A Higher-order Calculus and Theory Abstraction

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

Abstract

We present a higher-order calculus **ECC** which naturally combines Coquand-Huet's calculus of constructions and Martin-Löf's type theory with universes. **ECC** is very expressive, both for structured abstract reasoning and for program specification and construction. In particular, the strong sum types together with the type universes provide a useful module mechanism for abstract description of mathematical theories and adequate formalization of abstract mathematics. This allows comprehensive structuring of interactive development of specifications, programs and proofs.

After a summary of the meta-theoretic properties of the calculus, an ω–**Set** (realizability) model of **ECC** is described to show how its essential properties can be captured set-theoretically. The model construction entails the logical consistency of the calculus and gives some hints on how to adequately formalize abstract mathematics. Theory abstraction in **ECC** is discussed as a pragmatic application.

[*]Supported by a studentship of the University of Edinburgh and the ORS award

1 Introduction

The issue of abstraction and modularization has been one of the central problems considered in the design of programming and specification languages. It is also important in proof engineering. In recent years, the growing interests in the theory and methodology of computer-assisted reasoning (*c.f.*, [Bur86]) have led to the development of various proof development systems many of which are based on type systems such as type theories of Martin-Löf and NuPRL [ML73,84][Con86], the Edinburgh Logical Framework [HHP87] and the Calculus of Constructions [CH88]. To meet the challenge of 'proving in the large', one of the problems one faces is how to express abstract structures and modularize proof development so that large theorem-proving tasks can be conquered in a structured way. Practical experience shows that the lack of a module mechanism is a big obstacle to large proof development in applications.

In this paper, we present and study an *Extended Calculus of Constructions*, **ECC**, which can be seen as a natural combination of Coquand-Huet's calculus of constructions [CH88] and Martin-Löf's type theory with universes [ML73]. It extends the calculus of constructions with

- *Σ-types* (or, *strong sum types*), and
- *a fully cumulative type hierarchy.*

There are two main motivations to make these extensions. One is to consider the extended calculus as a programming logic for specification and development of programs (in a style similar to that of Martin-Löf's type theory, *c.f.*, [NPS88][Bur89]). Another motivation, which we emphasize in this paper, is *theory abstraction*. Σ-types in **ECC**, together with the type hierarchy, provide a powerful abstraction mechanism so that abstract structures can be naturally expressed and mathematical *theories* can be abstractly described and structured, leading to a comprehensive structuring of mathematical texts in interactive proof development and program specification. The cumulative type hierarchy, inspired by the work of Martin-Löf [ML73,84] and Coquand [Coq86a], also increases the expressiveness in another aspect so that, for example, abstract mathematics (*e.g.*, abstract algebras, categories) may be *adequately* formalized. Furthermore, as the type hierarchy provides a strong and flexible form of polymorphism, **ECC** provides a higher-order module mechanism which supports *structure sharing by parameterization* in the style of programming language Pebble [BLam84][LB88][Bur84].

The strong sum, $\Sigma x{:}A.B$, intuitively represents the set of (dependent) pairs of elements of A and B (B may be dependent on elements of A):

$$\{\ (a,b)\ \mid\ a \in A, b \in B(a)\ \}$$

Elements of $\Sigma x{:}A.B$ can be analyzed by using the two projections:

$$\pi_1(a,b) = a \quad \text{and} \quad \pi_2(a,b) = b$$

The basic idea of using strong sum to express abstract structures is best explained through an example. In a type system like Constructions, one postulates a (concrete) theory by assuming a context. For example, an arbitrary semigroup may be introduced by postulating the following context:

$$X{:}Type_0,\ \circ{:}X \to X \to X,\ p{:}P_{ASS}$$

where an arbitrary type X stands for the carrier, $\circ$ for the binary operation over X, and p is an assumed proof of the axiom of associativity $P_{ASS} \equiv \Pi x, y, z{:}X.(x \circ (y \circ z) = (x \circ y) \circ z)$. When a large proof uses many theories, which may depend on one and another in various ways, some notion of 'modularization' is needed to control the complexity. This is analogous to the need for modules in programming in the large.

The strong sum is a basic adequate mechanism to solve this problem, for it can be used to express abstract structures. For example, if strong sum is available, we can express an *abstract* theory of semigroups as consisting of two parts:

- an (abstract) *signature presentation* $Sig\_SG \equiv \Sigma X{:}Type_0.X \to X \to X$;
- the (abstract) *axiom* which is a map Ax_SG which, when given any structure s of type Sig_SG, returns the associativity axiom for s.

Furthermore, these two parts of the semigroup abstraction can be 'packaged' together as

$$SG \equiv \Sigma s{:}Sig\_SG.Ax\_SG(s)$$

Then, to postulate an arbitrary semigroup is just to assume a context $sg{:}SG$. The projection operators can be used to extract the components of any semigroup (*i.e.*, an object of type SG). Such a facility of expressing abstract structures, combined with the power of the type hierarchy, allows us to develop a nice approach to *structured abstract reasoning* (section 4).

However, there is a technical difficulty in extending the calculus of constructions by strong sums. That is, adding type-indexed strong sums (*i.e.*, $\Sigma x{:}A.P$ with A being a type and P a proposition) directly to the proposition level of the calculus of constructions results in a logically inconsistent system in which Girard's paradox can be derived [Coq86a][HH86][MH88]. As propositions play an essential role in expressing mathematical problems (*e.g.*, $Ax\_SG(s)$ in SG), this difficulty appears serious and prevents people from directly extending Constructions by strong sums to express abstract structures. We solve it by lifting propositions as types (see the following explanations).

The infinite type hierarchy in **ECC** is similar to that presented in [Coq86a] (and that of Martin-Löf's type theory [ML73,84]) but is *fully cumulative* in the following sense. First, unlike the original presentations of the calculi of constructions [CH88][Coq85][Coq86a], the propositions at the lowest impredicative level are *lifted* as higher-level types (of their proofs). This lifting is essential for Σ-types in **ECC** to play their role as an abstraction mechanism, and it solves the technical difficulty mentioned above. Secondly, type inclusions between the type universes are coherently expanded to the other types so that a strong form of type unicity is achieved; this yields a simple notion of *principal type* and a simple algorithm for type inference.

ECC has good proof-theoretic properties. Particularly, it is *strongly normalizing*, which shows the proof-theoretic consistency of **ECC** (and in general, Constructions with infinite type universes) and establishes the theoretical basis of an implementation (*e.g.*, decidability of convertibility and type checking).

We give in this paper an (intuitionistic) set-theoretic semantics of **ECC** in the framework of ω-sets [Mog85][LM88][Hyl87] which captures the intuitive meaning of the constructs in the calculus and reflects its essential properties. In addition to its importance in better understanding the calculus, such a model-theoretic semantics seems useful when considering pragmatics of the calculus, *e.g.*, how to *adequately* formalize mathematical problems in the calculus.

In section 2, **ECC** is described and its main meta-theoretic properties are briefly discussed. Section 3 describes the ω–**Set** model. One of the pragmatic aspects of **ECC** — theory abstraction — is discussed in section 4. As a conclusion, section 5 and section 6 discuss some related work and further research topics.

2 ECC

ECC consists of an underlying term calculus and a set of rules for inferring judgements.

2.1 The term calculus

The basic expressions of the term calculus, called *terms*, are inductively defined by the following clauses:

- The constants *Prop* and $Type_j$ ($j \in \omega$), called *kinds*, are terms;
- Variables (x,y,...) are terms;

- If M, N and A are terms, so are the following:

$$\Pi x{:}M.N,\ \lambda x{:}M.N,\ MN,\ \Sigma x{:}M.N,\ \mathbf{pair}_A(M,N),\ \pi_1(M),\ \pi_2(M)$$

In $\Pi x{:}M.N$, $\Sigma x{:}M.N$ and $\lambda x{:}M.N$, the free occurrences of variable x in N (but not those in M) are bound by the binding operators Πx, Σx and λx, respectively. Terms which are the same up to changes of bound variables are identified. ($\equiv$ is used for the syntactic identity between expressions such as terms.) For a term M, $FV(M)$ is the set of free variables occurring in M. When $x \notin FV(N)$, $\Pi x{:}M.N$ and $\Sigma x{:}M.N$ can be abbreviated as $M \to N$ and $M \times N$, respectively.

Reduction ($\rhd$) and conversion ($\simeq$) are defined as usual with respect to the following one-step contraction schemes:

$$(\beta) \qquad (\lambda x{:}A.M)N \rhd_1 [N/x]M$$

$$(\sigma) \qquad \pi_i(\mathbf{pair}_A(M_1,M_2)) \rhd_1 M_i \quad (i=1,2)$$

where $[N/x]M$, the substitution of term N for the free occurrences of variable x in M, is defined as usual with possible changes of bound variables.

The kinds are also called *type universes*. Every kind is assigned a number as its *level*:

$$\mathcal{L}(Prop) =_{\mathrm{df}} -1 \quad \text{and} \quad \mathcal{L}(Type_j) =_{\mathrm{df}} j \ (j \in \omega)$$

The type inclusions between the universes induce the type cumulativity that is syntactically characterized by the following relation.

Definition 2.1 (cumulativity relation) *Let $\preceq_i$ $(i \in \omega)$ be the binary relations over terms inductively defined as follows:*

1. *$A \preceq_0 B$ if and only if one of the following holds:*
   (a) $A \simeq B$; or
   (b) $A \simeq Prop$ and $B \simeq Type_j$ for some $j \in \omega$; or
   (c) $A \simeq Type_j$ and $B \simeq Type_k$ for some $j < k$.
2. *$A \preceq_{i+1} B$ if and only if one of the following holds:*
   (a) $A \preceq_i B$; or
   (b) $A \simeq \Pi x{:}A_1.A_2$ and $B \simeq \Pi x{:}B_1.B_2$ for some $A_1 \simeq B_1$ and $A_2 \preceq_i B_2$; or
   (c) $A \simeq \Sigma x{:}A_1.A_2$ and $B \simeq \Sigma x{:}B_1.B_2$ for some $A_1 \preceq_i B_1$ and $A_2 \preceq_i B_2$.

