Towards an Implicit Calculus of Inductive Constructions. Extending the Implicit Calculus of Constructions with Union and Subset Types.

Bruno Bernardo

Towards an Implicit Calculus of Inductive Constructions.

Extending the Implicit Calculus of Constructions with Union and Subset Types

Bruno Bernardo

Projet TypiCal
Ecole Polytechnique and INRIA Saclay, France
Bruno.Bernardo@lix.polytechnique.fr

Abstract. We present extensions of Miquel's Implicit Calculus of Constructions (ICC) and Barras and Bernardo's decidable Implicit Calculus of Constructions (ICC*) with union and subset types. The purpose of these systems is to solve the problem of interaction betweeen logical and computational data. This is a work in progress and our long term goal is to add the whole inductive types to ICC and ICC* in order to define a complete framework for theorem proving.

1 Introduction

The Calculus of Inductive Constructions (CIC) [8], the formalism on which the Coq proof assistant [10] is based, is a powerful type system for theorem proving. However, one drawback of this formalism is the interaction between logical and computational subterms. Proofs, type annotations, dependencies are not computationally relevant but their content is inspected by the type checker as if they were. This can lead to a certain lack of flexibility that makes theorem proving harder and less intuitive.

Let us for example consider the euclidean division `div`. Its type is :

```
forall a b, b > 0 -> {q & {r | a = b * q + r /\ b > r}}
```

Given two integers a and b and two proofs Π_1 and Π_2 that $b > 0$, it would quite natural to consider that `div` a b Π_1 and `div` a b Π_2 are equivalent, since these programs have the same computational behaviour. However, since we do not have proof-irrelevance, it will be hard to prove that `div` a b Π_1 $=$ `div` a b Π_2 holds because Π_1 and Π_2 may differ.

In order to distinguish proofs from algorithm, Coq has two sorts Prop and Set that are almost identical regarding typing rules, but that have different purposes. Prop is intended to include types that correspond to logical propositions whereas Set is intended to include datatypes. This distinction is useful for Coq's extraction procedure [4] that removes all the logical data, or more precisely the data whose type is in Prop, and keeps the data whose type is in Set, in order to produce certified programs. Even if Coq's extraction is very powerful, it is useless during the proof elaboration stage. Moreover, since it can only erase terms whose types are in Prop, it cannot remove logical data

whose type is in Set. If we consider lists indexed by their length (also known as vectors), their index will not be erased by the extraction (since we have `nat` : Set), even though this index is a static property and is thus useless for the computation.

Miquel's Implicit Calculus of Constructions (ICC) [7] seems to be a possible solution for the interference between logical and computational parts. ICC is a Curry-style Calculus of Constructions that also have an implicit product $\forall x : T.U$ that behaves as an intersection type rather than a function type. In ICC proofs and dependencies can be implicit and thus not interfere with computational data. Moreover abstractions do not carry type annotations. If we consider again the euclidean division program `div`, the proof Π that `b>0` does not need to appear and our previous problem shall not occur : there will only be one program `div` $a\ b$ whatever is the proof Π of $b > 0$ we have.

However, ICC has one major issue : it is unlikely that type inference is decidable in it. In [2], we have defined ICC*, a more verbose variant that has decidable type checking. We can see ICC* as a layer upon ICC where the implicit parts are made explicit and marked with a flag. ICC* and ICC are linked through an extraction function that removes the static part (annotations and flagged logical parts). This extraction is also used in the typing rules, mainly the conversion one: conversion is made between extracted terms. We designed ICC* so that it captures the nice behaviour of ICC while having decidable type inference.

Our long term goal is to add inductive types to both ICC and ICC* so we can have a more complete framework for theorem proving. Since inductive types can be decomposed into more basic types (sigma types, disjoint sums, fixpoint operators, equality, booleans, unit type and void type), we intend to do it by adding every basic type.

What we present here is a work in progress. First, we describe ICC$_\Sigma$, a version of ICC containing union types and subset types (Section 2), and then we present its decidable counterpart ICC$^*_\Sigma$ (Section 3).

