A-Translation and Looping Combinators in Pure Type Systems

Thierry Coquand, Hugo Herbelin

A-translation and Looping Combinators in Pure Type Systems

Thierry Coquand, coquand@cs.chalmers.se
Hugo Herbelin, herbelin@margaux.inria.fr

Abstract

We present here a generalization of A-translation to a class of Pure Type Systems.

We apply this translation to give a direct proof of the existence of a looping combinator in a large class of inconsistent type systems, class which includes type systems with a type of all types. This is the first non-automated solution to this problem.

Introduction

The term *A-translation* first appeared in a paper of Friedman [3]. It denotes there a technical tool used in a proof of closure under Markov's rule of several intuitionistic systems. Combined with Gödel's translation from classical arithmetic into intuitionistic arithmetic, this was used to give a new proof of the intuitionistic provability of classically provable Σ^0_1 formulas.

Leivant [8] is a good reference about A-translation. Recently, connections between A-translation and Continuation Passing Style have been investigated. See for instance Murthy's Ph. D. thesis[10].

We are going to generalise A-translation to a large class of Pure Type Systems, introduced recently by Barendregt [1, 4]. This generalisation is motivated by the following problem: to extract constructive informations from paradoxes in inconsistent type systems. More specifically, let us define a "looping combinator" as being a term having the same Böhm tree as the fixed-point combinator Y. It has been shown by Howe [6] that a type system with a type of all types contain a looping combinator. We will get this result as an application of A-translation for Pure Type Systems.

The basic idea motivating this use can be traced back to the earliest known translation from classical logic to intuitionistic logic due to Kolmogorov [7]. This translation was actually a translation of classical logic into minimal logic: the rule "ab falso quodlibet" is never used, and the absurd proposition $\perp$ in Kolmogorov's paper can thus be replaced formally by any proposition A.

Kolmogorov saw the use of his translation as the development of "pseudo-mathematics," where, intuitively speaking, all notions and all lemmas occurring in a proof are defined and proved "relatively to a fixed proposition A".

This is this feature of A-translation that we use essentially here. In general, it is hard to see how to transform a paradox into a looping combinator. Howe's argument [6] is rather involved, done with computer assistance, and shows only how to extract a looping combinator out of one specific paradox. Our approach is more general. We show how to build a looping combinator from any given paradox. Indeed, when we apply A-translation to a paradox, we get a proof of A where all notions and lemmas are defined and proved "relatively to A". This proof is then transformed without too much problems into a looping combinator.

The first section defines a class of "logical" Pure Type Systems in which we will define an A-translation. The second section describes the A-translation for logical Pure Type Systems. We state then a significant property of proofs obtained from the A-translation in the third section. This property is exploited to show the existence of a looping combinator in inconsistent Type Systems. The last section gives some examples of Type Systems containing looping combinators. We end by raising some questions suggested by our work.

1 Logical Pure Type Systems

We use here the standard definition of Pure Type Systems from Barendregt and Geuvers-Nederhof [1, 4]. In particular, we make fairly heavy implicit use of the general properties of Pure Types Systems as presented in [4].

Definitions : A Pure Type System L is **logical** iff it is functional (see [4]) and contains two distinguished sorts $Prop$ and $Type$ such that

- $Prop : Type$ is an axiom of L
- $(Prop, Prop, Prop)$ is a rule of L
- There are no sorts of type $Prop$

In a logical Pure Type System, the terms of type $Prop$ are called **propositions**, and the terms of type a proposition are called **proofs**.

Definition : A logical Pure Type System is **inconsistent** iff there exists a proof of A in the context $A : Prop$.

Definition : A logical Pure Type System is said to be **nondependent** iff the only rules concerning $Prop$ are of the form $(S, Prop, Prop)$ where S is some sort.

Remark : simply-typed λ-calculus, system F, F_ω (see [4]) are nondependent logical Pure Type Systems. On the other hand, a type system with a type of all types, with $Prop = Type$ is not logical because $Prop$ is then a sort of type $Prop$. The calculus of constructions, see [4], is logical, but is not nondependent because it has the rule $(Prop, Type, Type)$.