Define the cumulativity relation $\preceq$ *as*

$$\preceq \ =_{\mathrm{df}} \bigcup_{i\in\omega} \preceq_i$$

Furthermore, $A \prec B$ if and only if $A \preceq B$ and $A \not\simeq B$. □

Remark It can be proved, by using Church-Rosser theorem (theorem 2.2), that the cumulativity relation $\preceq$ is the smallest binary relation over terms such that

1. $\preceq$ is a partial order with respect to conversion; that is, (1) if $A \simeq B$, then $A \preceq B$, (2) if $A \preceq B$ and $B \preceq A$, then $A \simeq B$, and (3) if $A \preceq B$ and $B \preceq C$, then $A \preceq C$;
2. $Prop \preceq Type_0 \preceq Type_1 \preceq \ldots$;
3. if $A_1 \simeq B_1$ and $A_2 \preceq B_2$, then $\Pi x{:}A_1.A_2 \preceq \Pi x{:}B_1.B_2$;
4. if $A_1 \preceq B_1$ and $A_2 \preceq B_2$, then $\Sigma x{:}A_1.A_2 \preceq \Sigma x{:}B_1.B_2$. □

2.2 Judgements and inference rules

We now describe the judgement form and the inference rules of **ECC**.

Contexts are finite sequences of expressions of the form $x{:}M$, where x is a variable and M is a term. The empty context is denoted by $\langle\rangle$. The set of free variables in a context $\Gamma \equiv x_1{:}A_1, ..., x_n{:}A_n$, is defined as $FV(\Gamma) =_{\rm df} \bigcup_{1\leq i\leq n}(\{x_i\} \cup FV(A_i))$.

Judgements are of the form

$$\Gamma \vdash M : A$$

where Γ is a context, and M and A are terms. The intuitive meaning is that M *has type* A *in context* Γ. We write $\vdash M : A$ for $\langle\rangle \vdash M : A$.

The *inference rules* of **ECC** are listed as follows, where K and K' stand for arbitrary kinds, i, j and k for natural numbers and $\mathcal{L}(K)$ for the level of kind K:

(Ax)
$$\frac{}{\vdash Prop : Type_0}$$

(C)
$$\frac{\Gamma \vdash A : K}{\Gamma, x{:}A \vdash Prop : Type_0} \quad (x \notin FV(\Gamma))$$

(T)
$$\frac{\Gamma \vdash Prop : Type_0}{\Gamma \vdash Type_j : Type_{j+1}}$$

(var)
$$\frac{\Gamma, x{:}A, \Gamma' \vdash Prop : Type_0}{\Gamma, x{:}A, \Gamma' \vdash x : A}$$

$(\Pi 1)$
$$\frac{\Gamma, x{:}A \vdash P : Prop}{\Gamma \vdash \Pi x{:}A.P : Prop}$$

$(\Pi 2)$
$$\frac{\Gamma \vdash A : K \quad \Gamma, x{:}A \vdash B : Type_j}{\Gamma \vdash \Pi x{:}A.B : Type_k} \quad (k = max\{\mathcal{L}(K), j\})$$

(λ)
$$\frac{\Gamma, x{:}A \vdash M : B \quad \Gamma, x{:}A \vdash B : K}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B}$$

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

(Σ)
$$\frac{\Gamma \vdash A : K \quad \Gamma, x{:}A \vdash B : K'}{\Gamma \vdash \Sigma x{:}A.B : Type_k} \quad (k = max\{0, \mathcal{L}(K), \mathcal{L}(K')\})$$

$(pair)$
$$\frac{\Gamma \vdash M : A' \quad \Gamma \vdash N : B' \quad \Gamma, x{:}A \vdash B : K}{\Gamma \vdash \mathbf{pair}_{\Sigma x:A.B}(M, N) : \Sigma x{:}A.B} \quad (A' \preceq A, B' \preceq [M/x]B)$$

$(\pi 1)$
$$\frac{\Gamma \vdash M : \Sigma x{:}A.B}{\Gamma \vdash \pi_1(M) : A}$$

$(\pi 2)$
$$\frac{\Gamma \vdash M : \Sigma x{:}A.B}{\Gamma \vdash \pi_2(M) : [\pi_1(M)/x]B}$$

$(conv)$
$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A' : K}{\Gamma \vdash M : A'} \quad (A \simeq A')$$

(cum)
$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A' : K}{\Gamma \vdash M : A'} \quad (A \prec A')$$

A *derivation* of a judgement J is a finite sequence of judgements $J_1, ..., J_n$ with $J_n \equiv J$ such that, for all $1 \leq i \leq n$, J_i is the conclusion of some instance of an inference rule whose premises are in $\{ J_j \mid j < i \}$. A judgement J is *derivable* if there is a derivation of J.

We shall write $\Gamma \vdash M : A$ for '$\Gamma \vdash M : A$ is derivable'. We also often abbreviate $\Gamma \vdash Prop : Type_0$ as 'Γ is valid'. In fact, besides asserting that $Prop$ has type $Type_0$, $\Gamma \vdash Prop : Type_0$ also plays the role in the calculus of asserting that Γ is a valid context.

A term M is called a Γ-*term* (or *well-typed term under* Γ) if $\Gamma \vdash M : A$ for some A. A term A is called a Γ-*type* if $\Gamma \vdash A : K$ for some kind K. A Γ-type A is called a Γ-*proposition* if $\Gamma \vdash A' : Prop$ for some $A' \simeq A$ and called a *proper* Γ-*type* otherwise.

This completes our formal presentation of the calculus. The next section contains some informal remarks on design decisions.

2.3 Remarks

ECC is a natural combination of Coquand-Huet's calculus of constructions [CH88] and Martin-Löf's type theory with universes [ML73]. It extends the calculus of constructions by adding Σ-types and a cumulative type hierarchy. It can also be seen as an extension of the core of Martin-Löf's type theory with universes by adding a lowest *impredicative* level of propositions which stand for logical formulas by the Curry-Howard principle of formulas-as-types [CF58][How69].

The type universes provide us very rich type structures and make it possible to formalize the notion of arbitrary set by reflection principle — a basis to formalize abstract mathematics (*e.g.*, abstract algebras and categories). Viewing intuitively types as sets and ':' as the membership relation, we have

$$Prop \in Type_0 \in Type_1 \in ... \qquad \text{(by } (Ax) \text{ and } (T))$$

$$Prop \subseteq Type_0 \subseteq Type_1 \subseteq ... \qquad \text{(by } (cum))$$

In particular, unlike the original presentations of the calculi of constructions [CH88][Coq85][Coq86a], propositions are *lifted* to higher-level types ($Prop \subseteq Type_0$). It might appear that this would propagate the impredicativity at the level of propositions to the higher levels. For instance, we can derive $\vdash \Pi x{:}Type_j \Pi B{:}Type_j \rightarrow Prop.Bx : Type_j$. However, the type hierarchy, except the lowest level $Prop$, is still stratified (predicative) in the sense that the types can be ranked in such a way that the existence of any proper type depends only on the existence of types with lower ranks. This stratification of the type hierarchy is essential for the logical consistency of the calculus. (A more detailed analysis of this can be found in [Luo89b].)

The idea of lifting propositions to higher-level types is very important. (Philosophically, it is natural to think of propositions as types but *not* vice versa.) It is essential for Σ-types in **ECC** to be useful as an abstraction tool to express abstract mathematical theories and program specifications, since adding (type-indexed) Σ-types to the impredicative level of Constructions would produce an inconsistent system in which Girard's paradox can be derived [Coq86a][HH86][MH88]. Note that, in **ECC**, $\Sigma x{:}A.P$ is *not* a proposition even when P is. However, as propositions are lifted, we have

$$\frac{\Gamma \vdash A : Type_j \quad \Gamma, x{:}A \vdash P : Prop}{\Gamma \vdash \Sigma x{:}A.P : Type_j}$$

This $\Sigma x{:}A.P$ intuitively represents the set of pairs of an element a of A and a proof of the proposition $P(a)$, *i.e.*, the intuitionistic subset type (*c.f.*, [ML84]). It is this property that enables propositions to be used to express abstract axioms and program properties in abstract mathematical theories (see section 4) and program specifications expressed as Σ-types.

The type hierarchy is *fully cumulative*. The inference rule (cum) is a design decision which achieves a strong form of *type unicity* so that there is a simple notion of *principal type* (theorem 2.5) and a very straightforward algorithm for type inference (theorem 2.9) [Luo89a][Luo89b]. The other presentations in the literature of type universes with universe inclusions like those in [ML84][Coq86a] do not have this property: although every well-typed term has a *minimum type*, it is sometimes *not* the most general one [Luo88b]. For example, for the system presented on page 235 in [Coq86a], it is easy to show by induction on derivations that $x{:}Type_0 \to Type_0 \not\vdash x : Type_0 \to Type_1$.

The pairs are 'heavily typed' to avoid undesirable type ambiguity which would make type inference and type-checking difficult (perhaps impossible) [Luo88a]. For example, if untyped pairs were used, the untyped pair $(Type_0, Prop)$ would have both $\Sigma x{:}Type_1.x$ and $Type_1 \times Type_0$ as its types which are incompatible. But note that in **ECC**, thanks to the full cumulativity of types, we still have as expected, say $\vdash \mathbf{pair}_{Type_1 \times Type_0}(Type_0, Prop) : Type_2 \times Type_2$.

The cumulativity relation $\preceq$ defined in definition 2.1 is *not* completely contravariant for Π: for $\Pi x{:}A_1.A_2$ to be less than or equal to $\Pi x{:}B_1.B_2$, A_1 is required to be *convertible* to B_1 instead of $B_1 \preceq A_1$. One may take the latter decision and the proof-theoretic properties will still hold. The only difference from the proof-theoretic point of view is that some terms would get more types. For example, $\lambda x{:}Type_1.x$ would not only have types $Type_1 \to Type_j$, but have $Prop \to Type_j$ and $Type_0 \to Type_j$ $(j \geq 1)$ as its types as well. However, semantically, the type inclusions thus defined would be reflected by coercions instead of by set inclusions.

Finally, we note that the inference rules presented in this paper are slightly different from those in [Luo89a]. The presentation here enjoys the property that the use of rule (*cum*) can be postponed (lemma 3.10) which helps to give a good definition of the model by induction on derivations, while the presentation in [Luo89a] is simpler. However, the two presentations are derivably equivalent.

2.4 Main meta-theoretic properties

ECC has good meta-theoretic properties. Restricted by space, we only briefly describe the main properties.

First, the Church-Rosser property holds for the term calculus described in 2.1.