Preliminaries In this paper we will adopt the following usual conventions. We will consider terms up to α-conversion. The set of free variables of a term t will be written $\mathsf{FV}(t)$. Arrow types are explicit non-dependent products (when $x \notin \mathsf{FV}(U)$, we write $T \to U$ for $\Pi x\!:\!T.\,U$). Substitution of the free occurrences of variable x by N in term M is noted $M\{x/N\}$. We will consider a set of sorts $\mathcal{S} = \{\mathsf{Prop}\} \cup \{\mathsf{Type}_i \mid i \in \mathbb{N}\}$ designating the types of types. Prop is an impredicative sort intended to represent the types of logical data whereas sorts Type_i denote the usual predicative hierarchy of the Extended Calculus of Constructions[5]. As in the traditional presentation of Pure Type Systems [1], we define two sets **Axiom** $\subset \mathcal{S}^2$ and **Rule** $\subset \mathcal{S}^3$ by

$$\begin{aligned}\textbf{Axiom} &= \{(\mathsf{Prop}, \mathsf{Type}_0);\ (\mathsf{Type}_i, \mathsf{Type}_{i+1}) \mid i \in \mathbb{N}\}\\ \textbf{Rule} &= \{(\mathsf{Prop}, s, s);\ (s, \mathsf{Prop}, \mathsf{Prop}) \mid s \in \mathcal{S}\}\\ &\quad \cup \{(\mathsf{Type}_i, \mathsf{Type}_j, \mathsf{Type}_{max(i,j)}) \mid i, j, \in \mathbb{N}\}\end{aligned}$$

Note that these sets are functional and complete.

We will consider two judgements $\Gamma \vdash$ and $\Gamma \vdash M : T$. $\Gamma \vdash$ means that the context Γ is well-formed and $\Gamma \vdash M : T$ that M has type T under the context Γ.

2 ICC$_\Sigma$

ICC$_\Sigma$ is an extension of the fragment of ICC we used in [2]. Let us first recall briefly a few points about this fragment (cf. Fig1). More details about ICC's syntax can be found in [7] and [6].

2.1 Syntax of ICC

ICC is a Curry-style version of the Calculus of Constructions that also features an implicit product $\forall x : T.U$. This implicit product is interpreted as an intersection type: if we have $f : \forall x : T.U$, then we also have $f : U\{x/N\}$ for any well-typed $N : T$. (cf. rule (INST)). β- and η- reductions are defined in ICC . β-, η- and $\beta\eta$- reductions have the Church-Rosser property. Regarding typing rules, we take a restriction of the system described in [7]: we remove the (CUM),(EXT) and (STR) rules. (CUM) and (EXT) are related to subtyping, which we have not implemented yet. (STR) lets a proof of $\forall \_ : P.Q$ also be a proof of Q even if no proof of P is produced and such behaviour is not wanted. With these typing rules, $\beta\eta$ subject reduction holds. Type inference seems to be undecidable since it contains Curry-style System F, whose type inference is undecidable [11]. Coherence and strong normalization of the system are proven semantically (cf. subsection 2.3).

2.2 Adding sigma-types

In ICC$_\Sigma$, we have added a subset type $\{x : A \mid P\}$ and an union type $\exists x : A.B$ to ICC . Terms of subset type can be seen as dependent pairs where the second component (the proof that the property P holds) is implicit. Terms of union type can be seen as dependent pairs where the first component (the witness) is implicit.

There are six more typing rules (cf. Fig. 2). (SUB) and (U) are formation rules. They are very similar to the product formation rules except that every object is in the same sort s. The reason of this restriction is that the extension of the model has yet to be defined. In the near future, this restriction shall be removed.

There are two introduction rules : (SUB-I) and (U-I). (SUB-I) states that, in some context Γ, any a of type A has also type $\{x : A \mid B\}$ provided there is a proof b of type $B\{a/x\}$. (U-I) states that in some accurate context Γ, if b has type $B\{a/x\}$ with some $a : A$, then b has also type $\exists x : A.B$.

