An Extended Calculus of Constructions

Zhaohui Luo

Ph. D.
University of Edinburgh
1990

Abstract

This thesis presents and studies a unifying theory of dependent types **ECC** — Extended Calculus of Constructions. **ECC** integrates Coquand-Huet's (impredicative) calculus of constructions and Martin-Löf's (predicative) type theory with universes, and turns out to be a strong and expressive calculus for formalization of mathematics, structured proof development and program specification.

The meta-theory of **ECC** is studied and we show that the calculus has good meta-theoretic properties. The main proof-theoretic result is the *strong normalization* theorem, proved by using Girard-Tait's reducibility method based on a *quasi normalization* theorem which makes explicit the predicativity of the predicative universes. The strong normalization result shows the proof-theoretic consistency of the calculus; in particular, it implies the consistency of the embedded intuitionistic higher-order logic and the decidability of the theory. The meta-theoretic results establish the theoretical foundations both for pragmatic applications in theorem-proving and program specification and for computer implementations of the theory. **ECC** has been implemented in the proof development system LEGO developed by Pollack.

In **ECC**, dependent Σ-types are non-propositional types residing in the predicative universes and propositions are lifted as higher-level types as well. This solves the known difficulty that adding strong Σ-types to an impredicative system results in logical paradox and enables Σ-types to be used to express the intuitionistic notion of subsets. Σ-types together with type universes hence provide useful abstraction and module mechanisms for abstract description of mathematical theories and basic mechanisms for program specification and adequate formalization of abstract mathematics (*e.g.*, abstract algebras and notions in category theory). A notion of (abstract) mathematical theory can be described and leads to a promising approach to *abstract reasoning* and *structured reasoning*. Program specifications can be expressed by Σ-types, using propositions in the embedded logic to describe program properties (for example, by an equality reflection result, computational equality can be modeled by the propositional Leibniz's equality definable in the theory). These developments allow comprehensive structuring of formal or rigorous development of proofs and programs.

Also discussed is how the calculus can be understood set-theoretically. We explain an ω–**Set** (realizability) model of the theory. In particular, propositions can be interpreted as partial equivalence relations and the predicative type universes as corresponding to large set universes.

Acknowledgements

I would like most of all to express my gratitude to my supervisor Rod Burstall who not only brought me into the research area and provided many interesting ideas, able guidance and helpful advice, but also gave me a lot of help and encouragement in many aspects. From him, I have learned a lot about research methodology as well as scientific knowledge.

I would also thank Susumu Hayashi, Eugenio Moggi and Thierry Coquand for their insights and many helpful discussions, from which I have learned a lot about logics, type theories, categorical models *etc.*. Particularly, I am in debt to Hayashi for the wide-ranging discussions we had in Edinburgh from which I benefited very much, for his patience in listening to my detailed explanation of the proof of strong normalization, and for his many good suggestions; to Moggi for introducing me to the work on ω-sets and modest sets, reading a draft of this thesis and giving many suggestions; and to Coquand for the deep influence of his work and his remarks on and suggestions to my work.

Many discussions with Paul Taylor and Randy Pollack, in particular those about theory mechanisms, have proved very helpful; from them, I have learned much about proof development systems.

Thanks also go to Henk Barendregt, Stefano Berardi, Yves Lafont, Pierre-Louis Curien, Herman Geuvers, Furio Honsell, Gerard Huet, Martin Hyland, Josè Messeguer, Christian-Emil Ore, Andy Pitts, Gordon Plotkin, John Power, Anne Salvesen and Don Sannella for the helpful discussions we had.

I also thank the staff, students and my friends in Edinburgh who enabled me to study and work in so good an environment.

I was financially supported by the Studentship of the Edinburgh University and the Overseas Research Students Award.

My thanks to my wife Dejuan can not be expressed enough in any way.

Declaration

I hereby declare that this thesis has been composed by myself, that the work is my own, and the ideas and results that I do not attribute to others are due to myself.

Contents

Chapter 1

Introduction

Computer scientists and mathematicians consider various *constructions*: mathematical objects like proofs and theorems and computational objects like programs and specifications. We consider every construction as an object of certain type; in other words, mathematical objects and computer programs are all associated with their types. Such a view forms a basic starting point of type theories and strongly typed programming languages.

This thesis presents and studies a unifying theory of dependent types, **ECC** — Extended Calculus of Constructions. **ECC** integrates Coquand-Huet's calculus of constructions [CH88][Coq85] and Martin-Löf's type theory with universes [ML73,84], and turns out to be a strong expressive calculus for formalization of mathematics, structured proof development and program specification.

In this introduction, we first give a general and brief discussion about type theories as logical systems and their applications in computer science, by which we hope to provide enough background knowledge and references. Then, we discuss our research motivations and give an overview of the thesis. Related work is also briefly discussed.

1.1 Type Theories and Computer Science

1.1.1 Type theories as logical systems

The principle of propositions-as-types (or formulas-as-types), also known as the Curry-Howard correspondence, is the key idea for viewing (intuitionistic) type theories as logical systems and to apply type theories (and constructive mathematics in general[1]) to computer science. It was discovered by Curry [CF58] and Howard [How69], and further developed by many others (*c.f.*, [Sco70][ML73,84] [dB80]). This principle establishes the relationship between type systems and logical systems for natural deduction.

[1]We do not discuss the more general relationship between constructive mathematics and computer science but only discuss type theories. We then miss the other systems like type-free theories (*c.f.*, [Myh75][Fri77][Fef79][Bee85][HN88]). See [Hay89] for a general survey.

According to the principle of propositions-as-types, a proposition (formula) A corresponds to the type of its proofs, A°, and a construction of the truth of proposition A to an object in the corresponding type A°. For example, according to Heyting's intuitionistic semantics [Hey71], the proposition $A \supset B$ (A implies B) is asserted to hold if, and only if, we have a construction which, whenever given a construction of proposition A, gives a construction of proposition B. In other words, a construction of $A \supset B$ is a function that maps the proofs of A to proofs of B. The set of the constructions of proposition $A \supset B$ corresponds to the function space $A^\circ \to B^\circ$ whose elements (functions) can be expressed by λ-notation. Based on such a correspondence, various typed λ-calculi can be viewed as logical systems, where proof terms correspond to derivations. For instance,

- Simply typed λ-calculus (see [HS87][Bar84]) corresponds to the intuitionistic propositional logic. For this correspondence, see [How69].
- The Edinburgh Logical Framework (LF) [HHP87] (or Automath type theory [dB80][vD80]) corresponds to the intuitionistic first-order logic. For this correspondence, see [Ber89a][BGeu89] where it is proved that LF is a conservative extension of intuitionistic first-order logic.
- Girard's higher-order polymorphic λ-calculus F^ω [Gir72] corresponds to intuitionistic higher-order propositional logic. For this correspondence, see [Gir71].

We shall not give a general overview of development of type theories as logical systems. For details and general comparisons of these systems, we refer to the references above, and particularly, the recent work by Barendregt who describes a cube of typed λ-calculi which gives a clear picture of several related systems [Bar89a] and introduces generalized type systems (a notion due to Berardi and Terlouw) which may be viewed as logical systems [Bar89b].

Discussed below are several type theories which are well-known by now and based on which our extended calculus of constructions is developed.

Martin-Löf's type theory

In 1971, Martin-Löf formulated the first version of his type theory [ML71], aiming at a type system comparable with ZF set theory and formalizing category theory (*e.g.*, the category of all categories). A basic axiom of this system is that there is a type of all types.[2] However, this axiom is too strongly impredicative to be logically consistent; this first version of Martin-Löf's type theory is logically inconsistent in the sense that every type is non-empty — shown by Girard and known by now as Girard's paradox. The introduction of a type of all types was based on the following ideas. First, according to Russell's doctrine of types

[2]Burstall and Lampson proposed such an idea of a type of all types in designing the programming language Pebble [BLam84] with the motivation for modular typeful programming. Also see [Card86] for a further development of this idea.

[Rus03], the range of significance of a propositional function forms a type, *i.e.*, the class of propositions forms a type; secondly, quantification over propositions and predicates is allowed; finally, propositions and types are *identified*, *i.e.*, every proposition is a type and *vice versa*.[3] Suppose U is the type of all propositions, then U is also the type of all types by the identification. Therefore, type U of all types naturally occurs and, in particular, it is the type of itself. The discovery of the logical incoherence of the idea of a type of all types led Martin-Löf to completely dispense with the impredicativity occurring in simple type theory (*c.f.*, [Chu40]) and turn to predicative type systems.

In Martin-Löf's (predicative) type theory [ML73,84], propositions and types are still identified, but the type of all types is replaced by an infinite sequence of type universes $U_0 : U_1 : U_2 : ...$. Hence, one can not quantify over all propositions or predicates, although quantification over each universe is allowed. Reflection principle is used to make sure that the introduction of universes leads to stronger power of the theory, for example, to define transfinite types. Besides dependent product types (Π-types) and the basic types like finite types and the type of natural numbers, dependent strong sum (Σ-types) is introduced as a basic type constructor playing the roles of (strong) existential quantifier [How69] and expressing the intuitionistic notion of subsets [Bis67][Kre68]. A new equality type constructor is also introduced. There are basically two versions of Martin-Löf's predicative type theory, one with weak (intensional) equality types [ML73] and the other with strong (extensional) equality types [ML84]. The weak equality type reflects the definitional equality, while the strong equality is equivalent the judgemental equality. The system with weak equality types is decidable, but that with strong equality types is not.

Martin-Löf's type theory is one of the main attempts to formulate correct formal systems for formalizing constructive mathematics as described by Bishop [Bis67]. Because of the close relationship between type theories and (typed functional) programming languages, Martin-Löf's type theory can also be viewed as a programming language [ML82].

Girard-Reynolds' polymorphic λ-calculus

The higher-order polymorphic λ-calculus F^ω was introduced by Girard [Gir71,72, 86], and independently by Reynolds [Rey74].[4] The important idea is that of polymorphic types which allows quantification and abstraction over types. For example, for a type variable t ranging over all types, $\forall t.t \to t$ is also a type; it is the type of the polymorphic function $\Lambda t.\lambda x^t.x$. Such a type formation is impredicative or circular since $\forall t.t \to t$ is formed by quantifying over all types including itself and a polymorphic function can be applied to any type which

[3] This identification of types with propositions is a distinguishing feature of Martin-Löf's type theories, in his predicative theories [ML73,84] as well as the first impredicative version. However, the author thinks that propositions can be viewed as types, but *not* necessarily vice versa.

[4] Reynolds formulated the second-order λ-calculus as a programming language.

may be the type of the function as well. Girard showed that the (higher-order) polymorphic λ-calculus is strongly normalizing (hence logically consistent) using the reducibility method and extended the Gödel interpretation to higher-order arithmetic [Gir72]. F^ω can be viewed as a logical system of natural deduction for intuitionistic higher-order propositional logic [Gir71].

Impredicativity also allows one to represent many useful data types by a coding technique [BB85]. For example, the type of natural numbers can be represented as $\forall t.t \to (t \to t) \to t$ which has the Church numerals as its normal objects. Proof-theoretic results show that the number-theoretic functions representable in F^ω are exactly those provably total in higher-order arithmetic [Gir73]. Furthermore, the existential quantifier can also be defined [GP85][Rey83], which can be used to express abstract data types [GP85].

Semantically, types in polymorphic λ-calculus can not be understood as arbitrary sets in the usual sense. As shown by Reynolds [Rey84][RP88], the ordinary set-theoretic model for simply typed λ-calculus can not be extended to the polymorphic λ-calculus. However, it was shown by many people that polymorphism can be understood intuitionistically (*e.g.*,[Tro73b][Gir72][Mog85][LM88][Pit87]).

As shown by Girard [Gir72], an attempt to extend the impredicative polymorphism to the level of connectives would result in a logically inconsistent system. It was from this that Girard realized the inconsistency of the first version of Martin-Löf's type theory [ML71] in which the above extension can be translated. In the other direction, one may consider weakening the impredicative polymorphism into stratified polymorphism, as considered by Leivant [Lei89].

Coquand-Huet's calculus of constructions

The calculus of constructions (CC) was introduced by Coquand and Huet [CH88] [Coq85], based on ideas from Martin-Löf's type theory, Girard's higher-order polymorphic λ-calculus and de Bruijn's Automath [dB80]. Like Martin-Löf's type theory, it uses judgements with contexts and has dependent product as the basic type constructor. Like F^ω, it is impredicative as one can quantify over all propositions (of type $Prop$, the type of the propositions which is the only constant type) to form propositions. (F^ω is a subsystem of CC.) The basic idea is to relax Martin-Löf's requirement of identification of propositions with types so that the type $Prop$ of the propositions to be a large type instead of a proposition (propositions are considered as 'small types'). However, the product of a family of propositions indexed by any type is still a proposition (impredicativity).

