)$ can be given in turn the structure of a coarse pointed Π-set $\mathcal{U}'_i$ by setting:

$$\mathcal{U}'_0 = \langle \mathcal{U}_0,\ \emptyset,\ \langle\{\bullet\},\ \emptyset,\ \bullet\rangle\rangle \qquad \text{and} \qquad \mathcal{U}'_i = \langle \mathcal{U}_i,\ \emptyset,\ \mathcal{U}'_{i-1}\rangle \quad \text{for } i \geq 1\,.$$

Finally, the domain of all denotations $\mathcal{M}$ is taken as the union of all carriers of the Π-sets in the universes $\mathcal{U}_i$: $\mathcal{M} = \bigcup_{i\in\omega} \bigcup_{X\in\mathcal{U}_i} |X|$.

Fact 3 (Closure under Cartesian product) *For all $i \geq 1$:*

1. *If $X \in \mathcal{U}_i$ and $Y_x \in \mathcal{U}_0$ for all $x \in |X|$, then $\Pi x : X\,.\,Y_z \in \mathcal{U}_0$;*
2. *If $X \in \mathcal{U}_i$ and $Y_x \in \mathcal{U}_i$ for all $x \in |X|$, then $\Pi x : X\,.\,Y_z \in \mathcal{U}_i$.*

4.3 The interpretation function

A *valuation* in $\mathcal{M}$ is a partial function $\rho : \mathcal{X} \to \mathcal{M}$ (writing $\mathcal{X}$ the set of all variables) whose domain $\mathrm{dom}(\rho) \subset \mathcal{X}$ is finite. (Here, it is more convenient to keep the traditional set-theoretic encoding of functions to represent valuations.)

The set of all valuations is written $\mathrm{Val}_{\mathcal{M}}$. Given a valuation ρ, a variable $x \in \mathcal{X}$ and a value $v \in \mathcal{M}$, we write $(\rho, x \leftarrow v)$ the valuation defined by

$$(\rho, x \leftarrow v)(x) = v \qquad \text{and} \qquad (\rho, x \leftarrow v)(y) = \rho(y) \qquad (y \in \mathrm{dom}(\rho) \setminus \{x\})$$

To each term M we associate a partial function $[\![M]\!]_{\_} : \mathrm{Val}_{\mathcal{M}} \rightharpoonup \mathcal{M}$ which is inductively defined on M by the equations:

$$\begin{array}{ll} [\![x]\!]_\rho = \rho(x) & [\![\Pi x : T \,.\, U]\!]_\rho = \Pi v : [\![T]\!]_\rho \,.\, [\![U]\!]_{\rho; x \leftarrow v} \quad \text{(product of } \Pi\text{-sets)} \\ [\![\mathsf{Prop}]\!]_\rho = \mathcal{U}'_0 & [\![\lambda x : T \,.\, M]\!]_\rho = (v \in |[\![T]\!]_\rho| \mapsto [\![M]\!]_{\rho; x \leftarrow v}) \\ [\![\mathsf{Type}_i]\!]_\rho = \mathcal{U}'_i & [\![MN]\!]_\rho = [\![M]\!]_\rho([\![N]\!]_\rho) \end{array}$$

Since the function $\rho \mapsto [\![M]\!]_\rho$ is partial, it is important to precise when the denotation $[\![M]\!]\rho$ is defined:

- $[\![x]\!]_\rho$ is defined when $x \in \mathrm{dom}(\rho)$;
- $[\![\mathsf{Prop}]\!]_\rho$ and $[\![\mathsf{Type}_i]\!]_\rho$ are always defined;
- $[\![\Pi x : T \,.\, U]\!]_\rho$ is defined when
  - $[\![T]\!]_\rho$ is defined, and it is a pointed Π-set,
  - For all $v \in |[\![T]\!]_\rho|$, $[\![U]\!]_{\rho, x \leftarrow v}$ is defined, and it is a pointed Π-set,
  - $\Pi v : [\![T]\!]_\rho \,.\, [\![U]\!]_{\rho; x \leftarrow v}$ is an element of $\mathcal{M}$;
- $[\![\lambda x : T \,.\, M]\!]_\rho$ is defined when
  - $[\![T]\!]_\rho$ is defined, and it is a pointed Π-set,
  - For all $v \in |[\![T]\!]_\rho|$, $[\![M]\!]_{\rho, x \leftarrow v}$ is defined,
  - $(v \in |[\![T]\!]_\rho| \mapsto [\![U]\!]_{\rho; x \leftarrow v})$ is an element of $\mathcal{M}$;
- $[\![MN]\!]_\rho$ is defined when
  - $[\![M]\!]_\rho$ and $[\![N]\!]_\rho$ are defined,
  - $[\![M]\!]_\rho$ is a function, and $[\![M]\!]_\rho([\![N]\!]_\rho)$ is an element of $\mathcal{M}$.