Finally elimination rules (SUB-E) and (U-E) allow us to produce an object of any sort s from an object of union or subset type.

There is no need to add any reduction rule. No projection is definable for terms of union type : the first argument is implicit so there is no first projection; since the second projection depends on the first one, we do not have a second projection either. The first projection of terms of subset type can be emulated with the (SUB-E) rule by choosing $P = \lambda x.\, A$ and $f = \lambda x.\, x$.

Church-Rosser still holds for β-, η- and $\beta\eta$- reductions : we prove it using Tait's traditional method with parallel reduction. $\beta\eta$-subject reduction does not because Tatsuta's counter-example [9] applies here (adding an implicit second-order existential to Heyting's Arithmetic breaks subject-reduction).

Sorts

$$s ::= \mathsf{Prop} \mid \mathsf{Type}_i \ (i \in \mathbb{N})$$

Terms

$$M ::= x \mid s \mid \Pi x{:}M_1.\, M_2 \mid \forall x{:}M_1.M_2 \mid \lambda x.M \mid M_1\, M_2$$

Contexts

$$\Gamma ::= [] \mid \Gamma; x : M$$

Reduction

$$(\lambda x.\, M)\, N \rhd_\beta M\{x/N\}$$
$$\lambda x.\, M\, x \rhd_\eta M \qquad (\text{if } x \notin \mathsf{FV}(M))$$

Typing rules

$$\frac{}{[] \vdash}(\textsc{Wf-E}) \qquad \frac{\Gamma \vdash T : s \quad x \notin \mathsf{DV}(\Gamma)}{\Gamma; x : T \vdash}(\textsc{Wf-S})$$

$$\frac{\Gamma \vdash \quad (s_1, s_2) \in \textbf{Axiom}}{\Gamma \vdash s_1 : s_2}(\textsc{Sort}) \qquad \frac{\Gamma \vdash \quad (x : T) \in \Gamma}{\Gamma \vdash x : T}(\textsc{Var})$$

$$\frac{\Gamma \vdash T : s_1 \quad \Gamma; x : T \vdash U : s_2 \quad (s_1, s_2, s_3) \in \textbf{Rule}}{\Gamma \vdash \Pi x{:}T.\, U : s_3}(\textsc{ExpProd})$$

$$\frac{\Gamma \vdash T : s_1 \quad \Gamma; x : T \vdash U : s_2 \quad (s_1, s_2, s_3) \in \textbf{Rule}}{\Gamma \vdash \forall x{:}T.U : s_3}(\textsc{ImpProd})$$

$$\frac{\Gamma; x : T \vdash M : U \quad \Gamma \vdash \Pi x{:}T.\, U : s}{\Gamma \vdash \lambda x.\, M : \Pi x{:}T.\, U}(\textsc{Lam})$$

$$\frac{\Gamma \vdash M : \Pi x{:}T.U \quad \Gamma \vdash N : T}{\Gamma \vdash M\, N : U\{x/N\}}(\textsc{App})$$

$$\frac{\Gamma; x : T \vdash M : U \quad \Gamma \vdash \forall x{:}T.U : s \quad x \notin \mathsf{FV}(M)}{\Gamma \vdash M : \forall x{:}T.U}(\textsc{Gen})$$

$$\frac{\Gamma \vdash M : \forall x{:}T.U \quad \Gamma \vdash N : T}{\Gamma \vdash M : U\{x/N\}}(\textsc{Inst})$$

$$\frac{\Gamma \vdash M : T \quad \Gamma \vdash T' : s \quad T \cong_{\beta\eta} T'}{\Gamma \vdash M : T'}(\textsc{Conv})$$

