1.2

Introduction

In this paper, we present a theory of constructions for higher-order intuitionistic logic. The original inspirations were P. Martin-Löf Intuitionistic Theory of Types, J.Y. Girard's system $F\omega$, N.G. de Bruijn's Automath, and some ideas of G. Huet about a higher-order notion of Horn clauses. A first version of this calculus appeared in [17] and an early implementation, written by G. Huet in CAML, is presented in [20, 21].

After general motivations, we give a formal presentation of the present system of constructions. We present a skectch of different semantics. The third part is a survey of the main properties of this calculus (and connections with other logics). Finally, we discuss some possible extensions.

1 Informal motivation

A first attempt at formulating a system aiming at a representation of the full intuitionistic theory of finite types was P. Martin-Löf 1971 version of Type Theory [57]. However, this theory was based on the strongly impredicative axiom that there is a type of all types, and this assumption was shown to be inconsistent by J.Y. Girard [30]. This discovery led Martin-Löf to formulate theories that do not contain any more intuitionistic simple type theory [60]. We will present here a version of type theory that expresses naturally Heyting's semantics of intuitionistic higher-order logic.

Let us recall first what were Martin-Löf's motivation for having a type of all types. Three principles entail this idea.

- we want quantification over predicates and propositions,
- Russell's doctrine of types: the range of significance of a propositional function forms a type,
- the identification of propositions and types.

Indeed, by the first and second points, the collection of all propositions must form a type. If propositions and types are identified, then this type is also the type of all types.

The idea will be to keep the first and the second principles, but to restrict the third point. We retain only the principle that to each proposition corresponds a certain type, namely, the type of its proofs.

A proposition will be characterised by the way it can be proved. The proofs of a given proposition are to be considered as mathematical objects (cf. the "proof-as-objects" paradigm of [10]), and they must form a type, which is the type of proofs of the given proposition. To any proposition φ is thus associated the "type" of its proofs $\mathsf{T}(\varphi)$. Furthermore, the possibility of doing quantification over propositions, predicates, forces the introduction a type Prop of propositions. Heyting's semantics of the universal quantification can then be expressed by: if A is a type, φ a propositional function over A, that is an object of type $A{\to}\mathsf{Prop}$, to give a proof of the proposition $(\forall x : A)\varphi(x)$ is to give a function f such that for any object a of type A, $f(a)$ is a proof of $\varphi(a)$. Thus, $\mathsf{T}((\forall x : A)\varphi(x))$ has to be isomorphic to the product $(x : A)\mathsf{T}(\varphi(x))$ of the dependent family of type $\mathsf{T}(\varphi(x))$ over A.

Let us define $\varphi \Rightarrow \psi$ as $(\forall x : \mathsf{T}(\varphi))\psi$. A quick study of the type system we get shows that it contains the second-order calculus of Girard-Reynolds. For example, the generic identity will be the term $(\lambda A : \mathsf{Prop})(\lambda x : \mathsf{T}(A))x$ of type $(A : \mathsf{Prop})\mathsf{T}(A){\to}\mathsf{T}(A)$. It contains also naturally an intuitionistic version of simple type theory [13].

The nice point is that we do not need to state any further axioms or deduction rules. The logic becomes simply the expression of the fact that a proposition is true if, and only if, its corresponding type of proofs is inhabited.

We have thus extended Howard's theory of construction to an intuitionistic version of higher-order logic [43, 13]. In particular, this system leads to some new facts about the interpretation of the connector $\exists$, problem mentioned in [43], and will allow us to discuss the problem of semantics of evidence of classical second-order logic (see [14] for a discussion about the semantics of evidence).

2 The system (terms and typing rules)

2.1 Formal presentation

The terms have the following inductive structure.

1. identifiers,
2. the special constant Prop,
3. typed abstractions $(\lambda x : M)N$ and hypothetical formation $(\Lambda x{:}M)N$,
4. product, that we shall write $(x{:}M)N$, where M and N are terms, and universal quantification $(\forall x{:}M)N$,
5. application $M(N)$ and instantiation $\mathsf{app}(M, N)$ where M and N are terms.
6. type formation $\mathsf{T}(M)$, where M is a term.

We adopt the following notations: we write $M(M_1, ..., M_p)$ for $M(M_1)...(M_p)$. If the variable x does not appear free in N, we write $(M)N$ for $(x{:}M)N$ [1].

An *assignment*, or *context* is a list of pairs (x, t) where t is a term, and x an identifier. We shall write $\mathcal{V}(B)$ for the set of identifiers which appear in an assignment B. We define the following ordering relation on assignments: $B \subset B'$ which means that if (x, t) appears in B, then it appears in B' (i.e. the set which corresponds to B is a subset of the one which corresponds to B').

We define simultaneously what is a valid assignment, and when a term M is well-formed in an assignment B of type N, relation which is written $M : N\ [B]$. If $x \in \mathcal{V}(B)$, we denote by B_x the term such that (x, B_x) appears in B, and that is the first occurrence in the list B of the form (x, t). It should be noted that the definition of the entailment relation that follows is an instance of an inductive definition. It is a calculus of derivations of judgements of the three possible forms B is valid, T type $[B]$ and $M : T\ [B]$.

2.2 Type Inference Rules

Before reading the formal rules, it may be helpful to have in mind the following special interpretation: read M type as "M is a (finite) set", $x : A$ as "x belongs to the set A", and Prop as the set with two elements 0.1. $\mathsf{T}(0)$ as the empty set, and $\mathsf{T}(1)$ the singleton set 0. A valid assignement of length n will be interpreted as a set of n-tuples. All the rules we give receive then a straigthforward interpretation.

We have first the axiom Prop type $[B]$, if B is a valid assignment. The second axiom is that $x : B_x\ [B]$, if B is valid, and x is an identifier which appears in B. We have then the rule of context formation.

[1] We have to say a word about the problem of bound variables. We propose the following solution: use De Bruijn indexes (cf. [8]) for bound variables, but keep identifiers for free variables as in [18].

$$\frac{M \text{ type } [B]}{B, x : M \text{ is valid}}$$

In this rule, we must impose on x that it is an identifier which does not appear in B, and B may be empty.

$$\frac{M \text{ type } [B] \qquad N \text{ type } [B, x : M]}{(x : M)N \text{ type } [B]}$$

$$\frac{\varphi : \mathsf{Prop}\ [B]}{\mathsf{T}(\varphi) \text{ type } [B]}$$

$$\frac{P \text{ type } [B] \qquad Q \text{ type } [B, x : P] \qquad t : Q\ [B, x : P]}{(\lambda x : P)t : (x : P)Q\ [B]}$$

$$\frac{M \text{ type } [B] \qquad \varphi : \mathsf{Prop}\ [B, x : M]}{(\forall x : M)\varphi : \mathsf{Prop}\ [B]}$$

$$\frac{A \text{ type } [B] \qquad \varphi : \mathsf{Prop}\ [B, x : A] \qquad b : \mathsf{T}(\varphi)\ [B, x : A]}{\Lambda x : A.b : \mathsf{T}((\forall x : A)\varphi)\ [B]}$$

$$\frac{A \text{ type } [B] \qquad P \text{ type } [B, x : A] \qquad c : (x : A)P\ [B] \qquad a : A\ [B]}{c(a) : [a/x]P\ [B]}$$

$$\frac{A \text{ type } [B] \qquad \varphi : \mathsf{Prop}\ [B, x : A] \qquad t : \mathsf{T}((\forall x : A)\varphi)\ [B] \qquad a : A\ [B]}{\mathsf{app}(t, a) : \mathsf{T}([a/x]\varphi)\ [B]}$$

$$\frac{t : P\ [B] \qquad P = Q\ [B]}{t : Q\ [B]}$$

Here the equality used in $P = Q\ [B]$ is β-conversion, which can be defined directly at a syntactic level on the raw terms. The need for conversion rules is for instance explained in [60, 81].

We can define as well β-reduction, and then prove it is Church-Rosser by Tait's method (see [57]). This is used in the proof of the next result.

Lemma: (uniqueness of products) if $(\forall x : A_1)\varphi_1 = (\forall x : A_2)\varphi_2 : \mathsf{Prop}\ [B]$ then $A_1 = A_2\ [B]$ and $\varphi_1 = \varphi_2 : \mathsf{Prop}\ [B, x : A_1]$.

The following propositions are proved by induction. The meta-variables $E, E', ...$ will be used to denote arbitrary judgements.

Proposition 1: If B, B' are valid, $B \subset B'$, $E\ [B]$ then $E\ [B']$.

This allows us to consider assignment as sets, and not as lists.

Proposition 2: If $M : P\ [B]$, then $P \text{ type } [B]$.

Proposition 3: If $M : P\ [B]$, and $M = N : P\ [B]$, then $N : P\ [B]$.

Proposition 4: If $M : P\ [B]$, $M : Q\ [B]$ then $P = Q\ [B]$.

From this, we derive that any judgement has at most one derivation, if we identify terms up to β-conversion.

This expresses the "uniqueness" of types of a well-formed term (though this uniqueness is only modulo conversion).

Definition 3: Let B be a valid assignment, a *type* (of B) is a term M in B such that M type $[B]$. A *proposition* of B is a term φ such that $\varphi : \mathsf{Prop}$ $[B]$. A *small type* is a type of proofs of some proposition, that is a type of the form $\mathsf{T}(\varphi)$, with $\varphi : \mathsf{Prop}$. If φ is a proposition of B, a *proof* of φ is a term M such that $M : \mathsf{T}(\varphi)$ $[B]$, and we say that φ is *true* or *valid* in B if, and only if, φ has a proof.

One may think of Prop as a type of names of small types. Notice that the small types are intuitively closed by product over families of small types. If A is a type, and $B : (A)\mathsf{Prop}$, that is B is a family over A of names of small types, then the product $(x : A)\mathsf{T}(B(x))$ is intensionally isomorphic to the small type $\mathsf{T}(\forall x : A.B(x))$. It should then be clear from this remark in what way our system contains the second-order λ-calculus of Girard-Reynolds.

Let us give some examples to illustrate this point. We write Void for $(\forall\alpha : \mathsf{Prop})\alpha$, so that $\mathsf{Void} : \mathsf{Prop}$ (and we shall see that $\mathsf{T}(\mathsf{Void})$ is indeed empty). We define $\varphi{\rightarrow}\psi$ as being $(\forall x : \mathsf{T}(\varphi))\psi$, so that $\varphi{\rightarrow}\psi : \mathsf{Prop}$ if $\varphi : \mathsf{Prop}$ and $\psi : \mathsf{Prop}$. We write Unit for $(\forall\alpha : \mathsf{Prop})\alpha{\rightarrow}\alpha$, so that $\mathsf{Unit} : \mathsf{Prop}$ (and it is possible to show that the only element of type $\mathsf{T}(\mathsf{Unit})$ is $id = (\Lambda\alpha : \mathsf{Prop})(\Lambda x : \mathsf{T}(\alpha))x$, that is, the polymorphic identity).