The interpretation function is extended to all contexts by setting:

$$[\![\Gamma]\!] \quad = \quad \left\{\rho \in \mathrm{Val}_{\mathcal{M}} \mid \forall (x : T) \in \Gamma \rho(x) \in |[\![T]\!]_\rho|\right\}$$

4.4 Soundness

Definition 2 (Soundness conditions).

1. *A typing judgement $\Gamma \vdash M : T$ is sound w.r.t. $\mathcal{M}$ if for all $\rho \in [\![\Gamma]\!]$:*
   - *The denotations $[\![M]\!]_\rho$ and $[\![T]\!]_\rho$ are defined;*
   - *$[\![T]\!]_\rho$ is a Π-set; and*
   - *$[\![M]\!]_\rho \in |[\![T]\!]_\rho|$.*
2. *A subtyping judgement $\Gamma \vdash T \leq T'$ is sound w.r.t. $\mathcal{M}$ if for all $\rho \in [\![\Gamma]\!]$:*
   - *The denotations $[\![T]\!]_\rho$ and $[\![T']\!]_\rho$ are defined;*
   - *$[\![T]\!]_\rho$ and $[\![T']\!]_\rho$ are Π-sets;*
   - *$[\![T]\!]_\rho \leq [\![T']\!]_\rho$.*
3. *An equality judgement $\Gamma \vdash M = M' : T$ is sound w.r.t. $\mathcal{M}$ if for all $\rho \in [\![\Gamma]\!]$:*
   - *The denotations $[\![M]\!]_\rho$, $[\![M']\!]_\rho$ and $[\![T]\!]_\rho$ are defined;*
   - *$[\![T]\!]_\rho$ is a Π-set; and*
   - *$[\![M]\!]_\rho = [\![M']\!]_\rho \in |[\![T']\!]_\rho|$.*

Proposition 1 (Soundness). *If a typing judgement, a subtyping judgement or an equality judgement is derivable (Fig. 1), then it is sound w.r.t. $\mathcal{M}$.*

Proof. By induction of the derivation of the judgement. □

4.5 Adequacy

The basic extraction scheme To each (raw-)term M of CC_ω we associate a term M^* of λ_c which is inductively defined from the equations

$$x^* = x \qquad s^* = (\Pi x : T \,.\, U)^* = \lambda z \,.\, z \qquad \text{(or any quasi-proof)}$$
$$(\lambda x : T \,.\, M)^* = \lambda x \,.\, M^* \qquad (MN)^* = M^* N^*$$

Intuitively, this extraction function erases all non-computational information related to types, but preserves all the computational contents of proof-terms.

Substitutions A *substitution* is a partial function $\sigma : \mathcal{X} \mapsto \Lambda$ whose domain $\mathrm{dom}(\sigma) \subset \mathcal{X}$ is finite. Given an open term t of the λ_c-calculus and a substitution σ, we write $t[\sigma]$ the result of applying the substitution σ to t.