Fig. 1. Syntax, reduction rules and typing rules of ICC

Terms

$$M ::= \cdots \mid \{x{:}A \mid B\} \mid \Sigma[x{:}A].B$$

Typing rules

$$\frac{\Gamma \vdash A : s \qquad \Gamma; x : A \vdash B : s \qquad s \in \mathcal{S}}{\Gamma \vdash \{x{:}A \mid B\} : s}(\textsc{Sub})$$

$$\frac{\Gamma \vdash \{x{:}A \mid B\} : s \quad \Gamma \vdash a : A \quad \Gamma \vdash b : B\{a/x\}}{\Gamma \vdash a : \{x{:}A \mid B\}}(\textsc{Sub-I})$$

$$\frac{\Gamma \vdash P : \{x{:}A \mid B\} \to s \qquad \Gamma \vdash c : \{x{:}A \mid B\} \qquad \Gamma \vdash f : \Pi x{:}A.\, \forall y{:}B.P\, x}{\Gamma \vdash f\, c : P\, c}(\textsc{Sub-E})$$

$$\frac{\Gamma \vdash A : s \qquad \Gamma; x : A \vdash B : s \qquad s \in \mathcal{S}}{\Gamma \vdash \exists x{:}A.B : s}(\text{U})$$

$$\frac{\Gamma \vdash \exists x{:}A.B : s \quad \Gamma \vdash a : A \quad \Gamma \vdash b : B\{a/x\}}{\Gamma \vdash b : \exists x{:}A.B}(\text{U-I})$$

$$\frac{\Gamma \vdash P : \exists x{:}A.B \to s \qquad \Gamma \vdash c : \exists x{:}A.B \qquad \Gamma \vdash f : \forall x{:}A.\Pi y{:}B.\, P\, y}{\Gamma \vdash f\, c : P\, c}(\text{U-E})$$

Fig. 2. Additional syntax and typing rules of ICC_Σ

2.3 Semantics of ICC_Σ

Miquel designed in [7] two models based on coherence spaces [3]. One was used to prove that ICC is consistent; the other to prove that every well-typed term of ICC is strongly normalizing.

The extension of these models for ICC_Σ is a work in progress. Long discussions with Alexandre Miquel led us to an informal proof that ICC 's models are still valid for ICC_Σ.

In a nutshell, the interpretation of subset types is quite straightforward and follows the set theoretic intuition: $[\![\{x : A \mid B\}]\!] = \{x \in [\![A]\!] \mid [\![B]\!] \neq \emptyset\}$. Introduction and elimination rules are valid.

For union types, the argument is more intricate. The interpretation of $\Sigma[x{:}A].B\, x$ is the smallest semantical type that contains $\bigcup_{x \in [\![A]\!]} [\![B\, x]\!]$. Introduction and elimination rules seem valid. But the behaviour of union types is more understandable in realizability than in set theory.

3 ICC^*_Σ

ICC_Σ has two main drawbacks : subject reduction does not hold and type inference is probably undecidable. The purpose of ICC^*_Σ is to fix these problems in the same way that ICC^* fixed ICC 's type inference issue (cf. [2]). ICC^*_Σ should be seen as a wrap-

Terms

$$
\begin{array}{rl}
M ::= & x \mid s \\
& \mid \Pi x{:}M_1.\, M_2 \mid \lambda x{:}M_1.\, M_2 \mid M_1\, M_2 \qquad (explicit) \\
& \mid \Pi[x{:}M_1].\, M_2 \mid \lambda[x{:}M_1].\, M_2 \mid M_1[M_2] \quad (implicit)
\end{array}
$$

Extraction

$$
\begin{array}{rrrr}
s^* = s & & x^* = x & \\
(\Pi x{:}T.\, U)^* = \Pi x{:}T^*.\, U^* & & (\Pi[x{:}T].\, U)^* = \forall x{:}T^*.U^* & \\
(\lambda x{:}T.\, U)^* = \lambda x.U^* & & (\lambda[x{:}T].\, U)^* = U^* & \\
(M\,N)^* = M^*N^* & & (M[N])^* = M^* &
\end{array}
$$