Finally, the following notion is the expression of intuitionistic consistency.

Definition 4 We say that a valid context B is *consistent* if, and only if there exists a proposition φ in B which is not provable in B.

2.3 Some syntactic remarks

As we noticed already, our presentation is not minimal, and we have to relate it to the more concise presentations of [17, 20] (see below). Mainly since the work of [84, 77], we understand better the fact that we can present typed calculus in so many different ways. The important distinction is between *derivations* and *judgement*. A sound requirement for a calculus is that any judgement has at most one derivation. This is the case here, provided we identify terms up to β-conversion, and this was also the case for the formalisms presented in [17, 20].

Furthermore, the calculus presented here is more explicit than the one in [17, 20], so that there is a "forgetting map" between derivations in the present version in derivations of [17, 20]. It is then possible to define by induction on the derivations a "section" of this forgetting map which is an interpretation of the implicit calculi in the explicit system, and this defines as well an interpretation of implicit judgement in explicit judgements. The best way to give a semantics of the implicit calculi, is then first to define a syntactic interpretation of the implicit calculi in the explicit calculus, and then the semantics of the explicit calculus. See [84] for details. There are potential problems if we take untyped λ-abstraction (see the examples in [77]).

Notice that the Church-Rosser property plays a crucial role when we prove that any judgement has at most one derivation (in the uniqueness of products lemma above). All the troubles come because our syntax for terms is still not explicit enough. In general as explained in [84], the terms have to be still more explicit, i.e. we have to put explicitly types in the application. The syntax may even be made explicit in such a way that terms become identical to derivations, up to conversions [77]. Of course, the syntax becomes then more cumbersome, since the application becomes now something like $\mathsf{app}(A, \lambda x : A.B, t, u)$. However, for theoretical study, this calculus has to be preferred, and the Church-Rosser property is not needed any more at this level. It will be needed when we try to translate from the implicit system in the explicit one, since we give a translation of derivations, and the Church-Rosser property seems then crucial to establish that a given judgement has at most one derivation, see [84].

Whenever there is no possible confusion, we write φ for $\mathsf{T}(\varphi)$ if $\varphi : \mathsf{Prop}$, and we identify the

two λ-abstractions and the two applications. We will use the alternative notation $A \to B$ for $(A)B$, with A, B types, and $\varphi \Rightarrow \psi$ for $(\forall x : \mathsf{T}(\varphi))\mathsf{T}(\psi)$, with $\varphi, \psi : \mathsf{Prop}$. We consider also informally that Prop is included in the collection of types, so that we can talk of a small type for something which is of type Prop. There is a similar "abus de langage" in category theory with the notion of small complete category (see [41]). A final remark: the work of [84, 77] may be seen as an indication that all these "abus de langage" are more subtle than they seem, and need a careful meta-mathematical study. Essentially, the problem is that when we use such facilities, we are working in an implicit system in which the uniqueness property of derivations w.r.t. judgements may fail. We will however not carry this discussion any longer here, and limit ourselves to indicating this difficulty.

Thus, we suppose that we are working with the simplified system: first $B \vdash x : B_x$ and $\mathsf{Prop}\ \mathsf{type}\ [B]$ if B is valid, and

$$\frac{A\ \mathsf{type}\ [B]}{B, x : A\ \text{ is valid}} \qquad \frac{A : \mathsf{Prop}\ [B]}{B, x : A\ \text{ is valid}}$$

$$\frac{P\ \mathsf{type}\ [B, x : A]}{(x : A)P\ \mathsf{type}\ [B]} \qquad \frac{P : \mathsf{Prop}\ [B, x : A]}{(x : A)P : \mathsf{Prop}\ [B]}$$

$$\frac{M : P\ [B, x : A]}{\lambda x : A.M : (x : A)P\ [B]} \qquad \frac{N : (x : A)P\ [B] \quad M : A\ [B]}{N(M) : [M/x]P\ [B]}$$

$$\frac{M : P\ [B] \quad Q\ \mathsf{type}\ [B] \quad P =_\beta Q}{M : Q\ [B]} \qquad \frac{M : P\ [B] \quad Q : \mathsf{Prop}\ [B] \quad P =_\beta Q}{M : Q\ [B]}$$

As explained above, there is a syntactic translation of this calculus into the more explicit calculus, built by induction on the derivation. This justifies to use the more explicit version when we give the semantics, and to use (and implement) the more implicit version in the examples.

We have just seen that we can think of Prop as a collection of small types. The type Prop can thus be thought of as a "universe" of Martin-Löf type theory. The essential difference here is that this universe is closed under any products, not only products over small types. For instance, $(X : \mathsf{Prop})X$ is still a small type. The meaning of $(X : \mathsf{Prop})X$ involves a typical circularity: it is an object of type Prop but also defined by a quantification over Prop (see [75]). In particular, we cannot think of small types as sets ([74]).

Despite this fact, we have, by a suitable generalisation of Girard's argument [30, 17, 48].

Proposition 5: The calculus of constructions is strongly normalisable.

Corollary: the judgements B is valid, $P\ \mathsf{type}\ [B]$, $M : P\ [B]$ are decidable.

See the arguments of [57] for a proof of the corollary. Furthermore, this gives us an algorithm for testing the conversion of two typed terms: we compute the normal forms of the two terms (by β-reduction), and then compare syntactically these two normal forms.

It is possible to define a system in which we restrict the product formation only for a family of small types only over a small types. The axiom for $\forall$ becomes then

$$\frac{M : \mathsf{Prop}\ [B] \qquad \varphi : \mathsf{Prop}\ [B, x : \mathsf{T}(M)]}{(\forall x : \mathsf{T}(M))\varphi : \mathsf{Prop}\ [B]}$$

$$\frac{A : \mathsf{Prop}\ [B] \quad \varphi : \mathsf{Prop}\ [B, x : \mathsf{T}(A)] \quad b : (x{:}\mathsf{T}(A))\mathsf{T}(\varphi)\ [B]}{\Lambda(b) : \mathsf{T}((\forall x : \mathsf{T}(A))\varphi)\ [B]}$$

The system we get is then very close to some Automath systems [7, 36].

We will need also the following examples (see [6, 20]), given in the implicit syntax.

$$
\begin{array}{rcl}
\bot & = & (A{:}\mathsf{Prop})A \\
\neg(A) & = & A{\to}\bot \quad [A : \mathsf{Prop}] \\
\mathsf{Bool} & = & (A{:}\mathsf{Prop})A{\to}A{\to}A \\
\mathsf{true} & = & \lambda A{:}\mathsf{Prop}.\lambda x, y{:}A.x \\
\mathsf{false} & = & \lambda A{:}\mathsf{Prop}.\lambda x, y{:}A.y \\
\mathsf{Nat} & = & (A{:}\mathsf{Prop})A{\to}(A{\to}A){\to}A \\
0 & = & \lambda A{:}\mathsf{Prop}.\lambda x{:}A.\lambda f{:}A{\to}A.x \\
S & = & \lambda n{:}\mathsf{Nat}.\lambda A{:}\mathsf{Prop}.\lambda x{:}A.\lambda f{:}A{\to}A.f(n(A,x,f)) \\
\mathsf{Eq}(A,x,y) & = & (P{:}A{\to}\mathsf{Prop})P(y){\Rightarrow}P(x) \quad [A\ \mathsf{type}, x, y : A] \\
A \times B & = & (C{:}\mathsf{Prop})(A{\to}B{\to}C){\to}C \quad [A, B : \mathsf{Prop}] \\
(x,y) & = & \lambda C{:}\mathsf{Prop}.\lambda z{:}A{\to}B{\to}C.z(C,x,y) \quad [A, B : \mathsf{Prop}, x : A, y : B] \\
\pi_1(z) & = & z(A, \lambda x{:}A.\lambda y{:}B.x) \quad [A, B : \mathsf{Prop}, z : A \times B] \\
\pi_2(z) & = & z(B, \lambda x{:}A.\lambda y{:}B.y) \quad [A, B : \mathsf{Prop}, z : A \times B]
\end{array}
$$

2.4 Relationship with Automath languages

It turns out that the calculus presented here is mainly equivalent to the Automath version of [7] but where we allow quantification over (De Bruijn notion of) "type" as well, and this possibility was actually considered by De Bruijn (in the section 12.6). What follows shows the (maybe) surprising result that we still get a meaningful calculus by allowing this quantification (see also [9]).

A difference is that we did not take η-conversion. However, as noticed in [39], we have the following result:

Proposition: if $\lambda y : A.M(y) : T\ [B]$ and y is not free in M, then $M : T\ [B]$.

For a more precise comparison between all the different systems: Automath, LF, second-order λ-calculus,$\cdots$ we refer to [4].

3 Semantics

We present now a few semantics of the calculus. Besides proving consistency and suggesting some extensions, a semantics may be useful in order to establish independence results. These semantics are defined first for the system with the explicit syntax, and then, by composition of the translation from the implicit syntax to the explicit syntax.

3.1 Proof-irrelevance semantics

A first semantics consists in a classical reading of our rules, which forgets everything about proofs. Types are interpreted as Zermelo-Fraenkel sets, and Prop is interpreted as the two element set $\{0,1\}$. Let $\mathsf{T}(0)$ be the empty set, and $\mathsf{T}(1)$ be one singleton set. Furthermore, the product, the λ-abstraction, the application are interpreted as the ordinary set-theoretic product, function formation and application. Finally, if A is a type, and $\varphi : A{\to}\mathsf{Prop}$ $(\forall x : A)\varphi(x)$ is interpreted as 1 if, and only if, $[\![\varphi]\!](u)$ is 1 for every $u \in [\![A]\!]$. In this semantics, $\mathsf{T}(\forall x : A.\varphi(x))$ will not coincide in general with $(x{:}A)\mathsf{T}(\varphi(x))$. They are in general only sets in one-to-one correspondance (and this justifies the choice of the explicit system).

Notice a subtle point. When we define precisely this semantics, it is given by induction on the *derivation*. So the crucial property that judgements have at most one derivation is needed here. Notice that this argument does not work with a less explicit syntax, for instance in general if we use untyped λ-abstraction, or if we don't use the operators $\mathsf{T}(X)$ (see [77]). See [84] for a solution to the problem of the semantics of the present calculus with untyped abstraction.

