LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - University of Edinburgh

A Problem of Adequacy: conservativity of calculus of constructions over higher-order logic

by

Zhaohui Luo

LFCS Report Series ECS-LFCS-90-121

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

October 1990

A Problem of Adequacy: conservativity of calculus of constructions over higher-order logic[*]

Zhaohui Luo
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ, U.K.

August 3, 1990

[*]A draft of the paper was distributed on the Jumelage Meeting on the Typed Lambda Calculi held at Bari, Italy, in May of 1990.

Contents

Abstract

This paper discusses a problem of adequacy in formalization of mathematical notions in type theories. Based on a point of view that there should be a clear distinction between logical formulae and data types, we believe that, in an impredicative type theory like the calculus of constructions, arbitrary sets should be formalized as non-propositional types rather than propositional types which may be formed impredicatively.

We show that the calculus of constructions with type constants is a conservative extension of the intuitionistic higher-order logic, provided that the object set of the higher-order logic is interpreted as a non-propositional type constant. This result is proved by giving a formulae-as-types formulation of higher-order logic and using a projection technique developed by Berardi and Mohring. Comparing with the non-conservativity result of the (pure) calculus of constructions over higher-order logic [Ber89][Geu89], this provides an evidence to support the above view of adequate formalization and gives a better understanding of the formulae-as-types principle for higher-order logic.

We briefly discuss how abstract mathematics (*e.g.*, abstract algebras) can be adequately formalized in an impredicative type theory with predicative universes to support abstract reasoning.

1 A Problem of Adequacy

Various type theories have been developed and used in formalization of mathematical notions with applications to, for example, theorem-proving and program development. When using a type theory in applications, one must first give an adequate (or correct) formalization of the notions in the concerned problem; this is in fact the most important step. To formalize various mathematical notions adequately, it is essentially important to have a correct understanding of the type theory as well as the notions to be formalized.

In this paper, we discuss an adequacy problem in formalization of mathematical notions in (impredicative) type theories like the calculus of constructions [CH88]. More specifically, the question we intend to ask and answer is:

- *What is the (or an) adequate way to formalize the notion of arbitrary set in an impredicative type theory?*

To understand how an arbitrary set should be formalized is important in formalization of abstract mathematical notions like abstract algebras and notions in category theory. The formalization of these abstract notions is useful and important to do abstract reasoning in the development of proofs and programs. (See section 1.1 and section 3.2 for further explanations.)

In a type theory, a type is usually viewed intuitively as an entity representing a set; it may either stand for an ordinary set (data type) or the set of proofs of a logical formula by the Curry-Howard principle of formulae-as-types. It seems to be important, in both practical applications and theoretical research, to have a better understanding of these two kinds of types and their differences. In particular, when considering an impredicative type theory like the calculus of constructions, we believe that it is important and useful to have a clear distinction between a (small) type which represents the set of proofs of a logical formula and a (large) type which represents an ordinary set or data type.

In the calculus of constructions, a logical formula of the embedded higher-order logic is represented by a small type (called a proposition) which may be formed impredicatively. For example, the logical constant **false** can be represented by the proposition $\forall x{:}Prop.x$, where $Prop$ is the type of all propositions. Such small types with an impredicative formation rule are so special that, we believe, they should *not* be used to express arbitrary sets. In other words, trying to answer the above question of adequate formalization, we believe that

- *arbitrary sets should be formalized as non-propositional (predicative, large) types rather than propositional (impredicative, small) types.*

A model-theoretic explanation of the above idea of adequate formalization was discussed in [Luo89a], where it is shown, by giving a realizability model of an extended calculus of constructions with predicative universes, that predicative (large) types can be understood as arbitrary sets while it seems that the impredicative (small) types can not. Girard's paradox has also given us a hint on this; it shows that, roughly speaking, a type system with two impredicative levels is logically inconsistent (see [Coq86]).

In this paper, we shall discuss another evidence to support the above point of view of adequate formalization, by considering the conservativity problem of the calculus of constructions over the higher-order (predicate) logic. We shall show, by giving a formulae-as-types formulation of higher-order logic and using a projection technique developed by Berardi [Ber90] and Mohring [Moh89], that

- *the calculus of constructions with type constants* (**CC**$^+$) *is a conservative extension of the higher-order logic* (**HOL**), *provided that the object set of* **HOL** *is interpreted as a non-propositional type constant in* **CC**$^+$.

As proved by Berardi [Ber89] and Geuvers [Geu89], an interpretation which interprets the object set of higher-order logic as a propositional constant in the calculus of constructions will give a non-conservativity result. It is the difference of the interpretations of the object set as a non-propositional type (in our case) and as a proposition (in the case of Berardi and Geuvers) that results in the two completely different results about conservativity. These results conform with the conjecture in [Luo89a,90] that interpreting the object set of the higher-order logic as a small propositional type will not give a conservative extension result and interpreting it as a non-propositional type will.

Since the object set in a logic stands for an arbitrary set, these results about conservativity provide an evidence of the above view of adequate formalization on the one hand, and give a better understanding of the formulae-as-types principle for higher-order logic on the other. Such an understanding is also useful in applications where impredicative type theories are used to formalize program development or mathematics.

Based on the above point of view of adequate formalization, we shall briefly discuss how abstract mathematics (*e.g.*, abstract algebras) may be adequately formalized in an impredicative type theory with predicative universes such as the extended calculus of constructions [Luo89a,90], to support abstract reasoning.

1.1 Abstract reasoning

We give a very brief explanation of the motivation and the need to formalize abstract mathematics and, in particular, the notion of arbitrary set. Formalizing abstract mathematics like abstract algebra and category theory is an interesting application area in computer-assisted reasoning. However, what we want to emphasize here is another interesting aspect of this, from the point of view of computer scientists, that is, the need for abstract reasoning.

Computer scientists have a desire for abstraction in order to achieve clarity and efficiency. Abstract reasoning is such an approach to theorem proving and verification of proof obligations in rigorous program development. The basic idea is that, instead of reproving a theorem (program property) for many concrete theories (programs), one can prove an abstract theorem in an abstract theory and then simply instantiates the abstract proofs as concrete ones for free. For example, we can consider an abstract theory of groups and prove theorems about groups; then we can instantiate the proofs for different group structures (say a concrete theory for integers).