Typing rules

$$\frac{}{[] \vdash}(\textsc{Wf-E}) \qquad \frac{\Gamma \vdash T : s \quad x \notin \mathsf{DV}(\Gamma)}{\Gamma; x : T \vdash}(\textsc{Wf-S})$$

$$\frac{\Gamma \vdash \quad (s_1, s_2) \in \textbf{Axiom}}{\Gamma \vdash s_1 : s_2}(\textsc{Sort}) \qquad \frac{\Gamma \vdash \quad (x : T) \in \Gamma}{\Gamma \vdash x : T}(\textsc{Var})$$

$$\frac{\Gamma \vdash T : s_1 \quad \Gamma; x : T \vdash U : s_2 \quad (s_1, s_2, s_3) \in \textbf{Rule}}{\Gamma \vdash \Pi x{:}T.\, U : s_3}(\text{E-}\textsc{Prod})$$

$$\frac{\Gamma \vdash T : s_1 \quad \Gamma; x : T \vdash U : s_2 \quad (s_1, s_2, s_3) \in \textbf{Rule}}{\Gamma \vdash \Pi[x{:}T].\, U : s_3}(\text{I-}\textsc{Prod})$$

$$\frac{\Gamma; x : T \vdash M : U \quad \Gamma \vdash \Pi x{:}T.\, U : s}{\Gamma \vdash \lambda x{:}T.\, M : \Pi x{:}T.\, U}(\text{E-}\textsc{Lam})$$

$$\frac{\Gamma; x : T \vdash M : U \quad \Gamma \vdash \Pi[x{:}T].\, U : s \quad x \notin \mathsf{FV}(M^*)}{\Gamma \vdash \lambda[x{:}T].\, M : \Pi[x{:}T].\, U}(\text{I-}\textsc{Lam})$$

$$\frac{\Gamma \vdash M : \Pi x{:}T.\, U \quad \Gamma \vdash N : T}{\Gamma \vdash M\,N : U\{x/N\}}(\text{E-}\textsc{App}) \qquad \frac{\Gamma \vdash M : \Pi[x{:}T].\, U \quad \Gamma \vdash N : T}{\Gamma \vdash M[N] : U\{x/N\}}(\text{I-}\textsc{App})$$

$$\frac{\Gamma \vdash M : T \quad \Gamma \vdash T' : s \quad T^* \cong_{\beta\eta} T'^*}{\Gamma \vdash M : T'}(\textsc{Conv})$$

Fig. 3. Terms, extraction and typing rules in ICC*

ping of ICC_Σ designed to capture both the semantics of ICC_Σ and the good syntactic properties of a Church-style Calculus of Constructions extended with dependent pairs. ICC^*_Σ is an extension of ICC_Σ in the same way that ICC^* is an extension of ICC.

ICC^*'s syntax is the same as in a Church-style Calculus of Constructions except that we duplicate each operation (product, abstraction and application) in an explicit one and an implicit one. In ICC^*_Σ, we add two kinds of sigma-types : $\Sigma[x : A].B$, where the first component is implicit, and $\Sigma x : A.[B]$ where the second one is implicit. The corresponding dependent pairs are respectively $([a], b)_{\Sigma[xA].B}$ and $(a, [b])_{\Sigma xA.[B]}$. We also add an elimination operator Elim (cf Fig. 4).

We defined in [2] an extraction function that removes implicit abstractions and implicit applications as well as domains of abstractions (cf. Fig 3). Here, we extend this function by removing the implicit term in dependent pairs and by mapping $\Sigma x : A.[B]$ to the subset type $\{x : A \mid B\}$ and $\Sigma[x : A].B$ to the union type $\exists x : A.B$.

Since we want ICC_Σ and ICC^*_Σ to be tightly linked, we also add six rules, each one corresponding to a rule we have added in ICC_Σ . In order to capture in ICC^*_Σ terms the behaviour of ICC_Σ terms, ICC^*_Σ rules are designed so that they match exactly ICC_Σ rules after extraction.