CC is a higher-order functional system proposed for intuitionistic higher-order logic. Coquand [Coq86b][Coq85] studied its meta-theory and proved that it is strongly normalizing (and hence logically consistent and decidable) by extending Girard-Tait's reducibility method [Gir72][Tai75].[5] Combining impredicativity with dependent product types provides a rather strong power for formalization of mathematics. For example, Leibniz's equality between two objects of the same

[5] Another attempt to prove the strong normalization of CC is by Pottinger [Pot87].

type can be defined [CH85].

Semantically, as is the case with polymorphic λ-calculus, propositions can not be understood as arbitrary sets. The intuitionistic set-theoretic model for polymorphic λ-calculus can be extended to CC; this was considered by many people including [HPit87][Ehr88][Luo88a] who extended the model of partial equivalence relations for second-order λ-calculus to CC, and [Str88] who considered model construction for CC in Cartmell's framework of contextual categories [Car78,86].

CC is a very strong functional system. As Girard pointed out, any further attempt to extend the calculus must be very cautious [Gir86]. Adding another impredicative level to the calculus would result in a logically inconsistent system in which Girard's paradox can be deduced [Coq86a]. Similarly, adding (type-indexed) strong dependent sum as proposition constructor would also lead to logical inconsistency [Coq86a][HH86][MH88] (see section 2.2.4 for a further discussion).

1.1.2 Applications in computer science

Type theories have been related to many areas in computer science, especially in programming methodology and proof development systems. In researches in programming methodology, which is closely related to but has different emphasis from software engineering, computer scientists try to look for solid theoretical foundations on the basis of which they may develop a science of programming or program development. Proof development systems or proof engineering (*c.f.*, [Bur86]), besides being an important research area of its own interest, attracts more and more interest in computer science now that the need to verify various proof obligations in formal or rigorous program development has been recognized. Research on type theories has interesting applications in these aspects and provide good theoretical foundations for proof development systems, formal specifications and correct development of programs.

Proof development systems

Curry-Howard correspondence is the basis for type theories to be used in proof development systems. Under this paradigm, a type theory is viewed as a logical system to formalize mathematical problems; to prove a theorem expressed as a type is to find an object of that type, and proof-checking is just type-checking. We discuss several systems below.[6]

Automath, which was led by de Bruijn, is the earliest project of using type systems as the basis to 'check mathematics' on computers (see [dB80] for a survey of the project). Various typed λ-calculi have been proposed and used as basic languages of Automath to do proof checking, most of which correspond to first-

[6]We only discuss some proof development systems closed related to type theories and miss many other systems like Edinburgh LCF [GMW79], Cambridge LCF [Pau87], Isabelle [Pau88] and λProlog [MN87].

order languages (see [vD80] for a study of the language theory). In Automath project, de Bruijn developed a notational system to represent bound variables by their reference depths, known as de Bruijn notation [dB72,78], which has become a basic technique in implementations of proof checkers. A considerable amount of proof-checking has been done in Automath; for example, Jutting [Jut77] translated and checked Landau's book of analysis.

Based on Constable's idea of constructive mathematics as a programming language [Con71] and later on Martin-Löf's type theory, the Cornell group developed the proof development system NuPRL [Con86]. NuPRL's type theory is based on Martin-Löf's type theory with strong equality [ML84] and has quotient types, subset types, inductive types and partial function spaces. (For details, see [Con86].) The type system is not decidable; not every well-typed term has a normal form. NuPRL is a refinement proof development system with a sophisticated environment including a window system; users develop proofs by backward reasoning and using tactics.

An early proof checker for the calculus of constructions [CH88] was implemented by Huet at INRIA [CH85] and a new implementation and further development is now in progress [Hue89]. In [CH85], many examples are given to show how the calculus can be used to formalize mathematical problems in the early implementation. In Edinburgh, Pollack has developed a refinement proof development system LEGO [Pol89][LPT89] based on Huet's early implementation. LEGO implements various related type systems, including Edinburgh LF [HHP87] (see below), the calculus of constructions [CH88] and the Extended Calculus of Constructions [Luo89a,b] we are about to describe. (See [Bur89b] for a simple introduction to proof checking in LEGO.)

The Edinburgh Logical Framework (LF) [HHP87] was developed in Edinburgh as a system of dependent types for defining and implementing different logics. The general framework allows one to describe various logical systems so that their implementations can be done 'once for all' by the common proof checker for LF and makes it possible to study problems like proof-searching in a general way (*c.f.*, [Plo87]). Many logical systems have been described in LF [AHH87]. An early implementation was done by Griffin [Gri87] and the system LEGO implements a version of LF as well. LF type theory is predicative and corresponds to intuitionistic first-order logic [Ber89a][BGeu89]. As a formal system, it can be viewed as a subsystem of the calculus of constructions.

Programming methodology

λ-calculus has been used as a theoretical model of functional programming (*c.f.*, functional programming languages Lisp [McC62], Hope [BMS81], ML [Mil84], Pebble [BLam84][LB88] and Quest [Card89]). Playing a role of partial specification, types have been recognized as an important organizing mechanism for reliable program development and ordered evolution of software systems. Taking advantage of typing facilities in programming languages and using sophisticated type systems, this has even led to a distinguishable programming style called *typeful programming* [Card89]. One can find more discussions on this aspect in [CW85][Card89].

Besides this close relationship of type theories with programming languages, Curry-Howard correspondence also allows people to use type theories in program specification, program verification and program derivation. This is because the principle of propositions-as-types can also be explained as 'programs as objects' (or programs as proofs); in other words, a program which computes a value of a type corresponds to a construction of the type. For example, in Martin-Löf's type theory, a judgement of the form $a{:}A$ can be read in the following ways [ML82,84]:

- a is an object of type A;
- a is a proof of proposition A;
- a is a program (or implementation) satisfying specification A;
- a is a solution to problem A.

Therefore, based on Curry-Howard correspondence, suitable type theories may be viewed as programming logics in which one can describe and reason about program specification and program development.

Program specifications have been studied quite extensively in computer science (for example, algebraic specification languages like Clear [BGog80], Act1 [EFH83], OBJ [FGJM85] and ASL [ST88]). In type theory, a specification may be formulated as a type. In particular, Σ-types provide a nice mechanism for describing program specifications. For example, a specification of sorting programs for lists of natural numbers can be expressed as follows:

$$Sorting =_{\mathrm{df}} \Pi l{:}List(N).\ \Sigma l'{:}List(N).\ sorted(l, l')$$

where $sorted(l, l')$ is the proposition expressing that l' is the sorted list of l. An object of type $Sorting$ is a function $sorting$ which, when applied to a list l of natural numbers, returns a value which is a pair (l', p) such that l' is the sorted list of l and p is a proof of this.

Notice that, based on the above formulation of specifications, looking for a correct implementation of a specification like $Sorting$ is just to find an object (proof) of the type (proposition) $Sorting$. Program development corresponds to proof development in type theory. Such a programming methodology is seriously taken by the Göteberg group based Martin-Löf's type theory [NPS89]. Particularly, they investigated how to extract programs from proofs, called *program extraction*.[7] In

[7]Deductive approaches to program extraction or program synthesis have been considered by many others in different settings. Manna and Waldinger [MW71] considered an approach in classical logic. Some people use realizability method to consider program extraction in constructive settings; *e.g.*, Hayashi's program extraction system PX [HN88] and Mohring's work [Moh89] in the calculus of constructions.

order to delete redundant proof information from the extracted programs, they introduced subset types to Martin-Löf's type theory [NP83][SS88].

Programs in Martin-Löf's type theory are (primitive) recursive functions, which are just like ordinary functional programs. However, as mentioned above, in impredicative systems like F^ω or CC, data types may be defined and functions can be represented by coding. Based on this, an impredicative type system may provide a way of 'non-recursive programming'. Mohring [Moh89] studied how to extract F^ω programs from proofs in the calculus of constructions. However, it seems that the coding representation of data types and functions has not been well-understood yet and whether it supports a nice programming style is still a problem. How to do specifications in the calculus of constructions [CH88] does not seem to have been paid enough attention. This is partly because of the fact that (type indexed) Σ-types are inconsistent with impredicativity, which prevented people from using Σ-types to describe specifications. In the extended calculus of constructions we are going to describe, we show how Σ-types can be used to describe program specifications.

One may rewrite the specification of sorting programs in another way:

$$Sorting =_{\mathrm{df}} \Sigma f{:}List(N) \to List(N).\ \Pi l{:}List(N).\ sorted(l, f(l))$$

Then, an implementation of this specification is a pair consisting of a sorting program and a proof of the correctness of the program. This gives a view of program verification or correct program development. Burstall calls such a pair a '*deliverable*' and develops an approach to program development based on this and using the extended calculus of constructions [Bur89a].

Abstraction and modularization

The issue of abstraction and modularization has been one of the central concerns in the design of programming and specification languages. It is also important in proof engineering (proof development). Its importance becomes apparent when people start to do real program (software) development or to use computers to develop large proofs (say, program verifications); *i.e.*, programming or proving in the large. To meet such a challenge, one needs to express abstract structures and modularize program/proof development so that large programming/theorem-proving tasks can be conquered in a structured way. As in programming, types (and rich type structures) can also provide useful mechanisms for modular development of proofs and specifications. (See chapter 8 for more discussions.)

Two important aspects of modular mechanisms are information hiding and sharing between module structures. Abstract data types are considered in a type-theoretic setting (second-order λ-calculus) by Mitchell and Plotkin [MP85], where they explain how existential types can be used to express information hiding. Three existing ways to handle the problem of structure sharing are simply explained in [Bur84] which are: sharing by equation in Standard ML [HMM86][Mac86], sharing by parameterization in Pebble [BLam84][LB88] and sharing by history in Clear [BGog80]. In particular, rich type structures play an essential role in the second style as we shall explain in section 8.3.

In interactive theorem-proving systems, a notion of *theory* is needed to structure proof development. Burstall et al. considered how a notion of theory can be used to structure development of specifications in the design of specification language Clear [BGog80] and Sannella and Burstall considered how such an idea can be applied to theorem prover LCF [SB83]. Based on the extended calculus of constructions, we shall develop a notion of theory and an approach to abstract structured reasoning.

1.2 Motivations and Overview of the Thesis

Our basic motivations may be summarized as looking for a (logically) strong and (structurally) expressive formal system which provides

- strong reasoning power as a logical system,
- basic mechanisms for adequate formalization of mathematics and program specifications, and
- structural mechanisms for modular development of proofs and programs.

The basic approach we have taken is to develop such a formal system as a type theory which encorporates strong logical power (by the Curry-Howard correspondence) and rich type structures as structural mechanisms.

We present and study a type theory **ECC** — Extended Calculus of Constructions. **ECC** is developed based on Coquand-Huet's calculus of constructions [CH88][Coq85] and the ideas of type universes and Σ-types in Martin-Löf's type theory [ML73,84]. It extends the calculus of constructions by Σ-types and fully cumulative (predicative) type universes and may also be considered as an impredicative extension of Martin-Löf's type theory with universes by adding a new (impredicative) universe $Prop$ of propositions.

We have thus integrated the (logical) power of impredicativity with the (structural) power of predicative universes and Σ-types into a unifying theory of dependent types. The known difficulty that the introduction of Σ-types together with impredicativity leads to logical paradox [Coq86a][HH86][MH88] is solved by adding Σ-types as non-propositional types in the predicative universes of the calculus and *lifting propositions as higher-level types*. Type inclusions between universes are coherently generalized to the other types by introducing a syntactic cumulativity relation over terms so that a nice unicity of typing is obtained based on a simple notion of principal type.

This development results in a stronger and more expressive higher-order calculus which has an embedded intuitionistic higher-order logic and provides rich type structures for formalization of mathematics, abstract structured reasoning and program specification. Particularly, Σ-types in **ECC**, together with universes, provide a useful 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 development of proofs and programs. In another aspect, formalization of mathematics can be done based on a strong higher-order logic and the type universes make it possible to formalize abstract mathematics (*e.g.*, abstract algebras and notions in category theory). Furthermore, the strong and flexible polymorphism in the calculus provides a higher-order module mechanism which can describe parameterized modules and support structure sharing by parameterization in the style of Pebble.