This semantics, however simple it is (it can be seen as a higher-order generalisation of the truth-table methods), shows the consistency of our calculus. Indeed, the interpretation of $(\forall\alpha : \mathsf{Prop})\alpha$ is 0, and hence the type $(\alpha : \mathsf{Prop})\mathsf{T}(\alpha)$ cannot be inhabited. This was actually the method used by Gentzen [28] in order to establish the consistency of the simple theory of types. It is completely elementary, that is, it is "finitist" in the sense of Hilbert and can be carried out formally in a logically weak system (for instance, primitive recursive arithmetic). A similar semantics has been used by [82] in order to prove that we cannot derive all Peano axioms in Martin-Löf intuitionistic theory of types without universes. In our case too, we can use this semantics in order to show that the proposition $\mathsf{Eq}(\mathsf{Bool}, \mathsf{true}, \mathsf{false})\Rightarrow\bot$ is not inhabited.

The fact that there exist elementary consistency proofs has to be contrasted with the normalisation theorem, which cannot have an elementary proof. This can be understood intuitively: the normalisation theorem will entail not only the consistency of the pure calculus, that is the consistency of the empty context, but also the consistency of non trivial contexts.

As an example, let us consider a context which expresses the "axiom of infinity". It is the context Inf declaring one small type $A : \mathsf{Prop}$, one constant and one unary operation $a : A, f : A\rightarrow A$, one binary relation transitive and antireflexive $R : A\rightarrow A\rightarrow\mathsf{Prop}, h_1 : (x : A)R(x, x)\Rightarrow\bot, h_2 : (x, y, z : A)R(x, y)\Rightarrow R(y, z)\Rightarrow R(x, z)$ with the hypothesis $h_3 : (x : A)R(x, f(x))$.

Proposition: Inf is consistent.

Proof: Using the normalisation theorem, a purely combinatorial argument can be given. We claim that in the context Inf, by induction on the size of the construction in normal form M, if M has for type a proposition $R(u, v)$, then $u = f^n(a)$,$v = f^p(a)$ with $n < p$, and if M is of type A, then M is of the form $f^n(a)$. Indeed, by induction, such a construction will never use h_1.

It results that in the context Inf, $R(f^n(a), f^p(a))$ is provable only if $n < p$ (conversely, if $n < p$, $R(f^n(a), f^p(a))$ is provable). In particular, at least one proposition is not provable and so Inf is consistent.

Heuristically at least, this says that in the context Inf, we can define an infinity of "provably" distinct elements that are $a, f(a), f(f(a)), \cdots$. Formally, we can follow the development of arithmetic as done in [76] where an integer is interpreted as a class of classes over A. It is possible to do such a construction over an arbitrary type, but we cannot in the general case get all Peano axioms, in particular not the axiom that zero differs from a successor, and the axiom that the successor operation is one-to-one. In the context Inf and over the type A, all Peano axioms are provable (see [3] where such a development is done in a classical framework, and from which is inspired the context Inf). This shows, in an elementary way, that the normalisation theorem implies the consistency of higher-order classical arithmetic, and so, by Gödel's theorem, that the normalisation theorem cannot be proved by means of higher-order arithmetic.

Remark: what we have used here (and is thus not elementary provable) is the result a priori weaker than normalisation that if a small type is inhabited, then it is inhabited by a term in normal form. As noticed in another framework by Kreisel [32], this last result may become elementary inside a context. For instance, in any context which contains $x : (A : \mathsf{Prop})A\rightarrow A$, one direct inductive argument shows that any inhabited small type is inhabited by a term in normal form. We can use the variable x to "freeze" the redexes.

3.2 Realisability semantics

A more informative semantics is the realisability semantics. The basic idea behind these models is quite natural: consider this language really as a programming language, then "at compile time, we forget any type information to get a untyped program". This suggests to start with an untyped universe of combinatory expressions (the "programs") and to interpret a type as a set of such programs.

All this can be formalised. We take for the universe of untyped programs the set U of untyped λ-terms. A small type is interpreted as a subset of U (closed under β-conversion), and an element of a small type is interpreted by the corresponding untyped λ-term we get by forgetting all type informations. The "big" products over a type of a family of small types is interpreted by intersection (here we use impredicativity of set theory: an arbitrary intersection of a family of subsets of a given set is still a subset of this set!). Finally, a product $(x:A)B(x)$ over a small set A is interpreted as the set of λ-terms t such that, if $u \in [\![A]\!]$ then $t(u) \in [\![B(x)]\!][x/u]$ (see [21] for the details, this can be seen as a realisability interpretation of the constructions). For a generic example, $\lambda A : \mathsf{Prop}.\lambda x : A.x$ is interpreted by $\lambda x.x$ and its type $(A:\mathsf{Prop})(A)A$ is interpreted as the set of λ-terms t such that, for any subset A of U (closed under β-conversion), if u is in A, then $t(u) \in A$. This is indeed the case for $t = \lambda x.x$.

This model can be used in order to show the consistency of some extensions of the original calculus. For instance, one may want to add a small type $n : \mathsf{Prop}$, define $N = \mathsf{T}(n)$, together with one constant $O : N$, and one unary operation $S : (N)N$. We also add a recursive operator $\mathsf{rec} : (P : (N)\mathsf{Prop})P(O) \to ((x:N)P(x) \to P(S(x))) \to (x:N)P(x)$ with the new conversion rules

$$\mathsf{rec}(P, a, f)(O) = a \qquad \mathsf{rec}(P, a, f)(S(x)) = f(x, \mathsf{rec}(P, a, f)(x)).$$

It is possible to interpret this extension in the realisability model (but this was already possible, in a trivial way, in the proof-irrelevance model). The simplest way to see it is to add primitive operations to the "untyped programming languages" (i.e. to extend U with appropriate constants and δ rules). Furthermore, in this model, we can check that all Peano axioms are satisfied (and this does not hold in the proof-irrelevance model, where for instance $\mathsf{Eq}(N, O, S(O)) \Rightarrow \bot$ is not provable). Hence, we have shown the consistency of the extension of the pure calculus with Peano axioms.

We get a "more mathematically civilised" presentation of this model by using the notion of D-set of E. Moggi. One difference is that we start with an arbitrary combinatory algebra D instead of the combinatory algebra of untyped λ-terms, and we take partial equivalence relation instead of arbitrary subsets (see [84, 25]).

This semantics has been generalised to universes in [54].

Finally, let us mention that interesting independence results have been obtained by T. Streicher using variations of the realisability models, for building, for instance, models without strong sums of families of small types over a small type [84].

3.3 Model in domain theory

Domain theory has been developped in order to give a semantics of simply typed λ-calculus with fixed-point [78] (and it was realised later that it could be used also to give a semantics of untyped λ-calculus). It is then natural to try to extend this semantics to a non-standard model of a richer type theory, for instance the present theory of constructions, by interpreting a type as a domain. We get the interpretation of a family of types via the remark that there is a natural notion of "approximations" between domains: the embedding-projection pairs. We can thus define a "dependent domain" over a fixed domain D as a Scott-continuous (that is preserving directed

colimits) functor from D, seen as a category, into the category Dom^{EP}, of domains with embedding-projection pairs as morphisms.

We get in this way an interpretation of type theory with products over dependent families. Notice that in this interpretation, any type is inhabited (at least by $\bot$). The main problem is the interpretation of Prop and $\forall$. A candidate for Prop seems to be the "flat" domain $\mathsf{Bool}_\bot$. For a given domain D, we try then to interpret $\forall : (D \to \mathsf{Prop}) \to \mathsf{Prop}$ in the following way: if $f : D \to \mathsf{Prop}$ takes the value $\bot$ then $\forall(f)$ is $\bot$. Otherwise, if it takes the value false, $\forall(f)$ is false. Finally, in the remaining case, f takes only the value true and $\forall(f)$ is defined to be true. Over an infinite flat domain α however the universal quantification $\forall : (\alpha \to \mathsf{Prop}) \to \mathsf{Prop}$ is not "continuous": on each finite approximation f of the constant function $\lambda x.\mathsf{true}$, $\forall(f)$ is $\bot$, but $\forall(\lambda x.\mathsf{true})$ is true. Intuitively, the "Tarskian" computation of $\forall(f)$ which uses an infinity of values of f is not computable. Thus, we cannot use $\mathsf{Bool}_\bot$ as an interpretation of Prop.

3.3.1 Domain of domains

A first solution to this problem has been given by D. Scott [62, 80], using the notion of closures, following a suggestion of P. Hancock and P. Martin-Löf. Another solution using finitary projections has been found [1]. The rough idea is that everything is interpreted as a point of some "big" domain. The main drawback of these models is that they interpret a richer theory which is known to be "inconsistent" as a type theory, since there is a type of all types. Furthermore, these models are "not canonical" in the choice of the "big" domain to start with. However, direct generalisations are very convenient for showing the equational consistency of various extensions of the present calculus (strong sums, recursive types,$\cdots$ see [12]). This seems to indicate that there is nothing wrong with having a type of all types in a programming language which contains already a fixed-point operator [11]. However, what is missing here is an adequacy theorem that relates the operational and denotational semantics (like the one proved in [72] for the language PCF), which is a priori problematic, since the method used ordinary for proving these results is similar to the reducibility method used in the proof of normalisation, and we know that a calculus with a type of all types does not verify the normalisation property [30, 63, 18, 45].

3.3.2 Category of domains

As discovered by J.Y. Girard [31], it is possible to interpret polymorphism in a model where the interpretation of a type is an arbitrary domain (and not as a point of a special "big" domain), so that we get really a "canonical" model of type theory in domain theory. Quite naturally, a dependent family of domains over all domains is interpreted as a continuous functor from Dom^{EP} to Dom^{EP}. Girard used in [31] a somewhat non-standard notion of domain and morphisms between these domains, but in [22], it is shown that his semantics of the product of a functor F is nothing but the domain of sections of the Grothendieck cofibration of F, that is the domain of all continuous (and stable) family (t_X) such that $t_X \in F(X)$ (see [22, 64]). From this remark it is possible to develop a model similar to Girard's one, but with ordinary Scott-domains and continuous maps [22]. What is crucial in these models is that any domain is a directed colimit of finite domains.

The general picture that emerges from the study of this family of models is then that a small type is interpreted as a domain, and a type in general will be interpreted as a category, which shares a lot of properties of domains, but at a categorical level [19]. Typically, the category of domains with embedding-projection pairs as morphisms is such a category (and is the interpretation Dom^{EP} of the large type Prop), but also the category of Scott-continuous functors (that is, functors that preserve filtered colimits). All these categories C will have the property that for any Scott-continuous

functor F from C to Dom^{EP}, the collection of its continuous sections $t_X \in F(X)$ is a domain for the pointwise ordering. This has been done in details for $F\omega$ in [23], with complete algebraic lattices for small types and locally-finitely presented categories for general types. Some elements for a generalisation to Scott-domains or dI-domains are in [19] (see also [85]). One may object generally to these models that they are not faithful to the Curry-Howard notion of proposition as types, since any type is now inhabited (by $\bot$). However, it is possible to define a notion of total object (see the appendix of [31]) to remove $\bot$, and the notion of partial proofs suggested by these models may have some applications. We can also mention that there is no problem here to extend the adequacy theorem of [72] to the present framework (with the system of constructions instead of simple typed λ-calculus).