Lemma 1 *In a nondependent logical Pure Type System, if* $X = (X_1\ X_2)$ *and* X_1 *or* X_2 *is a proof, then* X *is a proof.*

Proof There exist Y_1, Y_2, S_1, S_2 and S such that $X_1 : (x_2 : Y_2)Y$, $X_2 : Y_2$, $Y : S$, $Y_2 : S_2$ and (S_2, S, S_1) is a rule. If X_1 is a proof, then $S_1 = Prop$ and so $S = Prop$. If X_2 is a proof, then $S_2 = Prop$, and so $S_1 = S = Prop$. ∎

Lemma 2 *In a nondependent logical Pure Type System, if* Y *is a subterm of* X *and* Y *is a proof, then* X *is a proof.*

Proof

By induction on the term X. We can as well assume that Y is a subterm of X distinct from X.

In such a case, the term X cannot be a variable, a constant.

if X is $\lambda x : X_1.X_2$ then, by induction hypothesis, since X_1 is not a proof, Y is a subterm of X_2, and hence by induction hypothesis, X_2 is a proof. Hence X is a proof.

if X is $(X_1\ X_2)$ then by induction hypothesis, X_1 or X_2 is a proof. By lemma 1, this implies that X is a proof.

The case where X is a product is impossible by induction hypothesis.

∎

Remark : This lemma implies that if $C : Prop$ in a context containing the declaration of a proof variable $h : B$, then h is not a subterm of C. Thus, any product $\Pi h : B.C$ built from the rule $(Prop, Prop, Prop)$ is nondependent and can be written $B \to C$.

Lemma 3 *Let* L *be nondependent logical Pure Type System and* p *a proof in a context* Γ.

Then p *is either a variable of the context, or a constant, or* $\lambda x : Y.q$ *where* q *is a proof in* $\Gamma, x : Y$, *or* $(q\ X)$ *where* q *is a proof in* Γ.

Proof Direct by case analysis. ∎

2 A-translation in nondependent logical Pure Type Systems

In all the section, we assume to be in a fixed nondependent logical Pure Type System, and inside the context $A : Prop$.

Notation: Let B be a proposition. We will write $[B]$ for the proposition $(B \to A) \to A$.

We now define a translation $^+$ on terms which are not proofs. This translation depends on the type of the subterms, and it is defined relatively to a context in which the term is well-formed. Notice that it is not clear a priori that M^+ is a well-formed term, so that a priori M^+ is defined only as a pseudo-term (see [4].) Proposition 1 will later show that M^+ is actually a well-formed term.

Definition : Let X be a well-formed term in the context Γ, different from a proof.

- X^+ is X if X is a variable, a constant or a sort
- $(X_1\ X_2)^+$ is $(X_1^+\ X_2^+)$
- $(\lambda x : X_1.X_2)^+$ is $\lambda x : X_1^+.X_2^+$
  where X_2^+ is defined in $\Gamma, x : X_1$
- the definition of $(\Pi x : X_1.X_2)^+$ depends on the type of X_2 and X_1:
  if X_2 is a proposition B_2 in Γ
  then if X_1 is a proposition B_1 in Γ
  then $(B_1 \to B_2)^+$ is $[B_1^+] \to [B_2^+]$
  else $(\Pi x : X_1.B_2)^+$ is $\Pi x : X_1^+.[B_2^+]$,
  where B_2^+ is defined in $\Gamma, x : X_1$
  else $(\Pi x : X_1.X_2)^+$ is $\Pi x : X_1^+.X_2^+$
  where X_2^+ is defined in $\Gamma, x : X_1$

Remark : lemma 3 justifies the previous definition by cases.

Lemma 4 *For any terms X well-formed in $\Gamma, y : Y$ and Z well-formed in Γ different from proofs, then $(X[y := Z])^+$ is identical to $X^+[y := Z^+]$.*

Proof It is straightforward. ■

Lemma 5 *For any terms X and Y well-formed in Γ different from proofs, $X =_\beta Y$ implies $X^+ =_\beta Y^+$.*

Proof It suffices to prove that $(\lambda z : Z.Z'\ Z'')^+$ reduces to $(Z'[z := Z''])^+$. This follows from lemma 4. ■

We now define a translation * on propositions and contexts

Definitions : Let B be a proposition in a certain context, B^* is defined as $[B^+]$. Let Γ be a well-formed context, Γ^* is defined inductively like this :

- if Γ is the empty context then Γ^* is the empty context
- if Γ is $\Gamma', x : X$, where X is not a proposition then Γ^* is $\Gamma'^*, x : X^+$
- if Γ is $\Gamma', h : B$, where B is a proposition then Γ^* is $\Gamma'^*, h : B^*$