Such an abstract reasoning has been (informally) adopted by mathematicians and is analogous to the notion of parameterization in modular programming. It is desirable and would not only save a lot of efforts from repeated proofs but also lead to a clearer structured approach to reasoning. A type theory can provide such facilities on the condition that it can be used to formalize abstract theories (*e.g.*, that for groups) adequately. This requires an adequate formalization of the notion of arbitrary set (*e.g.*, the carrier set of an arbitrary group). See [Luo89b,90] for more discussions on abstract reasoning, where an approach to structured abstract reasoning is studied as an application of the extended calculus of constructions. What we shall show below, through discussing the conservativity problem, is that arbitrary sets should be formalized as predicative large types instead of impredicative small types.

2 Conservativity of calculus of constructions over higher-order logic

The problem of conservativity, that is, whether a logical system is a conservative extension of another, has been an important research topic in logic. As well-known, by Curry-Howard principle of formulae-as-types, various typed λ-calculi can be viewed as logical systems (see [Bar89] for a recent account of this for various type systems using the notion of Generalized Type Systems). When one adds new type constructors to formulate a richer type theory, a natural question to ask is whether the enrichment fundamentally strengthens the logical power or not; in other words, one hopes to know whether the enriched theory is a conservative extension of the original system, which has been investigated before.

A typical and interesting instance of this problem is Coquand-Huet's calculus of constructions [CH88], which incorporates dependent types and was formulated as a type theory corresponding to higher-order predicate logic.

- *Is the calculus of constructions a conservative extension of higher-order logic?*

The intuition seems to suggest strongly that the answer be yes.

However, there is a subtle problem to be answered; that is, how should we interpret the object set of higher-order logic in the calculus of constructions? In the (pure) calculus of constructions, the only natural way to interpret the object type is to assume it to be a proposition (small type of type *Prop*), since one can not postulate new non-propositional type variable (or constant) in the system. Based on our point of view of adequate formalization, such an interpretation seems inadequate since a proposition should not be used to represent an arbitrary set, which is what the object set of a logic does stand for. In [Luo89a], the author gave the following conjectures based on the above considerations:

1. if the object set is interpreted as a proposition (of type *Prop*), the interpretation will *not* give a conservative extension of the higher-order logic;

2. if the object set is interpreted as a non-propositional type and the others interpreted in the obvious way, then the interpretation will be conservative.

This conjecture was based on the semantic understanding of the theory based on the realizability model [Luo89a]. The intuition behind the non-conservativity conjecture 1 is that too much computational power is embedded in the impredicative level of propositions. There should be a clear distinction between logical formulas (propositions) and sets (data types). Set-theoretically, an arbitrary non-propositional type can be understood as an arbitrary set, but a proposition can not. Interpreting object sets as non-propositional types (and formulas as propositions) conforms with such a distinction; on the other hand, interpreting object sets as propositions confuses such a difference and would cause problems. Recently, Berardi [Ber89] and Geuvers [Geu89] proved that the interpetation of the object set as a proposition in the calculus of constructions is non-conservative; in other words, viewing an arbitrary proposition in the calculus of constructions as representing the object set, one can prove some unexpected logical formulas that can not be proved in higher-order logic. This justifies our non-conservativity conjecture 1.

The above problem is caused by the interpretation which *identifies sets with logical formulas.* Arbitrary sets should be formalized as (large) predicative types rather than propositions (small types). Along such a line of consideration, it is natural to interpret the object set of higher-order logic as a non-propositional type. We shall consider the calculus of constructions with type constants and show that it is a conservative extension of higher-order logic, where the object set is interpreted as a non-propositional type constant. This conservativity result gives a positive answer to the above conservativity conjecture 2 and, together with the non-conservativity result by Berardi and Guevers mentioned above, gives a better understanding of the principle of formulae-as-types for higher-order logic and provides another evidence which supports our idea of adequate formalization.

2.1 A formulae-as-types formulation of higher-order logic

The conservativity result and its proof will be presented as follows: we first, in this subsection, give a faithful formulae-as-types formulation **HOL** of higher-order logic; then we present the calculus of constructions with type constants ($\mathbf{CC}^+$), of which **HOL** is a subsystem; finally, the proof of conservativity of $\mathbf{CC}^+$ over **HOL** is given.

In the literature, there exist several different formulations of higher-order logic (see [Chu40][Tak75][Sch77][LS86] among others). It is not our task here to compare their differences. We shall take Church's simple type theory [Chu40] as the system for higher-order logic and give a formulae-as-types formulation for it. However, different from the original classical extensional system of Church, what is presented below is the corresponding intuitionistic intensional system.

Notation The following notational conventions will be used in this paper.

1. substitution: $[N/x]M$ is used to express the substitution of the free occurrences of variable x in M by N, where possible α-conversion is done to avoid variable clashes.

2. syntactical identity: We write $M \equiv N$ to mean that M and N are syntactically equal up to possible changes of bound variables (under α-conversion).

3. set of free variables: We use $FV(\_)$ to denote the set of free variables in $\_$. □

2.1.1 Church's simple type theory

Putting it simply, Church's simple type theory is the simply typed λ-calculus with type constants *Prop* (the type of logical formulas) and *Obj* (the type of individuals or objects), and endowed with a notion of logical formula and a notion of provability.

Terms: The basic expressions, called *terms*, are inductively defined as follows:

1. Constants *Prop*, *Obj* and variables (x, ...) are terms;
2. If M and N are terms, so are the following:

$$M \to N,\ \lambda x{:}M.N,\ MN,\ M \supset N,\ \forall x{:}M.N$$

The conversion relation ($\simeq$) between terms is the β-conversion, induced by the contraction scheme $(\lambda x{:}A.M)N \rightsquigarrow [N/x]M$.

Types: Types are formed as follows:

1. *Prop* and *Obj* are types;
2. If A and B are types, so is $A \to B$.

Contexts: A (valid) context is a sequence of the form $x_1{:}A_1, ..., x_n{:}A_n$, where A_i are types and x_i are distinct variables.