In [42], besides a careful analysis of what is a categorical model of dependent types, is presented a topos-theoretic formulation of the complete algebraic lattice model, that may be a key step for a conceptual understanding of the general picture. One starting point is that any domain can be seen as a locale, hence as a special kind of topos. The idea would then be that a type in genera has to be interpreted as a logical theory, i.e. a topos. As an application, it is shown that this model contains a type of all types (which corresponds to the locally finitely presented category of left-exact categories). To look for a topos-theoretic version of Girard's model seems to be an interesting problem (see [47] for a topos-theoretic interpretation of stability).

Finally, an important remark is that the constructions of all these domain models of polymorphic systems are all elementary (this elementary character is only lost when we try to define what are the total objects, see the appendix of [31]).

4 A few properties

4.1 Conservativity over $F\omega$

In [29], J.Y. Girard introduced a functional system which is an extension of Gödel system T, in order to generalise the Dialectica interpretation to higher-order arithmetic (see [33]). It is also explained in [29] how this calculus can be seen as a system of natural deduction for intuitionistic higher-order propositional logic. It is thus not surprising that the language we have developed contains $F\omega$ as a subsystem.

In $F\omega$, $\Rightarrow$ is a primitive constant, and we need to introduce the clause that $M \Rightarrow N$ is a term if M and N are terms. We add the rules

$$\frac{M : \mathsf{Prop}\ [B] \qquad P : \mathsf{Prop}\ [B] \qquad N : P\ [B, x : M]}{(\Lambda x : M)N : M \Rightarrow P\ [B]}$$

$$\frac{M : \mathsf{Prop}\ [B] \qquad N : \mathsf{Prop}\ [B]}{M \Rightarrow N : \mathsf{Prop}\ [B]}$$

Furthermore we do not allow the rule of product formation

$$\frac{M\ \mathsf{type}\ [B] \qquad N\ \mathsf{type}\ [B, x : M]}{(x : M)N\ \mathsf{type}\ [B]}$$

if M is a small type. Typically, we cannot form a type like $(A{:}\mathsf{Prop})(P{:}A \to \mathsf{Prop})(x{:}A)P(x) \Rightarrow P(x)$.

A nice and simple result (noticed by V. Breazu-Tannen using a tool developped by Ch. Mohring) is that the calculus of constructions is a conservative extension of $F\omega$. This is shown by giving a "forgetting" map $\mathcal{E}$ from constructions to $F\omega$ (which can be seen as a modified realisability for construction into $F\omega$, see [67]). Intuitively, this map forgets all dependencies. For instance,

$\mathcal{E}((\forall\alpha:\mathsf{Prop})(\forall P:\alpha\to\mathsf{Prop})(\forall x:\alpha)P(x))$ is $(\alpha,\beta:\mathsf{Prop})\alpha\to\beta$ (see [67, 68]). This forgetting map is the identity on terms that are in $F\omega$, and preserves the typing: if $M:N$ in the system of constructions, then $\mathcal{E}(M):\mathcal{E}(N)$ in $F\omega$. From this, we deduce

Proposition If M,N are terms in $F\omega$, then $M:N$ is provable in the calculus of constructions if, and only if, $M:N$ is provable in $F\omega$. Furthermore if M is a type of $F\omega$, then M is inhabited in $F\omega$ if, and only if it is inhabited in the calculus of constructions.

4.2 Connection with Higher-Order Logic

An elegant formal system is "minimal" higher-order logic (this is the system used in [69]). The types and the terms of this system are the same as the terms of Church's simply typed theory [13]. The types are built from the ground type Prop by the arrow operation, written here $\alpha\to\beta$. The terms are built by λ-abstraction $\lambda x:\alpha.M$, by application $M(N)$, and by typed variables. There is a constant $\Rightarrow$: $\mathsf{Prop}\to\mathsf{Prop}\to\mathsf{Prop}$ (we write $\varphi\Rightarrow\psi$ for $\Rightarrow(\varphi,\psi)$) and a "polymorphic" constant $\forall:(\alpha\to\mathsf{Prop})\to\mathsf{Prop}$ (we write as usual $(\forall x:\alpha)\varphi$ for $\forall(\lambda x:\alpha.\varphi)$). A proposition is a term of type Prop.

We can define what is a "true" proposition relatively to a list of "hypotheses" Γ, relation that we write $\Gamma\vdash\varphi\ \mathsf{true}$, inductively as follows: first $\Gamma\vdash\varphi\ \mathsf{true}$ if $\varphi\in\Gamma$, then

$$\frac{\Gamma,\varphi\vdash\psi\ \mathsf{true}}{\Gamma\vdash\varphi\Rightarrow\psi\ \mathsf{true}}$$

$$\frac{\Gamma\vdash\varphi\Rightarrow\psi\ \mathsf{true}\qquad\Gamma\vdash\varphi\ \mathsf{true}}{\Gamma\vdash\psi\ \mathsf{true}}$$

$$\frac{\Gamma\vdash\varphi\ \mathsf{true}}{\Gamma\vdash(\forall x:\alpha)\varphi\ \mathsf{true}}\qquad(*)$$

$$\frac{\Gamma\vdash(\forall x:\alpha)\varphi\ \mathsf{true}}{\Gamma\vdash[t/x]\varphi\ \mathsf{true}}\qquad(**)$$

In the rule (**), it is supposed that t is a term of type α, and in the rule (*), that the variable x does not occur free in Γ.

This logic is known as "minimal" higher-order logic. It may seem quite poor, but (as known already from Russell [75]), other intuitionistic connectives are then definable (see [30, 21], the general principle is that the coding of a connective is the direct expression of its elimination rule). Let us give only here the representation of the existential quantification which will be used later: for a type A and a predicate φ over A we define $\exists x:A.\varphi(x)$ as the proposition $\forall p:\mathsf{Prop}.(\forall x:A.\varphi(x)\Rightarrow p)\Rightarrow p$. If a is an object of type A and b an object of type $\mathsf{T}(\varphi(a))$ then we can build as expected a proof $\mathsf{pair}(a,b)$ of $\exists x:A.\varphi(x)$. However, given an object x in $\mathsf{T}(\exists x:A.\varphi(x))$, there is no way to compute from x its components (this will be proved indirectly below by showing that the existence of the two projections contradicts the consistency result).

Furthermore, we can consider Leibniz' equality [76]: if x,y are two terms of type A, we define $\mathsf{Eq}(A,x,y)$ as $(P:A\to\mathsf{Prop})P(y)\Rightarrow P(x)$. This corresponds to the following rule: if $\mathsf{Eq}(A,x,y)$ and we want to prove the proposition $\varphi(x)$, then it's enough to prove $\varphi(y)$. We thus do not need to introduce a primitive notion of equality as in Martin-Löf type theory.

The system $F\omega$ can be seen as a language for expressing the proofs of this logic. We have seen how to embed $F\omega$ in the pure calculus of constructions. From this, we get a translation of intuitionistic higher-order logic in the system of constructions. From the conservativity result of the calculus of constructions over $F\omega$, we deduce also that this translation is conservative, that

is a proposition φ is provable in intuitionistic propositional higher-order logic if, and only if, it is inhabited as a small type of the system of constructions. This last result can be seen as a (constructive) completeness theorem: a proposition is provable in higher-order logic if, and only if, its translation in the system of constructions (its "semantics") is true. By using the normalisation theorem, we can prove a similar result for intuitionistic first-order logic (see [57]).

Proposition The calculus of constructions is a conservative extension of minimal propositional higher-order logic.

Remark: We thus get another consistency proof for the system of constructions by (elementary) reduction to the consistency of higher-order logic.

Higher-order logic is ordinarily presented with an extensionality axiom, In this case, as explained in [2], it is better to take the (extensional) equality as the only primitive symbol and it is then possible to define all other logical connectives from it (this is the solution chosen in [51] for instance, see also [35] that presents a proof-checker based on this logic). The connections between extensional higher-order logic and topos theory are explained in [51]. It is thus important to relate the intensional presentation of higher-order logic given above to an extensional one.

Such an interpretation was done by R. Gandy in [27]. But the spirit of this translation goes back to Principia [76]. The idea is to define for every type an extensional equality by induction. On the type Prop we take the logical equivalence as equality. On the type $\alpha{\to}\beta$ we take as extensional equality $\mathsf{Eq}_{\alpha\to\beta}(f,g) = \forall x,y:\alpha.\mathsf{Eq}_\alpha(x,y){\Rightarrow}\mathsf{Eq}_\beta(f(x),g(x))$. We then say that an element x of type α is extensional if, and only if $\mathsf{Eq}_\alpha(x,x)$, and we have an interpretation of extensional higher-order logic in intensional higher-order logic.

By composing these two interpretations: topos theory in extensional higher-order logic, and extensional higher-order logic in intensional higher-order logic, we thus get a translation of topos theory inside intensional higher-order logic. There is actually a direct way of building the free topos from intensional higher-order logic: an object of the free topos is defined as a pair of a type A, together with a partial equivalence relation on it $R : A{\to}A{\to}\mathsf{Prop}$. A morphism between (A,R) and (B,S) is a relation $f : A{\to}B{\to}\mathsf{Prop}$ which is "functional" with respect to R and S as in [51].

In [13], Church introduces a ground type ι of individuals. In the system of constructions described so far, we can introduce only "small" types $D : \mathsf{Prop}$. We can then translate faithfully any result of higher-order logic in the context with one proposition variable $D : \mathsf{Prop}$ (notice however that the conservativity over higher-order logic is not clear, and that the previous realisability method will not help).

4.3 Categorical semantics

We put this section here, since categorical "semantics" are really more translation of a type-theoretic formalism in categorical terms than true semantics. We have seen the connection between higher-order logic and topos theory: extensional higher-order logic is the "internal language" of topos theory. This connection is sometimes used in topos theory in order to prove logically a categorical result. We know also that the calculus we consider is richer than type theory: each proposition gives rise to a "small" type, the type of its proofs.