Lemma 6 *For any propositions B and C in Γ, $B =_\beta C$ implies $B^* =_\beta C^*$.*

Proof Straightforward by lemma 5. ■

Proposition 1 *If $\Gamma \vdash X : Y$ and X is not a proof then $\Gamma^* \vdash X^+ : Y^+$. If $\Gamma \vdash B : Prop$ then $\Gamma^* \vdash B^* : Prop$.*

Proof We prove this simultaneously by induction on the structure of the derivation of $\Gamma \vdash X : Y$ (resp. $\Gamma \vdash B : Prop$.) The case of conversion is done by lemma 5. Lemma 2 assures us that the derivation of $\Gamma \vdash X : Y$ (resp $\Gamma \vdash B : Prop$) encounters no proofs. ■

Lemma 7 *For any propositions B and C in Γ, if $\Gamma^* \vdash p : B^*$ and $B =_\beta C$ then $\Gamma^* \vdash p : C^*$.*

Proof By lemma 6 and the conversion rule in Pure Type Systems. Proposition 1 assures that $\Gamma^* \vdash C^* : Prop$. ■

We now define translation * on proofs. As for the translation $^+$, it is defined relatively to a context in which the term is a well-formed proof p, and it is not clear a priori that p^* is a well-formed term, so that p^* is defined only as a pseudo-term. Theorem 1 will actually show that p^* is indeed a well-formed term which is a proof.

Definition : Let p be a proof in the context Γ.

- p^* is p if p is a variable or a constant
- if p is $\lambda h : B.q$, with B a proposition, and $q : C$, then p^* is
  $\lambda k : ((B^* \to C^*) \to A).(k \;\; \lambda h : B^*.\lambda k' : (C^+ \to A).(q^* \;\; k'))$
  where q^* is defined in $\Gamma, h : B$
- if p is $\lambda x : Y.q$, with Y not a proposition, and $q : C$, then p^* is
  $\lambda k : ((\Pi x : Y.C^*) \to A).(k \; \lambda x : Y.\lambda k' : (C^+ \to A).(q^* \;\; k'))$
  where q^* is defined in $\Gamma, x : Y$
- if p is $(p_1 \; p_2)$ and $p_1 : B \to C$ then p^* is
  $\lambda k : (C^+ \to A).(p_1^* \;\; \lambda h_1 : (B^* \to C^*).(h_1 \;\; p_2^* \;\; k))$

- if p is $(p_1\ X)$, when X is not a proof, and $p_1 : \Pi x : Y.C$, then p^* is $\lambda k : (C[x := X]^+ \rightarrow A).(p_1^* \quad \lambda h_1 : (\Pi x : Y^+.C^*).(h_1 \quad X^+ \quad k))$

Remark : lemma 3 justifies the previous definition by cases.

Theorem 1 *Let B be a proposition in Γ. If $\Gamma \vdash p : B$ then $\Gamma^* \vdash p^* : B^*$*

PROOF By induction on the structure of the derivation of $\Gamma \vdash p : B$. The case of proposition conversion is done by lemma 7. Proposition 1 treats the case of judgements $\Gamma \vdash X : Y$ with X not a proof. ■

Remark 1: * is a Kolmogorov-like A-translation. It generalizes an A-translation of Paulin-Mohring [11] for the Calculus of Constructions with data types distinguished from propositions, and is inspired by a classical/intuitionistic translation of Girard [5] for higher order λ-calculi.
Remark 2: if we assume Church-Rosser property for the Pure Type System we are considering, lemma 5 holds also for $\beta\eta$-conversion and therefore proposition 1 and theorem 1 still hold in presence of $\beta\eta$-conversion. However, Church-Rosser property for general Pure Type Systems (not necessarilly normalisable) with $\beta\eta$-conversion seems still to be an open problem.

3 Long A-applicativity

As we said in the introduction, the original motivation in using A-translation was the fact that, intuitively, proofs that we get by A-translation "proves only A." Trying to make precise this remark leads to the following notion.