Let Γ be a context, ρ a valuation and σ a substitution. We say that σ *realises* ρ *in* Γ and write $\sigma \Vdash_\Gamma \rho$ when

1. $\mathrm{dom}(\sigma) = \mathrm{dom}(\Gamma)$
2. $\rho \in [\![\Gamma]\!]$
3. For all $(x : T) \in \Gamma$: $\quad \sigma(x) \Vdash_{[\![T]\!]_\rho} \rho(x)$

We can now extend the property of adequacy of second-order arithmetic [10] to CC_ω as follows:

Proposition 2 (Adequacy). *If $\Gamma \vdash M : T$, then for all valuations $\rho \in \Gamma$ and for all substitutions σ such that $\sigma \Vdash_\Gamma \rho$, we have*

$$M^*[\sigma] \;\Vdash_{[\![T]\!]_\rho}\; [\![M]\!]_\rho \,.$$

Proof. By induction of the derivation of the judgement. □

In particular, when the judgement $\vdash M : T$ is derivable in the empty context, the extracted term M^* realises the denotation of M: $\quad M^* \;\Vdash_{[\![T]\!]}\; [\![M]\!]$.

5 Extensions of the formalism

5.1 Peirce's law and the excluded middle

Let us now extend $\mathrm{CC}^{\mathrm{irr}}_\omega$ with a new constant

$$\mathsf{peirce} \quad : \quad \Pi A, B : \mathsf{Prop} \,.\, (((A \to B) \to A) \to A)$$

that we interpret in the model $\mathcal{M}$ as $[\![\mathsf{peirce}]\!]_\rho = \bullet$. We then extend the basic extraction scheme by setting $\mathsf{peirce}^* = \lambda\_ \,.\, \lambda\_ \,.\, \mathfrak{cc}$ and check that this extension is adequate in the sense of Prop. 2:

Fact 4 $\quad \mathsf{peirce}^* \;\in\; ([\![\Pi A, B : \mathsf{Prop} \,.\, (((A \to B) \to A) \to A)]\!]^\perp)^{\perp\!\!\!\perp}$.

From this extension of the calculus, it is easy to derive the law of excluded middle $\Pi A : \mathsf{Prop} \,.\, (A \vee \neg A)$ at the level of propositions (defining disjunction and negation by the mean of standard second-order encodings).

Remark. In the source calculus ($\mathrm{CC}^{\mathrm{irr}}_{\omega}$), it is not necessary to endow the extra constant peirce with specific equality rules, since the rule of proof-irrelevance already performs all possible identifications at the level of proof-terms. In the target calculus (λ_c), the constant peirce is extracted to the λ_c-term $\lambda\_\,.\,\lambda\_\,.\,\mathsf{cc}$ that evaluates as expected, by consuming two computationally irrelevant arguments (corresponding to types) before capturing the current continuation.

5.2 Decomposing the propositional dependent product

In intuitionistic and classical realisability [10], (non relativised) first- and second-order quantification is usually interpreted parametrically, that is, as an intersection (or as a union on the side of stacks). In $\mathrm{CC}^{\mathrm{irr}}_{\omega}$, universal quantification is represented by a dependent-product $\Pi x : T \,.\, U(x)$, that is, by a type of functions taking a value $x : T$ and returning a proof of $U(x)$.

To bridge both interpretations of universal quantification, we first extend the formalism with three new syntactic constructs, namely:

- An intersection type binder $\forall x : T \,.\, U$ corresponding to the parametric interpretation of the universal quantification. This construction is exactly the implicit dependent product of the implicit calculus of constructions [15], but here restricted to propositions.
- A construction $M \in T$ representing the *propositional contents* of the typing judgement $M : T$. As we shall see, the construction $M \in T$ represents the proposition whose proofs are the realisers of the term M in the type T.
- A constant $\top$ representing the proposition realised by all λ-terms.

$$\textbf{Terms} \qquad T, U \quad ::= \quad \cdots \quad | \quad \forall x : T \,.\, U \quad | \quad M \in T \quad | \quad \top$$

These new syntactic constructs that we interpret in $\mathcal{M}$ by

$$[\![\forall x : T \,.\, U]\!]_\rho \;=\; \Big\langle \{\bullet\},\ \{\bullet\} \times \bigcup_{v \in [\![T]\!]_\rho} [\![U]\!]^{\perp}_{\rho, x \leftarrow v},\ \bullet \Big\rangle$$

$$[\![M \in T]\!]_\rho \;=\; \big\langle \{\bullet\},\ \{\bullet\} \times [\![M]\!]_\rho^{\perp_{[\![T]\!]_\rho}},\ \bullet \big\rangle \qquad\qquad [\![\top]\!]_\rho \;=\; \langle \{\bullet\},\ \emptyset,\ \bullet \rangle$$

come with typing, subtyping and equality rules that are given in Fig. 2.