Typing rules: A typing judgement is of the form $\Gamma \vdash M : A$, where Γ is a (valid) context, M is a term and A is a type. $\Gamma \vdash M : A$ means that 'M is an object of type A in context Γ'. Derivable typing judgements are given by the following rules:

$$\frac{}{\Gamma_1, x{:}A, \Gamma_2 \vdash x : A} \qquad \frac{\Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

Logical formulas: Logical formulas are terms of type *Prop* (relative to contexts) introduced by the following rules:

$$\frac{\Gamma \vdash P : Prop \quad \Gamma \vdash Q : Prop}{\Gamma \vdash P \supset Q : Prop} \qquad \frac{\Gamma, x{:}A \vdash P : Prop}{\Gamma \vdash \forall x{:}A.P : Prop}$$

Provability: Whether a logical formula P is a logical consequence of (or is provable from) a sequence of hypothetical formulas H (relative to a context Γ) is given by the following rules:

1. If $P \in H$, then P is a logical consequence of H.
2. If Q is a logical consequence of H, P, then $P \supset Q$ is a logical consequence of H.
3. If P is a logical consequence of H, A is a type and $x \notin FV(H)$, then $\forall x{:}A.P$ is a logical consequence of H.
4. If $\forall x{:}A.P$ is a logical consequence of H and N is of type A (in Γ), then $[N/x]P$ is a logical consequence of H.

5. If P is a logical consequence of H and $Q \simeq P$, then Q is a logical consequence of H.

This completes the presentation of (the intuitionistic intensional version of) Church's simple type theory. Church's type theory is slightly different from some other presentations of higher-order logic (*c.f.*, [Tak75][Sch77]). In particular, there are higher-order functions of types like $(Obj \to Obj) \to (Obj \to Obj)$ or $(Obj \to Prop) \to Obj$.

2.1.2 HOL: a formulae-as-types formulation of higher-order logic

Now, we give a formulae-as-types formulation of higher-order logic, **HOL**; it is faithful with respect to Church's simple type theory as described above.

The idea of formulae-as-types is that one views a logical formula as the type of its proofs and the provability of a formula corresponds to the inhabitance of its corresponding type. We add to Church's simple type theory a type formation operator **Prf** and, for every logical formula P, $\mathbf{Prf}(P)$ is a type, the type of the proofs of P. The proofs of logical formulas are also explicitly expressible as terms. The provability of a logical formula P corresponds to the inhabitance of type $\mathbf{Prf}(P)$. This formulation gives us a clear understanding of the Curry-Howard principle of formulae-as-types for higher-order logic.

The *terms* of **HOL** are defined the same as that of Church's simple type theory except that we add the following clause:

- If M and N are terms, so are $\mathbf{Prf}(M)$, $\Lambda x{:}M.N$, $M \bullet N$.

The conversion relation ($\simeq$) between terms is the β-conversion, induced by $(\lambda x{:}A.M)N \leadsto_\beta [N/x]M$ and $(\Lambda x{:}A.M) \bullet N \leadsto_\beta [N/x]M$.

A *context* in **HOL** is a sequence of the form $x_1{:}A_1, ..., x_n{:}A_n$, where x_i and A_i are variable and term, respectively. We use $\langle\rangle$ to denote the empty context.

There are three judgement forms:

- Γ **valid**: Γ is a valid context;
- $\Gamma \vdash A$ **type**: A is a type in context Γ;
- $\Gamma \vdash M : A$: M is of type A in context Γ.

The inference rules of **HOL** for inferring judgements are the following:

Context Validity:

$$(C1)\ \frac{}{\langle\rangle\ \mathbf{valid}} \qquad (C2)\ \frac{\Gamma \vdash A\ \mathbf{type} \quad x \notin FV(\Gamma)}{\Gamma, x{:}A\ \mathbf{valid}}$$

Type Formation:

$$(Obj)\ \frac{\Gamma\ \textbf{valid}}{\Gamma \vdash \mathit{Obj}\ \textbf{type}} \qquad (Prop)\ \frac{\Gamma\ \textbf{valid}}{\Gamma \vdash \mathit{Prop}\ \textbf{type}}$$

$$(\rightarrow)\ \frac{\Gamma \vdash A\ \textbf{type}\quad \Gamma \vdash B\ \textbf{type}\quad A, B \not\equiv \mathbf{Prf}(...)}{\Gamma \vdash A \rightarrow B\ \textbf{type}} \qquad (\mathbf{Prf})\ \frac{\Gamma \vdash P : \mathit{Prop}}{\Gamma \vdash \mathbf{Prf}(P)\ \textbf{type}}$$

Basic Typing:

$$(var)\ \frac{\Gamma_1, x{:}A, \Gamma_2\ \textbf{valid}}{\Gamma_1, x{:}A, \Gamma_2 \vdash x : A}$$

$$(\lambda)\ \frac{\Gamma, x{:}A \vdash M : B\quad A, B \not\equiv \mathbf{Prf}(...)}{\Gamma \vdash \lambda x{:}A.M : A \rightarrow B} \qquad (app)\ \frac{\Gamma \vdash M : A \rightarrow B\quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

Propositions:

$$(\supset)\ \frac{\Gamma \vdash P : \mathit{Prop}\quad \Gamma \vdash Q : \mathit{Prop}}{\Gamma \vdash P \supset Q : \mathit{Prop}} \qquad (\forall)\ \frac{\Gamma, x{:}A \vdash P : \mathit{Prop}\quad A \not\equiv \mathbf{Prf}(...)}{\Gamma \vdash \forall x{:}A.P : \mathit{Prop}}$$

Proofs:

$$(\Lambda_\supset)\ \frac{\Gamma, x{:}\mathbf{Prf}(P) \vdash M : \mathbf{Prf}(Q)}{\Gamma \vdash \Lambda x{:}\mathbf{Prf}(P).M : \mathbf{Prf}(P \supset Q)} \qquad (App_\supset)\ \frac{\Gamma \vdash M : \mathbf{Prf}(P \supset Q)\quad \Gamma \vdash N : \mathbf{Prf}(P)}{\Gamma \vdash M \bullet N : \mathbf{Prf}(Q)}$$

$$(\Lambda_\forall)\ \frac{\Gamma, x{:}A \vdash M : \mathbf{Prf}(P)\quad A \not\equiv \mathbf{Prf}(...)}{\Gamma \vdash \Lambda x{:}A.M : \mathbf{Prf}(\forall x{:}A.P)} \qquad (App_\forall)\ \frac{\Gamma \vdash M : \mathbf{Prf}(\forall x{:}A.P)\quad \Gamma \vdash N : A}{\Gamma \vdash M \bullet N : \mathbf{Prf}([N/x]P)}$$

Type Conversion:

$$(conv)\ \frac{\Gamma \vdash M : A\quad \Gamma \vdash B\ \textbf{type}\quad A \simeq B}{\Gamma \vdash M : B}$$

We call a type of the form $\mathbf{Prf}(...)$ a *small type* (or *propositional type*). A type A is Γ-*inhabited* if and only if $\Gamma \vdash M : A$ for some term M. A term M is *well-typed* if and only if $\Gamma \vdash M : A$ or $\Gamma \vdash M$ **type** for some Γ and A.