This suggests in a natural way two questions: is there an extensional version of the calculus of constructions, and what is the categorical version of such a calculus? For the first question, it seems possible to introduce a primitive equality type, as in one version of Martin-Löf type theory (the one described in [61], and used in [15]). The system we get has not been studied yet. The new formation, introduction and elimination and conversion rules are the following.

$$\frac{\mathsf{T}(A) \quad x : A \quad y : A}{\mathsf{T}(I(A,x,y))}$$

$$\frac{x = y : A}{r : I(A,x,y)}$$

$$\frac{c : I(A,x,y)}{x = y : A}$$

$$\frac{c : I(A,x,y)}{c = r : I(A,x,y)}$$

The categorical notion that generalises topos theory in that we have an explicit representation of proofs (the principle of "proof-irrelevance" holds in topos theory as well as in classical set theory) seems to be the following one: a locally cartesian closed category with a small complete category [41, 25]. In [25] (where such categories are coined "dictos") an alternative description is given which emphasizes one interesting phenomenon [41]: if we have a small complete category C in a locally cartesian closed category E, then there is a reflection from E to C, so that, intuitively, in these models, each type has "a best approximation" in term of small types or propositions. This fact is also true in the algebraic lattices/ locally finitely presented category model [42]: given a locally finitely presented category its "reflected" complete algebraic lattice is the ideal completion of the canonical preorder associated to any small aubcategory of generators.

The categorical presentation of the intensional theory is described in [42, 49, 50, 25, 84].

4.4 Representation of data types

There are representations of data structures in the system F (originated in [59], see for instance [6, 52, 20]) that we can use as types of individuals. We have given already the coding of the type of booleans and natural numbers.

Let us give only some remark for the case of the natural number. We can define the iterator over small types $\lambda n{:}\mathsf{Nat}.\lambda A{:}\mathsf{Prop}.\lambda x{:}A.\lambda f{:}A{\to}A.n(A,x,f)$, which is reminiscent of the expression of a weak natural number object in category theory. One can then represent primitive recursive functions, via the usual coding of iterations and products [52, 20]. For instance, for the predecessor function we define first the function $F(z) = (S(\pi_1(z)), \pi_1(z))$ of type $(\mathsf{Nat}\times\mathsf{Nat}){\to}(\mathsf{Nat}\times\mathsf{Nat})$ and then $\mathsf{pred}(x) = \lambda x{:}\mathsf{Nat}.\pi_2(x(\mathsf{Nat}\times\mathsf{Nat}, F, (0,0)))$.

Let $\bar{n}$ be the canonical representation of the integer n in the system F, that is $\lambda A{:}\mathsf{Prop}.\lambda x{:}A.\lambda f{:}A{\to}A.f^n(x)$. Let us say that a function f from integers to integers is representable (in the system of constructions) if, and only if, there exists a term $M{:}\mathsf{Nat}{\to}\mathsf{Nat}$ such that, for any integer n, we have $M(\bar{n}) = \overline{f(n)}$. Though it may seem difficult even to represent the predecessor function, the class of functions that we can define of type $\mathsf{Nat}{\to}\mathsf{Nat}$ is actually very large.

Proposition: A function is representable (in the system of constructions) if, and only if, it is provably total in higher-order arithmetic.

Proof: The simplest proof is the one described in [67] (see [32] for the definition of provably total): by using the modified realisability of the system of constructions in $F\omega$ described above, we are reduced to a similar statement for $F\omega$ which was proved in [29] using the Dialectica interpretation and in [67] using a realisability method (from an idea of P. Martin-Löf [59]).

It should be said however that this representation of data types and recursors is not completely understood yet. As emphasised by J.L. Krivine, these representations theorems are *extensional*, and

what matters as much is the *intensional* aspect. But the coding of primitive recursion via iteration and pairing (though all right from a denotational point of view) seems pretty bad intensionally. For instance, the computation of the predecessor of $\bar{n}$ will be in n β-reduction steps (for the normal order evaluation), and it is in one step with a system that has primitive recursion built-in (like Gödel's system T).

It should be noted also that, although we can state the induction principle

$$(P{:}\mathsf{Nat}{\to}\mathsf{Prop})P(0){\Rightarrow}((x{:}\mathsf{Nat})P(x){\Rightarrow}P(S(x))){\Rightarrow}(x{:}\mathsf{Nat})P(x)$$

we have the following negative result:

Proposition: The induction principle over Nat is not provable.

Proof: By the normalisation theorem, we are reduced to show that in the context

$$P : \mathsf{Nat}{\to}\mathsf{Prop}, h_1 : P(0), h_2 : (x : \mathsf{Nat})P(x){\Rightarrow}P(S(x)), n : \mathsf{Nat}$$

there is no term in normal form of type $P(n)$. Indeed a term built from h_1 or h_2 will prove something of the form $P(0)$ or $P(S(x))$ and the *variable* n is not convertible to 0 or to a term of the form $S(x)$ (since they are distinct normal forms).

All these facts are strong motivations for the extension of the system with inductively defined types as primitive (see below, and [71, 24]).

4.5 Inconsistent extensions

In our proof of consistency, the possibility of interpreting a type as a set (in Zermelo-Fraenkel sense) was crucial. Roughly speaking, this is the only known consistency criterion. That is, whenever a functional/logical system does not have such a property, it is likely that some form of Russell's paradox, or Burali-Forti paradox will apply and show the inconsistency.

4.5.1 The system U

The first example is Girard's system U (see [30]), that I will adapt to the present formalism. Following [18], we explain it first for logical systems, and then we derive what it means in term of constructions.

The type structure of minimal higher-order logic is the simply typed λ-calculus (and simply typed λ-calculus was historically created in [13] for that). As noticed in [58], simply typed structure presents unnatural restriction, since we cannot formulate a statement for an arbitrary type. A possible attempt to overcome this restriction is to start with the second-order λ-calculus instead of the simply typed λ-calculus.

We thus enlarge the types of higher-order logic with type variables and product over type variables $\Pi\alpha.T(\alpha)$ where $T(\alpha)$ is a type expression where α may occur as a variable. At the level of terms, we have to introduce an abstraction over type variables, so that $\lambda\alpha.M$ is of type $\Pi\alpha.T(\alpha)$ if M is of type $T(\alpha)$ (with the usual restriction that α cannot occur free in the type of the free variables of M), and instantiation of a term to a given type, so that $M(\tau)$ is of type $[\tau/\alpha]T$ if τ is a type and M a term of type $\Pi\alpha.T(\alpha)$. Notice that the constant $\forall$ is now a constant of type $\Pi\alpha.(\alpha{\to}\mathsf{Prop}){\to}\mathsf{Prop}$. The system we get is called U^- in J.Y. Girard's thesis [30].

But we want also to state (and prove) generic statement. For that, we introduce a new constant $\bigwedge$ which is of type $(\Pi\alpha.\mathsf{Prop}){\to}\mathsf{Prop}$ (and we write $\bigwedge\alpha.\varphi$ for $\bigwedge(\lambda\alpha.\varphi)$). We express thus universal quantification over all types. Another view of this system would have been to consider it as an

extension of the second order λ-calculus, with one special type Prop, and special constants for implications and quantifications.

We add then the new rules of inference:

$$\frac{\Gamma \vdash \varphi \text{ true}}{\Gamma \vdash \bigwedge \alpha.\varphi \text{ true}} \quad (*)$$

$$\frac{\Gamma \vdash \bigwedge \alpha.\varphi \text{ true}}{\Gamma \vdash [\alpha/\tau]\varphi \text{ true}} \quad (**)$$

In the rule (**), it is supposed that τ is a type, and in the rule (*), that the variable α doesn't occur free in Γ.

If we think of our types as sets, it seems clear that we get an inconsistent system: we cannot form an arbitrary product over all sets and get a set (in general it will be "too big"). We have to be careful however since it is not always obvious to translate a set-theoretical result in term of types (for instance, Russell's paradox shows at once that there cannot be a set of all sets, but it's not so clear that there cannot be a type of all types). This intuition is however correct: we can derive a paradox similar to Burali-Forti paradox [30, 18, 45].

It is less clear what is the situation for the system U^- where, intuitively speaking, the power of second-order constructions seems to have no counterpart in the logic. Actually, in [73], Reynolds stated a conjecture that is roughly equivalent to the consistency of U^-, which is the existence of a set-theoretic model of polymorphism. In [74], Reynolds found actually a proof that there is no set0-theoretic model of polymorphism, with a quite general notion of set-theoretic model. This was using a paradox similar to Cantor's paradox It is possible to translate in type theoretic terms the argument of Reynolds. The trick is that the construction of an initial T-algebra (see [74]) which uses in set-theory a "big" equalizer becomes in type theory the expression of an induction predicate over the type which is a weak initial T-algebra (see [24]). This shows that the system U^- is also inconsistent.

A striking corollary of the inconsistency of the system U is that a type system with a type of all types contains non-normalisable terms [30, 18]. Indeed, it is possible to interpret the system U in such a type system in such a way that propositions are interpreted by types, and that a proposition is provable in U if, and only if, its corresponding type is inhabited. In particular, the type $(X : \mathsf{Type})X$ is inhabited. We know that β-reduction preserves typing, and a closed term in $(X : \mathsf{Type})X$ cannot be in head-normal form. Hence, a closed term of type $(X : \mathsf{Type})X$ is not normalisable.

Intuitively, we cannot say that all types are (isomorphic to) small types in a consistent way.

4.5.2 Representation of existence

We have seen in the representation of existential quantification that from a given proof of $\exists x : A.\varphi(x)$, we have "no access to the two components of this proof". We know however (by using the normalisation theorem) that any closed proof of this proposition reduces to one term of the form $\mathsf{pair}(a, b)$ where a is of type A and b a proof of $\varphi(a)$. We can thus try to internalise this remark and to add to the system a "choice operator" which extracts the two components of the proof of an existential statement [43]. Notice that this conflicts with the intuition that we get from the proof-irrelevance (or the realisability, or the domain model). Indeed, the existential type $\exists x : A.\varphi(x)$ becomes them a name for the type $\Sigma x : A.\mathsf{T}(\varphi(x))$, but this type is not "small" in general. For instance, in the domain model, it will be in general a large category, and not at all a domain. There is nothing wrong a priori with having these choice operators if A is itself small.

From these remarks, it is not surprising that we can prove a contradiction from the introduction of the two projections for existential types. This can be done directly as in [18]. It has also been noticed [44, 38] that we can interpret $Type : Type$ in this system by building a small type of all small type $B = \exists x : \mathsf{Prop}.\mathsf{Unit} : \mathsf{Prop}$. We can indeed build two operations $\epsilon : \mathsf{Prop} \to B$ and $E : B \to \mathsf{Prop}$ such that for $A : \mathsf{Prop}$, $E(\epsilon(A))$ is convertible to A (we take $\epsilon(A) = \mathsf{pair}(A, id)$, with $id : \mathsf{Unit}$ its canonical proof, and $E(x) = \pi_1(x)$). Now, we have the following results.
Lemma: If we add to the system of construction a small type $B : \mathsf{Prop}$ together with operations $E : B \to \mathsf{Prop}$, $\epsilon : \mathsf{Prop} \to B$ and with the conversion rule $A = E(\epsilon(A))$ $[A : \mathsf{Prop}]$, then we any small type is inhabited.