Fact 5 *The typing rules, subtyping rules and equality rules of Fig. 2 are sound w.r.t. the interpretation of the constructs $\forall x : T \,.\, U$, $M \in T$ and $\top$ in $\mathcal{M}$.*

In the extended formalism, the propositional dependent product can now be decomposed in terms of $\forall$ and $\in$ as

$$\Pi x : T \,.\, U \quad = \quad \forall x : T \,.\, ((x \in T) \to U)$$

using the decomposition rule of Fig. 2. Intuitively, this equality rule expresses that in $\mathrm{CC}^{\mathrm{irr}}_{\omega}$, the propositional dependent product corresponds exactly to the relativised universal quantification in the sense of AF2. In subsection 5.3, we will exploit this fact in order to recover the usual interpretation of the numeric quantification in classical realisability.

Typing rules

$$\frac{\Gamma, x : T \vdash U : \mathsf{Prop}}{\Gamma \vdash \forall x : T \,.\, U : \mathsf{Prop}} \qquad \frac{\Gamma \vdash M : T}{\Gamma \vdash M \in T : \mathsf{Prop}} \qquad \frac{\vdash \Gamma \ \mathbf{ctx}}{\Gamma \vdash \top : \mathsf{Prop}}$$

Subtyping rules

$$\frac{\Gamma \vdash T = T' : s \qquad \Gamma \vdash U \leq U' \qquad \Gamma \vdash U' : \mathsf{Prop}}{\Gamma \vdash \forall x : T \,.\, U \leq \forall x : T' \,.\, U'}$$

$$\frac{\Gamma \vdash M : T \qquad \Gamma \vdash T \leq T'}{\Gamma \vdash (M \in T) \leq (M \in T')} \qquad \frac{\Gamma \vdash T : \mathsf{Prop}}{\Gamma \vdash T \leq \top}$$

Equality rules

$$\frac{\Gamma \vdash T = T' : s \qquad \Gamma, x : T \vdash U = U' : \mathsf{Prop}}{\Gamma \vdash \forall x : T \,.\, U = \forall x : T' \,.\, U' : \mathsf{Prop}}$$

$$\frac{\Gamma \vdash M = M' : T \qquad \Gamma \vdash T = T' : s}{\Gamma \vdash (M \in T) = (M' \in T') : \mathsf{Prop}}$$

(Decomposition of Π)

$$\frac{\Gamma, x : T \vdash U : \mathsf{Prop}}{\Gamma \vdash \Pi x : T \,.\, U = \forall x : T \,.\, (x \in T \to U) : \mathsf{Prop}}$$

$$\frac{\Gamma \vdash T : s}{\Gamma \vdash \Pi x : T \,.\, \top = \top : \mathsf{Prop}}$$

(Simplification of $\in$)

$$\frac{\Gamma \vdash T : s}{\Gamma \vdash (T \in s) = \top : \mathsf{Prop}} \qquad \frac{\Gamma \vdash M : \top}{\Gamma \vdash (M \in \top) = \top : \mathsf{Prop}}$$

$$\frac{\Gamma \vdash M : \Pi x : T \,.\, U}{\Gamma \vdash (M \in \Pi x : T \,.\, U) = \forall x : T \,.\, ((x \in T) \to M x \in U) : \mathsf{Prop}}$$

$$\frac{\Gamma \vdash M : \forall x : T \,.\, U}{\Gamma \vdash (M \in \forall x : T \,.\, U) = \forall x : T \,.\, (M \in U) : \mathsf{Prop}}$$