Moreover, regarding conversion and reduction rules, ICC^*_Σ works exactly the same way than ICC^* . This means that, in ICC^*_Σ , conversion is still made between extracted terms and not between annotated ones. Thus, there is still no need for reduction rules in ICC^*_Σ , since all the computation occurs between extracted terms.

We keep the main metatheoretical properties that we had in ICC^* . We prove by mutual structural induction that the extraction is sound and complete: for every judgement $\Gamma \vdash$ or $\Gamma \vdash M : T$ in ICC^*_Σ , $\Gamma^* \vdash$ or $\Gamma^* \vdash M^* : T^*$ holds; and vice-versa for every judgement Γ or $\Gamma \vdash M : T$ in ICC_Σ , there exists Δ or Δ, N and U such that $\Delta \vdash \wedge\Delta^* = \Gamma$ or $\Delta \vdash N : U \wedge \Delta^* = \Gamma \wedge N^* = M \wedge U^* = T$. From this, we deduce the relative consistency of ICC^*_Σ : if ICC_Σ is consistent, then ICC^*_Σ is also consistent.

We prove decidability of type inference as in ICC^* by induction using the strong normalization of ICC_Σ terms and β-reduction rules that we introduce[1] in ICC^*_Σ . Σ-types are treated the same way products are. Dependent pairs have type annotations. For the Elim operator, the induction step is straightforward.

4 Perspectives and Future Work

The final goal is to have inductive types in our system so that it could eventually be used in Coq.[2] We are going to proceed by adding each basic type that appears in the decomposition of inductive types.

The very next step is to complete the addition of Σ-types. We need to prove rigorously how Miquel's models are still valid in ICC_Σ . We shall also add explicit Σ-types $\Sigma x : A.B$, explicit dependent pairs $(a, b)_{\Sigma x:A.B}$ and allow different universes to be

[1] These rules are just defined for the sake of the proof, they play no role during computation.

[2] A prototype is already available as a darcs repository at
`http://www.lix.polytechnique.fr/Labo/Bruno.Barras/coq-implicit`

Terms

$$M ::= \cdots \mid \Sigma[x\!:\!A].B \mid \Sigma x\!:\!A.[B] \mid ([a],b)_{\Sigma[xA].B} \mid (a,[b])_{\Sigma xA.[B]} \mid \mathsf{Elim}(P,f,c)$$

Extraction

$$\begin{array}{rlrl} (\Sigma[x\!:\!A].B)^* & = \exists x\!:\!A^*.B^* & (\Sigma x\!:\!A.[B])^* & = \{x\!:\!A^* \mid B^*\} \\ (([a],b)_{\Sigma[xA].B})^* & = b^* & ((a,[b])_{\Sigma xA.[B]})^* & = a^* \end{array}$$

$$(\mathsf{Elim}(P,f,c))^* = f^*\, c^*$$

Typing rules

$$\frac{\Gamma \vdash A : s \qquad \Gamma; x : A \vdash B : s \qquad s \in \mathcal{S}}{\Gamma \vdash \Sigma x\!:\!A.[B] : s}\,(\Sigma_{\text{SUB}})$$

$$\frac{\Gamma \vdash \Sigma x\!:\!A.[B] : s \quad \Gamma \vdash a : A \quad \Gamma \vdash b : B\{a/x\}}{\Gamma \vdash (a,[b])_{\Sigma xA.[B]} : \Sigma x\!:\!A.[B]}\,(\Sigma_{\text{SUB}}\text{-I})$$

$$\frac{\begin{array}{c}\Gamma \vdash P : \Sigma x\!:\!A.[B] \to s \qquad \Gamma \vdash c : \Sigma x\!:\!A.[B] \\ \Gamma \vdash f : \Pi x\!:\!A.\,[y\!:\!B].\,P\,(x,[y])_{\Sigma xA.[B]}\end{array}}{\Gamma \vdash \mathtt{Elim}(P,f,c) : P\,c}\,(\Sigma_{\text{SUB}}\text{-E})$$