ECC has good meta-theoretic properties. For example, we have

- Church-Rosser property for the basic untyped term calculus;
- Type-preserving substitution (or the *Cut* operation) is admissible;
- Subject reduction theorem (closedness of typing over reduction);
- Weakening and strengthening lemmas;
- The existence of principal (or the most general) types.

The main proof-theoretic result about the calculus is:

- **Strong Normalization**: Every well-typed term is strongly normalizable.

i.e., every reduction sequence starting from a well-typed term is terminating. This result shows the proof-theoretic consistency of the calculus. Its proof uses Girard-Tait's reducibility method [Gir72][Tai75] and is based on the proofs of strong normalization for the calculus of constructions [CH88] 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 predicative universes. We do this by proving a *quasi-normalization* theorem which enables us to define a two-dimensional ranking measure of types.

Several important corollaries of the strong normalization theorem are:

- Logical consistency of the embedded logic;
- Decidability of conversion and the cumulativity relation for well-typed terms;
- Decidability of type inference and type checking;
- Equality reflection, *i.e.*, the propositional Leibniz's equality definable in the calculus reflects the definitional (or computational) equality (conversion).

These results establish the theoretical foundations for both pragmatic applications and machine implementations. The logical consistency is the most basic requirement for the system to be used for formalization of mathematics, theorem-proving and program specification. The decidability results and type-inference algorithm can be directly applied to a computer implementation of proof development systems based on the calculus and the meta theorems are useful for a good implementation.[8]

Based on the rich type structures (Σ-types and universes, in particular) of **ECC**, we discuss a notion of (abstract) theory for abstract structured reasoning. Such a theory mechanism allows a good modularization of proof development and makes it possible to build up useful theory bases for large theorem-proving. Program specifications can be expressed by Σ-types in a similar style to using Martin-Löf's type theory as we discussed in section 1.1.2, but propositions in the embedded higher-order logic are used to express program properties. In particular, Leibniz's equality can be used to model the definitional (or computational) equality (*c.f.*, [Bur89a]); the theoretical soundness of this modeling is justified by the equality reflection result mentioned above.

We also discuss the model-theoretic aspect of the calculus. We explain how the calculus can be understood set-theoretically in the ω–**Set** framework developed by Moggi and Hyland [Mog85][Hyl82,87]. In particular, propositions in the impredicative universe are interpreted as partial equivalence relations and the predicative type universes can be interpreted as corresponding to large set universes.

In chapter 2, the calculus **ECC** is formally presented and informal explanations of the primitive notions in the theory are given together with some remarks on design decisions.

Chapter 3 studies the basic meta-theoretic properties of the calculus, including those about conversion and the cumulativity relation, properties of derivable judgements, admissibility results like subject reduction, weakening and strengthening, and the typing properties like those about principal types.

Chapter 4 proves the quasi normalization theorem and defines a two-dimensional ranking measure of types, which make explicit the predicativity of the non-propositional types and establish a necessary result for us to prove strong normalization.

Chapter 5 proves the strong normalization theorem, using Girard-Tait's reducibility method based on a slightly more general definition of saturated sets and the quasi-normalization result.

Chapter 6 considers two important corollaries of the normalization property — logical consistency and decidability. In section 6.1, the embedded higher-order logic is described and proved to be consistent. We also conjecture that it is a conservative extension of the intensional intuitionistic higher-order logic and give a discussion. Decidability results are discussed in section 6.2; in particular, we describe an algorithm for type inference (and type checking) and prove its correctness.

Chapter 7 discusses a set-theoretic interpretation of the calculus in the ω–**Set** framework. We explain how the main constructs can be understood set-

[8] **ECC** has been implemented in the proof development system LEGO developed by Pollack [Pol89][LPT89].

theoretically.

Chapter 8 describes an approach to abstract structured reasoning, based on a notion of abstract theory. We show how abstract reasoning can be done by proof instantiation and structured reasoning by proof inheritance and parameterized sharing.

Chapter 9 considers some issues in program specification and programming. We show that Leibniz's equality reflects the conversion relation and hence can be used to model definitional equality in specifications. We also briefly discuss how the predicative part of **ECC** may provide programming facilities by stratified polymorphism and rich type structures (*e.g.*, abstract data types).

In chapter 10, we discuss further research topics and directions, including some open problems.

1.3 Related Work

We briefly discuss some related work below, some of which has been mentioned in section 1.1.

The calculus of constructions (CC) was first studied by Coquand in his thesis [Coq85] and also in [CH88][CH85] *etc.*. Its meta theory, proof of normalization theorem in particular, can be found in [Coq85][Coq86b] and [Pot87]. Girard-Tait's reducibility method [Gir72][Tai75] is the general method used to prove normalization of CC. Our proof of strong normalization for **ECC** is also based on the method.

Type universes were first introduced in Martin-Löf's type theory [ML73,84] and also appeared in NuPRL's type theory [Con86]. The idea of extending CC by universes appeared in [Coq86a], where the Generalized Calculus of Constructions (GCC) was presented.[9] The strong normalization theorem for Constructions with infinite type universes was first proved in [Luo88b] (and this thesis, also see [Luo89a,b]). Based on the results in [Luo88b], the type-checking problem for GCC was considered by Harper and Pollack [HPol89]; because GCC does not have the property of type unicity (see section 2.2.5), the resulted algorithm is rather complicated compared with that for **ECC** (as sketched in [Luo89a] and described in this thesis).

Σ-types were considered by Howard [How69] and become well-known by Martin-Löf's work [ML73,84]. A similar idea of using Σ-types to express modular structures occurs in researches of programming languages (*e.g.*, [BLam84] and [Mac86]). The idea of lifting propositions (in the impredicative universes of Constructions) as higher-level types, in order to use Σ-types to express abstract structures and mathematical theories, was investigated in [Luo88a][Luo89a,b]. Recently, Coquand [Coq89] and Streicher [Str88] considered using an explicit lifting operator to lift propositions, and view the calculus with type inclusions as

[9] In the presentation of GCC in [Coq86a] (page 235), the rules stating $Type_j : Type_{j+1}$ were inadvertently missing.

an abbreviation [Coq89].

There are several existing works 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 a semantics of CC based on the notion of contextual category [Cart78,86]. After Moggi [Mog85] found out that there is a small internal complete category in the category of ω-sets [Hyl82], with a notion of completeness which is enough to interpret Girard-Reynolds' polymorphic λ-calculus, Hyland [Hyl87] defined a stronger notion of completeness which can be used to model the calculus of constructions. Ehrhard [Ehr88] sketched an ω–**Set** model of CC. An ω–**Set** model of CC (with Σ-types and lifting of propositions as types) was described in [Luo88a]. In [Luo89a] (and this thesis), the model in [Luo88a] is also extended to the type universes (using an idea of Hayashi).

Burstall [Bur89a] developed an approach to program development, using the extended calculus of constructions and the system LEGO. He uses Σ-types to express specifications in a similar style as [NPS89], but uses Leibniz's equality to model computational equality and gives a categorical explanation of the approach. The formulation and proof of the equality reflection result in this thesis was motivated by Burstall's work on specifications.

The proof development system LEGO is developed by Pollack in Edinburgh [Pol89][LPT89]. It implements several related type theories, including **ECC** presented in this thesis. One of the interesting features of the system is that it supports 'universe polymorphism' [Hue87][HPol89] so that indices of universes may be omitted in practice.

Chapter 2

ECC: an Extended Calculus of Constructions

In this chapter, the calculus **ECC** is formally described, followed by some informal explanations and remarks on design decisions.

2.1 A Formal Presentation of ECC

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

2.1.1 The term calculus

The basic expressions of the calculus are called *terms*, given by the following definition.

Definition 2.1.1 (terms) 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:*

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

We use $\mathcal{T}$ to denote the set of terms. □

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 Π, Σ and λ, respectively. The usual conventions of parenthesis omitting are adopted; for example, $M_1M_2...M_n$ stands for $(...((M_1M_2)M_3)...M_{n-1})M_n$[1] and the scopes of the binding operators Π, Σ and λ extend to the right as far as possible. For a term M,

[1] We also sometimes write $M_1M_2...M_n$ as $M_1(M_2,...,M_n)$ for readability consideration.

$FV(M)$ 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.

α-convertible terms (*i.e.*, terms which are the same up to changes of bound variables) are identified. $\equiv$ is used for the syntactical identity between expressions such as terms, *i.e.*, $A \equiv B$ means that A and B are the same up to α-conversion.

Definition 2.1.2 (reduction and conversion) Reduction *($\rhd$) and* conversion *($\simeq$) are defined as usual with respect to the following contraction schemes:*

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

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

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. More precisely,

1. *The terms of the forms $(\lambda x{:}A.M)N$ and $\pi_i(\mathbf{pair}_A(M_1, M_2))$ $(i = 1, 2)$ are called* β-redexes *and* σ-redexes*, with $[N/x]M$ and M_i being their* contractums*, respectively, and $\lambda x{:}A.M$ and $\mathbf{pair}_A(M_1, M_2)$ are called* major terms *of these redexes.*

2. *If a term P contains an occurrence of a redex R and we replace that occurrence by its contractum, and the resulting term is P', we say P* one-step reduces to *P' (notation $P \rhd_1 P'$).*

3. *We say P* reduces to *Q (notation $P \rhd Q$) if and only if Q is obtained from P by a finite (possibly empty) series of contractions.*

4. *We say P* is convertible to *Q (notation $P \simeq Q$) if and only if Q is obtained from P by a finite (possibly empty) series of contractions and reversed contractions, i.e., there exist $M_1, ..., M_n$ $(n \geq 1)$ such that $P \equiv M_1$, $Q \equiv M_n$ and $M_i \rhd_1 M_{i+1}$ or $M_{i+1} \rhd_1 M_i$ for $i = 1, ..., n-1$.*

A term is in normal form *if and only if it does not contain any redex. A term M_1 is* strongly normalizable *if and only if every reduction sequence of the form $M_1 \rhd_1 M_2 \rhd_1 M_3 \rhd_1 ...$ is finite.* □

The kinds, also called *type universes*, and the type inclusions between them induce the type cumulativity that is syntactically characterized by the following relation.

Definition 2.1.3 (cumulativity relation) *The* cumulativity relation *$\preceq$ is defined to be the smallest binary relation over terms such that*

1. *$\preceq$ is a partial order with respect to conversion, that is,*
   (a) *if $A \simeq B$, then $A \preceq B$;*
   (b) *if $A \preceq B$ and $B \preceq A$, then $A \simeq B$; and*

(c) *if* $A \preceq B$ *and* $B \preceq C$, *then* $A \preceq C$.

2. $Prop \preceq Type_0 \preceq Type_1 \preceq ...$;

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* $\Sigma x{:}A_1.A_2 \preceq \Sigma x{:}B_1.B_2$.

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

Remark The well-definedness (*i.e.*, the existence) of the cumulativity relation will be justified in section 3.1 by giving an alternative inductive definition. □

2.1.2 Judgements and inference rules

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

Definition 2.1.4 (contexts) 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 $\Gamma \equiv x_1{:}A_1, ..., x_n{:}A_n$, $FV(\Gamma)$, *is defined as* $\bigcup_{1\le i\le n}(\{x_i\} \cup FV(A_i))$. □

Definition 2.1.5 (judgements) Judgements *are of the form*

$$\Gamma \vdash M : A$$

where Γ *is a context and* M *and* A *are terms. We shall write* $\vdash M : A$ *for* $\langle\rangle \vdash M : A$. □

The *inference rules* of **ECC** are listed as follows, where j stands for an arbitrary natural number:

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

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

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

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

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

$$(\Pi 2) \qquad \frac{\Gamma \vdash A : Type_j \quad \Gamma, x{:}A \vdash B : Type_j}{\Gamma \vdash \Pi x{:}A.B : Type_j}$$

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

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

$$(\Sigma) \qquad \frac{\Gamma \vdash A : Type_j \quad \Gamma, x{:}A \vdash B : Type_j}{\Gamma \vdash \Sigma x{:}A.B : Type_j}$$

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

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

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

$$(\preceq) \qquad \frac{\Gamma \vdash M : A \quad \Gamma \vdash A' : Type_j}{\Gamma \vdash M : A'} \quad (A \preceq A')$$

Definition 2.1.6 (derivations) *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', and $\Gamma \not\vdash M : A$ for '$\Gamma \vdash M : A$ is not derivable'.* □

Definition 2.1.7 (valid contexts) *A context Γ is* valid *if and only if $\Gamma \vdash Prop : Type_0$. We also often abbreviate $\Gamma \vdash Prop : Type_0$ as 'Γ is valid'.* □

Definition 2.1.8 *Let Γ be a 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 *(or* well-typed type under Γ*) 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* non-propositional Γ-type *(or* proper Γ-type*) otherwise.*

- *A term M is called a* Γ-proof *if $\Gamma \vdash M : P$ for some Γ-proposition P.*

- *A term A is* inhabited *(under Γ) if $\Gamma \vdash M : A$ for some M.* □

This completes our formal presentation of the calculus.