Definition : The notion of **long A-applicative** proof in a context Γ is defined by the following cases:

- the variable h of type B with $B : Prop$ is a long A-applicative proof if $h : B$ is in Γ
- $\lambda x_1 : Y_1 \ \ldots \lambda x_n : Y_n.p$ is a long A-applicative proof in Γ if p is a long A-applicative proof in $\Gamma, x_1 : Y_1, \ldots, x_n : Y_n$ and if p is of type A
- $(p\ q)$ is a long A-applicative proof in Γ if p and q are long A-applicative proofs in Γ.
- $(p\ X)$ where X is not a proof is a long A-applicative proof in Γ if p is a long A-applicative proof in Γ.

Proposition 2 *If p is a proof in Γ then p^* is long A-applicative in Γ^*.*

PROOF Direct from the definition of p^*. ■

Lemma 8 *If p is a long A-applicative proof in a context $\Gamma, h : B$ and q is long A-applicative in Γ then $p[h := q]$ is long A-applicative in Γ.*

If p is a long A-applicative proof in a context $\Gamma, x : Y$ and X is not a proof in Γ then $p[x := X]$ is long A-applicative in Γ.

Proof By induction on the structure of p. ■

4 Looping combinators

The idea of Meyer and Reinhold [9] to obtain a recursion combinator in the inconsistent system $Type : Type$ was to exploit the non normalisability of the proof of the inconsistency by inserting some "f" in it in order to obtain a term p_0 such that p_0 reduces to $(f\ p_1)$ and then p_1 to $(f\ p_2)$, and so on... From such a sequence, it is direct to build a family of terms $Y_n : \Pi A : Type.(A \to A) \to A$ such that $(Y_n\ A\ f) = f\ (Y_{n+1}\ A\ f)$.

Definition : Let T be a Pure Type System and S a sort of T. A **looping combinator of sort** S in T is a term $Y : \Pi A : S.(A \to A) \to A$ such that there exists a sequence of terms $Y_0 \equiv Y, Y_1, \ldots Y_n \ldots$, of type $\Pi A : S.(A \to A) \to A$ such that for any $A : S,\ f : A \to A$

$(Y_n\ A\ f) =_\beta f(Y_{n+1}\ A\ f)$

Howe [6] applied the same idea to transform the paradox of Girard [5] into a looping combinator by a direct mechanichal analysis of the term corresponding to this paradox.

We are now going to show how to build a looping combinator in any inconsistent nondependent logical Pure Type System. The last section will show that this implies in particular the existence of a looping combinator also for $Type : Type$.

From now on, we assume to be in a fixed inconsistent nondependent logical Pure Type System, and inside the context $A : Prop$.

Proposition 3 *There exists a long A-applicative proof of A.*

Proof Since the type system is inconsistent, there exists a proof q_A of A in the context $A : Prop$. By theorem 1, $(q_A)^*$ is a proof of A^* in the context $A : Prop$ and by proposition 2, this proof is long A-applicative. But A^* is $(A \to A) \to A$, and $p_A = ((q_A)^*\ \lambda x : A.x)$ is a long A-applicative proof of A. ■

We now precise what kind of term is a long A-applicative proof of A :

Lemma 9 *A long A-applicative proof of A is of the following form:*

$$((\lambda x^1 : Y^1\ \ldots \lambda x^m : Y^m.q)\ \ X^1 \ldots X^m)$$

with $m \geq 1$, $q : A$ and each X^i is either long A-applicative or not a proof.

Proof Let p be a long A-applicative proof of A in the context $A : Prop$. Since A is atomic, A cannot be convertible to a product by Church-Rosser. Hence, by uniqueness of type, p does not begin with an abstraction.

Therefore it is of the following form:

$(p'\ X^1 \ldots X^m)$ with $m \geq 0$ and p' either a variable or an abstraction

Since we are in the context $A : Prop$ p' cannot be a variable, $m \geq 1$ and p' begins with an abstraction. And since p is long A-applicative, p' is of the following form:

$\lambda x^1 : Y^1\ \ldots \lambda x^{m'} : Y^{m'}.q$ with $m' \geq 1$ and $q : A$.

The type of q remains A by instantiation, hence m cannot be greater than m'. And since p proves A, m' cannot be greater than m. Hence we have $m = m'$, i.e. p has the desired form. ■

We now define a strategy of reduction applicable to long A-applicative proofs of type A.

Definition : Let p be a long A-applicative proofs of type A. By lemma 9, p is

$((\lambda x_1 : Y_1\ \ldots \lambda x_n : Y_n.q)\ \ X_1 \ldots X_n)$, with $n \geq 1$ and $q : A$,

$red(p)$ is then the following term of type A

$q[x_1 := X_1] \ldots [x_n := X_n]$.