Theorem 2.2 (Church-Rosser theorem) *If $M_1 \simeq M_2$, then there exists M such that $M_1 \rhd M$ and $M_2 \rhd M$.*

Proof Similar to that in [ML72]. See [Luo89b]. □

Remark The inclusion of either η-reduction or the rule for surjective pairing would make Church-Rosser fail [vD80][Klo80] for the untyped term calculus. In fact, with either of these rules, Church-Rosser even fails for well-typed terms of **ECC** because of the existence of type inclusions induced by universes. □

We now state some basic properties of the calculus whose proofs can be found in [Luo89b] (also see [Luo88b]).

Theorem 2.3 *In* **ECC**, *we have*

1. *Any derivation of $\Gamma, x{:}A, \Gamma' \vdash M : B$ has a sub-derivation of $\Gamma \vdash A : K$ for some kind K.*
2. *(context validity) Any derivation of $\Gamma, \Gamma' \vdash M : A$ has a sub-derivation of $\Gamma \vdash Prop : Type_0$.*
3. *(weakening) If $\Gamma \vdash M : A$ and Γ' is a valid context which contains every component of Γ, then $\Gamma' \vdash M : A$.*
4. *(context replacement) If $\Gamma, x{:}A, \Gamma' \vdash M : C$ and $B \preceq A$ is a Γ-type, then $\Gamma, x{:}B, \Gamma' \vdash M : C$.*
5. *(cut) If $\Gamma, x{:}A, \Gamma' \vdash N : B$ and $\Gamma \vdash M : A$, then $\Gamma, [M/x]\Gamma' \vdash [M/x]N : [M/x]B$.*
6. *If $\Gamma \vdash M : A$, then A is a Γ-type.*
7. *(subject reduction) If $\Gamma \vdash M : A$ and $M \rhd N$, then $\Gamma \vdash N : A$.*

8. *(strengthening) If* $\Gamma, y{:}Y, \Gamma' \vdash M : A$ *and* $y \notin FV(M) \cup FV(A) \cup FV(\Gamma')$, *then* $\Gamma, \Gamma' \vdash M : A$;

9. *(type cumulativity) Let* A *and* B *be* Γ*-types. Then,* $A \preceq B$ *if and only if* $\Gamma, x{:}A \vdash x : B$, *where* $x \notin FV(\Gamma)$. □

Because of the type inclusions induced by type universes, type uniqueness (up to conversion) fails. However, we have a simple notion of principal type which characterizes the set of types of a well-typed term.

Definition 2.4 (principal type) *A is called a* principal type *of* M *(under* Γ*) if and only if*

1. $\Gamma \vdash M : A$, *and*

2. *for any term* A', $\Gamma \vdash M : A'$ *if and only if* $A \preceq A'$ *and* A' *is a* Γ*-type.* □

Theorem 2.5 (existence of principal type) *Every* Γ*-term* M *has a principal type under* Γ, *denoted as* $T_\Gamma(M)$; $T_\Gamma(M)$ *is the minimum type of* M *under* Γ *with respect to the cumulativity relation* $\preceq$.

Proof The theorem follows from the properties 6 and 9 of theorem 2.3 and the 'diamond property' of the cumulativity relation: $\Gamma \vdash M : A$ and $\Gamma \vdash M : B$ implies $\Gamma \vdash M : C$ for some $C \preceq A, B$. □

Remark The principal type $T_\Gamma(M)$ is obviously unique (up to conversion). The existence of the principal type is not only a good proof-theoretic property but allows simple implementation of the type hierarchy for **ECC**. □

Theorem 2.6 (strong normalization) **ECC** *is strongly normalizing, i.e., if* $\Gamma \vdash M : A$, *then* M *is strongly normalizable.* □

Remark This result shows the proof-theoretic consistency of Constructions with infinite type universes and establishes the theoretical foundation of an implementation. The proof is quite difficult. It uses Girard-Tait's reducibility method [Gir72][Tai75] and is based on the proofs of strong normalization for the pure calculus of constructions by Coquand [Coq86b] and Pottinger [Pot87]. One of the special key points of this proof is to find a suitable *ranking* of the types to make explicit the predicativity of the type universes $Type_j$. We do this by proving a *quasi-normalization* theorem which enables us to define a two-dimensional ranking measure. Details and analysis of this will appear in a paper in preparation (also see [Luo89b][Luo88b]). □

Corollary 2.7 (consistency) **ECC** *is logically consistent. In particular, for any term* M, $\nvdash M : \Pi x{:}Prop.x$. □

Remark The consistency of the calculus also follows from our model construction (theorem 3.11) or even a proof-irrelevant model-theoretic argument (*c.f.*, [Coq89]) which are easier than proving normalization. □

Corollary 2.8 (decidability of conversion) *It is decidable whether* $M \simeq N$ *for arbitrary well-typed terms* M *and* N. □

As the convertibility for well-typed terms is decidable, so is the cumulativity relation $\preceq$. Hence, we have

Theorem 2.9 (type inference) *There is a simple algorithm such that, when given a context* Γ *and a term* M, *it checks whether* M *is a* Γ*-term, and if so, returns the principal type of* M *under* Γ. □

Corollary 2.10 (decidability of type-checking) **ECC** *is decidable, i.e., it is decidable whether* $\Gamma \vdash M : A$ *for arbitrary* Γ, M *and* A. □

3 An ω–Set Model of ECC

In this section, we construct a realizability model of **ECC** which gives an (intuitionistic) set-theoretic semantics of the calculus. The model captures the intuitive meanings of the constructs in the calculus and reflects its essential properties such as logical consistency. It also gives some hints on how abstract mathematics may be adequately formalized.

3.1 Basic ideas and basic notions

The main question in interpreting **ECC** set-theoretically is how to interpret the type universes and the type formation operators Π and Σ so that, intuitively,

1. $Prop \in Type_0 \in Type_1 \in ...$;
2. $Prop \subseteq Type_0 \subseteq Type_1 \subseteq ...$;
3. $Type_j$ is closed under Π and Σ;
4. $Prop$ is closed under Π.

These requirements prevent us from giving a naive non-trivial classical set-theoretic model of **ECC**. (See [Rey84][RP88][LM88][Pit87][CGW87][Mes88] for discussions for modeling the second-order λ-calculus [Gir86][Rey74] which is a sub-system of **ECC**.)

Fortunately, the idea of interpreting types as partial equivalence relations [Gir72][Tro73][Mog85] provides us a nice framework of ω-sets and modest sets [Mog85][LM88][Hyl87] in which there is an interpretation of **ECC** satisfying the above requirements. In particular, proper types can be interpreted as ω-sets and propositions as objects of a full subcategory **PROP** of the category of ω-sets which is isomorphic to the category of partial equivalence relations. The lowest impredicative universe (*Prop*) corresponds to the category **PROP** and the predicative universes ($Type_j$) correspond to large set universes. (This idea of interpreting $Type_j$ as large set universes is suggested to the author by Hayashi, Moggi and Coquand.)

The rest of this subsection is devoted to the basic notions of ω-sets and modest sets, and to the basic ideas for interpreting **ECC** in order to satisfy the requirements mentioned above. In fact, the basic framework is slightly extended in order to gain a good interpretation of the type hierarchy.

Definition 3.1 (ω-sets) *An* ω-set $A = (|A|, \Vdash_A)$ *consists of a (carrier) set* $|A|$ *and a binary (realizability) relation* $\Vdash_A \subseteq \omega \times |A|$ *such that*

$$\forall a \in |A|.\ \exists n \in \omega.\ n \Vdash_A a$$

A morphism f between two ω-sets A and B is a function $f : |A| \to |B|$ such that $\exists n \in \omega.\ n \Vdash_{A,B} f$, where (with nm denoting the result of Kleene application of n to m)

$$n \Vdash_{A,B} f \text{ if and only if } \forall a \in |A|\ \forall m \in \omega.\ m \Vdash_A a \Rightarrow nm \Vdash_B f(a)$$

The ω-sets and the morphisms between them form the category of ω-sets*, denoted as* ω–**Set**. □

The category of ω-sets, and the category of modest sets defined below, are concrete locally cartesian closed categories [Mog85][LM88][Hyl87].

We now define three ω-set constructors σ, σ_Γ and π_Γ which will be used as basic operations to interpret valid contexts, Σ-types and Π-types in our model, respectively.

Definition 3.2 (σ, σ_Γ and π_Γ) *Suppose that Γ is an ω-set and $A : |\Gamma| \to \omega$–***Set***.*

(σ) *$\sigma(\Gamma, A)$ is defined to be the following ω-set:*

$$|\sigma(\Gamma, A)| =_{\mathrm{df}} \{ (\gamma, a) \mid \gamma \in |\Gamma|, a \in |A(\gamma)| \}$$

$$\langle m, n\rangle \Vdash_{\sigma(\Gamma,A)} (\gamma, a) \text{ if and only if } m \Vdash_{\Gamma} \gamma \text{ and } n \Vdash_{A(\gamma)} a$$

(σ_Γ) *Let $B : |\sigma(\Gamma, A)| \to \omega$–**Set**. $\sigma_\Gamma(A, B)$ is the function from $|\Gamma|$ to ω–**Set** defined as, for $\gamma \in |\Gamma|$,*

$$|\sigma_\Gamma(A, B)(\gamma)| =_{\mathrm{df}} \{ (a, b) \mid a \in |A(\gamma)|, b \in |B(\gamma, a)| \}$$

$$\langle m, n\rangle \Vdash_{\sigma_\Gamma(A,B)(\gamma)} (a, b) \text{ if and only if } m \Vdash_{A(\gamma)} a \text{ and } n \Vdash_{B(\gamma,a)} b$$

(π_Γ) *Let $B : |\sigma(\Gamma, A)| \to \omega$–**Set**. $\pi_\Gamma(A, B)$ is the function from $|\Gamma|$ to ω–**Set** defined as, for $\gamma \in |\Gamma|$, $|\pi_\Gamma(A, B)(\gamma)|$ is the set of the functions $f : |A(\gamma)| \to \bigcup_{a\in|A(\gamma)|} |B(\gamma, a)|$ such that*

$$\forall a \in |A(\gamma)|.\ f(a) \in |B(\gamma, a)| \text{ and } \exists n \in \omega.\ n \Vdash_{\pi_\Gamma(A,B)(\gamma)} f$$

and $n \Vdash_{\pi_\Gamma(A,B)(\gamma)} f$ if and only if

$$\forall a \in |A(\gamma)|\ \forall p \in \omega.\ p \Vdash_{A(\gamma)} a \Rightarrow np \Vdash_{B(\gamma,a)} f(a)$$

□

Large set universes are used to interpret the predicative universes $Type_j$ so that the closedness requirement 3 is satisfied. A basic insight is that the notions of ω-sets and modest sets have nothing to do with *sizes* of the sets under consideration. Consider ZFC set theory with infinite inaccessible cardinals $\kappa_0 < \kappa_1 < \ldots$ (Recall that a cardinal κ is (strongly) inaccessible if it is uncountable and regular, and $2^\lambda < \kappa$ for all $\lambda < \kappa$. See, *e.g.*, [Lev79][Dev79].), and let V_α be the cumulative hierarchy of sets. Then $Type_j$ is interpreted to correspond to the following category ω–**Set**(j).