$$\frac{\Gamma \vdash A : s \qquad \Gamma; x : A \vdash B : s \qquad s \in \mathcal{S}}{\Gamma \vdash \Sigma[x\!:\!A].B : s}\,(\Sigma_{\text{U}})$$

$$\frac{\Gamma \vdash \Sigma[x\!:\!A].B : s \quad \Gamma \vdash a : A \quad \Gamma \vdash b : B\{a/x\}}{\Gamma \vdash ([a],b)_{\Sigma[xA].B} : \Sigma[x\!:\!A].B}\,(\Sigma_{\text{U}}\text{-I})$$

$$\frac{\begin{array}{c}\Gamma \vdash P : \Sigma[x\!:\!A].B \to s \qquad \Gamma \vdash c : \Sigma[x\!:\!A].B \\ \Gamma \vdash f : \Pi[x\!:\!A].\,(y\!:\!B).\,P\,([x],y)_{\Sigma[xA].B}\end{array}}{\Gamma \vdash \mathtt{Elim}(P,f,c) : P\,c}\,(\Sigma_{\text{U}}\text{-E})$$

Fig. 4. Extended syntax, extraction and typing rules in ICC^*_Σ

used in the formation rules (i.e. allow that e.g. $\Gamma \vdash \Sigma x : A.B : \mathsf{Type}_{(max(i,j))}$ when $\Gamma \vdash A : \mathsf{Type}_i$ and $\Gamma \vdash B : \mathsf{Type}_j$). These additions would require little effort with regard to syntax, but it should be harder to prove that models are still valid (or to adapt them if it is not the case).

We will then add more basic types such as unit type and void, which should be much easier than for the Σ-types.

We have already started to think about equality. The syntactic work is almost finished but the semantic part needs to be done. Our system will support heterogeneous equality, which would let us prove easily that e.g. `div 11 5 = div 13 6`, where `div` is the euclidean division.

The final step would be to add fixpoints operators so we could express recursion and have full inductive types.

Acknowledgements Many thanks to Alexandre Miquel for thorough discussions about his models of ICC and his help in extending them, to Bruno Barras for his continuous support and advice and to Mathieu Boespflug for his remarks. The author is funded by the DGA.

References

1. H. Barendregt. Lambda Calculi with Types. Technical Report 91-19, Catholic University Nijmegen, 1991. In Handbook of Logic in Computer Science, Vol II.
2. B. Barras and B. Bernardo. The implicit calculus of constructions as a programming language with dependent types. In R. M. Amadio, editor, *FoSSaCS*, volume 4962 of *Lecture Notes in Computer Science*, pages 365–379. Springer, 2008.
3. J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and types*. Cambridge University Press, New York, NY, USA, 1989.
4. P. Letouzey. *Programmation fonctionnelle certifiée – L'extraction de programmes dans l'assistant Coq*. PhD thesis, Université Paris-Sud, July 2004.
5. Z. Luo. *An Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1990.
6. A. Miquel. The implicit calculus of constructions. extending pure type systems with an intersection type binder and subtyping. In *Proceedings of the fifth International Conference on Typed Lambda Calculi and Applications (TLCA'01)*, Krakow (Poland), 2001.
7. A. Miquel. *Le Calcul des Constructions implicite: syntaxe et sémantique*. PhD thesis, Université Paris 7, Dec. 2001.
8. C. Paulin-Mohring. *Définitions Inductives en Théorie des Types d'Ordre Supérieur*. Habilitation à diriger les recherches, Université Claude Bernard Lyon I, Dec. 1996.
9. M. Tatsuta. Simple saturated sets for disjunction and second-order existential quantification. In S. R. D. Rocca, editor, *TLCA*, volume 4583 of *Lecture Notes in Computer Science*, pages 366–380. Springer, 2007.
10. The Coq development team. The coq proof assistant reference manual v8.2. Technical report, INRIA, France, February 2009. `http://coq.inria.fr/doc/main.html`.
11. J. B. Wells. Typability and type checking in System F are equivalent and undecidable. *Ann. Pure Appl. Logic*, 98(1–3):111–156, 1999.