Remarks Some remarks about the above formulae-as-types formulation follow.

1. Note that, although proofs become explicit in the theory, there is *no* arrow types which contain small types. For example, $\mathbf{Prf}(P) \rightarrow \mathit{Prop}$ is not a type. Because of this, Leibniz's equality (see section 2.2.1) can *not* be defined over a small type, although it can be defined over other types.

2. The above formulation has nice formal properties as a type system. In particular, we note that the strengthening lemma holds, *i.e.*,

   - If $y \notin FV(\Gamma', M, A)$, then
     - $\Gamma, y{:}A, \Gamma'$ **valid** implies Γ, Γ' **valid**,
     - $G, y{:}A, \Gamma' \vdash A$ **type** implies $\Gamma, \Gamma' \vdash A$ **type**, and
     - $\Gamma, y{:}A, \Gamma' \vdash M : A$, implies $\Gamma, \Gamma' \vdash M : A$.

   This strengthening lemma is proved using the method in [Luo88,90] to prove the same property of type theories like the calculus of constructions; it is an important lemma used to prove the projection theorem in the next section.

□

The formulae-as-types formulation of higher-order logic is faithful to Church's formulation, as the following theorem shows.

Theorem 2.1 *A logical formula P is a logical consequence of $H \equiv P_1, ..., P_n$ in Γ (in Church's simple type theory) if and only if* $\mathbf{Prf}(P)$ *is Γ_H-inhabited, where $\Gamma_H \equiv \Gamma, y_1{:}\mathbf{Prf}(P_1), ..., y_n{:}\mathbf{Prf}(P_n)$ with $y_1, ..., y_n$ fresh.* □

2.2 Conservativity of CC$^+$ over HOL

In this section, we prove that the calculus of constructions with type constants is a conservative extension of **HOL** (and hence of Church's simple type theory).

2.2.1 the calculus of constructions with type constants: CC$^+$

The following formulation of the calculus of constructions with type constants follows that by Coquand [Coq89], with the only difference (and enrichment) that a new non-propositional type constant *Obj* is added. (One may allow arbitrary many constants following the style of presentation of Edinburgh LF [HHP87]. But, for our purpose here, one type constant is enough. The proof of conservativity can easily be extended to the many-sorted case.)

The basic expressions, called *terms*, are inductively defined as follows:

- Constants *Prop* and *Obj*, and variables (x, ...) are terms;
- If M and N are terms, so are the following:

$$\Pi x{:}M.N,\ \lambda x{:}M.N,\ MN,\ \forall x{:}M.N,\ \mathbf{Prf}(M),\ \Lambda x{:}M.N,\ M \bullet N$$

Conversion, contexts and judgement forms are defined in the same way as those for **HOL**. The inference rules of $\mathbf{CC}^+$ are the following:

Context Validity:

$$(C1)\ \frac{}{\langle\rangle\ \textbf{valid}} \qquad (C2)\ \frac{\Gamma \vdash A\ \textbf{type}\quad x \notin FV(\Gamma)}{\Gamma, x{:}A\ \textbf{valid}}$$

Type Formation:

$$(Obj)\ \frac{\Gamma\ \textbf{valid}}{\Gamma \vdash Obj\ \textbf{type}} \qquad (Prop)\ \frac{\Gamma\ \textbf{valid}}{\Gamma \vdash Prop\ \textbf{type}}$$

$$(\Pi)\ \frac{\Gamma \vdash A\ \textbf{type}\quad \Gamma, x{:}A \vdash B\ \textbf{type}}{\Gamma \vdash \Pi x{:}A.B\ \textbf{type}} \qquad (\mathbf{Prf})\ \frac{\Gamma \vdash P : Prop}{\Gamma \vdash \mathbf{Prf}(P)\ \textbf{type}}$$

Basic Typing:

$$(var)\ \frac{\Gamma_1, x{:}A, \Gamma_2\ \textbf{valid}}{\Gamma_1, x{:}A, \Gamma_2 \vdash x : A}$$

$$(\lambda)\ \frac{\Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B} \qquad (app)\ \frac{\Gamma \vdash M : \Pi x{:}A.B\quad \Gamma \vdash N : A}{\Gamma \vdash MN : [N/x]B}$$

Propositions:

$$(\forall)\ \frac{\Gamma, x{:}A \vdash P : Prop}{\Gamma \vdash \forall x{:}A.P : Prop}$$

Proofs:

$$(\Lambda)\ \frac{\Gamma, x{:}A \vdash M : \mathbf{Prf}(P)}{\Gamma \vdash \Lambda x{:}A.M : \mathbf{Prf}(\forall x{:}A.P)} \qquad (App)\ \frac{\Gamma \vdash M : \mathbf{Prf}(\forall x{:}A.P)\quad \Gamma \vdash N : A}{\Gamma \vdash M \bullet N : \mathbf{Prf}([N/x]P)}$$

Type Conversion:

$$(conv)\ \frac{\Gamma \vdash M : A\quad \Gamma \vdash B\ \textbf{type}\quad A \simeq B}{\Gamma \vdash M : B}$$

The notion of type inhabitance is defined the same as that for **HOL**.

Remarks

1. **HOL** is a sub-system of $\mathbf{CC}^+$. This becomes clear when we define in $\mathbf{CC}^+$,

$$A \to B =_{\text{df}} \Pi x{:}A.B$$

$$A \supset B =_{\text{df}} \forall x{:}\mathbf{Prf}(A).B$$

where $x \notin FV(B)$. Note that this inclusion gives an interpretation of higher-order logic in $\mathbf{CC}^+$ in such a way that *the object set Obj is interpreted as a non-propositional type constant.* Under this (inclusion) interpretation, $\mathbf{CC}^+$ is a conservative extension of higher-order logic, as we shall show below. If we the object set were interpreted as a small type, the interpretation is non-conservative (see [Ber89][Geu89]).