Indeed, it is possible to interpret the $Type : Type$ calculus, where the types are interpreted as the small types of the calculus, and $Type$ itself is B. We will see in the next section that we can weaken the hypotheses, and suppose only that we have $(A : \mathsf{Prop})A \Leftrightarrow E(\epsilon(A))$ (but then, we must use the inconsistency of the system U, and not only the one of $\mathsf{T}(:)\mathsf{T}()$).

From this, we deduce that if we introduce the two projections for the existential types, then we can build an element of type $(A : \mathsf{Prop})A$, and such an element cannot be normalisable (for there is no normal form of type $(A : \mathsf{Prop})A$ and reduction preserves typing).

This shows that for an impredicative theory of constructions, we cannot have an existential quantification with choice operators. This is to be contrasted with a predicative version of a theory of constructions, such as Martin-Löf Type Theory [60], where existence is presented with two projections (this is motivated by the constructive meaning of existence).

This property can be interpreted by saying that the use of the existential type "hides" some information in such a way that it is impossible to get it back. It has been suggested by J. Mitchell and G. Plotkin [65] to use existential types for the representation of abstract data types.

4.5.3 Semantics of evidence for classical logic

It seems clear that we can extend the interpretation of propositions as sets for intuitionistic propositional logic to classical logic simply by adding a special element $?_A : A + \neg A$. The goal of this section is to show that such an addition for an impredicative logic will trivialise any explicit considerations of proofs, so that the proof-irrelevance principle follows from the assumption that the logic is classical and impredicative.

The intuition behind this result is the following. It has been noticed by Spector [83] that if we add to predicative analysis the axiom of choice and the law of excluded middle, then we get a logic as strong as impredicative analysis. We can thus expect to get two levels of impredicativity if we add the magic element $?_A : A + \neg A$, and we can then apply the inconsistency of the system U to ge the inconsistency of this extension (with the "propositions-as-types" meaning, that is all types become inhabited).

Lemma: The following context

$$B : \mathsf{Prop},\ E : B \to \mathsf{Prop},\ \epsilon : \mathsf{Prop} \to B,\ H : (A : \mathsf{Prop})A \Leftrightarrow E(\epsilon(A)),$$

is inconsistent.

Intuitively, this says that there cannot be any "reflection" from the "category" of small types into one small type.

Proof: The proof of this lemma is by a direct interpretation of the system U in that context. The small type B is used for the representation of the type of truth-value of the system U and in general the types of

the system U are interpreted by small types. We define $\Rightarrow_1 : B \to B \to B$, $\forall_1 : (A : \mathsf{Prop})(A \to B) \to B$, and $\bigwedge : (\mathsf{Prop} \to B) \to B$.

$$\begin{aligned} \Rightarrow_1 &= (\lambda p, q : B)\epsilon(E(p) \to E(q)) \\ \forall_1 &= (\lambda A : \mathsf{Prop})(\lambda f : A \to B)\epsilon((x : A)E(f(x))) \\ \bigwedge &= (\lambda F : \mathsf{Prop} \to B)\epsilon((A : \mathsf{Prop})F(A)). \end{aligned}$$

This gives an interpretion of the system U. We thus get that for any $p : B$, $E(p)$ is inhabited. In particular, if $A : \mathsf{Prop}$, $E(\epsilon(A))$ is inhabited, and so A is inhabited. This means that the given context is inconsistent.

In order even to state the next result, we need to extend the calculus of constructions with disjoint sums à la Martin-Löf. This means that we add a new type forming operation $A + B$ type, if A type and B type, with the rules that $i(M) : A + B$ if $M : A$, $j(N) : A + B$ if $N : B$. Finally, if A type, B type, $C(z)$ type $[z : A + B]$, then

$$\frac{M(x) : C(i(x))[x : A] \quad N(y) : C(j(y))[y : B] \quad P : A + B}{\mathsf{if}(P, M, N) : C(P)}$$

with the conversions rules that $\mathsf{if}(i(x), M, N) = M(x) : C(i(x))\ [x : A]$ and $\mathsf{if}(j(y), M, N) = N(y) : C(j(y))\ [y : B]$.

Proposition: In the calculus of construction with disjoint sums, the existence of a classical operator $? : (p{:}\mathsf{Prop})\mathsf{T}(p) + \mathsf{T}(\neg(p))$ implies that any small type has at most one element for Leibniz equality.

Proof: We consider the small type of booleans $\mathsf{Bool} = (A{:}\mathsf{Prop})A \to A \to A$, with $\mathsf{true} = (\lambda A : \mathsf{Prop})(\lambda x, y : A)x : \mathsf{Bool}$ and $\mathsf{false} = (\lambda A : \mathsf{Prop})(\lambda x, y : A)y : \mathsf{Bool}$. It is enough to show that $\mathsf{Eq}(\mathsf{Bool}, \mathsf{true}, \mathsf{false})$, where the equality used is Leibniz equality, since for any small type A and any $a, b : A$, we can define $f : \mathsf{Bool} \to A$ such that $f(\mathsf{true})$ is convertible to a, and $f(\mathsf{false})$ is convertible to b.

If we have $? : (p{:}\mathsf{Prop}).\mathsf{T}(p) + \mathsf{T}(\neg(p))$, we are in classical logic and we can reason by cases. Hence, we can suppose that $\neg(\mathsf{Eq}(\mathsf{Bool}, \mathsf{true}, \mathsf{false}))$. We then show $\bot$ by reduction to the previous lemma.

We define the map $E : \mathsf{Bool} \to \mathsf{Prop}$ by $E(p) = \mathsf{Eq}(\mathsf{Bool}, p, \mathsf{true})$, for $p : \mathsf{Bool}$. For the map $\epsilon : (\mathsf{Prop})\mathsf{Bool}$, we consider, for $p : \mathsf{Prop}$, $?(p) : \mathsf{T}(p) + \mathsf{T}(\neg(p))$, and we take

$$\epsilon(p) = \mathsf{if}(?(p), (\lambda x : \mathsf{T}(p))\mathsf{true}, (\lambda x : \mathsf{T}(\neg(p)))\mathsf{false}).$$

We can then show, by using the hypothesis $\neg(\mathsf{Eq}(\mathsf{Bool}, \mathsf{true}, \mathsf{false}))$, that, for an arbitrary $A : \mathsf{Prop}$, A is equivalent to $E(\epsilon(A))$, hence the result by the lemma.

This implies that the only "model" of such a theory is the "proof-irrelevance" model, where we interpret Prop as a set with two elements, and each type is either empty, or with only one element. In particular, a non-trivial "small complete category" [41] cannot satisfy the choice principle. This says also intuitively that there cannot be any realisability like interpretation of impredicative classical logic (see [14] for such an interpretation for predicative classical logic).

5 Some possible extensions

Does the constructive proof of Gödel's Incompleteness Theorem suggest any reflection principles that could be added to the theory preserving its constructive character? Would this be a way in which an "abstract" theory of proofs might become interesting again? There seems to be quite a number of things to think about this area, and the theory of constructions, in this form or another, gives us a way to making the questions and answers precise (D. Scott [79]).

5.1 Addition of universes

As pointed out in [58], the simple theory of types, although proof theoretically quite strong, has some unnatural limitations: it does not talk about arbitrary set, but instead, talks about individuals or sets of individuals, or sets of sets of individuals, $\cdots$. In [57], the assumption that there is a type of all types which was formulated avoids this problem, but, as we have seen, this is inconsistent with the proof-as-objects idea. There is however another possible solution to this difficulty, which is to use a reflection principle (as in [60]).

For the expression of this reflection principle, we introduce a special type U, called a universe, together with the assumptions that U is closed under products: if A is a type in U and $B : A \to U$, then $(x{:}A)B(x)$ is a type in U, and $\mathsf{Prop} : U$. With this addition, we can introduce type variables $X : U$, then state, and prove, theorems in a generic way in X, like the fact that the inclusion relation between predicate over X is transitive. The consistency of this extension is clear since we can extend the proof-irrelevance semantics to this new calculus, by taking for $[\![U]\!]$ the set V_ω of all hereditarily finite sets. Notice however that this consistency proof is no longer elementary, and indeed, by considering the type of "predicative" Church numerals $(X : U)(X \to X) \to X \to X$, it is possible to interpret higher-order arithmetic in the empty context, so that there cannot be any elementary consistency proof any more. In [54], the realisability model (and in [55, 5], the normalisation proof) is extended to the system with universes.

An interesting problem is to characterize the proof theoretic strength of higher-order logic together with the reflection principle. It seems likely that we can interpret Zermelo set theory by a direct extension of the usual representation of Zermelo set theory with the bounded comprehension axiom in higher-order logic (see [34] for details), to a representation of the full Zermelo set theory in higher-order logic with the reflection principle.

In practice, it seems that we do not need to use more than one or two universe levels. One natural question however is to relate this kind of reflection principle to the one studied by Turing-Feferman [26] (see [46] for examples of the use of the reflection principle). One is tempted to iterate this "universe principle", at first over the integers [61, 18, 54] (in particular notice the slight improvement in the formulation of [54], which introduces explicitely a subtyping relation that expresses really the cumulativity of the universes). G. Huet has proposed a notion of "universe polymorphism" [40, 37]. We then get a system which combines impredicative formations (at the level of propositions), and predicative logic (at the level of types), and we know furthermore from Girard's paradox that such an extension is, in some way "the best possible" [18]. It may be interesting to illustrate ideas from [53] in the system with universe polymorphism, and to try to develop a mechanical study of predicative reasoning (as started in the introduction of the second edition of Principia [76]).

5.2 Strong sums

We have seen that we cannot add strong elimination rules for the existential quantification, so that existential quantification becomes a "strong sum". However, the proof-irrelevance semantics suggests that it is possible to add a strong sum for the types. We add the terms $\Sigma(M, N)$, the pairing operation $\mathsf{pair}_A(M, N)$ and the rules

$$\frac{A \ \mathsf{type} \qquad P \ \mathsf{type}\ [x : A]}{\Sigma(A, P)\ \mathsf{type}}$$

In the following three rules, we assume implicit the hypotheses:
A type and P type $[x : A]$.

$$\frac{M : A \quad N : P\ [x : A]}{\mathsf{pair}_{\Sigma(A,P)}(M,N) : \Sigma(A,P)}$$

$$\frac{Q\ \mathsf{type}\ [z : \Sigma(A,P)] \quad M : (x{:}A)(y{:}P)Q[\mathsf{pair}_{\Sigma(A,P)}(x,y)]}{\mathsf{Elim}(M) : (z : \Sigma(A,P))Q}$$

$$\frac{Q\ \mathsf{type}\ [z : \Sigma(A,P)] \quad M : (x{:}A)(y{:}P)Q[\mathsf{pair}_{\Sigma(A,P)}(x,y)] \quad a : A \quad b : P[a]}{\mathsf{Elim}(M)(\mathsf{pair}_{\Sigma(A,P)}(a,b)) = M(a,b){:}Q[\mathsf{pair}_{\Sigma(A,P)}(a,b)]}$$

As in [61], from these operators, we can define the two projections. Since we have an interpretation of these rules in the proof-irrelevance model, we know that this extension is consistent. We write $\Sigma x : A.B(x)$ for $\Sigma(A,B)$ with A type, and $B : A{\to}\mathsf{Prop}$.