Definition 3.3 *ω–**Set**(j) is the full subcategory of ω–**Set** whose objects are those ω-sets whose carrier sets are in V_{κ_j}.* □

As V_{κ_j} is a model of ZFC set theory, we have

Lemma 3.4 *σ_Γ and π_Γ are closed for ω–**Set**(j), that is, if $A : |\Gamma| \to \omega$–**Set**(j) and $B : |\sigma(\Gamma, A)| \to \omega$–**Set**$(j)$, then $\sigma_\Gamma(A, B), \pi_\Gamma(A, B) : |\Gamma| \to \omega$–**Set**$(j)$.* □

Remark The above lemma meets the closedness requirement 3. As $V_{\kappa_i} \subseteq V_{\kappa_{i+1}}$, ω–**Set**(j) is a full subcategory of ω–**Set**$(j + 1)$, satisfying the inclusion requirement 2 between the $Type_j$. Note that ω–**Set**(j) are *small* categories. Therefore, they can be naturally viewed as ω-sets through the embedding functor Δ from the category of sets **Set** to ω–**Set** defined as $\Delta(X) =_{\mathrm{df}} (X, \omega \times X)$ for $X \in \mathrm{Obj}(\textbf{Set})$, and $\Delta(f) =_{\mathrm{df}} f$ for $f : X \to Y$ in **Set**. As $V_{\kappa_j} \in V_{\kappa_{j+1}}$, we have $\Delta(\mathrm{Obj}(\omega\text{–}\textbf{Set}(j))) \in \mathrm{Obj}(\omega\text{–}\textbf{Set}(j + 1))$. This satisfies the membership requirement 1 between the $Type_j$. □

To interpret the propositions, the notion of modest set is essential, as the category of modest sets **M** is closed for arbitrary products.

Definition 3.5 (modest sets) *A* modest set *is an ω-set A such that*

$$\forall n \in \omega\ \forall a, b \in |A|.\ n \Vdash_A a \text{ and } n \Vdash_A b \ \Rightarrow\ a = b$$

The category of modest sets*, denoted as* **M***, is the full subcategory of ω–**Set** with the modest sets as its objects.* □

Lemma 3.6 *π_Γ is closed for the modest sets in the sense that, for all $A : |\Gamma| \to \omega$–**Set**, $B : |\sigma(\Gamma, A)| \to \mathbf{M}$, we have $\pi_\Gamma(A, B) : |\Gamma| \to \mathbf{M}$.*

Proof See [LM88]. □

Unlike the case of models (*e.g.*, [LM88]) of the second-order lambda calculus, where the only universe is itself not a type, although **M** is closed for arbitrary products as the above lemma shows it can not be directly used to interpret the impredicative universe *Prop* in Constructions-like calculi. The reason is that **M** itself is *not* a small category. If *Prop* were interpreted as **M**, there would be no way to justify $Prop \in Type_0$. Fortunately, **M** is a *small complete* category in the sense that it is equivalent to the following small category **PROP**, which is isomorphic to the category of partial equivalence relations over ω. (Recall that R is a partial equivalence relation if R is symmetric and transitive.)

Definition 3.7 (PROP) *The category* **PROP** *is the full subcategory of* **M** *(hence, of ω–***Set***) with the following object set:*

$$\mathrm{Obj}(\mathbf{PROP}) =_{\mathrm{df}} \{\, (Q(R), \in) \mid R \subseteq \omega \times \omega \textit{ is a partial equivalence relation} \,\}$$

where, $Q(R) = \{\, [n]_R \mid (n, n) \in R \,\}$ is the quotient set with respect to R. □

Lemma 3.8 *There is an equivalence of categories* **back** : **M** → **PROP** *such that* **back**$(A) \cong A$ *for $A \in \mathrm{Obj}(\mathbf{M})$, and* **back**$(P) = P$ *for $P \in \mathrm{Obj}(\mathbf{PROP})$.*

Proof Define **back** : **M** → **PROP** as follows: for $A \in \mathrm{Obj}(\mathbf{M})$,

$$\mathbf{back}(A) =_{\mathrm{df}} (Q(R_A), \in)$$

where $R_A = \{\, (n, m) \mid \exists a \in A.\ n \Vdash_A a \text{ and } m \Vdash_A a \,\}$ is the partial equivalence relation induced by A, and, for any morphism $f : A \to B$ in **M**,

$$\mathbf{back}(f)([p]_{R_A}) =_{\mathrm{df}} [np]_{R_B} \quad \text{where } n \Vdash_{A,B} f$$

back is a category equivalence with the inclusion functor **inc** : **PROP** → **M** as its inverse. In fact, we have the identity natural transformation $id : id_{\mathbf{PROP}} \to \mathbf{back} \circ \mathbf{inc}$ and a natural transformation

$$\eta : id_{\mathbf{M}} \to \mathbf{inc} \circ \mathbf{back}$$

defined as follows: for $A \in \mathrm{Obj}(\mathbf{M})$ and $a \in |A|$, $\eta_A(a) =_{\mathrm{df}} [n]_{R_A}$, where $n \Vdash_A a$. Hence, for all $A \in \mathrm{Obj}(\mathbf{M})$, $\mathbf{back}(A) = \mathbf{inc} \circ \mathbf{back}(A) \cong A$. Furthermore, for $P = (Q(R), \in) \in \mathrm{Obj}(\mathbf{PROP})$, it is easy to show that $R_P = R$, and so $\mathbf{back}(P) = (Q(R_P), \in) = (Q(R), \in) = P$. □

Remark *Prop* is interpreted to correspond to the category **PROP**. The existence of the equivalence **back** is important to satisfy the closedness requirement 4 as **M** is closed under π_Γ. The requirement $Prop \subseteq Type_0$ is satisfied by the fact that **PROP** is a full subcategory of ω–**Set**(0) and, the requirement $Prop \in Type_0$ by the fact $\Delta(\mathrm{Obj}(\mathbf{PROP})) \in \mathrm{Obj}(\omega\text{–}\mathbf{Set}(0))$. □

3.2 Model construction

We give every valid context and every derivable judgement a unique denotation such that they satisfy some desirable properties. A valid context is interpreted as an ω-set and a derivable judgement $\Gamma \vdash M : A$ as a 'Γ-indexed element of A'.

Notation (FPP property) Let $\Gamma \in \omega$–**Set** and $A : |\Gamma| \to \omega$–**Set**. A morphism $f : \Gamma \to \sigma(\Gamma, A)$ in ω–**Set** *satisfies the first projection (FPP) property*, written as

$$f : \Gamma \to_{FPP} \sigma(\Gamma, A)$$

if and only if $p(\Gamma, A) \circ f = id_\Gamma$, where $p(\Gamma, A) : \sigma(\Gamma, A) \to \Gamma$ is the morphism defined by $p(\Gamma, A)(\gamma, a) =_{\rm df} \gamma$. Intuitively, $f : \Gamma \to_{FPP} \sigma(\Gamma, A)$ is a 'Γ-indexed element of A'. □

Theorem 3.9 (Interpretation) *There is an interpretation* $[\![\_]\!]$ *of the valid contexts and the derivable judgements of* **ECC** *such that*

1. *if* $\Gamma \vdash Prop : Type_0$, *then* $[\![\Gamma]\!] \in \mathrm{Obj}(\omega\text{–}\mathbf{Set})$;
2. *if* $\Gamma \vdash M : A$, *then* $[\![\Gamma \vdash M : A]\!] : [\![\Gamma]\!] \to_{FPP} [\![\Gamma, x{:}A]\!]$;
3. *if* $A \preceq A'$ *are* Γ*-types, then there is an inclusion morphism* $\mathbf{inc}_\Gamma(A, A') : [\![\Gamma, x{:}A]\!] \hookrightarrow [\![\Gamma, x{:}A']\!]$ *such that, if* $\Gamma \vdash M : A$, $\Gamma \vdash N : A'$ *and* $M \simeq N$, *then* $[\![\Gamma \vdash N : A']\!] = \mathbf{inc}_\Gamma(A, A') \circ [\![\Gamma \vdash M : A]\!]$. □

Before defining the interpretation, we discuss a notational convention and a notion of canonical derivation. First, different from the traditional simpler cases (*c.f.*, [See84]), types and objects in Constructions-like calculi are mixed up. Types are also objects with kinds as their types. Therefore, a type has a 'double identity' in the model, playing slightly different roles when viewed as a type or as an object. This is reflected technically as a correspondence between functions and a special kind of morphisms in ω–**Set**.

Convention Suppose $\Gamma \in \omega\text{–}\mathbf{Set}$ and $K : |\Gamma| \to \omega\text{–}\mathbf{Set}$ is a constant function such that, for some set X, $K(\gamma) = \Delta(X) = (X, \omega \times X)$ for all $\gamma \in |\Gamma|$. Then, there is a one-one correspondence between the morphisms from Γ to $\sigma(\Gamma, K)$ which satisfy the first projection property and the functions from $|\Gamma|$ to X.

- Given $f : \Gamma \to_{FPP} \sigma(\Gamma, K)$, the corresponding function $f^\star : |\Gamma| \to X$ is defined by $f^\star(\gamma) =_{\rm df} x$, where $\gamma \in |\Gamma|$ and $f(\gamma) = (\gamma, x)$;
- Given $g : |\Gamma| \to X$, the corresponding morphism $g^\circ : \Gamma \to_{FPP} \sigma(\Gamma, K)$ is defined by $g^\circ(\gamma) =_{\rm df} (\gamma, x)$, where $\gamma \in |\Gamma|$ and $g(\gamma) = x$.