2. The most important enrichment of $\mathbf{CC}^+$ compared with **HOL** is that proofs can be formally manipulated in $\mathbf{CC}^+$. For example, Leibniz's equality over small types can be defined in $\mathbf{CC}^+$:

$$(p =_{\mathbf{Prf}(P)} q) \ =_{\mathrm{df}} \ \forall f{:}(\mathbf{Prf}(P) \to Prop).\ f(p) \supset f(q)$$

3. Dependent Π-type $\Pi x{:}A.\mathbf{Prf}(P)$ is not identified with (although isomorphic to) the small type $\mathbf{Prf}(\forall x{:}A.P)$. This allows a proof-irrelevant semantics of the theory (see [Coq89]). □

2.2.2 conservativity

That $\mathbf{CC}^+$ is a conservative extension of **HOL** is proved by using a projection technique developed by Berardi [Ber90] and Mohring [Moh89]. Berardi used the technique to prove that the calculus of constructions is a conservative extension of Girard's system F^ω [Gir72] and Mohring used it to consider program extraction in the calculus of constructions. (Such an idea of projection also appears in [HHP87] where it is used to prove the normalization property of Edinburgh Logical Framework by reducing it to that of simply typed λ-calculus.)

The problem of conservativity of $\mathbf{CC}^+$ over **HOL** can be stated as follows (we shall use $\vdash^{\mathbf{HOL}}$ and $\vdash^{\mathbf{CC}^+}$ to distinguish judgements from the two systems under consideration when confusion may occur):

- *If* $\Gamma \vdash^{\mathbf{HOL}} A$ **type**, *then* A *is* Γ*-inhabited in* $\mathbf{CC}^+$ *if and only if* A *is* Γ*-inhabited in* **HOL**.

Since **HOL** is a subsystem of $\mathbf{CC}^+$, the if-part of the above statement is obvious. Its only-if-part is a corollary of the projection theorem 2.3. We first define the projection map.

Definition 2.2 (projection ′) *The projection ′ from* $\mathbf{CC}^+$ *to* **HOL** *is defined by induction on the structure of the well-typed terms* M *in* $\mathbf{CC}^+$ *(and then extended to contexts in the obvious way).*

1. $M \equiv Obj$ or $Prop$, *then* $M' =_{\mathrm{df}} M$.

2. $M \equiv x$ *is a variable, then* $M' =_{\mathrm{df}} x$.

3. $M \equiv \mathbf{Prf}(P)$, *then* $M' =_{\mathrm{df}} \mathbf{Prf}(P')$.

4. $M \equiv \Pi x{:}A.B$, *then*

$$M' =_{\mathrm{df}} \begin{cases} \mathbf{Prf}(P_A \supset P_B) & \textit{if } A' \equiv \mathbf{Prf}(P_A) \textit{ and } B' \equiv \mathbf{Prf}(P_B) \\ \mathbf{Prf}(\forall x{:}A'.P_B) & \textit{if } A' \not\equiv \mathbf{Prf}(...) \textit{ and } B' \equiv \mathbf{Prf}(P_B) \\ B' & \textit{if } A' \equiv \mathbf{Prf}(...) \textit{ and } B' \not\equiv \mathbf{Prf}(...) \\ A' \to B' & \textit{if } A', B' \not\equiv \mathbf{Prf}(...) \end{cases}$$

5. $M \equiv \lambda x{:}A.M_0$. *We introduce a terminology here. A (well-typed) term M is called proof-valued if it has a type of the form* $\Pi x_1{:}A_1 ... \Pi x_n{:}A_n.\ \mathbf{Prf}(...)$. *$M$ is called non-proof-valued if it has a type of the form* $\Pi x_1{:}A_1 ... \Pi x_n{:}A_n.\ K$, *where K is Obj or $Prop$. Now, for* $M \equiv \lambda x{:}A.M_0$,

$$M' =_{\mathrm{df}} \begin{cases} \Lambda x{:}A'.M_0' & \textit{if } M \textit{ is proof-valued} \\ M_0' & \textit{if } M \textit{ is non-proof-valued and } A' \equiv \mathbf{Prf}(...) \\ \lambda x{:}A'.M_0' & \textit{if } M \textit{ is non-proof-valued and } A' \not\equiv \mathbf{Prf}(...) \end{cases}$$

6. $M \equiv M_1 M_2$, *then*

$$M' =_{\mathrm{df}} \begin{cases} M_1' \bullet M_2' & \textit{if } M_1 \textit{ is proof-valued} \\ M_1' & \textit{if } M_1 \textit{ is non-proof-valued and } M_2 \textit{ is proof-valued} \\ M_1' M_2' & \textit{otherwise} \end{cases}$$

7. $M \equiv \forall x{:}A.P$, *then*

$$M' =_{\mathrm{df}} \begin{cases} P_A \supset P' & \textit{if } A' \equiv \mathbf{Prf}(P_A) \\ \forall x{:}A'.P' & \textit{if } A' \not\equiv \mathbf{Prf}(...) \end{cases}$$

8. $M \equiv \Lambda x{:}A.M_0$, *then* $M' =_{\mathrm{df}} \Lambda x{:}A'.M_0'$.

9. $M \equiv M_1 \bullet M_2$, *then* $M' =_{\mathrm{df}} M_1' \bullet M_2'$.

□

The projection map removes systematically the λ-abstraction and Π-quantification over small types. For example, consider the Leibniz's equality = uniformly defined over the small types:

$$= \ \equiv \ \lambda P{:}Prop \lambda x, y{:}\mathbf{Prf}(P).\ \forall f{:}(\Pi z{:}\mathbf{Prf}(P).Prop) \forall p{:}\mathbf{Prf}(fx).fy$$

which is of type $\Pi P{:}Prop\Pi x,y{:}\mathbf{Prf}(P).\ Prop$ (in $\mathbf{CC}^+$). Under the projection, we have

$$=' \ \equiv\ \lambda P{:}Prop.\,\forall f{:}Prop.f \supset f$$

which is of type $Prop \to Prop$ (in **HOL**).

Remark As in [Ber90] and [Moh89], the above projection respects substitution in the sense that $([N/x]M)' \equiv [N'/x]M'$. □

Theorem 2.3 (projection theorem) *The projection $'$ from $\mathbf{CC}^+$ to* **HOL** *has the following properties:*

1. *If Γ* **valid** *in $\mathbf{CC}^+$, then Γ'* **valid** *in* **HOL***;*
   if $\Gamma \vdash^{\mathbf{CC}^+} A$ **type***, then $\Gamma' \vdash^{\mathbf{HOL}} A'$* **type***; and*
   if $\Gamma \vdash^{\mathbf{CC}^+} M : A$, then $\Gamma' \vdash^{\mathbf{HOL}} M' : A'$.