2.2 Informal Explanations

The extended calculus of constructions **ECC** presented above may be seen as a combination of Coquand-Huet's calculus of constructions [CH88] and Martin-Löf's type theory with universes [ML73,84]. It extends the calculus of constructions with Σ-types and fully cumulative type universes. One may also consider it as an impredicative extension of Martin-Löf's type theory with universes by adding a new (and the lowest) *impredicative* universe *Prop* of propositions. It turns out that such an integration results in a stronger and more expressive higher-order calculus for formalization of mathematics, abstract structured reasoning and program specification.

We now informally explain the primitive notions of the calculus and give some remarks on design decisions.

2.2.1 Judgements and validity of contexts

A context $\Gamma \equiv x_1{:}A_1, ..., x_n{:}A_n$ is informally viewed as a list of assumptions that x_i is an object of type A_i.

The intuitive meaning of a judgement $\Gamma \vdash M : A$ is that *M has type A in context* Γ, *i.e.*, under the assumptions Γ, M is an object of type A.

The only axiom of the system is $\vdash Prop : Type_0$. Besides asserting that *Prop* has type $Type_0$ in Γ, the judgement $\Gamma \vdash Prop : Type_0$ also plays the role in the calculus of asserting that Γ is a valid context. The validity of contexts are proved by the rules (Ax) and (C). We may replace the rules $(Ax)(C)(T)(var)$ by the following, with an additional judgement form 'Γ valid':

$$\frac{}{\langle\rangle\ valid}$$

$$\frac{\Gamma \vdash A : Type_j}{\Gamma, x{:}A\ valid} \quad (x \notin FV(\Gamma))$$

$$\frac{\Gamma\ valid}{\Gamma \vdash Prop : Type_0}$$

$$\frac{\Gamma\ valid}{\Gamma \vdash Type_j : Type_{j+1}}$$

$$\frac{\Gamma, x{:}A, \Gamma'\ valid}{\Gamma, x{:}A, \Gamma' \vdash x : A}$$

Then we gain an equivalent system as presented in [Luo89a]. These rules may give a clearer picture of context validity in the calculus.

As we shall show in section 3.2, for any derivable judgement $\Gamma \vdash M : A$, every prefix subsequence of Γ is a valid context and A is a Γ-type. If $x_1{:}A_1, ..., x_n{:}A_n$ is a valid context, then $x_1, ..., x_n$ are distinct and A_i is a type only dependent on variables $x_1, ..., x_{i-1}$.

2.2.2 Propositions and the impredicative universe $Prop$

Inheriting the impredicative type structure from the calculus of constructions, **ECC** has an embedded intuitionistic higher-order logic. Provability of a formula corresponds to the inhabitation of the corresponding proposition. The propositions (more precisely, Γ-propositions) play the role of logical formulas by the Curry-Howard principle of propositions-as-types [CF58][How69]. For example, logical implication between two propositions P_1 and P_2 is expressed by $P_1 \rightarrow P_2$ and, if P is a predicate over type A (*i.e.*, P is a propositional function), then the formula for universal quantification stating that 'for all x in A, $P(x)$' is expressed by proposition $\Pi x{:}A.P(x)$ formed by product operator Π. The other ordinary logical connectives and existential quantifier can be defined by (impredicative) coding of their elimination rules as in higher-order logic (*c.f.*, [Pra65][CH85]). (See section 6.1 for details.)

The universe $Prop$ of propositions is impredicative. By rule (Π1), $Prop$ is closed under arbitrary dependent products. In other words, for arbitrary type A and any propositional function P over A, $\Pi x{:}A.P(x)$ is a proposition. For example, we can derive

$$\vdash \Pi x{:}Prop.x : Prop$$

This proposition $\Pi x{:}Prop.x$ stands for logical constant **false** as it implies every proposition (see section 6.1.1) and is not inhabited in the empty context (theorem 6.1.5). The circularity in such a type formation is clear: $\Pi x{:}Prop.x$ is formed by quantifying over the type $Prop$ which has $\Pi x{:}Prop.x$ as its object. Because of such an impredicative polymorphism, as in polymorphic λ-calculus [Gir72][Rey74] and the calculus of constructions [CH88], propositions in the calculus can not be understood as arbitrary sets (see section 7.5 for a discussion).

Note that, unlike Martin-Löf's type theory, we do not identify types with propositions. Propositions are types, but *not* vice versa. There are non-propositional types like $Prop$, $Prop \rightarrow Prop$ and Σ-types which are not regarded as representing logical formulas in the system. This provides a conceptual distinction between logical formulas (propositions) and data types (non-propositional types). Philosophically, it does not seem to be natural to identify data types with logical formulas, although it is possible.

2.2.3 Non-propositional types and predicative universes $Type_j$

Besides the impredicative universe $Prop$, there are infinite predicative universes $Type_0$, $Type_1$, $Type_2$, ..., where, roughly speaking, the non-propositional types reside. The type universes in **ECC** provide us very rich type structures and make the system become stronger and more expressive for formalization of mathematics and structured abstract reasoning. Particularly, it makes it possible to formalize *abstract mathematics* (*e.g.*, abstract algebras and notions in category theory like the category of all small categories) — one of the two bases (the other is Σ-types) for *structured abstract reasoning*. The idea is that it is possible to represent arbitrary sets by non-propositional types in predicative universes. Furthermore, universes uniformly provide a strong form of polymorphism which enables us to do structured reasoning or programming. For example, universes allow us to express parameterized modules and hence *parameterized structure sharing* following the idea of Burstall and Lampson in the programming language Pebble [BLam84][Bur84]. All these will be further discussed in chapter 8.

Formally, the predicative universes in **ECC** is further developed from the formulations of universes of Martin-Löf [ML73,84] and Coquand [Coq86a]. $Prop$ is an object of type $Type_0$ (by rule (Ax)) and $Type_j$ is an object of $Type_{j+1}$ (by rule (T)). Viewing intuitively types as sets and ':' as the membership relation, we have

$$Prop \in Type_0 \in Type_1 \in Type_2 \in \ldots \tag{1}$$

With infinite universes, every object in the calculus has a type (types have universes as their types), as in Martin-Löf's type theory. Note that a universe is not an object of itself or any universe lower than it.

Furthermore, by rule $(\preceq)$, we can infer that every object of type $Prop$ is an object of type $Type_0$ and every object of type $Type_j$ is an object of type $Type_{j+1}$; *i.e.*, intuitively,

$$Prop \subseteq Type_0 \subseteq Type_1 \subseteq Type_2 \subseteq \ldots \tag{2}$$

Type inclusions between universes are uniformly extended to other types (see section 2.2.5 below) and the lifting of propositions to higher-level types ($Prop \subseteq Type_0$) is particularly important for Σ-types in the calculus to be used as an abstraction mechanism (see section 2.2.4 below). In general, one only works with finite many universes. With universe inclusions, one can work uniformly in a universe big enough without worrying about indices of universes.[2]

Remark There are basically two approaches to formulating type universes and the associated reflection principle, called by Martin-Löf as 'formulation à la Russell' and 'formulation à la Tarski' [ML84]. The former uses explicit universe inclusions following the style of Russell's ramified type theory and is adopted in our formulation of **ECC**. In the later approach, a new higher universe, say $Type_j$, is introduced as a type consisting of the *names* of the types residing in the universe; each of these types is introduced by a type constructor T_j as $T_j(a)$ which has a in $Type_j$ as its name. Following this view of distinction between types and their names, in the former approach using universe inclusions, a type symbol stands for both a type and the name of the type. □

The universes $Type_j$ are *predicatively* closed under formation of dependent products (Π-types) and dependent strong sums (Σ-types). By rules ($\Pi 2$) and

[2] In LEGO proof development system [Pol89][LPT89], such a 'typical ambiguity' is allowed [HP89].

(Σ), for any Γ-type A and any $(\Gamma, x{:}A)$-type B in the same universe $Type_j$, their dependent product type $\Pi x{:}A.B$ and dependent sum type $\Sigma x{:}A.B$ are of type $Type_j$. In fact, because of the type inclusions between universes ((2) above), rules ($\Pi 2$)(Σ) are the more economic expressions of the following (derivable) rules:[3]

$$\frac{\Gamma \vdash A : K \quad \Gamma, x{:}A \vdash B : K'}{\Gamma \vdash \Pi x{:}A.B : K_{max}}$$

$$\frac{\Gamma \vdash A : K \quad \Gamma, x{:}A \vdash B : K'}{\Gamma \vdash \Sigma x{:}A.B : K_{max}}$$

where K and K' are arbitrary kinds, and $K_{max} \equiv max_{\preceq}\{Type_0, K, K'\}$ is the maximum kind among $Type_0, K$ and K' subject to the cumulativity relation $\preceq$. In other words, the dependent product/sum of any two types which are in universes lower than or the same with $Type_j$ is an object of $Type_j$. For example, $\Pi x{:}Prop.Type_0$ is of type $Type_i$ for $i \geq 1$ but not of type $Prop$ or $Type_0$.

Therefore, the universes $Type_j$ are predicative in the sense that there is no circularity in formations of non-propositional types. This predicativity will be made formally explicit in chapter 4 and is essential for **ECC** to be logically consistent and not to suffer from logical paradox. For example, if $Type_0$ were closed for arbitrary dependent product types as $Prop$ does, Girard's paradox [Gir72][Coq86a] could be deduced.

The Π-type $\Pi x{:}A.B$ is the type of functions which take an object N of type A into an object of type $[N/x]B$. Functions are represented by λ-expressions (*c.f.*, rule (λ)) whose applications to objects are expressed by rule (*app*). When B is not dependent on the objects of A, *i.e.*, x does not occur free in B, $\Pi x{:}A.B$ (abbreviation $A \to B$) is the type of functions from A to B.

The Σ-type $\Sigma x{:}A.B$ is the type of pairs (a, b) where a is an object of type A and b is of type $[a/x]B$. Intuitively, it 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:

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

When B is not dependent on the objects of A, $\Sigma x{:}A.B$ (abbreviation $A \times B$) is the usual product type of pairs from A and B.

Formal objects for pairs in our calculus are 'heavily typed'. We use $\mathbf{pair}_A(M, N)$ instead of the usual untyped term (M, N) as in Martin-Löf's type theory. This avoids the undesirable type ambiguity which would make type inference and type checking difficult (perhaps impossible) [Luo88a]. For example, if untyped pairs were used, $(Type_0, Prop)$ would have both $\Sigma x{:}Type_1.x$ and $Type_1 \times Type_0$ as its types which are incompatible.

[3] A rule R of the form $\frac{J_1 \; \dots \; J_n}{J}$ is called *derivable* if there is a finite sequence of judgements $J_{n+1}, \dots, J_{n+m}$ with $J_{n+m} \equiv J$ such that, for all $n+1 \leq i \leq n+m$, J_i is either one of $J_1, \dots, J_n$ or the conclusion of some instance of an inference rule whose premises are in $\{\, J_j \mid 1 \leq j < i \,\}$.

2.2.4 Lifting of propositions and Σ-types as subsets

As explained by Martin-Löf, Σ-types in his type theory can be used to express the intuitionistic notion of subsets; *i.e.*, $\Sigma x{:}A.B(x)$ expresses the set of the objects a in A such that $B(a)$ holds. From intuitionistic point of view, to give an object of type A such that $B(a)$ is to give a together with a proof of $B(a)$ (*c.f.*, [Bis67][Kre68][ML73,84]).

Different from Martin-Löf's type theory, **ECC** has propositions as logical formulas and propositions are not identified with types. Therefore, such an expression of subsets is possible only if we can form Σ-type $\Sigma x{:}A.B$ when B is a proposition in context $\Gamma, x{:}A$.