Fig. 2. Typing, subtyping and equality rules of $\forall$, $\in$ and $\top$

5.3 Adding a primitive type of natural numbers

Let us now extend $\mathrm{CC}^{\mathrm{irr}}_{\omega}$ with the following set of typed constants ($i \geq 1$):

$$\begin{array}{rcl} \mathsf{nat} & : & \mathsf{Type}_1 \qquad \mathsf{0} : \mathsf{nat} \qquad \mathsf{s} : \mathsf{nat} \to \mathsf{nat} \\ \mathsf{nat\_ind} & : & \Pi X : \mathsf{nat} \to \mathsf{Prop} \,.\, (X\mathsf{0} \to \Pi y : \mathsf{nat} \,.\, (Xy \to X(\mathsf{s}\, y)) \to \Pi x : \mathsf{nat} \,.\, Xx) \\ \mathsf{nat\_rec}_i & : & \Pi X : \mathsf{nat} \to \mathsf{Type}_i \,.\, (X\mathsf{0} \to \Pi y : \mathsf{nat} \,.\, (Xy \to X(\mathsf{s}\, y)) \to \Pi x : \mathsf{nat} \,.\, Xx) \end{array}$$

The constants $\mathsf{nat\_ind}$ and $\mathsf{nat\_rec}_i$ ($i \geq 1$) respectively implement the induction principle and (dependently-typed) recursion in Type_i.

Interpreting nat The constant nat is interpreted as the pointed Π-set defined by $|[\![\mathsf{nat}]\!]| = \mathbb{N}$, $\varepsilon_{[\![\mathsf{nat}]\!]} = 0$, and whose orthogonality relation is given by

$$n^{\perp_{[\![\mathsf{nat}]\!]}} \;=\; [\![\forall X : \mathsf{nat} \to \mathsf{Prop} \,.\, (X\mathsf{0} \to \forall y : \mathsf{nat} \,.\, (Xy \to X(\mathsf{s}\, y)) \to Xx)]\!]^{\perp}_{x \leftarrow n}$$

for all $n \in \mathbb{N}$. Notice that the definition above is not circular, since the r.h.s. only depends on the definition of the carrier of nat, but not on its orthogonality relation. The constants $\mathsf{0}$ and s are then interpreted as expected.

The interest of this definition is that the proposition $x \in \mathsf{nat}$ is interpreted exactly as the relativisation predicate which is traditionally used in second-order arithmetic to define numeric quantification:

$$\mathsf{Nat}(x) \;\equiv\; \forall X : \mathsf{nat} \to \mathsf{Prop} \,.\, (X\mathsf{0} \to \forall y : \mathsf{nat} \,.\, (Xy \to X(\mathsf{s}\, y)) \to Xx)$$

Fact 6 *The following equality rule is sound in* $\mathcal{M}$*:*

$$\frac{\Gamma \vdash M \,:\, \mathsf{nat}}{\Gamma \vdash (M \in \mathsf{nat}) = \mathsf{Nat}(M) \,:\, \mathsf{Prop}}$$

Combining the latter with the decomposition of the dependent product (cf subsection 5.2) we get the equality $\Pi x : \mathsf{nat} \,.\, P(x) \;=\; \forall x : \mathsf{nat} \,.\, (\mathsf{Nat}(x) \to P(x))$ expressing that the PTS-style quantification $\Pi x : \mathsf{nat} \,.\, P(x)$ is interpreted in $\mathcal{M}$ exactly the same way as the numeric quantification in classical realisability [10].

Interpreting $\mathsf{nat\_ind}$ *and* $\mathsf{nat\_rec}_i$ The constant $\mathsf{nat\_ind}$ is interpreted as the proof object $\bullet$ whereas the constants $\mathsf{nat\_rec}_i$ are interpreted the obvious way (i.e. as the corresponding set-theoretic recursors in the universes $\mathcal{U}_i$). From the latter definition, it is immediate that:

Fact 7 *The following equality rules are sound in* $\mathcal{M}$*:*

$$\frac{\begin{array}{ll} \Gamma \vdash P \,:\, \mathsf{nat} \to \mathsf{Type}_i & \Gamma \vdash N \,:\, \mathsf{nat} \\ \Gamma \vdash M_0 \,:\, P\,\mathsf{0} & \Gamma \vdash M_1 \,:\, \Pi p : \mathsf{nat} \,.\, P\,p \to P(\mathsf{s}\,p) \end{array}}{\begin{array}{l} \Gamma \vdash \mathsf{nat\_rec}_i\, P\, M_0\, M_1\, \mathsf{0} = M_0 \,:\, P\,\mathsf{0} \\ \Gamma \vdash \mathsf{nat\_rec}_i\, P\, M_0\, M_1\, (\mathsf{s}\, N) = M_1\, N\, (\mathsf{nat\_rec}_i\, P\, M_0\, M_1\, N) \,:\, P\,(\mathsf{s}\, N) \end{array}}$$