2. *If M is a well-typed term in* **HOL***, then $M' \equiv M$.*

Proof The proof for the second statement can readily be done by induction on the structure of well-typed term M in **HOL**. The proof for the first statement is by induction on the derivation of the judgements in $\mathbf{CC}^+$, given as follows.

For the rules $(C1)(C2)(Obj)(Prop)(\mathbf{Prf})(var)(\Lambda)(App)$, the induction argument can readily be verified by the corresponding rule(s) in **HOL** and induction hypothesis. For rule $(conv)$, the fact that $A' \simeq B'$ whenever $A \simeq B$ is used. For rule (app), the property that the projection respects substitution is used. For rules (Π), (λ) and $(\forall)$, the strengthening lemma for **HOL** must be used. Here, we give the argument for rule (Π).

$$(\Pi) \qquad \frac{\Gamma \vdash^{\mathbf{CC}^+} A\ \mathbf{type} \quad \Gamma, x{:}A \vdash^{\mathbf{CC}^+} B\ \mathbf{type}}{\Gamma \vdash^{\mathbf{CC}^+} \Pi x{:}A.B\ \mathbf{type}}$$

By induction hypothesis, we have $\Gamma' \vdash^{\mathbf{HOL}} A'$ **type** and $\Gamma', x{:}A' \vdash^{\mathbf{HOL}} B'$ **type**. There are four cases to consider.

1. $A' \equiv \mathbf{Prf}(P_A)$ and $B' \equiv \mathbf{Prf}(P_B)$, then $(\Pi x{:}A.B)' \equiv \mathbf{Prf}(P_A \supset P_B)$. By induction hypothesis, $\Gamma' \vdash^{\mathbf{HOL}} P_A : Prop$ and $\Gamma', x{:}A' \vdash^{\mathbf{HOL}} P_B : Prop$. Since $A' \equiv \mathbf{Prf}(P_A)$, we have $x \notin FV(P_B)$. So, by the strengthening lemma for **HOL**, $\Gamma' \vdash^{\mathbf{HOL}} P_B : Prop$. Hence, by $(\supset)^{\mathbf{HOL}}$ and $(\mathbf{Prf})^{\mathbf{HOL}}$, we have

$$\Gamma' \vdash^{\mathbf{HOL}} \mathbf{Prf}(P_A \supset P_B)\ \mathbf{type}$$

2. $A' \not\equiv$ **Prf**(...) and $B' \equiv \mathbf{Prf}(P_B)$, then $(\Pi x{:}A.B)' \equiv \mathbf{Prf}(\forall x{:}A'.P_B)$. By induction hypothesis, $\Gamma', x{:}A' \vdash^{\mathbf{HOL}} P_B : Prop$. Then, by $(\forall)^{\mathbf{HOL}}$ and $(\mathbf{Prf})^{\mathbf{HOL}}$, we have

$$\Gamma' \vdash^{\mathbf{HOL}} \mathbf{Prf}(\forall x{:}A'.P_B)\ \mathbf{type}$$

3. $A' \equiv$ **Prf**(...) and $B' \not\equiv$ **Prf**(...), then $(\Pi x{:}A.B)' \equiv B'$. Since $A' \equiv$ **Prf**(...), $x \notin FV(B')$. By induction hypothesis and the strengthening lemma for **HOL**,

$$\Gamma' \vdash^{\mathbf{HOL}} B'\ \mathbf{type}$$

4. $A', B' \not\equiv$ **Prf**(...), then $(\Pi x{:}A.B)' \equiv A' \to B'$. Since $B' \not\equiv$ **Prf**(...), $x \notin FV(B')$. So, by induction hypothesis and the strengthening lemma for **HOL**, $\Gamma' \vdash^{\mathbf{HOL}} B'$ **type**. Then, by $(\to)^{\mathbf{HOL}}$ and induction hypothesis,

$$\Gamma' \vdash^{\mathbf{HOL}} A' \to B'\ \mathbf{type}$$

So, we have $\Gamma' \vdash^{\mathbf{HOL}} (\Pi x{:}A.B)'$ **type**. □

Theorem 2.4 (conservativity) $\mathbf{CC}^+$ *is a conservative extension of* **HOL**; *i.e., if* $\Gamma \vdash^{\mathbf{HOL}} A$ **type**, *then A is Γ-inhabited in* $\mathbf{CC}^+$ *if and only if A is Γ-inhabited in* **HOL**.

Proof By the projection theorem and the inclusion of **HOL** into $\mathbf{CC}^+$. □

Corollary 2.5 *A logical formula in Church's simple type theory is provable if and only if its corresponding small type is inhabited in* $\mathbf{CC}^+$. □

Remark The above conservativity result can also be thought of as a *faithfulness* or *completeness* result, which can be read as that the interpretation of higher-order logic in $\mathbf{CC}^+$ is faithful or, viewing type theories as a semantic base, the interpretation is (sound and) complete. □

3 Abstract Reasoning and Discussions

The above conservativity result gives us a better understanding of the formulae-as-types principle for higher-order logical systems. Such an understanding of type theory is important both in theory and in pragmatic applications where type theories like the calculus of constructions are used in, for example, theorem proving and program development. We now discuss briefly some of the related issues.

3.1 Predicativity vs. impredicativity

First, we have noticed that the notions of sets and logical formulas are in some sense *not* identifiable in an impredicative type theory in which impredicative types represent logical formulae. In fact, in traditional studies of logic, logicians rarely regarded sets as formulas in the strong sense as, for example, in Martin-Löf's type theory. In particular, the object set of a logic is not (and should not) be regarded as identifiable with a logical formula; for example, in Church's formulation of higher-order logic [Chu40], the object set is a type which has nothing to do with a formula. It is some modern type theories (Martin-Löf's theory, the calculus of constructions among others) that make it possible to have a 'double identity' for a type to denote. However, it seems that, both in theoretical and pragmatical aspects, we should be careful about such an identification of concepts. The conservativity problem discussed above gives us an example where a distinction between sets and formulas is important both in theory and applications, when an impredicative type theory is considered. Note that we have always been emphasizing that we are considering *impredicative* type systems. For predicative type systems, *e.g.*, Edinburgh LF [HHP87], the story seems different. For example, one can show that the Edinburgh LF, viewed as a logic, is a conservative extension of first-order logic (see [Ber90]), where the object sets can be represented by (predicative) small types which represent logical formulae as well.