Lemma 10 *For any long A-applicative proof p of A in $A : Prop$, $red(p)$ is a long A-applicative proof of A in $A : Prop$.*

Proof By lemma 8. ■

We now define the transformation p^f which inserts "marks" inside long A-applicative proofs p in such a way that for any long A-applicative proofs p of A $red(p^f)$ is $(f\ (red(p))^f)$.

Definition : Let p be a long A-applicative proof in a context Γ
p^f is defined inductively in the context $\Gamma, f : A \rightarrow A$ as follows:

- if p is a variable h in Γ then p^f is h in Γ,
- if p is $\lambda x_1 : Y_1\ \ldots \lambda x_n : Y_n.q$ in Γ then p^f is $\lambda x_1 : Y_1\ \ldots \lambda x_n : Y_n.(f\ q^f)$ in Γ where q^f is defined in $\Gamma, f : A \rightarrow A, x_1 : Y_1, \ldots, x_n : Y_n$,
- if p is $(p_1\ p_2)$ then p^f is $(p_1^f\ p_2^f)$,

- if p is $(p_1\ M)$ with M not a proof, then p^f is $(p_1^f\ M)$.

Remark : p^f is of same type as p and is long A-applicative also.

Lemma 11 *If p is an A-applicative proof in the context $\Gamma, h : B$ and q an A-applicative proof of B in Γ then $p^f[h' := q^f]$ is $(p[h := q])^f$.*

If p is an A-applicative proof in the context $\Gamma, R : T$ and $M : T$ not a proof then $p^f[R := M]$ is $(p[R := M])^f$.

Proof By structural induction on p and by lemma 8 ■

Lemma 12 *For any long A-applicative proof p of A, $red(p^f)$ is $(f\ (red(p))^f)$.*

Proof p is of the form

$$((\lambda x^1 : Y^1\ \ldots \lambda x^m : Y^m.q)\ \ X^1 \ldots X^m),$$

and then p^f is

$$((\lambda x^1 : Y^1\ \ldots \lambda x^m : Y^m.(f\ q^f))\ \ (X^1)^f \ldots (X^m)^f),$$

which reduces by lemma 11 to $(f\ (red(p))^f)$. ■

Lemma 13 *There exists a sequence of terms $M_0, M_1, \ldots, M_n, \ldots$ defined in the context $A : Prop, f : A \to A$ such that $M_n =_\beta (f\ M_{n+1})$.*

Proof We define a sequence of terms p_n as follows. First we define p_0 to be any long A-applicative proof of A in the context $A : Prop$, using proposition 3. We then define p_{n+1} to be $red(p_n)$. Each proof term p_n is long A-applicative proof of A in $A : Prop$ by lemma 10.

Let M_n be p_n^f. The sequence $M_0, \ldots, M_n, \ldots$ satisfies lemma 13 by lemma 12. ■

Theorem 2 *In any inconsistent nondependent logical Pure Type System, there exists a looping combinator of type Prop.*

Proof Direct from lemma 13 ■

Remark : The proof given here is constructive. We can effectively transform any proof of A in the context $A : Prop$ into a looping combinator.

5 Applications

We describe here the systems U^-, U and $Type : Type$ as Pure Type Systems.

The system U^- is the Pure Type System defined by the following sorts:

$Prop$, $Type$ and $Class$,

the axioms:

$Prop : Type$ and $Type : Class$

and the rules:

$(Prop, Prop, Prop)$
$(Type, Prop, Prop)$
$(Type, Type, Type)$
$(Class, Type, Type)$.

System U is the same as system U^- plus the following rule:

$(Class, Prop, Prop)$

The system $Type : Type$ is the Pure Type System with the only sort:

$Type$

the only axiom:

$Type : Type$

and the only rule:

$(Type, Type, Type)$

Both systems U and U^- are nondependent logical Pure Type System. They are both inconsistent, as shown in [2, 5]. Hence, by theorem 2, they contain a looping combinator of sort $Prop$. It is clear that a looping combinator for one of this system translates directly in a looping combinator of sort $Type$ for $Type : Type$.

Here is a direct application. Call a nondependent logical type system **impredicative** iff it contains the rule $(Type, Prop, Prop)$.