Extraction We finally extend the extraction mechanism to the new constants nat, $\mathsf{0}$, s, $\mathsf{nat\_ind}$ and $\mathsf{nat\_rec}_i$ by setting:

$$\mathsf{nat}^* = \lambda z\,.\,z \qquad \text{(or any quasi-proof)}$$
$$\mathsf{0}^* = \lambda xf\,.\,x \qquad \mathsf{s}^* = \lambda nxf\,.\,f(nxf) \qquad \mathsf{nat\_rec}_i^* = \mathsf{nat\_ind}^*$$
$$\mathsf{nat\_ind}^* = \lambda\_xfn\,.\,n\,(\lambda z\,.\,z\,\mathsf{0}^*x)\,(\lambda p\,.\,p\ (\lambda myz\,.\,z\,(s^*m)\,(fmy)))\,(\lambda xy\,.\,y)$$

Proposition 3. *The extraction scheme extended to the constants* nat, $\mathsf{0}$, s, $\mathsf{nat\_ind}$ *and* $\mathsf{nat\_rec}_i$ *is adequate w.r.t.* $\mathcal{M}$ *(in the sense of Prop. 2).*

References

1. P. Aczel. On relating type theories and set theories. In Altenkirch, Naraschewski, and Reus, editors, *Proceedings of Types'98*, 1999.
2. T. Altenkirch. *Constructions, Inductive Types and Strong Normalization*. PhD thesis, University of Edinburgh, November 1993.
3. T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 120(76):95, 1988.
4. J.H. Geuvers and M.J. Nederhof. A modular proof of strong normalization for the calculus of constructions. In *Journal of Functional Programming*, volume 1,2(1991), pages 155–189, 1991.
5. J. M. E. Hyland. The effective topos. In A. S. Troelstra and D. van Dalen, editors, *The L. E. J. Brouwer Centenary Symposium*. North Holland, 1982.
6. J.-L. Krivine. *Lambda-calcul, types et modèles*. Masson, 1991.
7. J.-L. Krivine. A general storage theorem for integers in call-by-name lambda-calculus. *Th. Comp. Sc.*, 129:79–94, 1994.
8. J.-L. Krivine. Typed lambda-calculus in classical Zermelo-Fraenkel set theory. *Arch. Math. Log.*, 40(3):189–205, 2001.
9. J.-L. Krivine. Dependent choice, 'quote' and the clock. *Th. Comp. Sc.*, 308:259–276, 2003.
10. J.-L. Krivine. Realizability in classical logic. Unpublished lecture notes (available on the author's web page), 2005.
11. P. Letouzey. A new extraction for Coq. In H. Geuvers and F. Wiedijk, editors, *TYPES*, volume 2646 of *Lecture Notes in Computer Science*, pages 200–219. Springer, 2002.
12. G. Longo and E. Moggi. A category-theoretic characterization of functional completeness. *Theor. Comput. Sci.*, 70(2):193–211, 1990.
13. Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science*. Oxford University Press, 1994.
14. P.-A. Melliès and B. Werner. A generic normalisation proof for pure type systems. In E. Giménez and C. Paulin-Mohring, editors, *TYPES*, volume 1512 of *Lecture Notes in Computer Science*, pages 254–276. Springer, 1996.
15. A. Miquel. *Le calcul des constructions implicite: syntaxe et sémantique*. PhD thesis, Université Paris 7, 2001.
16. C. Paulin-Mohring. Extracting $F\omega$'s programs from proofs in the calculus of constructions. In *POPL'89*, pages 89–104, 1989.
17. C. Paulin-Mohring. *Définitions Inductives en Théorie des Types d'Ordre Supérieur*. Habilitation à diriger les recherches, Université Claude Bernard Lyon I, 1996.
18. The Coq Development Team (LogiCal Project). The Coq Proof Assistant Reference Manual – Version 8.1. Technical report, INRIA, 2006.
19. T. Streicher. *Semantics of Type Theory*. Birkhäuser, 1991.