There is a known problem for extending impredicative type theories by strong sum (Σ-types); that is, arbitrary strong sum is logically inconsistent with impredicativity [Coq86a][HH86][MH88]. Adding arbitrary type-indexed Σ-types to the impredicative level of the calculus of constructions would produce an inconsistent system in which Girard's paradox can be derived. In other words, the following inference rule is problematic and, together with the rules for two projections, inconsistent with the impredicativity of $Prop$:

$$(*) \qquad \frac{\Gamma, x{:}A \vdash B : Prop}{\Gamma \vdash \Sigma x{:}A.B : Prop}$$

where A is not restricted as a small type (proposition).[4] As propositions play a necessary and significant role in expressing mathematical problems and specifications, the above difficulty appears serious and seems to have prevented people from directly extending a Constructions-like calculus by Σ-types in order to have the power of expressing abstract structures.

However, the above result does *not* prevent us from adding Σ-types as large types (non-propositional types) as we do for **ECC**. The only problem is how to regard propositions also as large types to form Σ-types. In formulation of **ECC**, we propose an idea of *lifting propositions to higher-level types*.[5] Every object of type $Prop$ is also an object of type $Type_0$, *i.e.*, $Prop \subseteq Type_0$, as we described in section 2.2.3. This can be understood as lifting a proposition as the type of its proofs; or putting in another way, a logical formula is regarded as the name of the type of its proofs. This lifting of propositions is essential for Σ-types to express the intuitionistic notion of subset. Note that, in **ECC**, $\Sigma x{:}A.P$ is *not* a proposition even when P is; in other words, rule $(*)$ above is not included or

[4] A simple and intuitive argument to see this problem is that, if rule $(*)$ were allowed, then we would be able to derive $\vdash \Sigma x{:}Prop.\{*\} : Prop$, where $\{*\}$ stands for a non-empty type, say unit type. Then, we have $\Sigma x{:}Prop.\{*\}$ is 'isomorphic' to $Prop$, which shows that we would essentially have $Prop : Prop$. If we add a premise $\Gamma \vdash A : Prop$ to rule $(*)$, the rule would become of no problem; it is the rule for *small* Σ*-types*. We do not have small Σ-types in **ECC**, not because it can not be added, but because we do not see its necessity.

[5] This idea was considered in [Luo88c,a] in order to get a good formulation of an extension of the calculus of constructions by Σ-types. In the original presentations of Constructions [CH88][Coq85][Coq86a], propositions are not higher-level types.

admissible. However, as propositions are lifted as types, we can derive (by rules ($\preceq$) and (Σ))

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

This non-propositional type $\Sigma x{:}A.P$ intuitively represents the intuitionistic subset type. It is this that enables propositions to be used to express axioms of a mathematical theory and program properties in a specification when we use Σ-type to express abstract theories and specifications. (See chapter 8 and chapter 9.)

One might wonder whether the lifting of propositions would propagate the impredicativity at the level of propositions to the higher levels. For instance, we can derive

$$\vdash \Pi x{:}Type_j \Pi B{:}Type_j \to Prop.B(x) : Type_j$$

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 formations of non-propositional types are only dependent on the types with lower ranks (see chapter 4).

The intuitionistic expression of 'such that' is based on the idea of treating proofs as mathematical objects. Note that we can quantify over the proofs of a proposition to form propositions or types. This makes it possible, for example, to use propositions to express properties of proofs or programs (*c.f.*, [ML73]). A typical example is the Leibniz's equality definable in the calculus. (See definition 6.1.4 and section 9.1.)

Remark We can define (weak) existential types by dependent product types at the predicative universes of **ECC** [Luo89a] as well as at the impredicative universe [Rey83]. (See section 9.2.2.) They can be used to express abstract data types as discussed by Mitchell and Plotkin in [MP85]. However, they can not be used to express the intuitionistic notion of subset. □

2.2.5 Conversion and full cumulativity

We now informally explain the term calculus, mainly to explain the conversion relation $\simeq$ and the cumulativity relation $\preceq$, both of which are defined for the untyped terms. The conversion relation has the Church-Rosser property [ML72] (see theorem 3.1.1) and the cumulativity relation $\preceq$ is a partial order over terms with respect to conversion. At the untyped term level, neither of them is decidable. However, essentially, they are only used for well-typed terms in the calculus and in this case, they are decidable as we shall show (lemma 6.2.1).

Conversion between well-typed terms may be regarded as formally expressing definitional equality which is purely for abbreviation of linguistic expressions [ML73], and reduction may be regarded as evaluation of defined functions applied to their arguments. β-conversion corresponds to the following definitional schema of functional abstraction: if a term M is of type B assuming variable x is an arbitrary object of type A, we can define a function f of type $\Pi x{:}A.B$ by

$$f(x) =_{\rm df} M$$

f thus defined is formally expressed by $\lambda x{:}A.M$. Then a β-reduction step contracting $f(N)$ to $[N/x]M$ corresponds to an evaluation step of the function f applied to an object N of type A.

σ-reduction corresponds to extracting the components from a pair by evaluating the projection functions. σ-conversion can be explained by considering the following definitional schema: for a binary function M of type $\Pi x{:}A\Pi y{:}B(x).C(x,y)$, we can define a unary function f of type $\Pi z{:}(\Sigma x{:}A.B(x)).C(\pi_1(z),\pi_2(z))$ by

$$f(z) =_{\rm df} M(\pi_1(z),\pi_2(z))$$

Formally, $f =_{\rm df} \lambda z{:}(\Sigma x{:}A.B(x)).M(\pi_1(z),\pi_2(z))$. Then, by σ-conversion (together with β-conversion), we have

$$f(\mathbf{pair}_{\Sigma x:A.B}(M_1,M_2)) \simeq M(M_1,M_2)$$

The cumulativity relation subsumes conversion and reflects the type inclusions between universes. Splitting the cumulativity rule $\preceq$ to $\simeq$ and $\prec$, the cumulativity rule ($\preceq$) in fact stands for the following two rules:

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

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

As conversion reflects definitional equality, the rule ($conv$) of type conversion, as explained in [ML73], allows us to apply the *principle of replacing a type (proposition) by a definitionally equal type (proposition)*, *i.e.*, if M is an object (proof) of type (proposition) A and A is definitionally equal to type (proposition) A', then M is an object (proof) of A'.

Rule (cum) generalizes the universe inclusions coherently to the other types, achieving a nice unicity of types. It results in a simple notion of *principal type* (or the most general type, see definition 3.3.5 and theorem 3.3.6) and a simple algorithm for type inference (definition 6.2.2 and theorem 6.2.3). For example, the principal types of $M\equiv\lambda x{:}Type_1.x$ and $N\equiv\mathbf{pair}_{Type_1\times Type_0}(Type_0,Prop)$ are $Type_1\rightarrow Type_1$ and $Type_1\times Type_0$, respectively. By the cumulativity rule, we have $\vdash M : Type_1\rightarrow Type_i$ ($i\geq 1$) and $\vdash N : Type_j\times Type_k$ ($j\geq 1$ and $k\geq 0$).

This generalization clarifies the feature of type inclusions in a type system with universes and leads to a simple implementation of the type hierarchy of

ECC. In the formulations of universes by Martin-Löf [ML84] and Coquand [Coq86a], the following rules are used:

$$\frac{\Gamma \vdash A : Type_j}{\Gamma \vdash A : Type_{j+1}}$$

Although in such a formulation every well-typed term has a minimum type with respect to the cumulativity relation, as shown in [Luo86b], the minimum type is sometimes not the most general one (principal type). 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 cumulativity relation $\preceq$ defined in definition 2.1.3 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 $B_1 \preceq A_1$.[6] 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. The algorithm for type inference remains the same except that the basic relation $\preceq$ is changed. However, from a set-theoretic semantic point of view, the type inclusions with a cumulativity relation being completely contravariant would be reflected by coercions instead of by set inclusions if we think of functions as relations.

A final remark is about rule $(\preceq)$. In the rule, $A \preceq A'$ is a *side condition*. This means that we do *not* take its justifications as part of a derivation in **ECC**. The premise $\Gamma \vdash A' : Type_j$ is then important and necessary to guarantee that A' is a well-typed type. One may consider equality judgements as in Martin-Löf's type theory [ML73][7] and take justifications of the cumulativity relation as parts of derivations. This is possible because $\preceq$ is decidable (and axiomatizable) for well-typed terms.

[6] Here, one may compare with languages with subtyping. See [Card89], for example.

[7] Not in the sense of the judgemental equality in Martin-Löf's system with strong equality [ML84].

Chapter 3

Basic Meta-theoretic Properties

We study in this chapter the basic meta-theoretic properties of the calculus. The main properties of the underlying term calculus are concerned about conversion and reduction (Church-Rosser theorem) and the cumulativity relation. Properties about derivable judgements and some important admissibility results for derivability are proved in section 3.2. A notion of principal type which characterizes the type cumulativity in the calculus is studied in section 3.3.

3.1 Properties of the Term Calculus

The most important property of the term calculus is the Church-Rosser theorem about the relations of reduction and conversion.

Theorem 3.1.1 (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 Sketch By definition of conversion, we only have to show that reduction has the diamond property, *i.e.*, if $M \rhd M_1$ and $M \rhd M_2$, then $M_1 \rhd M'$ and $M_2 \rhd M'$ for some M'. Following [ML72], we give a proof sketch of the diamond property as follows.[1]

1. Definitions:

   (a) *parallel one-step reduction:* $M \rhd^1 N$ if and only if N is got by contracting some (possibly all or none) of the redexes in M, starting from within and proceeding outwards.

   (b) *parallel n-step reduction:* $M \rhd^0 N$ if and only if $M \equiv N$; $M \rhd^{n+1} N$ if and only if $M \rhd^n M' \rhd^1 N$ for some M'.

   Note that $M \rhd N$ if and only if $M \rhd^n N$ for some $n \in \omega$.

[1] Martin-Löf in [ML72] refers to Tait for the basic ideas of the proof. Similar proofs for simpler λ-calculi can be found in other places, *e.g.*, Appendix 1 of [HS87] among others.

2. Lemma: $M \rhd^1 M'$ implies $[N/x]M \rhd^1 [N/x]M'$. (Obvious by definition of $\rhd^1$.)

3. Lemma: If $M \rhd^1 M_1$ and $M \rhd^1 M_2$, then $M_1 \rhd^1 M'$ and $M_2 \rhd^1 M'$ for some M'. (By induction on the structure of M and using the lemma above.)

4. Lemma: If $M \rhd^m M_1$ and $M \rhd^n M_2$, then $M_1 \rhd^n M'$ and $M_2 \rhd^m M'$ for some M'. (By $m \times n$ times applications of the above lemma.)

From the last lemma above, the diamond property for reduction holds, and hence the theorem. □

Corollary 3.1.2 (uniqueness of normal forms) *The normal form of a term is unique (up to syntactical identity), if it exists.* □

Remark Note that **ECC** does *not* include the η-contraction scheme

$$(\eta) \qquad \lambda x{:}A.Mx \leadsto_\eta M \quad (x \notin FV(M))$$

or the contraction scheme of surjective pairing

$$(\pi) \qquad \mathbf{pair}_A(\pi_1(M), \pi_2(M)) \leadsto_\pi M$$

either of which would make Church-Rosser property fail to hold [vD80][Klo80] for the term calculus. The examples to show this would be, with $A \not\simeq B$,

$$\lambda x{:}A.(\lambda x{:}B.x)x$$

$$\mathbf{pair}_{B\times B}(\pi_1(\mathbf{pair}_{A\times A}(a,a)), \pi_2(\mathbf{pair}_{A\times A}(a,a)))$$

The first would reduce to $\lambda x{:}A.x$ by (β) and $\lambda x{:}B.x$ by (η); the second would reduce to $\mathbf{pair}_{B\times B}(a,a)$ by (σ) and $\mathbf{pair}_{A\times A}(a,a)$ by (π). It is also worth remarking that, with either of them, Church-Rosser even fails for well-typed terms of **ECC** because of the existence of type inclusions induced by universes. In fact, whenever $x{:}A \vdash x : B$ and $\vdash a : A$, the above two terms are well-typed. □

In the rest of this section, we prove the existence of the cumulativity relation as defined in definition 2.1.3 and some of its properties. We will show in section 3.3 that the cumulativity relation does characterize the type cumulativity in the calculus.

We first give an inductive definition of a binary relation over terms which will be shown to be the cumulativity relation as defined in definition 2.1.3.

Definition 3.1.3 (cumulativity relation: inductive definition) *Let* $\preceq_i \subseteq \mathcal{T} \times \mathcal{T}$ $(i \in \omega)$ *be the 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 $A_2 \preceq_i B_2$.*

$A \prec_i B$ if and only if $A \preceq_i B$ and $A \not\simeq B$.