Theorem 3 *Convertibility is undecidable for inconsistent impredicative logical Pure Type System. Furthermore, convertibility and type-checking is undecidable for* $Type : Type$.

Proof The arguments of [9], which assumed the existence of a fixed-point combinator, apply directly using a looping combinator instead.

For sake of completeness, we include a sketch of these arguments. First, it is standard [5] how to represent primitive recursive functions as terms of type $N \to N$, where N is the proposition $\Pi X.X \to (X \to X) \to X$, and the number n is represented by the term $\lambda X.\lambda x.\lambda f.(f^n\ x)$. A looping combinator family allows the numeralwise representation of any *partial* recursive function ϕ by a term Φ : namely $\Phi t_n =_\beta t_k$ iff $\phi(n) = k$. This entails the undecidablity of convertibility in any inconsistent impredicative logical Pure Types System.

The same reasoning will apply to $Type : Type$ by taking N to be the type $\Pi X.X \to (X \to X) \to X$. Furthermore, in this case the problem whether $\phi(n) = 0$ reduces to the question whether $(f\ x)$ is typable in the context $P : N \to Type,\ f : P(t_0) \to N,\ x : P(\Phi(t_n))$. Likewise, checking specific type judgements is undecidable, since $\phi(n) = 0$ reduces to the question whether x has type $P(\Phi(t_n))$ in the context $P : N \to Type,\ x : P(t_0)$. ■

Notice however that the normalisation theorem for system F [5] implies directly the decidability of type-checking for the system U^- and the system U.

Conclusion

We would like to raise some problems:

- The problem of the existence of a fixed-point combinator for the system $Type : Type$ is still open.
- Is it possible to derive the existence of a looping combinator from the existence of a paradox in a more direct way than by using A-translation?
- For the system U^- it is possible to define a "stripping" operation that associates to any proof term the untyped λ-term we get by forgetting the type information. We conjecture that the usual direct proof of non typability of the term $(\lambda x\ (x\ x)\ \lambda x\ (x\ x))$ in system F extends to show that this term is not typable in system U^-.

Acknowledgements

The authors want to thank Herman Geuvers, Chet Murthy, Erik Palmgren and Benjamin Werner both for their remarks about the paper and for enjoyable discussions about A-translation and Pure Type Systems. Thanks also to Henk Barendregt for his definition of looping combinators.

References

[1] H. Barendregt. *Introduction to Generalized Type System.* In the Journal of Functional Programming, Volume 1, Part 2, April 1991, pages 125 - 154.

[2] T. Coquand. *A New Paradox in Type Theory.* To appear in the Proceedings of the 9th International Congress of Logic, Methodology and Philosophy of Science, Uppsala, August 1991.

[3] H. Friedman. *Classically and Intuitionistically Provably Recursive Functions.* In Higher Set Theory, ed. G.H. Müller and D.S. Scott. Springer Verlag Lecture Notes 669, 1978, pages 21 - 27.

[4] H. Geuvers and M.-J. Nederhof. *Modular proof of strong normalisation for the calculus of constructions.* In the Journal of Functional Programming, Volume 1, Part 2, April 1991, pages 155 - 189.

[5] J.Y. Girard. *Interprétation fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieur.* Thèse de doctorat d'état de l'université Paris 7, 1972.

[6] D. J. Howe. *The computational behaviour of Girard's paradox.* In Proceedings of the Second Symposium of Logic in Computer Science (Ithaca, N.y.), IEEE, 1987, pages 205 - 214.

[7] A.N. Kolmogorov. *On the principle of excluded middle.* 1925. in From Frege to Gödel : a source book in mathematical logic, 1879-1931. J. Van Heijenoort. Harvard University Press, 1967.

[8] D. Leivant. *Syntactic translations and provable recursive functions.* Journal of Symbolic Logic 50, 1985, pages 682 - 688.

[9] A. R. Meyer, M. B. Reinhold. *"type" is not a type.* In Conference record of the thirteenth annual ACM symposium on principles of programming languages, Association for Computing Machinery, SIGACT, SIGPLAN, 1986, pages 287 - 295.

[10] C. Murthy *Extracting Constructive Content From Classical Proofs.* Ph.D. Thesis, Cornell University, 1990.

[11] C. Paulin *Extraction de programmes dans le Calcul des Constructions.* Thèse de doctorat de l'université Paris 7, 1989.