One interesting application of strong sums is the representation of mathematical theories, and this is particularly powerful when combined with universes [70, 56]. A theory will be defined by its carrier part which is a type, and its axiomatisation, which is a predicate over this type. For instance, the theory of reflexive relation is defined by the carrier part $T = \Sigma(U, \lambda A : U.A{\to}A{\to}\mathsf{Prop})$, and the axiomatisation is the predicate $\psi = \mathsf{Elim}(\lambda A : U.\lambda R : A{\to}A{\to}\mathsf{Prop}.(x : A)R(x,x))$ which is of type $T{\to}\mathsf{Prop}$. A reflexive relation can then be seen as an object of type $\Sigma(T,\psi)$. A way to think of this last type is as the "subset" of objects in T that satisfies ψ, or as the type of models of the theory defined by T and ψ.

We can now consider the notion of morphisms, or interpretations, between theories. For instance, we can consider the notion of converse of a relation. We define first a function $\mathsf{conv} : T{\to}T$ by $\mathsf{conv} = \mathsf{Elim}(\lambda A : U.\lambda R : A{\to}A{\to}\mathsf{Prop}.\mathsf{pair}_T(A, \lambda x, y : A.R(y,x)))$. We can then remark that we have a proof of $(x{:}T)\psi(x){\Rightarrow}\psi(\mathsf{conv}(x))$. This interpretation has thus two parts: one part is defined purely at the level of support, and the other part is purely logical.

This seems to be a general phenomenon: an interpretation between two theories $\Sigma(T_1,\psi_1)$ and $\Sigma(T_2,\psi_2)$ will have one carrier part $f : T_1{\to}T_2$, and one logical part in $(x{:}T_1)\psi_1(x){\Rightarrow}\psi_2(f(x))$.

We can think of $\Sigma(T,\psi)$ as the type of models of the theory defined by the pair (T,ψ). Interpretations between two theories (T_1,ψ_1) and (T_2,ψ_2) are the maps $f : T_1 \to T_2$ such that $(x : T_1)(\psi_1(x))\psi_2(f(x))$ holds, so that they are themselves elements of the type of models of the theory defined by $(T_1{\to}T_2, \lambda f{:}T_1{\to}T_2.(x{:}T_1)\psi_1(x){\Rightarrow}\psi_2(f(x)))$.

If we want to internalize this discussion, we have to use two levels of universes. We define the type $\mathsf{Theory} : U_2$ as $\Sigma(U_1, \lambda A : U_1.A{\to}\mathsf{Prop})$. We have an evaluation mapping $\mathsf{Mod} : \mathsf{Theory}{\to}U_1$ defined by $\mathsf{Mod}(Th) = \mathsf{Elim}(Th)(\lambda A : U_1.\lambda\psi : A{\to}\mathsf{Prop}.\Sigma(A,\psi))$. We can now define what is a morphism between two theories $\mathsf{MOR} : \mathsf{Theory}{\to}\mathsf{Theory}{\to}\mathsf{Theory}$ by defining

$$\mathsf{mor}(T_1,\psi_1,T_2,\psi_2) = \mathsf{pair}_{\mathsf{Theory}}(T_1{\to}T_2, \lambda f{:}T_1{\to}T_2.(x{:}T_1)\psi_1(x){\Rightarrow}\psi_2(f(x)))$$

and $\mathsf{MOR}(Th_1,Th_2) = \mathsf{Elim}(Th_1)(\lambda T_1{:}U_1.\lambda\psi_1{:}T_1{\to}\mathsf{Prop}.$
$\mathsf{Elim}(Th_2)(\lambda T_2{:}U_1.\lambda\psi_2{:}T_2{\to}\mathsf{Prop}.\mathsf{mor}(T_1,\psi_1,T_2,\psi_2)))$.

If F is an object in $\mathsf{Mod}(\mathsf{MOR}(Th_1,Th_2))$, and S_1 an object in $\mathsf{Mod}(Th_1)$, then we can apply F to S_1, yielding an object in $\mathsf{Mod}(Th_2)$. That is, we have an object application in

$$(Th_1,Th_2{:}\mathsf{Theory})\mathsf{Mod}(\mathsf{MOR}(Th_1,Th_2)){\to}\mathsf{Mod}(Th_1){\to}\mathsf{Mod}(Th_2).$$

The remark that it was possible to internalize the representation of theories in the calculus itself was explained to me by R. Pollack (from discussions with Z. Luo).

From what we just said, there is no object in

$$(Th_1,Th_2{:}\mathsf{Theory})(\mathsf{Mod}(Th_1){\to}\mathsf{Mod}(Th_2)){\to}\mathsf{Mod}(\mathsf{MOR}(Th_1,Th_2)),$$

so that, a priori, there is a problem to interpret the operation of simple type λ-calculus with theories. However, it is quite possible to define a product operation $\times : \mathsf{Theory} \to \mathsf{Theory} \to \mathsf{Theory}$, and to interpret the equational presentation of cartesian closed category [51] in this framework.

There are three (at least formal) analogies that seem quite interesting: the first is the distinction between runtime and compile time that is discussed in [12] as an argument for avoiding dependent types (for compilation), the other is E. Moggi's notion of modules for SML [66], the last one is the interpretation of the implication in realisability (see [68]).

5.3 Inductive Types

As we noticed before, there are some problems in the impredicative representation of inductive types. It is thus natural to try to extend the core calculus with a primitive notion of inductively defined types. We shall not give here the full theory (see [71, 24]), but only illustrate some points about such an extension.

For the natural number object, we want to introduce one type Nat (a priori, it is not clear whether we want this type to be small or not) with one constant $0 : \mathsf{Nat}$ and one successor operation $S : \mathsf{Nat} \to \mathsf{Nat}$. We then introduce one elimination operator rec with the rule

$$\frac{P(x)\ \mathsf{type}\ [x : \mathsf{Nat}] \quad a : P(0) \quad f : (x{:}\mathsf{Nat})(P(x))P(S(x))}{\mathsf{rec}(a, f) : (x{:}\mathsf{Nat})P(x)}$$

We add the new conversion rules that $\mathsf{rec}(a, f)(0) = a$ and $\mathsf{rec}(a, f)(S(x)) = f(a, \mathsf{rec}(a, f)(x))$, for $x : \mathsf{Nat}$.

With the operator rec, we have simultaneously the possibility of building terms and of proving properties by induction over Nat. Furthermore, since Prop is a type, we can define by induction propositions, predicates,... For instance, we can define the property Z of being equal to 0 by $\mathsf{rec}(\mathsf{true}, \lambda x : \mathsf{Nat}.\lambda y : \mathsf{Prop}.\mathsf{false})$ with $\mathsf{true} = (A{:}\mathsf{Prop})(A)A$ (actually any true proposition will do) and $\mathsf{false} = (A{:}\mathsf{Prop})A$. We can also define the predecessor function as $\mathsf{rec}(0, \lambda x, y : \mathsf{Nat}.x)$. From this, we can deduce the other Peano axioms: the axiom $(x{:}\mathsf{Nat})\neg\mathsf{Eq}(\mathsf{Nat}, S(x), 0)$ follows from the existence of the property Z, and the axiom $(x, y : \mathsf{Nat})\mathsf{Eq}(\mathsf{Nat}, S(x), S(y)) \Rightarrow \mathsf{Eq}(\mathsf{Nat}, x, y)$ follows from the existence of the predecessor function.

If we do not ask Nat to be a small type, then the proof irrelevance semantics gives a consistency proof. As expected, this semantics is not elementary, since we need an infinite set in order to interpret Nat. If we want Nat as a *small* type, then the semantics has to be more subtle than the proof irrelevance semantics, since in this semantics, small types are interpreted as sets with at most one element. It turns out that the realisability interpretation works here, by interpreting Nat as the set of untyped λ-term of the form $\lambda f.\lambda x.f^n(x)$ (or by adding special constants $0, S, \mathsf{rec}$ to the untyped λ-calculus with the new conversion rules $\mathsf{rec}(a, f, 0) = a$ and $\mathsf{rec}(a, f, S(x)) = f(a, \mathsf{rec}(a, f, x))$).

With one universe U, we can internalize the recursion operator as a constant of type $(P{:}\mathsf{Nat} \to U)P(0) \to ((x : Nat)P(x) \to P(S(x))) \to (x : \mathsf{Nat})P(x)$. With this extension, we can also (as in [60]) define by induction families like $P(n) = A^n$, with A type, and we can define, for instance the notion of sums of a n-tuple of integers as a term of type $(n{:}\mathsf{Nat})\mathsf{Nat}^n \to \mathsf{Nat}$, which could be defined only at the meta-level before. The combination of universes and inductive types is thus extremely powerful for the internalisation of meta-arguments (see [46]).

Conclusion

We have presented a possible expression of Heyting's semantics for intuitionistic higher-order logic, and extended for this purpose Howard's theory of constructions. The notion of construction developped here is however still too crude for being suitable for the interpretation of intuitionistic mathematics (in particular we have not considered at all the difficult problem of the representation of choice sequences). The goal of our approach was only to show that a type-theoretic presentation of intuitionistic mathematics is possible (in opposition to a categorical or set-theoretical setting). We hope that it will be interesting to mechanize and illustrate proof-theoretic results (like the realisability method or the Dialectica interpretation) in such a framework and that it will help towards the understanding of the mystery of impredicativity.

Acknowledgments

Serge Yoccoz suggested to me the problem of the semantics of evidence for classical impredicative analysis. I want also thank G. Huet for many useful suggestions, and H. Herbelin, Ch. Paulin-Mohring and L. Colson for proof-reading.

References

[1] R. Amadio, K. Bruce, and G. Longo. "The finitary projection model for second order lambda calculus and solutions to higher order domain equations." Proceeding of the first LICS, Boston, 1986.

[2] P. Andrews. "General Models and Extensionality." JSL, vol 37, No 2, 1972.

[3] P. Andrews. "An introduction to Mathematical Logic and Type Theory: To Truth through Proofs." Academic Press, 1986.

[4] H. Barendregt. "The forest of lambda calculi with types." Talk given at the Workshop of Lambda Calculus and Category Theory, CMU, 1988.

[5] S. Berardi. "A subsystem λ_Z of λ_U without Girard's paradox." Unpublished draft, 1988.