Define $\preceq$ as

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

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

We show below that $\preceq$ defined above is the smallest binary relation over terms such that the four conditions in definition 2.1.3 are satisfied; in other words, the above is in fact an alternative definition of the cumulativity relation (corollary 3.1.7).

Lemma 3.1.4 *Let $\preceq$ be the relation defined by definition 3.1.3. Then, $A \preceq B$ if and only if one of the following holds:*

- *$A \preceq_0 B$;*
- *$A \simeq \Pi x{:}A_1.A_2$ and $B \simeq \Pi x{:}B_1.B_2$ for some $A_1 \simeq B_1$ and $A_2 \preceq B_2$;*
- *$A \simeq \Sigma x{:}A_1.A_2$ and $B \simeq \Sigma x{:}B_1.B_2$ for some $A_1 \preceq B_1$ and $A_2 \preceq B_2$.*

Proof Obvious from definition 3.1.3 of $\preceq$. □

Remark We may define a relation $\approx$ between terms: $A \approx B$ if and only if there exists a sequence of terms $M_0, ..., M_n$ such that $A \equiv M_0$, $B \equiv M_n$, and $M_i \preceq M_{i+1}$ or $M_{i+1} \preceq M_i$ for $0 \leq i < n$. Then, the relationship between $\approx$ and $\preceq$ is similar to that between $\simeq$ and $\triangleright$. The lemma 3.1.4 implies that, if $A \approx B$, then A and B have the same *sort* of forms up to conversion. □

The following lemma will be used to prove that the relation defined in definition 3.1.3 is a partial order (lemma 3.1.6) and that the cumulativity relation is well-founded (corollary 3.1.8).

Lemma 3.1.5 *Let A, B, C and D be terms, $i \in \omega$, and $\preceq$ be the relation defined in definition 3.1.3. If $A \prec_i B$, then*

1. *$B \preceq C$ implies $B \preceq_i C$, and*

2. *$D \preceq A$ implies $D \preceq_i A$.*

Proof By induction on $i \in \omega$. We only give this proof for the first part, *i.e.*, $B \preceq C$ implies $B \preceq_i C$, if $A \prec_i B$. The second part is symmetric and omitted.

For $i = 0$, A and B are convertible to some kinds. By Church-Rosser theorem, $B \preceq C$ must be because B and C are convertible to some kinds, and hence $B \preceq_0 C$.

For $i = k + 1$, we have the following two cases to consider:

1. $A \prec_k B$.

2. $A \not\prec_k B$ and $A \prec_{k+1} B$.

For the first case, $B \preceq C$ implies $B \preceq_k C$ by induction hypothesis and hence $B \preceq_{k+1} C$.

For the second case, for some $Q \in \{\Pi, \Sigma\}$, $A \simeq Qx{:}A_1.A_2$ and $B \simeq Qx{:}B_1.B_2$ such that

- $A_1 \simeq B_1$ and $A_2 \prec_k B_2$, if $Q \equiv \Pi$,
- $A_1 \prec B_1$ and $A_2 \preceq B_2$, or $A_1 \preceq B_1$ and $A_2 \prec B_2$, if $Q \equiv \Sigma$.

By Church-Rosser theorem and induction hypothesis, $B \not\prec_k C$, and $B \preceq C$ is either because $B \simeq C$, which implies $B \preceq_{k+1} C$, or because $B \simeq Qx{:}B_1'.B_2'$ and $C \simeq Qx{:}C_1.C_2$ for some $B_1' \begin{cases} \simeq C_1 & \text{if } Q \equiv \Pi \\ \preceq C_1 & \text{if } Q \equiv \Sigma \end{cases}$ and $B_2' \preceq C_2$. By Church-Rosser theorem, $B_j \simeq B_j'$ $(j = 1, 2)$. So, $B_1 \begin{cases} \simeq C_1 & \text{if } Q \equiv \Pi \\ \preceq C_1 & \text{if } Q \equiv \Sigma \end{cases}$ and $B_2 \preceq C_2$. Hence, by induction hypothesis, $B_1 \begin{cases} \simeq C_1 & \text{if } Q \equiv \Pi \\ \preceq_k C_1 & \text{if } Q \equiv \Sigma \end{cases}$ and $B_2 \preceq_k C_2$. Therefore, $B \preceq_{k+1} C$. □

Lemma 3.1.6 *The relation $\preceq$ defined in definition 3.1.3 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 $A \preceq C$.*

Proof We only have to show that every $\preceq_i$ is a partial order with respect to conversion. By induction on i.

The base case for $\preceq_0$ can readily be verified. Consider $\preceq_{k+1}$.

1. Reflexivity: Obvious by definition 3.1.3.

2. Anti-symmetry: Suppose $A \preceq_{k+1} B$ and $B \preceq_{k+1} A$. As $A \preceq_{k+1} B$, we have two cases to consider.

   (a) $A \preceq_k B$;

   (b) $A \not\preceq_k B$ and $A \prec_{k+1} B$.

   For the first case, either $A \simeq B$ or $A \prec_k B$, and we have $B \preceq_k A$ by definition of $\preceq_k$ and lemma 3.1.5. Then, $A \simeq B$, by induction hypothesis. For the second case, for some $Q \in \{\Pi, \Sigma\}$, $A \simeq Qx{:}A_1.A_2$ and $B \simeq Qx{:}B_1.B_2$ such that

   - $A_1 \simeq B_1$ and $A_2 \prec_k B_2$, if $Q \equiv \Pi$,
   - $A_1 \prec B_1$ and $A_2 \preceq B_2$, or $A_1 \preceq B_1$ and $A_2 \prec B_2$, if $Q \equiv \Sigma$.

   By Church-Rosser theorem and lemma 3.1.5, $B \preceq_{k+1} A$ is also due to the same reason, *i.e.*, $B \simeq Qx{:}B_1'.B_2'$ and $A \simeq Qx{:}A_1'.A_2'$ for some $B_1' \begin{cases} \simeq A_1' & \text{if } Q \equiv \Pi \\ \preceq_k A_1' & \text{if } Q \equiv \Sigma \end{cases}$ and $B_2' \preceq_k A_2'$. By Church-Rosser theorem, $A_j \simeq A_j'$ and $B_j \simeq B_j'$ $(j = 1, 2)$, and hence, $B_1 \begin{cases} \simeq A_1 & \text{if } Q \equiv \Pi \\ \preceq_k A_1 & \text{if } Q \equiv \Sigma \end{cases}$ and $B_2 \preceq_k A_2$. By induction hypothesis, we have $A_j \simeq B_j$ $(j = 1, 2)$ and hence $A \simeq B$.

3. Transitivity: Suppose $A \preceq_{k+1} B$ and $B \preceq_{k+1} C$. As $A \preceq_{k+1} B$, we have the same two cases as the above case. For the first case, we have $B \preceq_k C$ by definition of $\preceq_k$ and lemma 3.1.5. By induction hypothesis, $A \preceq_k C$ and hence $A \preceq_{k+1} C$. For the second case, by Church-Rosser theorem and lemma 3.1.5, $B \preceq_{k+1} C$ is also due to the same reason, *i.e.*, $B \simeq Qx{:}B_1'.B_2'$ and $C \simeq Qx{:}C_1.C_2$ for some $B_1' \begin{cases} \simeq C_1 & \text{if } Q \equiv \Pi \\ \preceq_k C_1 & \text{if } Q \equiv \Sigma \end{cases}$ and $B_2' \preceq_k C_2$. By Church-Rosser theorem, $B_j \simeq B_j'$ $(j = 1, 2)$, and hence, by induction hypothesis, $A_1 \begin{cases} \simeq C_1 & \text{if } Q \equiv \Pi \\ \preceq_k C_1 & \text{if } Q \equiv \Sigma \end{cases}$ and $A_2 \preceq_k C_2$, which implies $A \preceq_{k+1} C$.

As each $\preceq_i$ is a partial order with respect to conversion, so is the relation $\preceq$ defined in definition 3.1.3. □

Corollary 3.1.7 *The relation $\preceq$ defined in definition 3.1.3 is the smallest partial order over terms with respect to conversion such that*

1. *$Prop \preceq Type_0 \preceq Type_1 \preceq \ldots$;*
2. *if $A \simeq A'$ and $B \preceq B'$, then $\Pi x{:}A.B \preceq \Pi x{:}A'.B'$;*
3. *if $A \preceq A'$ and $B \preceq B'$, then $\Sigma x{:}A.B \preceq \Sigma x{:}A'.B'$.*

Proof By lemma 3.1.6, $\preceq$ is a partial order w.r.t. conversion and it obviously satisfies the three conditions. For minimality, suppose $R \subseteq \mathcal{T} \times \mathcal{T}$ to be a partial order w.r.t. conversion satisfying the conditions. We only have to show that $\preceq_i \subseteq R$ for every $i \in \omega$, which can easily be done by induction on i. □

Remark Definition 3.1.3 and the above corollary show that the cumulativity relation (definition 2.1.3) is well-defined; in other words, definition 3.1.3 gives an alternative inductive definition of the cumulativity relation. □

Using lemmas 3.1.5 and 3.1.4, we can also show that the cumulativity relation is well-founded.

Corollary 3.1.8 (well-foundedness of $\preceq$) *The cumulativity relation $\preceq$ is well-founded in the sense that there is no infinite decreasing sequence of the form $A_0 \succ A_1 \succ A_2 \succ \ldots$.*

Proof If there exists an infinite sequence $A_0 \succ A_1 \succ A_2 \succ \ldots$, we have by lemma 3.1.5, $A_0 \succ_i A_1 \succ_i A_2 \succ_i \ldots$ for some $i \in \omega$. So, we only have to show that $\preceq_i$ is well-founded for every $i \in \omega$.

By induction on i. $\preceq_0$ is obviously well-founded. Consider $\preceq_{i+1}$. If $A \succ_{i+1} B$, then there are three possibilities:

1. $A \succ_i B$,
2. $A \simeq \Pi x{:}A_1.A_2$ and $B \simeq \Pi x{:}B_1.B_2$ for some $A_1 \simeq B_1$ and $A_2 \succ_i B_2$, or
3. $A \simeq \Sigma x{:}A_1.A_2$ and $B \simeq \Sigma x{:}B_1.B_2$ for some A_1, A_2, B_1 and B_2 such that, $A_1 \succ_i B_1$ and $A_2 \succeq_i B_2$, or $A_1 \succeq_i B_1$ and $A_2 \succ_i B_2$.

For the first case, there is no infinite decreasing sequence starting from $A \succ B$ by induction hypothesis and lemma 3.1.5. For the second case, by lemma 3.1.4, every component of a decreasing $\succ_{i+1}$-sequence starting from $A \succ B$ is convertible to a term of Π-form. Hence, if such a sequence is infinite, there must be an infinite decreasing sequence starting from $A_2 \succ B_2$, which is impossible by induction hypothesis and lemma 3.1.5. The third case for Σ can be similarly proved. Hence, every $\preceq_i$ is well-founded and so is $\preceq$ by lemma 3.1.5. □

3.2 Derivable Judgements and Derivability

Shown in this section and the next are the basic properties of **ECC**. We show in this section that, if a judgement $x_1{:}A_1, \ldots, x_n{:}A_n \vdash M : A$ is derivable, then

- $x_1{:}A_1, \ldots, x_i{:}A_i$ $(i = 1, \ldots, n)$ are valid contexts (followed by lemma 3.2.3);
- A and A_i are all types (followed by lemma 3.2.1 and theorem 3.2.7);
- the variables $x_1, \ldots, x_n$ are distinct, the free variables in M and A are among $x_1, \ldots, x_n$ and those in A_i are among $x_1, \ldots, x_{i-1}$. (lemma 3.2.2).

These give us a better understanding of the forms of derivable judgements.

We also show that the following operations on derivable judgements are admissible:

- Context replacement by $B \preceq A$ (lemma 3.2.5);
- Type-preserving substitution or Cut (theorem 3.2.6);
- Subject reduction (theorem 3.2.8);
- Weakening and strengthening (lemmas 3.2.4 and 3.2.9).

These provide us important admissible rules[2] which not only enable one to understand the calculus (derivability, in particular) better but also allow one to use them in implementations of the calculus (*c.f.*, [LPT89]).