We have, $f^{\star\circ} = f$ and $g^{\circ\star} = g$. Based on this, we shall in the proof below of theorem 3.9 adopt the convention that we omit $\star$ and $\circ$ for convenience. □

Secondly, we introduce a notion of canonical derivation in order to give a good inductive definition of the interpretation. As pointed out by Coquand [Coq89], it is a nice meta-theoretic property for a calculus that a judgement have at most one derivation (up to conversion). The work by Streicher [Str88] shows that this property is very helpful, and sometimes necessary, to define a semantics by induction on derivations. When a calculus lacks such a property, a definition of semantics may assign different denotations to the same judgement. Streicher [Str88] gives a way of solving this problem, first defining denotations by induction on the structure of pre-judgements (instead of on derivations), and then proving that the definition gives a unique denotation to every derivable judgement.

It is obvious that our presentation of **ECC** does *not* have the property that every judgement has at most one derivation because the universe inclusions induce general inclusions between types (rule (*cum*)). However, because of the existence of principal types, we can introduce a notion of *canonical derivation* which exists for every derivable judgement. The essential idea is to postpone all of the uses of the (*cum*) rule to the last step. The existence of canonical derivations enables us to construct the model by induction on canonical derivations.

Lemma 3.10 (postponing (*cum*)**)** *In* **ECC**, *every derivable judgement has a derivation in which rule* (*cum*) *is not used except in the last step.*

Proof By induction on derivations. □

An immediate consequence of the above lemma is that every derivable judgement has a derivation $J_1, ..., J_n$ such that $J_1, ..., J_{n-1}$ are all of the form $\Gamma \vdash M : T_\Gamma(M)$ and J_n is deduced by rule (*conv*) or (*cum*). We call a derivation of this form a *canonical derivation*. Note that the canonical derivations of a judgement are essentially the same (up to conversion). Furthermore, all judgements in a canonical derivation of a judgement of the form $\Gamma' \vdash N : T_{\Gamma'}(N)$ have the form $\Gamma \vdash M : T_\Gamma(M)$. Now, we return to prove theorem 3.9 and define the model.

Proof (of theorem 3.9) An interpretation satisfying the stated properties is defined by induction on the length of the shortest canonical derivations of a judgement.

First, we interpret valid contexts. Recall that a context Γ is valid if and only if $\Gamma \vdash Prop : Type_0$ is derivable. By rule (Ax), the empty context is valid. It is interpreted as the terminal object of ω–**Set**:

$$[\![\langle\rangle]\!] =_{\text{df}} \mathbf{1} = (1, \omega \times 1)$$

A valid context $\Gamma, x{:}A$ (by rule (C)) is interpreted as the ω-set

$$[\![\Gamma, x{:}A]\!] =_{\text{df}} \sigma([\![\Gamma]\!], [\![\Gamma \vdash A : K]\!])$$

Note that the interpretation of a kind (see below) is an FPP-morphism corresponding to a constant function. By the convention we discussed above, $[\![\Gamma, x{:}A]\!]$ is well-defined. (We will not detail such correctness claims below but leave them to the reader.)

The derivable judgements are interpreted by considering the last rule used in canonical derivations.

(Ax)(C) $\Gamma \vdash Prop : Type_0$ is interpreted as

$$[\![\Gamma \vdash Prop : Type_0]\!](\gamma) =_{\text{df}} (\gamma, \Delta(\text{Obj}(\mathbf{PROP})))$$

where $\gamma \in |\,[\![\Gamma]\!]\,|$ and Δ is the embedding functor from **Set** to ω–**Set**.

(T) $\Gamma \vdash Type_j : Type_{j+1}$ is interpreted as, for $\gamma \in |\,[\![\Gamma]\!]\,|$,

$$[\![\Gamma \vdash Type_j : Type_{j+1}]\!](\gamma) =_{\text{df}} (\gamma, \Delta(\text{Obj}(\omega\text{–}\mathbf{Set}(j))))$$

(var) Define $[\![\Gamma, x{:}A, \Gamma' \vdash x : A]\!]$ to be the function with domain $|\,[\![\Gamma, x{:}A, \Gamma']\!]\,|$ by

$$[\![\Gamma, x{:}A, \Gamma' \vdash x : A]\!](\gamma, a, \gamma') =_{\text{df}} ((\gamma, a, \gamma'), a)$$

(Π1) $\Gamma \vdash \Pi x{:}A.P : Prop$ is interpreted as

$$[\![\Gamma \vdash \Pi x{:}A.P : Prop]\!] =_{\text{df}} \mathbf{back} \circ \pi_{[\![\Gamma]\!]}([\![\Gamma \vdash A : T_\Gamma(A)]\!], [\![\Gamma, x{:}A \vdash P : Prop]\!])$$

where **back** is the category equivalence defined in the proof of lemma 3.8.

(Π2) $\Gamma \vdash \Pi x{:}A.B : Type_k$ is interpreted as

$$[\![\Gamma \vdash \Pi x{:}A.B : Type_k]\!] =_{\text{df}} \pi_{[\![\Gamma]\!]}([\![\Gamma \vdash A : K]\!], [\![\Gamma, x{:}A \vdash B : Type_j]\!])$$

(λ) There are two cases to consider.

1. $\Gamma \not\vdash \Pi x{:}A.B : Prop$. Then, for $\gamma \in |\,[\![\Gamma]\!]\,|$,

$$[\![\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B]\!](\gamma) =_{\text{df}} (\gamma, g_\gamma)$$

where, supposing $P = \pi_{[\![\Gamma]\!]}([\![\Gamma \vdash A : T_\Gamma(A)]\!], [\![\Gamma, x{:}A \vdash B : T_{\Gamma,x:A}(B)]\!])$, $g_\gamma \in |P(\gamma)|$ is the function such that, if $[\![\Gamma, x{:}A \vdash M : B]\!](\gamma, a) = ((\gamma, a), b)$, then $g_\gamma(a) = b$.

2. $\Gamma \vdash \Pi x{:}A.B : Prop$. Then, for $\gamma \in |\,[\![\Gamma]\!]\,|$,

$$[\![\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B]\!](\gamma) =_{\mathrm{df}} (\gamma, \eta_{P(\gamma)}(g_\gamma))$$

where P and g_γ are as in the above case and η is the natural transformation defined in the proof of lemma 3.8.

(app) There are two cases to consider:

1. $\Gamma, x{:}A \not\vdash B : Prop$. Then, for $\gamma \in |\,[\![\Gamma]\!]\,|$, if $[\![\Gamma \vdash M : \Pi x{:}A.B]\!](\gamma) = (\gamma, f)$ and $[\![\Gamma \vdash N : A']\!](\gamma) = (\gamma, a)$, then
$$[\![\Gamma \vdash MN : [N/x]B]\!](\gamma) =_{\mathrm{df}} (\gamma, f(a))$$
2. $\Gamma, x{:}A \vdash B : Prop$. Then, for $\gamma \in |\,[\![\Gamma]\!]\,|$, if $[\![\Gamma \vdash M : \Pi x{:}A.B]\!](\gamma) = (\gamma, [n]_{R_{[\![\Gamma \vdash \Pi x:A.B:T_\Gamma(\Pi x:A.B)]\!](\gamma)}})$, $[\![\Gamma \vdash N : A']\!](\gamma) = (\gamma, a)$ and $p \Vdash_{[\![\Gamma \vdash A:T_\Gamma(A)]\!](\gamma)} a$, then
$$[\![\Gamma \vdash MN : [N/x]B]\!](\gamma) =_{\mathrm{df}} (\gamma, [np]_{R_{[\![\Gamma, x:A \vdash B:T_{\Gamma, x:A}(B)]\!](\gamma, a)}})$$
where the R's are the partial equivalence relations as defined in the proof of lemma 3.8.

(Σ) $\Gamma \vdash \Sigma x{:}A.B : Type_k$ is interpreted as,

$$[\![\Gamma \vdash \Sigma x{:}A.B : Type_k]\!] =_{\mathrm{df}} \sigma_{[\![\Gamma]\!]}([\![\Gamma \vdash A : K]\!], [\![\Gamma, x{:}A \vdash B : K']\!])$$

(pair) For $\gamma \in |\,[\![\Gamma]\!]\,|$, if $[\![\Gamma \vdash M : A']\!](\gamma) = (\gamma, m)$ and $[\![\Gamma \vdash N : B']\!](\gamma) = (\gamma, n)$,

$$[\![\Gamma \vdash \mathbf{pair}_{\Sigma x:A.B}(M, N) : \Sigma x{:}A.B]\!](\gamma) =_{\mathrm{df}} (\gamma, (m, n))$$

(π1)(π2) For $\gamma \in |\,[\![\Gamma]\!]\,|$, if $[\![\Gamma \vdash M : \Sigma x{:}A.B]\!](\gamma) = (\gamma, (a, b))$, then

$$[\![\Gamma \vdash \pi_1(M) : A]\!](\gamma) =_{\mathrm{df}} (\gamma, a)$$

$$[\![\Gamma \vdash \pi_2(M) : [N/x]B]\!](\gamma) =_{\mathrm{df}} (\gamma, b)$$

(conv) Define $[\![\Gamma \vdash M : A']\!]$ as $[\![\Gamma \vdash M : A]\!]$.