3.2 Predicative universes and abstract reasoning

Abstract reasoning, as motivated in section 1.1, requires to formalize abstract mathematics like theories for abstract algebras and notions in category theory. An adequate formalization of a notion of (arbitrary) set is needed when, for example, formalizing the carrier set of an abstract algebra or the object class of the category of small categories. We briefly show how this can be done by considering predicative universes.

3.2.1 predicative universes

Predicative universes like those in Martin-Löf's type theory can be introduced into an impredicative type theory like the calculus of constructions, as shown in [Coq86][Luo88,89a]. In the extended calculus of constructions (**ECC**) [Luo89a,90], the impredicative type universe is viewed as providing higher-order logical mechanisms, while the predicative universes, where the ordinary sets reside, providing abstraction mechanisms which are rich and expressive for formalization of mathematical problems and program development.

Adding a predicative universe to the calculus of constructions is to introduce a type which has as objects the names of the existing types. Such a universe $Type_0$ can be given by the following rules (here, we use type equality judgement in Martin-Löf's style, although it is not necessary):

$$\frac{\Gamma\ \mathbf{valid}}{\Gamma \vdash Type_0\ \mathbf{type}} \qquad \frac{\Gamma \vdash a : Type_0}{\Gamma \vdash \mathbf{T}_0(a)\ \mathbf{type}}$$

The type *Prop* is reflected in $Type_0$:

$$\frac{\Gamma\ \mathbf{valid}}{\Gamma \vdash prop : Type_0} \qquad \frac{\Gamma\ \mathbf{valid}}{\Gamma \vdash \mathbf{T}_0(prop) = Prop}$$

Π-types are reflected by:

$$\frac{\Gamma \vdash a : Type_0 \quad \Gamma, x{:}\mathbf{T}_0(a) \vdash b : Type_0}{\Gamma \vdash \pi x{:}a.b : Type_0} \qquad \frac{\Gamma \vdash a : Type_0 \quad \Gamma, x{:}\mathbf{T}_0(a) \vdash b : Type_0}{\Gamma \vdash \mathbf{T}_0(\pi x{:}a.b) = \Pi x{:}\mathbf{T}_0(a).\mathbf{T}_0(b)}$$

Small types are (lifted and) reflected by:

$$\frac{\Gamma \vdash P : Prop}{\Gamma \vdash \mathbf{t}_0(P) : Type_0} \qquad \frac{\Gamma \vdash P : Prop}{\Gamma \vdash \mathbf{T}_0(\mathbf{t}_0(P)) = \mathbf{Prf}(P)}$$

Note that the universe $Type_0$ is not reflected in itself, for otherwise, the resulting system would be logically inconsistent (by Girard's paradox). Hence, $Type_0$ does not contain (the names of) all of the types, *e.g.*, it does not contain a name of itself. However, one can iterate the above process to introduce higher universes $Type_1$, $Type_2$, ..., so that the existing types after $Type_i$ is introduced, are reflected in $Type_{i+1}$. Introducing such a reflection principle into the calculus of constructions is logically consistent and the resulting type system has nice proof-theoretic properties like Church-Rosser and strong normalization [Luo88,90]. A realizability model of such a calculus can be found in [Luo89a], where predicative universes are interpreted as large set universes while the impredicative 'universe' *Prop* corresponds to the category of partial equivalence relations.

It is worth remarking that, in the above formulation of type theory, we do *not* have type variables; in other words, we do not allow to have an assumption like X **type** in a context and hence there is no way to quantify over *all* types. Besides assuring that the system does not suffer logical paradoxes, this in particular allows the type theory to remain *open* as Martin-Löf's type theory does [ML73,84]. That is, new types (and new type constructors) can be added to the system when it is needed. For example, one may introduce strong sum types (Σ-types) as in the extended calculus of constructions [Luo89a,90] or inductive types [CM90][Ore89]. The desire to do abstract reasoning can be fulfilled by using predicative universes which reflect the classes of sets which have been defined and under consideration. This reflection principle, although it does not give us the (unachievable) power to talk about all types including those which have not been introduced or defined, seems to be the best one can gain. On the other hand, it does provide us a powerful tool to do abstract reasoning and is enough in applications. In practice, the tension of worrying about universe levels can be eased by considering a proper notion of universe subtyping, as formulated in **ECC** [Luo89a,90] and a notion of universe polymorphism [Hue87][HP89][Pol90a] where one can omit the universe levels to write *Type* instead of $Type_i$, as implemented in the proof development system Lego [LPT89]. In such a setting, to assume X:*Type* in a context, is in some sense equivalent to assume that X be an arbitrary type. One can also quantify over *Type* to talk about 'all types', bearing in mind that it is the machine who does the work to avoid universe circularity (by giving an error message when it occurs).

3.2.2 formalization and reasoning of abstract mathematics

Our semantic consideration and the results about the conservativity problem both show that, in an impredicative type theory used as a logical system, arbitrary sets should not be represented as impredicative small types; for otherwise, one might be able to prove some unexpected logical consequences, *i.e.*, the formalization would be inadequate. For example, it would be inadequate to introduce an arbitrary monoid by the following context:

$$X{:}Prop,\ e{:}\mathbf{Prf}(X),\ \circ{:}\mathbf{Prf}(X) \to \mathbf{Prf}(X) \to \mathbf{Prf}(X),\ \dots\dots$$

The way we have suggested is to use predicative universes (and the mechanism for universe polymorphism) to formalize an arbitrary monoid as follows:

$$X{:}Type,\ e{:}X,\ \circ{:}X \to X \to X,\ \dots\dots$$

Strong sum types can be introduced as in **ECC** so that a notion of abstract theory (as opposed to a context which we call a concrete theory) can be considered and used to do abstract reasoning and structured reasoning (see [Luo89b,90] for details). An example of proof development using such a theory facility of abstract reasoning embodied in **ECC** is Pollack's proof of Tarski's fixpoint theorem in Lego [Pol90b]. It is also possible to formalize category theory in such a setting.