Lemma 3.2.1 *Any derivation of* $\Gamma, x{:}A, \Gamma' \vdash M : B$ *has a sub-derivation of* $\Gamma \vdash A : Type_j$ *for some* j*.*

Proof By induction on derivations.[3] □

Lemma 3.2.2 (free variables) *Suppose* $\Gamma \vdash M : A$*. Then,*

1. $FV(M) \cup FV(A) \subseteq FV(\Gamma)$.
2. Γ *has the form* $x_1{:}A_1, ..., x_n{:}A_n$ *such that* $x_1, ..., x_n$ *are distinct and* $FV(A_i) \subseteq \{x_1, ..., x_{i-1}\}$ *for* $i = 1, ..., n$*.*

Proof By induction on derivations. When proving the first, use 3.2.1 for $(\Pi 1)(\lambda)$. When proving the second, use the first for (C). □

Lemma 3.2.3 (context validity) *Any derivation of* $\Gamma, \Gamma' \vdash M : A$ *has a sub-derivation of* $\Gamma \vdash Prop : Type_0$*.*

Proof By induction on derivations. □

Lemma 3.2.4 (weakening) *If* $\Gamma \vdash M : A$ *and* Γ' *is a valid context which contains every component of* Γ*, then* $\Gamma' \vdash M : A$*.*

Proof By induction on derivations. For the rules other than $(\Pi 1)(\lambda)$, apply induction hypothesis and the same rule. For $(\Pi 1)(\lambda)$, use lemma 3.2.1 and then similar. □

Remark The weakening lemma expresses the monotonicity of the calculus, *i.e.*, postulating more assumptions does not invalidate provable results. □

---

[2]A rule R of the form $\frac{J_1 \;\; ... \;\; J_n}{J}$ is called *admissible* if J is derivable whenever $J_1, ..., J_n$ are derivable.

[3]We will say 'by induction on derivations (of ...)' to mean 'by induction on the lengthes of derivations (of ...)'.

Lemma 3.2.5 (context replacement) *If* $\Gamma, x{:}A, \Gamma' \vdash M : C$ *and* $B \preceq A$ *is a* Γ*-type, then* $\Gamma, x{:}B, \Gamma' \vdash M : C$*.*

Proof By induction on derivations of $\Gamma, x{:}A, \Gamma' \vdash M : C$. The only two non-trivial cases are rule (C) and rule (var). For rule (C), we have two possibilities:

1. $\Gamma' \equiv \Gamma_1', y{:}A_1$, $M \equiv Prop$ and $C \equiv Type_0$:

$$\frac{\Gamma, x{:}A, \Gamma_1' \vdash A_1 : Type_j}{\Gamma, x{:}A, \Gamma_1', y{:}A_1 \vdash Prop : Type_0} \quad (y \notin FV(\Gamma, x{:}A, \Gamma_1'))$$

   By induction hypothesis, $\Gamma, x{:}B, \Gamma_1' \vdash A_1 : Type_j$. By lemma 3.2.2, $y \notin FV(\Gamma, x{:}B, \Gamma_1')$. So, applying (C) suffices.

2. $\Gamma' \equiv \langle\rangle$:

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

   As B is a Γ-type, $\Gamma \vdash B : Type_j$ for some j. Applying (C) suffices.

For rule (var), with $M \equiv x$ and $C \equiv A$,

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

By induction hypothesis, $\Gamma, x{:}B, \Gamma' \vdash Prop : Type_0$. By rule (var), $\Gamma, x{:}B, \Gamma' \vdash x : B$. As A is a Γ-type by lemma 3.2.1, we have by lemma 3.2.4 that A is a $(\Gamma, x{:}B, \Gamma')$-type. Hence, $\Gamma, x{:}B, \Gamma' \vdash x : A$ by rule $(\preceq)$ as $B \preceq A$. □

Remark As a special case of the above lemma, replacing a type in the context of a judgement by a convertible type results in an 'equivalent' judgement subject to derivability. The above lemma is another sort of 'weakening' lemma as one gets a possibly stronger assumption when replacing A by $B \preceq A$. □

Theorem 3.2.6 (Cut) *If* $\Gamma, x{:}N, \Gamma' \vdash P : A$ *and* $\Gamma \vdash M : N$*, then* $\Gamma, [M/x]\Gamma' \vdash [M/x]P : [M/x]A$*.*

Proof By induction on derivations of $\Gamma, x{:}N, \Gamma' \vdash P : A$. Here, we only check the rules (var), $(\Pi 1)$ and $(pair)$. The other cases are simpler or similar. For rule (var), with $\Gamma, x{:}N, \Gamma' \equiv \Gamma_1, y{:}B, \Gamma_2$,

$$\frac{\Gamma_1, y{:}B, \Gamma_2 \vdash Prop : Type_0}{\Gamma_1, y{:}B, \Gamma_2 \vdash y : B}$$

there are two cases:

1. $x{:}N \equiv y{:}B$, $\Gamma \equiv \Gamma_1$ and $\Gamma' \equiv \Gamma_2$. By lemma 3.2.2, $x \notin FV(N)$. So, we only have to show $\Gamma, [M/x]\Gamma' \vdash M : N$. This is true by induction hypothesis and lemma 3.2.4, as $\Gamma \vdash M : N$.

2. $x{:}N$ occurs in Γ_1 or Γ_2. By induction hypothesis, $\Gamma, [M/x]\Gamma' \vdash Prop : Type_0$. As $x \not\equiv y$ by lemma 3.2.2, $\Gamma, [M/x]\Gamma'$ contains the component $y{:}[M/x]B$. So, an application of rule (var) yields the result.

For rule $(\Pi 1)$, with $P \equiv \Pi x{:}P_1.P_2$ and $A \equiv Prop$,

$$\frac{\Gamma, x{:}N, \Gamma', y{:}P_1 \vdash P_2 : Prop}{\Gamma, x{:}N, \Gamma' \vdash \Pi x{:}P_1.P_2 : Prop}$$

As $x \not\equiv y$ by lemma 3.2.2, $\Gamma, [M/x]\Gamma', y{:}[M/x]P_1 \vdash [M/x]P_2 : Prop$ by induction hypothesis. By rule $(\Pi 1)$, $\Gamma, [M/x]\Gamma' \vdash \Pi y{:}[M/x]P_1.[M/x]P_2 : Prop$. Since M is a Γ-term, $y \notin FV(M)$ by lemma 3.2.2. So, $\Gamma, [M/x]\Gamma' \vdash [M/x]\Pi x{:}P_1.P_2 : Prop$ as required.

For rule $(pair)$, (write Γ_1 for $\Gamma, x{:}N, \Gamma'$,)

$$\frac{\Gamma_1 \vdash M_1 : A_1 \quad \Gamma_1 \vdash N_1 : [M_1/y]B_1 \quad \Gamma_1, y{:}A_1 \vdash B_1 : Type_j}{\Gamma_1 \vdash \mathbf{pair}_{\Sigma y:A_1.B_1}(M_1, N_1) : \Sigma y{:}A_1.B_1}$$

Note that $x \not\equiv y$ and $x \notin FV(M)$ by lemma 3.2.2. By induction hypothesis, we have

$$\Gamma, [M/x]\Gamma' \vdash [M/x]M_1 : [M/x]A_1$$
$$\Gamma, [M/x]\Gamma' \vdash [M/x]N_1 : [M/x][M_1/x]B_1$$
$$\Gamma, [M/x]\Gamma', y{:}[M/x]A_1 \vdash [M/x]B_1 : Type_j$$

Noticing that $[M/x][M_1/y]B_1 \equiv [[M/x]M_1/y][M/x]B_1$, we have by rule (Σ),

$$\Gamma, [M/x]\Gamma' \vdash \mathbf{pair}_{\Sigma y:[M/x]A_1.[M/x]B_1}([M/x]M_1, [M/x]N_1) : \Sigma y{:}[M/x]A_1.[M/x]B_1$$

As $x \not\equiv y$, this judgement is

$$\Gamma, [M/x]\Gamma' \vdash [M/x]\mathbf{pair}_{\Sigma y:A_1.B_1}(M_1, N_1) : [M/x]\Sigma y{:}A_1.B_1$$

as required. □

Remark The name of the above lemma (also used in [Pot87]) is due to the analogy with the cut rule in sequent calculus of the form

$$\frac{\Gamma, N \vdash A \quad \Gamma \vdash N}{\Gamma \vdash A}$$

□

Theorem 3.2.7 *If $\Gamma \vdash M : A$, then A is a Γ-type.*

Proof By induction on derivations of $\Gamma \vdash M : A$. For the rules except (*app*) and ($\pi 2$), it is easy. (We only remark that lemmas 3.2.1 and 3.2.4 are used for (var), and lemma 3.2.1 for (λ).) The cases for (*app*) and ($\pi 2$) are similar. We check ($\pi 2$) here. With $M \equiv \pi_2(M')$ and $A \equiv [\pi_1(M')/x]A_2$,

$$\frac{\Gamma \vdash M' : \Sigma x{:}A_1.A_2}{\Gamma \vdash \pi_2(M') : [\pi_1(M')/x]A_2}$$

By induction hypothesis, $\Gamma \vdash \Sigma x{:}A_1.A_2 : K$ for some kind K. Any derivation D of this judgement must have (Σ) or ($\preceq$) as the last rule used. So, D must have a subderivation which is a derivation of $\Gamma \vdash \Sigma x{:}A_1.A_2 : Type_j$ with (Σ) as the last rule; *i.e.*, we have

$$\frac{\Gamma \vdash A_1 : Type_j \quad \Gamma, x{:}A_1 \vdash A_2 : Type_j}{\Gamma \vdash \Sigma x{:}A_1.A_2 : Type_j}$$

As $\Gamma \vdash \pi_1(M') : A_1$, we have $\Gamma \vdash [\pi_1(M')/x]A_2 : Type_j$ by theorem 3.2.6. So, A is a Γ-type. □

Remark This theorem says that every inhabited term is a type. However, the converse is not true in general; not every Γ-type is necessarily inhabited under Γ (see theorem 6.1.5). □

Theorem 3.2.8 (subject reduction) *If* $\Gamma \vdash M : A$ *and* $M \rhd N$, *then* $\Gamma \vdash N : A$.

Proof We only need to show that, if $\Gamma \vdash M : A$ and $M \rhd_1 N$, then $\Gamma \vdash N : A$. This is proved by induction on derivations of $\Gamma \vdash M : A$.

1. $(Ax)(C)(T)(var)$: Trivial.

2. ($\preceq$): By induction hypothesis and applying ($\preceq$).

3. ($\Pi 1$)($\Pi 2$)(λ)(Σ)(*pair*): These cases are similar in which lemma 3.2.5 and (or) lemma 3.2.1 and (or) theorem 3.2.6 are used. We check (*pair*) here.

$$\frac{\Gamma \vdash M_1 : A_1 \quad \Gamma \vdash N_1 : [M_1/x]B_1 \quad \Gamma, x{:}A_1 \vdash B_1 : Type_j}{\Gamma \vdash \mathbf{pair}_{\Sigma x:A_1.B_1}(M_1, N_1) : \Sigma x{:}A_1.B_1}$$

   So, $M \equiv \mathbf{pair}_{\Sigma x:A_1.B_1}(M_1, N_1) \rhd_1 \mathbf{pair}_{\Sigma x:A_1'.B_1'}(M_1', N_1') \equiv N$. There are four cases:

   (a) $M_1 \rhd_1 M_1'$: By induction hypothesis, $\Gamma \vdash M_1' : A_1$; by theorem 3.2.6 and applying rule ($\preceq$), we have $\Gamma \vdash N_1 : [M_1'/x]B_1$.

   (b) $N_1 \rhd_1 N_1'$: By induction hypothesis, $\Gamma \vdash N_1' : [M_1/x]B_1$.

   (c) $A_1 \rhd_1 A_1'$: By lemma 3.2.1, induction hypothesis, lemma 3.2.5 and rule ($\preceq$), $\Gamma \vdash M_1 : A_1'$ and $\Gamma, x{:}A_1' \vdash B_1 : Type_j$.

(d) $B_1 \rhd_1 B_1'$: By induction hypothesis, theorem 3.2.6 and rule $(\preceq)$, $\Gamma, x{:}A_1 \vdash B_1' : Type_j$ and $\Gamma \vdash N_1 : [M_1/x]B_1'$.

Then applying $(pair)$ suffices in every case above.