(cum) $\Gamma \vdash M : A'$ is interpreted as, for $\gamma \in |\,[\![\Gamma]\!]\,|$,

$$[\![\Gamma \vdash M : A']\!](\gamma) =_{\mathrm{df}} [\![\Gamma \vdash M : A]\!](\gamma)$$

It can be verified that the interpretation satisfies the conditions stated in the theorem. Here, we explain how substitution is understood semantically and verify that the interpretation preserves conversion. Suppose that $\Gamma \vdash N : A$ and $\Gamma, x{:}A \vdash M : B$. The judgement $\Gamma \vdash [N/x]M : [N/x]B$ is interpreted as the 'composition' of their interpretations, that is, for $\gamma \in |\,[\![\Gamma]\!]\,|$, if

$$[\![\Gamma, x{:}A \vdash M : B]\!]([\![\Gamma \vdash N : A]\!](\gamma)) = ((\gamma, a), b)$$

then

$$[\![\Gamma \vdash [N/x]M : [N/x]B]\!](\gamma) = (\gamma, b)$$

To verify the preserveness of the convertibility, by Church-Rosser theorem and subject reduction, we only have to show that if $\Gamma \vdash M : A$ and $M \rhd_1 N$, then $[\![\Gamma \vdash M : A]\!] = [\![\Gamma \vdash N : A]\!]$. In other words, we only have to show that

$$[\![\Gamma \vdash (\lambda x{:}A.M)N : [N/x]B]\!] = [\![\Gamma \vdash [N/x]M : [N/x]B]\!]$$

and, let $P \equiv \mathbf{pair}_{\Sigma x:A.B}(M_1, M_2)$,

$$[\![\Gamma \vdash \pi_1(P) : A]\!] = [\![\Gamma \vdash M_1 : A]\!]$$

$$[\![\Gamma \vdash \pi_2(P) : [\pi_1(P)/x]B]\!] = [\![\Gamma \vdash M_2 : [\pi_1(P)/x]B]\!]$$

We only check the case for β-reduction when $\Gamma \vdash N : A$, $\Gamma, x{:}A \vdash M : B$ and $\Gamma, x{:}A \vdash B : Prop$. The other cases can be similarly verified. For $\gamma \in |\,[\![\Gamma]\!]\,|$, by definition, suppose

$$[\![\Gamma \vdash N : A]\!](\gamma) = (\gamma, a) \quad \text{and} \quad p \Vdash_{snd([\![\Gamma\vdash A:T_\Gamma(A)]\!](\gamma))} a$$

$$[\![\Gamma, x{:}A \vdash M : B]\!](\gamma, a) = ((\gamma, a), [m]_{R_{[\![\Gamma,x:A\vdash B:Prop]\!](\gamma,a)}})$$

$$[\![\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B]\!](\gamma) = (\gamma, [n]_{R_{[\![\Gamma\vdash\Pi x:A.B:Prop]\!](\gamma)}})$$

then, by definition, $np \in [m]_{R_{[\![\Gamma,x:A\vdash B:Prop]\!](\gamma,a)}}$. So, we have

$$\begin{array}{rcl} [\![\Gamma \vdash (\lambda x{:}Prop.M)N : [N/x]B]\!](\gamma) & = & (\gamma, [np]_{R_{[\![\Gamma,x:A\vdash B:Prop]\!](\gamma,a)}}) \\ & = & (\gamma, [m]_{R_{[\![\Gamma,x:A\vdash B:Prop]\!](\gamma,a)}}) \\ & = & [\![\Gamma \vdash [N/x]M : [N/x]B]\!](\gamma) \end{array}$$

This completes the proof of the theorem. □

3.3 Consistency and discussions

One of the most important features of the realizability model described above is that it entails the logical consistency of **ECC**.

Theorem 3.11 (consistency) **ECC** *is logically consistent in the sense that there is a proposition which is not inhabited by any term. In particular, for any term M, $\not\vdash M : \Pi x{:}Prop.x$.*

Proof In fact, $[\![\vdash \Pi x{:}Prop.x : Prop]\!]$ is the function from the singleton set $\{*\}$ with $(*, (\emptyset, \emptyset))$ as its image. So, there is no morphism from $[\![\langle\rangle]\!] = \mathbf{1}$ to $[\![y{:}\Pi x{:}Prop.x]\!] = \sigma(\mathbf{1}, [\![\vdash \Pi x{:}Prop.x : Prop]\!])$. By theorem 3.9, we have $\not\vdash M : \Pi x{:}Prop.x$ for any term M. □

By this theorem, the higher-order intuitionistic logic embedded in **ECC** by the Curry-Howard correspondence [CF58][How69] is consistent. This is the most basic requirement for **ECC** to be suitable for theorem-proving and program specification. This is one of the reasons that we view such a model as appropriate. There are other reasonable models. For example, we can give a truth-value model of **ECC** where propositions are interpreted as 0 or 1. Some other models (*e.g.*, domain-theoretic ones) do not capture the essential properties of the calculus like logical consistency.

Besides the consistency, the model described above gives an (intuitionistic) set-theoretic semantics of the calculus. (ω-sets and modest sets can be characterized in the framework of effective topos (**Eff**) [Hyl82][Hyl87], which is a topos-theoretic model of intuitionistic set theory.) The set-theoretic flavor of such a semantics makes possible a deeper understanding of the calculus, and the semantics may be used as the basis of an informal but precise explanation for users doing theorem proving and program specification (*e.g.*, [LPT89]). For example, the intuitions that $a : A$ means that a denotes an element of the set denoted by A, that $P : Prop$ as a proposition and as a lifted type $P : Type_j$ denotes the same set, and that the syntactic type inclusions ($A \preceq A'$) are set inclusions are all reflected by the model.

Another insight one may gain from the model is about how to formalize mathematical problems *adequately*. As we know, one of the basic motivations for introducing type universes is to allow formalization of the notion of an *arbitrary set* by reflection [ML73][Coq89]. Our model gives semantical support to such an idea. For example, it seems to be *not* adequate to formalize an arbitrary group by assuming its carrier by $X{:}Prop$, as we know that X, as a proposition, can not be viewed as an *arbitrary set*. Assuming $X{:}Type_0$ is more adequate as we can view $Type_0$ as containing *almost all* sets as shown by the above model. More research is needed on this aspect.

4 Theory Abstraction in ECC

In this section, we briefly discuss a pragmatic aspect of **ECC** — expressing and structuring mathematical theories in proof development. We show what facilities for abstraction and modularization the calculus provides and how abstract reasoning and structured reasoning can be done in our setting. The idea of structuring theories in proof development is originally introduced to the author by Rod Burstall and the formulation in this paper also benefits from discussions with Coquand, Taylor and Pollack (see [Coq89][Luo89a][TLP89]). One may find a further development of these ideas in [TLP89].

4.1 A notion of (abstract) theory

What is a theory? Different theory manipulation mechanisms give rather different impressions of what a theory might be. Here, we take a simple view that a *theory* in a proof development system basically consists of a *signature* (a group of basic notions, say constants and function symbols), a group of *hypotheses* (say axioms) and the *proved theorems* (possibly together with their *proofs*).

We also conceptually distinguish *concrete* theories and *abstract* theories. For example, a concrete theory of natural numbers would be expressed in **ECC** as a context Γ_{Nat} of the following form

$$nat{:}Type_0,\ 0{:}nat,\ suc{:}nat \to nat,\ +{:}nat \to nat \to nat,\ ...$$

where '...' contains the assumptions of the axioms for natural numbers. Proved theorems of such a concrete theory are then the inhabited propositions proved under it (*i.e.*, some P's such that $\Gamma_{Nat} \vdash p : P$ for some p).

More interestingly, we can express a notion of *abstract theory* as well by using Σ-types and the type hierarchy. For instance, as discussed in the introduction, we can express the abstract theory of semigroups as the Σ-type:

$$SG \equiv \Sigma s{:}Sig\_SG.Ax\_SG(s)$$

where $Sig\_SG \equiv \Sigma X{:}Type_0.X \to X \to X$ and $Ax\_SG(s)$ is the proposition for the associativity axiom.

In general, a presentation of an (abstract) mathematical theory T in **ECC** consists of

- a *signature presentation* Sig_T, which is in general a Σ-type, and
- the *abstract axioms* over the signature, which can be expressed as a propositional function Ax_T of type $Sig\_T \to Prop$.

Then the abstract theory T is expressed by the Σ-type:

$$T \ \equiv \ \Sigma s{:}Sig\_T.Ax\_T(s)$$

The proved (abstract) *theorems* of T are expressed as a function

- Thm_T of type $Sig\_T \to Prop$, which is generally of the form $\lambda s{:}Sig\_T.\ P_1 \wedge P_2 \wedge ... \wedge P_n$, where $\wedge$ is the propositional AND operator defined in the calculus.

The proofs of these theorems constitute a function

- Prf_T of type $\Pi t{:}T.Thm\_T(\pi_1(t))$ which, when given any T-structure satisfying the T-axioms, results in the (concrete) proofs of the theorems for the structure.

Remark It is easy to see that, in this setting, any abstract universal algebra with finitely many sorts, operators and axioms can be formalized as an abstract theory. One can also formalize categorical notions in a similar way. $\square$

4.2 Abstract reasoning

We use the phrase 'abstract reasoning' here in the sense of Paulson [Pau87], where he points out its desirability and the fact that the theory mechanism of Cambridge LCF, which is based on ideas of [SB83], does not support it. The idea of abstract reasoning is that, instead of re-proving a theorem for many concrete theories, we can prove an (abstract) theorem in an (abstract) theory, then simply *instantiate* the abstract proofs as concrete ones for free. The notion of abstract theories for computer-assisted reasoning is analogous to the notion of 'parameterized modules' for modular programming. It becomes more useful as the task of proof development becomes large.

How this idea of abstract reasoning by proof instantiation can be expressed in the notion of theory we presented above is best explained by a simple example. Consider the abstract theory SG of semigroups and suppose that we have proved some (abstract) theorems about it:

$$Thm\_SG \equiv \lambda s{:}Sig\_SG.\ P_1 \wedge ... \wedge P_n$$

$$Prf\_SG \equiv \lambda sg{:}SG.\ and\_intro(p_1, ..., p_n)$$

We can then, for instance, instantiate these theorems and proofs to the concrete ones about natural numbers and $+$ (or other similar concrete theories) whenever we have proved that the structure consisting of nat and $+$ satisfies the associativity axiom (say, with proof ass_nat_plus). The instantiated proofs are then easily constructed as (from now on, we elide the explicitly typed pair operator for notational convenience)

$$Prf\_Nat\_SG \equiv Prf\_SG((nat, +), ass\_nat\_plus)$$

Remark The facility of abstract reasoning comes from the power of Π-abstraction. However, the type universes make it possible to formalize abstract mathematics (like the theory of semigroups) adequately and Σ-types are important for 'packaging' the formalization in a well-structured way. $\square$

4.3 Structured reasoning

In larger proof development activities, one hopes to conquer a big and complex task by dividing it into smaller and simpler ones and then putting the results together in a structured way. We discuss here two aspects of this idea.

4.3.1 proof inheritance

Proof inheritance between theories through theory morphisms [TLP89][Coq89] allows the theorems and proofs of a smaller and weaker theory to be inherited as those of a bigger and stronger theory.