[6] C. Böhm and A. Berarducci. "Automatic synthesis of typed λ-programs on term algebras." Theoretical Computer Science 39, 1985.

[7] N.G. de Bruijn. "The mathematical language AUTOMATH, its usage and some of its extensions." Symposium on Automatic Demonstration, IRIA, Versailles, 1968. Lecture Notes in Mathematics **125**, Springer-Verlag, 1970, pp. 29–61.

[8] N.G. de Bruijn. "Lambda-Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem." Indag. Math. **34,5** (1972), 381–392.

[9] N.G. de Bruijn. "Some extensions of Automath: the AUT-4 family." Internal Automath memo M10 (Jan. 1974).

[10] N.G. de Bruijn. "A survey of the project Automath." (1980) in to H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, Eds Seldin J. P. and Hindley J. R., Academic Press (1980).

[11] R. Burstall and B.Lampson. "Pebble, a Kernel Language for Abstract Data Types and Modules." Information and Computation, Volume 76, 1989.

[12] L. Cardelli. "A Polymorphic λ-calculus with Type:Type." Private communication (1986).

[13] A. Church. "A formulation of the simple theory of types." Journal of Symbolic Logic 5, 1940.

[14] R. Constable "On the semantics of evidence." Manuscript, Marktoberdorf summer school, 1988.

[15] R. Constable & alii. "Implementing mathematics with the Nuprl Proof Deevlopment System." Prentice-Hall edition, 1986.

[16] R.L. Constable and N.P. Mendler. "Recursive Definitions in Type Theory." In Proc. Logic of Programs, Springer-Verlag Lecture Notes in Computer Science **193** (1985).

[17] T. Coquand. "Une théorie des constructions." Thèse de troisième cycle, Université Paris VII (Jan. 85).

[18] T. Coquand. "An Analysis of Girard's Paradox." Proceeding of the first LICS, Boston, 1986.

[19] T. Coquand. "Categories of Embeddings." Proceeding of the third LICS, Edimburgh, 1988.

[20] T. Coquand, G. Huet. "Constructions: A Higher Order Proof System for Mechanizing Mathematics." EUROCAL85, Linz, Springer-Verlag LNCS 203 (1985).

[21] T. Coquand, G. Huet. "The Calculus of Constructions." Information and Computation, Volume 76, 1988.

[22] T. Coquand, C. Gunter, G. Winskel. "Domain Theoretic Models of Polymorphism." Cambridge Technical Report No 116 (1987), to appear in Information and Computation.

[23] T. Coquand, T. Ehrhard. "An equational presentation of Higher-Order Logic." Proceedings of Category theory and computer science, Edimburgh, Springer Lecture Notes in Computer Science, vol. 283, Springer-Verlag, 1987.

[24] T. Coquand, Ch. Paulin-Mohring. "About inductively defined types."' To appear in the proceedings of the Tallin conference, 1988.

[25] Th. Ehrhard. "Une sémantique catégorique des types dépendants." Thèse, Université Paris VII, 1988.

[26] S. Feferman. "Turing in the land of O(z)" Unpublished draft, Stanford University, 1987.

[27] R.O. Gandy. "On the axiom of extensionality-Part I." J.S.L. 21, 1956.

[28] G. Gentzen. "The Collected Papers of Gerhard Gentzen." North Holland, 1969.

[29] J.Y. Girard. "Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et dans la théorie des types." Proc. 2nd. Scand. Log. Symp., North Holland, 1971.

[30] J.Y. Girard. "Interprétation fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieure." Thèse d'Etat, Université Paris VII (1972).

[31] J.Y. Girard. "System F: fifteen years later" Theoretical Computer Science, vol. 45, 1986.

[32] J.Y. Girard. "Proof Theory and Logical Complexity, Part I" Bibliopolis, 1988.

[33] K. Gödel. "On a Hitherto Unexploited Extension of the Finitary Standpoint." Translation by W. Hodges, Journal of Philossphical Logic 9, 1980.

[34] R. Goldblatt. "Topoi: the Categorical Analysis of Logic." North-Holland, 1979.

[35] M. Gordon. "HOL A Machine Oriented Formulation of Higher Order Logic." Cambridge Technical Report, no. 68, 1985.

[36] R. Harper, F. Honsell, G. Plotkin. "A Framework for defining Logics." Proceedings of the second LICS symposium, IEEE, Ithaca, 1987.

[37] B. Harper and R. Pollack. "Type-Checking, Universe Polymorphism, and Typical Ambiguity in the Calculus of Constructions." To appear in the proceedings of CCIPL, 1989.

[38] B. Harper and J. Mitchell. "The Essence of ML." 15th POPL 1988, ACM Press.

[39] L. Hehmink and R. Ahn. "Goal Directed Proof Construction in Type Theory." Unpublished manuscript, Philips Research Laboratory, 1988.

[40] G. Huet. "A calculus with Type:Type." Unpublished manuscript, INRIA, 1987.

[41] M. Hyland. "A Small Complete Category." In proceedings of Church's Thesis Conference, to appear in Ann. Pure. Appl. Logic, 1986.

[42] J.M.E. Hyland, and A.M. Pitts. "The theory of constructions: categorical semantics and topos-theoretic models." in Proceedings of the Boulder Conference, Contemporary Mathematics, AMS, Providence RI, 1988

[43] W. A. Howard. "The formulæ-as-types notion of construction." In Hindley, Seldin To H.B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism, Academic Press, 1980.

[44] J. Hook and D. Howe. "Impredicative strong existential equivalent to type:type." Technical Report, Cornell University, 1986.

[45] D.J. Howe. "The Computational Behaviour of Girard's Paradox." Proceedings of the second LICS symposium, IEEE, Ithaca (1987).

[46] D. Howe. "Automatic Reasoning in an Implementation of Constructive Type Theory." Ph. D. thesis, Cornell University, 1988.

[47] P. Johnstone. "A Topos-Theoretic Look at Dilators." Unpublished manuscript, Cambridge 1988.

[48] B. Jutting. "Normalisation in the system of constructions." Unpublished Manuscript, 1986.

[49] François Lamarche. "A model of the theory of constructions." in Proceedings of the Boulder Conference, Contemporary Mathematics, AMS, Providence RI, 1988

[50] François Lamarche. "Modelling Polymorphism with Categories." Ph. D. thesis, McGill University 1988

[51] J. Lambek and P. J. Scott. "Introduction to higher order categorical logic" Cambridge studies in advanced mathematics, Cambridge University Press, 1986.

[52] D. Leivant. "Reasoning about functional programs and complexity classes associated with type disciplines." 24th FOCS, 1983.

[53] D. Leivant. "Stratified Polymorphism." In Proceedings of the fourth LICS, IEEE, 1989.

[54] Z. Luo. "ECC, an Extended Calculus of Constructions." In Proceedings of the fourth LICS, IEEE, 1989.

[55] Z. Luo. "$CC^{\infty}_{\sqsubset}$ and its Meta Theory." Report ECS-LFCS-88-58, Laboratory for Foundations of Computer Science, Edimburgh, 1988.

[56] D.B. MacQueen. "Using Dependent Types to Express Modular Structure." 13th POPL, ACM Press, 1986.

[57] P. Martin-Löf. "A Theory of Types." Unpublished manuscript, 1971.

[58] P. Martin-Löf. "On the strength of intuitionistic reasoning." International Congress for Logic, Methodology and Philosophy of Science, 1971.

[59] P. Martin-Löf. "A construction of the provable wellorderings of the theory of species." Unpublished manuscript, 1971.

[60] P. Martin-Löf. "An Intuitionistic Theory of Types: predicative part." Logic Colloquium '73, North-Holland, 1975, 73-118.

[61] P. Martin-Löf. "Intuitionistic Type Theory." Bibliopolis, 1984.

[62] N. McCracken. "An investigation of a programming language with a polymorphic type structure." Ph.D. Dissertation, Syracuse University (1979).

[63] A. Meyer and M. Reinholt. "Type is not a type" 13th POPL, ACM Press, 1986.

[64] J. Messeguer. "Relating Models of Polymorphism." 16th POPL, ACM Press, 1989.

[65] J. Mitchell and G. Plotkin. "Abstract types have existential types." 12th POPL, ACM Press, 1985.

[66] E. Moggi. Personal Communication.

[67] Ch. Paulin-Mohring. "Extraction de programmes dans le Calcul des Constructions." Thèse, Université Paris 7, 1989.

[68] Ch. Paulin-Mohring. "Extracting Fω programs from proofs in the calculus of construction." 16th POPL, ACM Press, 1989.

[69] L. Paulson "The Representation of Logic in Higher-Order Logic." Cambridge Technical Report, 113, 1987.

[70] B. Nordström and K. Peterson. "The Semantics of Module Specifications in Martin-Löf's Type Theory." Chalmers Technical Report 36, 1985.

[71] F. Pfenning. "Inductively defined types for the constructions." To appear in Proceedings of MFPLS, 1989.

[72] G. Plotkin. "LCF as a programming language." Theoretical Computer Science 5, 1976.

[73] J. C. Reynolds. "Types, abstraction and parametric polymorphism." IFIP Congress' 83 Paris (1983).

[74] J.C. Reynolds. "Polymorphism is not Set-Theoretic." Lecture Notes in Computer Science 173, Springer-Verlag, 1984.

[75] B. Russell. "The Principles of Mathematics." 1903.

[76] B. Russell and A.N. Whitehead. "Principia Mathematica." Volume 1,2,3 Cambridge University Press, 1912.

[77] A. Salvesen. "Polymorphism and Monomorphim in Martin-Löf's Type Theory." To appear in Logic Colloquium' 88, Padova, 1988.

[78] D. Scott. "A Type-Theoretical Alternative to CUCH, ISWIM, OWHY." Unpublished manuscript, Oxford, 1969.

[79] D. Scott. "Constructive validity." Symposium on Automatic Demonstration, Springer-Verlag Lecture Notes in Mathematics, **125**, Springer-Verlag, 1970, 237–275.

[80] D. Scott. "Data Types as Lattices." SIAM Journal of Computing **5** (1976) 522–587.

[81] J.P. Seldin. "Progress report on generalized functionality." Ann. Math. Logic. 17 (1979).

[82] J. Smith. "Non derivability of Peano Axioms in Type Theory without Universes." Journal of Symbolic Logic, 53(3), 1986.

[83] C. Spector "Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles formulated in current intuitionistic mathematics." Recursive function theory, Proc. Symp. Pure Math., vol. V, A.M.S. Providence, 1962.

[84] Th. Streicher. "Correctness and Completeness of a Categorical Semantics of the Calculus of Constructions." Ph. D. Thesis, Passau, 1988.

[85] P. Taylor. "The Trace Factorisation and Cartesian Closure for Stable Categories." Unpublished manuscript, 1989.