Adequate formalization is in fact a general problem in computer science (*c.f.*, requirements specification). What we have discussed is a particular case when using a type theory as a language of specification. There is no general way using which one can get an adequate specification, except that we should get a good understanding of the language (a type theory in our case) and the problem to be formalized (abstract algebras, for example). This paper addresses a particular problem in this aspect.

3.3 Extensionality: a pragmatic example?

We now discuss an example which concerns about the following extensionality axiom, which Church mentioned in [Chu40] and is a consistent extension of Church's simple type theory:

$$Ext \equiv \forall P, Q{:}Prop.\ (P \supset Q) \supset (Q \supset P) \supset (P =_{Prop} Q)$$

where $=_{Prop}$ is the Leibniz's equality between propositions defined as:

$$(P =_{Prop} Q) =_{\mathrm{df}} \forall f{:}Prop \to Prop.\ f(P) \supset f(Q)$$

Berardi [Ber90] uses this to prove the non-conservativity of the calculus of constructions over higher-order logic as follows. Assuming *Ext*, one would get the following consequence in the calculus of constructions

- For any proposition X of type *Prop*, X is isomorphic to $X \to X$.

In other words, one can show that a proposition X with different proofs can not be finite (up to the intensional equality) under the extensionality axiom! This is in fact not surprising, because the extensionality axiom is actually saying that 'there are in consequence only two propositions' [Chu40], one inhabited and the other not. This is certainly against the idea of representing the object set in logic as a proposition. The Church's simple type theory extended by the extensionality axiom should certainly allow arbitrary interpretation of the object set, since it is an arbitrary type but not a proposition. Using this fact, Berardi [Ber90] argues model-theoretically that the interpretation of object set as a proposition in the calculus of constructions is not conservative. As we can readily see, in **HOL** as presented in this paper, *Obj* can not be proved to be isomorphic to *Obj* $\to$ *Obj* by assuming the extensionality axiom *Ext* above.

A question can now be asked about adequate formalization. Does the above example about extensionality give us a counter-example against the way to use propositions to represent arbitrary sets? To some extent it does, because it shows that considering *Prop* as the universe of sets conflicts with the proof-irrelevance principle people normally believe in. However, one who tries to argue for using small types to represent sets might still think that the axiom *Ext* should not be allowed at all. The author could not think of a *real* pragmatic counter-example giving more direct justifications of the view of adequate formalization (or it does not exist?) than the conservativity problem discussed in this paper, although we believe that the material in this paper has given a strong support. To conclude this paper, we leave the problem to find such an example or to disprove its existence as a further open question to consider.

Acknoledgement Thanks to Rod Burstall and Randy Pollack for many helpful discussions.

References

[Bar89] H.P Barendregt, *'Introduction to Generalized Type Systems'*, to appear in Proc. of the 3rd Italian Conf. on Theoretical Computer Science, Mandera.

[Ber89] S. Berardi, *Non-conservativity of Coquand's Calculus with respect to Higher-order Intuitionistic Logic,* Talk given in the 3rd Jumelage meeting on Typed Lambda Calculi, Edinburgh.

[Ber90] S. Berardi, *Type Dependence and Constructive Mathematics*, thesis manuscript.

[CH88] Th. Coquand and G. Huet, *'The Calculus of Constructions'*, Information and Computation 76(2/3).

[Chu40] A. Church, *'A Formulation of the Simple Theory of Types'*, J. Symbolic Logic 5(1).

[CM90] Th. Coquand and Ch. Paulin-Mohring, *'Inductively Defined Types'*, Lecture Notes in Computer Science 417.

[Coq86] Th. Coquand, *'An Analysis of Girard's Paradox'*, Proc. 1st Ann. Symp. on Logic in Computer Science.

[Coq89] Th. Coquand, *'Metamathematical Investigations of a Calculus of Constructions'*, manuscript.

[Geu89] H. Geuvers, Talk given in the 3rd Jumelage meeting on Typed Lambda Calculi, Edinburgh, Sept. 1989.

[Gir72] J.-Y. Girard, *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur,* These, Université Paris VII.

[HHP87] R. Harper, F. Honsell and G. Plotkin, *'A Framework for Defining Logics'*, Proc. 2nd Ann. Symp. on Logic in Computer Science.

[HP89] R. Harper and R. Pollack, *'Type Checking, Universe Polymorphism, and Typical Ambiguity in the Calculus of Constructions'*, To appear in Theoretical Computer Science.

[Hue87] G. Huet, *'Extending the Calculus of Constructions with Type:Type'*, unpublished manuscript.

[LPT89] Z. Luo, R. Pollack and P. Taylor, *How to Use LEGO: a preliminary user's manual,* LFCS Technical Notes LFCS-TN-27, Dept. of Computer Science, Edinburgh University.

[LS86] J. Lambek and P.J. Scott, *Introduction to higher-order categorical logic,* Cambridge University Press.

[Luo88] Zhaohui Luo, $\mathrm{CC}^{\infty}_{\subset}$ *and Its Meta Theory,* LFCS report ECS-LFCS-88-58, Dept. of Computer Science, Univ. of Edinburgh.

[Luo89a] Zhaohui Luo, 'ECC, *an Extended Calculus of Constructions',* Proc. of the Fourth Ann. Symp. on Logic in Computer Science, June 1989, Asilomar, California, U.S.A.

[Luo89b] Zhaohui Luo, *'A Higher-order Calculus and Theory Abstraction',* To appear in Information and Computation.

[Luo90] Zhaohui Luo, *An Extended Calculus of Constructions,* PhD thesis, University of Edinburgh.

[ML73] Per Martin-Löf, *'An Intuitionistic Theory of Types: Predicative Part',* in Logic Colloquium'73, (eds.) H.Rose and J.C.Shepherdson.

[ML84] Per Martin-Löf, *Intuitionistic Type Theory,* Bibliopolis.

[Moh89] Ch. Paulin-Mohring, *'Extracting* F^{ω} *Programs from Proofs in the Calculus of Constructions'*, Proc. Principles of Programming Languages 1989.

[Ore89] C-E. Ore, *'Notes about the Extensions of* **ECC** *for Including Inductive (Recursive) Types'*, draft.

[Pol90a] R. Pollack, *'The Implicit Syntax'*, manuscript.

[Pol90b] R. Pollack, *'The Tarski Fixpoint Theorem'*, private communication.

[Sch77] K. Schütte, *Proof Theory*, Springer-Verlag.

[Tak75] G. Takeuti, *Proof Theory,* Stud. Logic 81.