A morphism from an (abstract) theory T to another T' is a pair of functions (f, g) where

$$f : Sig\_T \to Sig\_T'$$

$$g : \Pi s{:}Sig\_T.\ Ax\_T(s) \to Ax\_T'(f(s))$$

The existence of such a morphism means that T is *stronger* than T'. A typical example of such a morphism is when T (say, theory of rings) is a theory which contains more sorts or operators and stronger axioms than a theory T' (say, SG); there is a 'forgetful' morphism whose first component, f, forgets the extra sorts and operators and whose second component gives proofs of the axioms of T' under the translation of f.

Given such a morphism, we can inherit the proofs of theorems in the weaker theory T' as the proofs of the corresponding theorems in T in the following way. Suppose Prf_T' is the (abstract) proofs of the theorems proved for T' which is of type $\Pi t'{:}T'.\ Thm\_T'(\pi_1(t'))$. Then, the corresponding (abstract) theorems in T

$$Thm(T, T') \equiv \lambda s{:}Sig\_T.\ Thm\_T'(f(s))$$

are proved by the following proofs inherited from Prf_T':

$$Prf(T,T') \equiv \lambda t{:}T.\ Prf\_T'(f(\pi_1(t)), g(\pi_1(t), \pi_2(t)))$$

For example, the theorems about semigroups can be inherited as theorems about rings through a forgetful morphism. (There are indeed two forgetful morphisms which concern the operators plus and multiplication, respectively.) The idea of divide-and-conquer (and separation of concerns) is embodied in proof inheritance. Simpler and more general theorems are dealt with in simpler and weaker theories, and then inherited (or lifted) to more complex and stronger theories.

4.3.2 sharing by parameterization

Structure sharing is important for modular proof development just as it is for modular programming. The type hierarchy of **ECC** provides a strong form of polymorphism and hence a facility of defining *higher-order modules*. With this, one can define functions between abstracted modules and express *sharing by parameterization* to structure proof development in the style of Pebble [Bur84][LB88], where the type of all types exists. We explain this by an example.

Example We define a function *ringGen* which results in a ring structure when given as arguments a semigroup and an abelian group with the same carrier, and a proof of the extra axiom (the distributive laws). Suppose the theories of semigroups and abelian groups are defined as follows:

$$SG \equiv \Sigma s{:}\Sigma X{:}Type_0.SGwrt(X).\ Ax\_SG(s)$$

$$AG \equiv \Sigma g{:}\Sigma X{:}Type_0.AGwrt(X).\ Ax\_AG(g)$$

where $SGwrt, AGwrt : Type_0 \to Type_0$ and, when given $X : Type_0$ as carrier, give as results the types of the operations for semigroups and abelian groups with respect to X, respectively, and $Ax\_SG(s)$ and $Ax\_AG(g)$ are the propositions expressing the axioms of theories for semigroups and abelian groups.

ringGen can then be defined as

$$\begin{aligned} ringGen \quad \equiv \quad & \lambda X{:}Type_0 \\ & \lambda * {:}SGwrt(X)\ \lambda p{:}Ax\_SG(X,*) \\ & \lambda(+,0,\ '){:}AGwrt(X)\ \lambda q{:}Ax\_AG(X,+,0,\ ') \\ & \lambda d{:}P_{DISTR}.\ ((X,+,0,\ ',*), and\_intro(p,q,d)) \end{aligned}$$

which is of type

$$\begin{aligned} & \Pi X{:}Type_0 \\ & \Pi * {:}SGwrt(X)\ \Pi p{:}Ax\_SG(X,*) \\ & \Pi g{:}AGwrt(X)\ \Pi q{:}Ax\_AG(X,g) \\ & \Pi d{:}P_{DISTR}.\ Ring \end{aligned}$$

where P_{DISTR} is the proposition for the distributive laws and *Ring* is the Σ-type for the abstract theory of rings defined similarly to *SG* and *AG*. *ringGen* guarantees that its two arguments have the same carrier. □

Note that *SGwrt* and *AGwrt* are sort of 'parameterized modules'. Supported by such a facility, the idea of divide-and-conquer can be successfully used for proof development. For example, *ringGen* is useful to organize proof inheritance when a structure can be viewed as a ring in different ways. When some proofs of justifying the construction of a required structure (ring in this case) are more complicated, this is desirably useful to make proof development structured.

Remark There are several different ways to control structure sharing which appear in programming and specification languages ML [HMM86][Mac86], Pebble [LB88] and Clear [BG80] (see [Bur84] for a simple explanation). Although propositional equality (*e.g.*, Leibniz's equality) can be defined in Constructions, it can *not* be used to express sharing constraints in the style of ML, as Thierry Coquand pointed out to the author. □

4.4 Discussion

We have shown above that the extensions of the calculus of constructions by Σ-types and type universes provide expressive mechanisms to express a notion of (abstract) theory for doing abstract and structured reasoning. These mechanisms can even be internally expressed in **ECC** (an idea due to Pollack and Coquand) [TLP89]; for this the fourth level of the type hierarchy (types of type $Type_2$) is used.

As well-known, existential types (weak sums) [MP85][Rey83][Pra65] can be defined in the calculus of constructions. (They can also be defined at the predicative levels of **ECC** [Luo89a].) However, they are *not* useful to express mathematical theories or program specifications, because the elimination operator for the weak sum is too weak and, in particular, there is no way to prove that the first component of a 'weak pair' of type $\exists x{:}A.B$ satisfies the property B. A comparison of strong and weak sums in the context of modular programming can be found in [Mac86].

The approach to theory abstraction discussed above adopts a view of '*theories as types*'. More precisely, abstract theories are expressed as Σ-types. There is another approach to theory structuring [SB83][BLuo88][HST89] borrowing an idea from research in algebraic specification languages like Clear [BG80]. This latter approach may be called '*theories as values*', as there are theory operations to 'put theories together'. In the AUTOMATH project, ideas like telescope of organizing mathematical texts through manipulating contexts were informally studied [dB80][Zuc75]. (Thanks to a referee for referring the author to the paper [Zuc75].) Further research and experience are needed to show what is necessary and whether it is possible to combine these ideas together.

5 Related Work

The calculus of constructions (CC for short) is studied in [Coq85][CH88][CH85] *etc.*. Its meta theory is developed in [Coq85][Coq86b] and [Pot87]. There are several existing (independent) work on the semantic aspects of Constructions including the following. Hyland and Pitts [HPit87] developed a general approach to categorical semantics of Constructions-like calculi, where an extension of CC with Σ-types and unit types is presented with the motivation of investigating semantics. Streicher [Str88] studied semantics of CC based on the notion of contextual category [Car86] and raised the question of well-definedness of interpretations of categorical semantics. Hyland [Hyl87] gives a clue how a model of CC may be constructed in the framework of ω-sets and Ehrhard [Ehr88] sketched an ω–**Set** model of CC. A full description of an ω–**Set** model of CC (with Σ-types and lifting of propositions as types) can be found in [Luo88a]. In this paper, we have extended the model in [Luo88a] to the type hierarchy (using an idea of Hayashi) and simplified it by using principal types and canonical derivations.

Type universes are first introduced in Martin-Löf's type theory [ML73,84] and also appear in NuPRL's type theory [Con86]. The idea of extending the calculus of constructions by infinite universes appeared in [Coq86a], where the Generalized Calculus of Constructions (GCC for short) is presented. The strong normalization theorem for constructions with infinite type universes was proved in [Luo88b]. The type-checking problem for GCC is considered in [HPol89]; because GCC does not have the property of type unicity, the resulted algorithm is rather complicated compared with that for **ECC** [Luo89a][Luo89b][Pol89].

Σ-types are well-known in Martin-Löf's type theory [ML73,84]. A similar idea of using Σ-types to express modular structures occurs in researches of programming languages (*e.g.*, [BLam84] and

[Mac86]). For programming language research, one does not need to consider logical consistency problem as we do. The idea of lifting propositions as types in Constructions in order to use Σ-types to express abstract structures and mathematical theories was investigated in [Luo88a][Luo89a]. Coquand and Streicher considered more explicit lifting operators to lift propositions [Coq89][Str88], and view the more implicit calculus as an abbreviation of the explicit one [Coq89].

6 Conclusion and Further Research Topics

In this paper, the Extended Calculus of Constructions (**ECC**) is presented and studied. It is an expressive and promising calculus for formalizing mathematical problems in computer-assisted reasoning and program specification. In particular, an ω–**Set** realizability model is given and the pragmatic aspect of theory abstraction is discussed.

By the Curry-Howard principle of formulas-as-types [CF58][How69], there is an embedded logic in **ECC**. We conjecture that this logic is a conservative extension (with respect to some reasonable interpretation) of the (intensional intuitionistic) higher-order logic HOL (*c.f.*, [Chu40][Tak75] for classical ones). This is also relevant to the problem of adequate formalization of abstract mathematics discussed at the end of section 3. The connection is concerned with the following question: *What is a proper way of interpreting the object set in HOL?* We conjecture that *it should be interpreted as a proper type instead of a proposition*; in other words, if the object set of individuals is interpreted as a proposition $Obj{:}Prop$, the interpretation will not give a conservative extension of HOL (the intuition is that too much *computational* power is provided at the impredicative level), and if the object set is interpreted as a proper type (say, $Obj{:}Type_0$), it will give a conservativity result of the embedded logic with respect to HOL. Further research is needed in this aspect. (We also conjectured this in [Luo89a]. Recently, Geuvers [Geu89] and Berardi [Ber89] have independently proved that interpreting the object set as a proposition in the pure calculus of constructions does not yield conservativity, which shows that the first part of our conjecture is right, while the second part is still open.)

The proof-theoretic power of the calculus is unknown. The realizability model given in this paper uses large set universes to interpret the type universes $Type_j$. But it may be possible to give a *small* model of **ECC**.

Doing specifications in type theory has been studied by the Göteborg group [NPS89] and the NuPRL group [Con86] based on type theories of Martin-Löf and NuPRL. An idea called *deliverables* is recently proposed by Burstall [Bur89] using **ECC**. We view **ECC** as a core of a programming logic in whose impredicative level ($Prop$) the embedded logic resides and whose predicative levels provide programming facilities (*c.f.*, [Lei89]). Using **ECC** as a programming logic has certain advantages compared with some other type theories. For example, unlike Martin-Löf's type theories [ML73,84], we do not need to add a new propositional equality in our setting since Leibniz's equality (over any type A, notation $=_A$) is definable in our calculus and, more importantly, it *reflects* the computational equality (conversion) in the sense that $a \simeq b$ whenever $\vdash p : a =_A b$ for some p [Luo89b]. This property of equality reflection gives a good justification of the practice in program specifications in **ECC** where Leibniz's equality is used to reflect computational equality (*c.f.*, [Bur89]). However, it needs further investigations to see whether and how such a calculus can be further developed and put into practice as a real specification and programming language.

Further possible extensions of **ECC** are possible. Inductive types at the predicative levels (*c.f.*, [CP89][Ore89]) may be very useful for program specification and construction. It may be possible to give model-theoretic justifications of such extensions by further extending the framework of ω-sets; but it is still not clear how this can be done.

An interactive proof development system LEGO [Pol89][LPT89] has been implemented by Pollack in Edinburgh. It supports several type theories, of which **ECC** is the strongest at the current time. Further experience with the system should lead to a powerful environment for proof development and program specification.

Acknowledgements I am very grateful to Susumu Hayashi who kindly helped me check a draft of the SN proof and pointed out that the notion of inaccessible cardinal can be used to interpret type universes. Thanks to Eugenio Moggi and Thierry Coquand for their insights and many valuable suggestions. Thanks to Gerard Huet, Randy Pollack, John Power and Paul Taylor for many helpful discussions, and particularly to Pollack who kindly reads an earlier version of this paper and gives many helpful suggestions. Thanks also to the referees for many good and helpful suggestions.

Finally and most gratefully, special thanks to my supervisor Rod Burstall, whose ideas on structured theories and sharing originated my motivations, for his very helpful and continuous guidance in the work.

References

[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, Sept. 1989.

[BG80] R. Burstall and J. Goguen, *'The Semantics of CLEAR, a Specification Language',* Lecture Notes in Computer Science 86.

[BLam84] R. Burstall and B. Lampson, *'Pebble, a Kernel Language for Modules and Abstract Data Types',* Lecture Notes in Computer Science 173.

[BLuo88] R. Burstall and Zhaohui Luo, *'A Set-theoretic Setting for Structuring Theories in Proof Development',* Circulated notes. Apr. 1988.

[Bur84] R. Burstall, *'Programming with Modules as Typed Functional Programming',* Proc. Inter. Conf. on Fifth Generation Computer Systems, Tokyo.

[Bur86] R. Burstall, *Research in Interactive Theorem Proving at Edinburgh University,* Proc. of 20th IBM Computer Science Symposium, Shizuoka, Japan. Also, LFCS Report ECS-LFCS-86-12, Dept. of Computer Science, Univ. of Edinburgh.

[Bur89] R. Burstall, *An Approach to Program Specification and Development in Constructions*, Talk given in Workshop on Programming Logic, Bastad, Sweden, May 1989.

[Car86] J. Cartmell, *'Contextual Category and Generalized Algebraic Theories',* Annals of Pure and Applied Logic 32.

[CF58] H. B. Curry and R. Feys, *Combinatory Logic, Vol. 1,* North Holland Publishing Company.

[CGW87] Th. Coquand, C. Gunter and G. Winskel, *Domain Theoretic Models of Polymorphism,* Tech. Report No. 116, Computer Laboratory, University of Cambridge.

[CH85] Th. Coquand and G. Huet, *'Constructions:a Higher Order Proof System for Mechanizing Mathematics',* EUROCAL'85, Lecture Notes in Computer Science 203.

[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).

[Con86] R. L. Constable et al., *Implementing Mathematics with the NuPRL Proof Development System,* Pretice-Hall.

[Coq85] Th. Coquand, *'Une Theorie des Constructions',* PhD thesis, University of Paris VII.

[Coq86a] Th. Coquand, *'An Analysis of Girard's Paradox'*, Proc. 1st Ann. Symp. on Logic in Computer Science.

[Coq86b] Th. Coquand, *'A Calculus of Constructions'*. manuscript, Nov. 1986.

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

[CP89] Th. Coquand and C. Paulin, *'Inductively Defined Types'*, draft.

[dB80] N. G. de Bruijn, *'A Survey of the Project AUTOMATH'*, In To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, (eds., J. Hindley and J. Seldin), Academic Press.

[Dev79] K. Devlin, *Fundamentals of Contemporary Set Theory,* Springer-Verlag.

[Ehr88] T. Ehrhard, *'A Categorical Semantics of Constructions'*, Proc. 3rd Ann. Symp. on Logic in Computer Science, Edinburgh.

[Geu89] H. Geuvers, *A Modular Proof of Strong Normalization for the Calculus of Constructions*, Talk given in the 3rd Jumelage meeting on Typed Lambda Calculi, Edinburgh, Sept. 1989.

[Gir72] J.-Y. Girard, *Interpretation fonctionelle et elimination des coupures de l'arithmetique d'ordre superieur,* These, Universite Paris VII.

[Gir86] J.-Y. Girard, *'The System F of Variable Types, Fifteen Years Later'*, Theoretical Computer Science 45.

[HH86] J. Hook and D. Howe, *Impredicative Strong Existential Equivalent to Type:Type,* Technical Report TR86-760, Cornell University.

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

[HMM86] R. Harper, D. MacQueen and R. Milner, *Standard ML,* LFCS Report ECS-LFCS-86-2, Dept. of Computer Science, Univ. of Edinburgh.

[How69] W. A. Howard, *'The Formulae-as-types Notion of Construction'*, In To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, (eds., J. Hindley and J. Seldin), Academic Press, 1980.

[HPit87] M. Hyland and A. Pitts, *'The Theory of Constructions: Categorical Semantics and Topos-theoretic Models'*, Categories in Computer Science and Logic, Boulder.

[HPol89] R. Harper and R. Pollack, *'Type Checking, Universe Polymorphism, and Typical Ambiguity in the Calculus of Constructions'*, To appear in Theoretical Computer Science.

[Hyl82] M. Hyland, *'The Effective Topos'*, in The Brouwer Symposium, (eds., A.S.Troelstra and Van Dalen) North-Holland.

[Hyl87] M. Hyland, *'A Small Complete Category'*, To appear in Ann. Pure Appl. Logic.

[Klo80] J. W. Klop, *Combinatory Reduction Systems,* Mathematical Center Tracts 127.

[LB88] B. Lampson and R. Burstall, *'Pebble, a Kernel Language for Modules and Abstract Data Types'*, Information and Computation 76(2/3).

[Lei89] D. Leivant, *'Stratified Polymorphism',* Proc. of the Fourth Symp. on Logic in Computer Science, Asilomar, California, U.S.A.

[Lev79] A. Levy, *Basic Set Theory,* Springer-Verlag.

[LM88] G. Longo and E. Moggi, *Constructive Natural Deduction and Its 'Modest' Interpretation,* Report CMU-CS-88-131, Computer Science Dept., Carnegie Mellon Univ.

[LPT89] Z. Luo, R. Pollack and P. Taylor, *How to Use LEGO: a preliminary user's manual,* LFCS, Dept. of Computer Science, Edinburgh Univ., Apr. 1989.

[Luo88a] Zhaohui Luo, *A Higher-order Calculus and Theory Abstraction,* LFCS report ECS-LFCS-88-57, Dept. of Computer Science, Univ. of Edinburgh.

[Luo88b] Zhaohui Luo, $\mathrm{CC}^{\infty}_{\subseteq}$ *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, *An Extended Calculus of Constructions,* thesis in preparation.

[Mac86] D. MacQueen, *'Using Dependent Types to Express Modular Structure',* Proc. 13th Principles of Programming Languages.

[Mes88] J. Meseguer, *Relating Models of Polymorphism,* SRI-CSL-88-13, Computer Science Lab, SRI International.

[MH88] J. Mitchell and R. Harper, *'The Essence of ML',* Proc. 15th Principles of Programming Languages.

[ML72] Per Martin-Löf, *An Intuitionistic Theory of Types,* manuscript.

[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.

[MP85] J. Mitchell and G. Plotkin, *'Abstract Types Have Existential Type',* Proc. 12th Principles of Programming Languages.

[Mog85] E. Moggi, *'The PER-model as Internal Category with All Small Products',* manuscript.

[NPS88] B. Nordström, K. Petersson and J. Smith, *Programming in Martin-Löf's Type Theory: an introduction,* book to appear.

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

[Pau87] L. Paulson, *Theorem Proving in Cambridge LCF,* Cambridge Press.

[Pit87] A. Pitts, *'Polymorphism is Set Theoretic, Constructively',* Summer Conf. on Category Theory and Computer Science, Edinburgh.

[Pol89] R. Pollack, *'The Theory of LEGO',* manuscript.

[Pot87] G. Pottinger, *Strong Normalization for Terms of the Theory of Constructions,* TR 11-7, Odyssey Research Associates.

[Pra65] D. Prawitz, *Natural Deduction, a Proof-Theoretic Study,* Almqvist & Wiksell.

[Rey74] J. C. Reynolds, *'Towards a Theory of Type Structure',* Lecture Notes in Computer Science 19.

[Rey83] J. C. Reynolds, *'Types, Abstraction and Parameter Polymorphism',* Information Processing'83.

[Rey84] J. C. Reynolds, *'Polymorphism is Not Set-theoretic',* Lecture Notes in Computer Science 173.

[RP88] J. C. Reynolds and G. D. Plotkin, *On Functors Expressible in the Polymorphic Typed Lambda Calculus,* LFCS report, ECS-LFCS-88-53, Dept. of Computer Science, Univ. of Edinburgh.

[SB83] D. Sannella and R. Burstall, *'Structured Theories in LCF',* 8th Colloquium on Trees in Algebra and Programming.

[See84] R. A. G. Seely, *'Locally Cartesian Closed Categories and Type Theory,'* Math. Proc. Camb. Phil. Soc. 95.

[Str88] T. Streicher, *Correctness and Completeness of a Categorical Semantics of the Calculus of Constructions,* PhD Dissertation, Passau.

[Tai75] W. W. Tait, *'A Realizability Interpretation of the Theory of Species',* Logic Colloquium (ed. R. Parikh), Lecture Notes in Computer Science 453.

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

[TLP89] P. Taylor, Z. Luo and R. Pollack, *'Theories, Mathematical Structures and Strong Sums',* in preparation.

[Tro73] A. S. Troelstra, *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis,* Lecture Notes in Mathematics 344.

[vD80] D. T. van Daalen, *The Language Theory of Automath,* PhD Thesis. Technologicval Univ., Eindhoven.

[Zuc75] J. Zucker, *'Formalization of Classical Mathematics in AUTOMATH',* Colloque Internationaux du CNRS 249, Clermont-Ferrand.