4. (app): With $M \equiv M_1 N_1$ and $A \equiv [N_1/x]B_1$,

$$\frac{\Gamma \vdash M_1 : \Pi x{:}A_1.B_1 \quad \Gamma \vdash N_1 : A_1}{\Gamma \vdash M_1 N_1 : [N_1/x]B_1}$$

There are two cases:

(a) $N \equiv M_1' N_1'$ and either $M_1 \rhd_1 M_1'$ or $N_1 \rhd_1 N_1'$. In this case, by induction hypothesis, $\Gamma \vdash M_1' : \Pi x{:}A_1.B_1$ and $\Gamma \vdash N_1' : A_1$. So, applying (app) yields $\Gamma \vdash N : [N_1'/x]B_1$. Since $[N_1'/x]B_1 \simeq [N_1/x]B_1$, we have $\Gamma \vdash N : [N_1/x]B_1$ by theorem 3.2.7 and rule $(\preceq)$.

(b) $M \equiv M_1 N_1 \equiv (\lambda x{:}A_1'.M_1')N_1 \rhd_1 [N_1/x]M_1' \equiv N$. The last rule used in any derivation of $\Gamma \vdash M_1 : \Pi x{:}A_1.B_1$ must be (λ) or $(\preceq)$. If it is (λ), applying theorem 3.2.6 suffices. If it ends with $(\preceq)$, we have for some $X \preceq \Pi x{:}A_1.B_1$,

$$\frac{\Gamma \vdash \lambda x{:}A_1'.M_1' : X \quad \Gamma \vdash X : Type_j}{\Gamma \vdash \lambda x{:}A_1'.M_1' : \Pi x{:}A_1.B_1}$$

We may assume that the last rule used to derive $\Gamma \vdash \lambda x{:}A_1'.M_1' : X$ is not $(\preceq)$, then it must be (λ), *i.e.*,

$$\frac{\Gamma, x{:}A_1' \vdash M_1' : B_1'}{\Gamma \vdash \lambda x{:}A_1'.M_1' : \Pi x{:}A_1'.B_1'}$$

where $X \equiv \Pi x{:}A_1'.B_1'$. By lemma 3.1.4, $X \simeq \Pi x{:}A_1''.B_1'' \preceq \Pi x{:}A_1.B_1$ for some A_1'' and B_1'' such that $A_1'' \simeq A_1$ and $B_1'' \preceq B_1$. By Church-Rosser theorem, $X \rhd \Pi x{:}A_0.B_0$ and $\Pi x{:}A_1''.B_1'' \rhd \Pi x{:}A_0.B_0$ for some A_0 and B_0 such that $A_1' \rhd A_0$, $A_1'' \rhd A_0$, $B_1' \rhd B_0$ and $B_1'' \rhd B_0$. So, we have

$$A_1' \simeq A_0 \simeq A_1'' \simeq A_1 \quad \text{and} \quad B_1' \simeq B_0 \simeq B_1'' \preceq B_1$$

By theorem 3.2.7, lemma 3.2.5 and rule $(\preceq)$, we have $\Gamma, x{:}A_1 \vdash M_1' : B_1$. Then, by theorem 3.2.6, we have $\Gamma \vdash [N_1/x]M_1' : [N_1/x]B_1$, *i.e.*, $\Gamma \vdash N : [N_1/x]B_1$.

5. $(\pi 1)(\pi 2)$:

$$\frac{\Gamma \vdash M_1 : \Sigma x{:}A_1.B_1}{\Gamma \vdash \pi_1(M_1) : A_1}$$

$$\frac{\Gamma \vdash M_1 : \Sigma x{:}A_1.B_1}{\Gamma \vdash \pi_2(M_1) : [\pi_1(M_1)/x]B_1}$$

$M \equiv \pi_i(M_1) \rhd_1 N$ $(i = 1, 2)$ and A is A_1 and $[\pi_1(M_1)/x]B_1$, respectively. There are two cases:

(a) $N \equiv \pi_i(M_1')$ and $M_1 \rhd_1 M_1'$. By induction hypothesis and applying $(\pi 1)$ and $(\pi 2)$.

(b) $M \equiv \pi_i(M_1) \equiv \pi_i(\mathbf{pair}_C(M_{11}, M_{12})) \rhd_1 M_{1i} \equiv N$. Then, any derivation D of $\Gamma \vdash M_1 : \Sigma x{:}A_1.B_1$ must use $(pair)$ or $(\preceq)$ as the last rule. If the last rule used in D is $(pair)$, we have $\Gamma \vdash M_{11} : A_1$, and, by theorem 3.2.7 and rule $(\preceq)$, $\Gamma \vdash M_{12} : [\pi_1(M_1)/x]B_1$. That is, $\Gamma \vdash N : A$. If the last rule used in D is $(\preceq)$, we have, for some $X \preceq \Sigma x{:}A_1.B_1$,

$$\frac{\Gamma \vdash M_1 : X \quad \Gamma \vdash X : Type_j}{\Gamma \vdash M_1 : \Sigma x{:}A_1.B_1}$$

We may assume that the last rule used to derive $\Gamma \vdash M_1 : X$ is not $(\preceq)$, then it must be $(pair)$, *i.e.*,

$$\frac{\Gamma \vdash M_{11} : A_1' \quad \Gamma \vdash M_{12} : [M_{11}/x]B_1' \quad \Gamma, x{:}A_1' \vdash B_1' : Type_j}{\Gamma \vdash M_1 : \Sigma x{:}A_1'.B_1'}$$

where $X \equiv \Sigma x{:}A_1'.B_1'$. By lemma 3.1.4, $X \simeq \Sigma x{:}A_1''.B_1'' \preceq \Pi x{:}A_1.B_1$ for some A_1'' and B_1'' such that $A_1'' \preceq A_1$ and $B_1'' \preceq B_1$. By Church-Rosser theorem, $X \rhd \Sigma x{:}A_0.B_0$ and $\Sigma x{:}A_1''.B_1'' \rhd \Sigma x{:}A_0.B_0$ for some A_0 and B_0 such that $A_1' \rhd A_0$, $A_1'' \rhd A_0$, $B_1' \rhd B_0$ and $B_1'' \rhd B_0$. So, we have

$$A_1' \simeq A_0 \simeq A_1'' \preceq A_1 \quad \text{and} \quad B_1' \simeq B_0 \simeq B_1'' \preceq B_1$$

and the later implies $[M_{11}/x]B_1' \preceq [\pi_1(M_1)/x]B_1$. By theorem 3.2.7 and rule $(\preceq)$, $\Gamma \vdash M_{11} : A_1$ and $\Gamma \vdash M_{12} : [\pi_1(M_1)/x]B_1$, *i.e.*, $\Gamma \vdash N : A$.

This completes the proof of the theorem. □

Remark The theorem of subject reduction is one of the most important properties of a type system like **ECC**. Besides its importance in meta theory, it also saves much work in implementation, *e.g.*, it saves type-checking when reductions or normalizations are performed.

Although subject reduction holds, the following rule is *not* admissible in **ECC**:

$$(**) \qquad \frac{\Gamma \vdash M{:}A \quad \Gamma \vdash N{:}B}{\Gamma \vdash M{:}B} \quad (M \rhd N)$$

For example, we have $\vdash Prop : Type_0$, but $\not\vdash (\lambda x{:}Type_1.x)Prop : Type_0$. In fact, we only have $\vdash (\lambda x{:}Type_1.x)Prop : Type_i$ for $i \geq 1$, *i.e.*, its principal type is $Type_1$ (see section 3.3). □

Lemma 3.2.9 (strengthening) *If* $\Gamma, y{:}Y, \Gamma' \vdash M : A$ *and* $y \notin FV(M) \cup FV(A) \cup FV(\Gamma')$, *then* $\Gamma, \Gamma' \vdash M : A$.

Proof Note that a straightforward induction on derivations does not work as the (app) rule loses the information of variable occurrences (in A). To solve this problem, we notice that we only have to prove the following statement:

$(*)$ if $\Gamma, y{:}Y, \Gamma' \vdash M : A$ and $y \notin FV(M) \cup FV(\Gamma')$, then there exists $A' \preceq A$ such that $\Gamma, \Gamma' \vdash M : A'$.

for then, supposing $\Gamma, y{:}Y, \Gamma' \vdash M : A$ and $y \notin FV(M) \cup FV(A) \cup FV(\Gamma')$, we have by $(*)$ that there exists $A' \preceq A$ such that $\Gamma, \Gamma' \vdash M : A'$. We only have to show $\Gamma, \Gamma' \vdash A : K$ for some kind K in order to apply rule $(\preceq)$ to show $\Gamma, \Gamma' \vdash M : A$. By theorem 3.2.7, $\Gamma, y{:}Y, \Gamma' \vdash A : K$ for some kind K. As $y \notin FV(A) \cup FV(\Gamma')$, by $(*)$, there exists $B \preceq K$ such that $\Gamma, \Gamma' \vdash A : B$. Because kind K is a Γ, Γ'-type (by rules $(Ax), (C)$ and (T)), we can apply rule $(\preceq)$ to have $\Gamma, \Gamma' \vdash A : K$.

$(*)$ is proved by induction on derivations of $\Gamma, y{:}Y, \Gamma' \vdash M : A$.

1. (Ax): Trivial.

2. (C): With $\Gamma, y{:}Y, \Gamma' \equiv \Gamma_1, x{:}A_1$,

$$\frac{\Gamma_1 \vdash A_1 : Type_j}{\Gamma_1, x{:}A_1 \vdash Prop : Type_0} \quad (x \notin FV(\Gamma_1))$$

   If $y{:}Y$ does not occur in Γ_1 (*i.e.*, Γ' is empty and $y{:}Y \equiv x{:}A_1$), then we have $\Gamma_1 \vdash Prop : Type_0$ by lemma 3.2.3. Otherwise, $\Gamma, y{:}Y, \Gamma' \equiv \Gamma, y{:}Y, \Gamma'', x{:}A_1$ and, by induction hypothesis, there exists $C \preceq Type_j$ such that $\Gamma, \Gamma'' \vdash A_1 : C$. By Church-Rosser theorem, $C \simeq K \preceq Type_j$ for some kind K. So, $\Gamma, \Gamma'' \vdash A_1 : Type_k$ for some k by rule $(\preceq)$ and hence $\Gamma, \Gamma'', x{:}A_1 \vdash Prop : Type_0$ by rule (C), *i.e.*, $\Gamma, \Gamma' \vdash Prop : Type_0$.

3. (T): With $\Gamma, y{:}Y, \Gamma' \equiv \Gamma_1$,

$$\frac{\Gamma_1 \vdash Prop : Type_0}{\Gamma_1 \vdash Type_j : Type_{j+1}}$$

   By induction hypothesis, $\Gamma, \Gamma' \vdash Prop : C$ for some $C \preceq Type_0$. By Church-Rosser theorem, $C \simeq K \preceq Type_0$ for some kind K. So, $\Gamma, \Gamma' \vdash Prop : Type_0$ and hence $\Gamma, \Gamma' \vdash Type_j : Type_{j+1}$ by rule (T).

4. (var): With $\Gamma, y{:}Y, \Gamma' \equiv \Gamma_1, x{:}A_1, \Gamma_2$,

$$\frac{\Gamma_1, x{:}A_1, \Gamma_2 \vdash Prop : Type_0}{\Gamma_1, x{:}A_1, \Gamma_2 \vdash x : A_1}$$

   Note that $y \not\equiv x$ by assumption. By induction hypothesis, we have $\Gamma, \Gamma' \vdash Prop : C$ for some $C \preceq Type_0$ which implies $\Gamma, \Gamma' \vdash Prop : Type_0$ by Church-Rosser theorem. So, $\Gamma, \Gamma' \vdash x : A_1$.

5. ($\Pi 1$): With $\Gamma, y{:}Y, \Gamma' \equiv \Gamma_1$,

$$\frac{\Gamma_1, x{:}A_1 \vdash P : Prop}{\Gamma_1 \vdash \Pi x{:}A_1.P : Prop}$$

   We have $y \not\equiv x$ by lemma 3.2.2. By induction hypothesis, there exists $C \preceq Prop$ such that $\Gamma, \Gamma', x{:}A \vdash P : C$. By Church-Rosser theorem, $C \simeq Prop$, so $\Gamma, \Gamma', x{:}A \vdash P : Prop$ by rule ($\preceq$). Then applying ($\Pi 1$) we have $\Gamma, \Gamma' \vdash \Pi x{:}A_1.P : Prop$.

6. ($\Pi 2$): With $\Gamma, y{:}Y, \Gamma' \equiv \Gamma_1$,

$$\frac{\Gamma_1 \vdash A_1 : Type_j \quad \Gamma_1, x{:}A_1 \vdash B : Type_j}{\Gamma_1 \vdash \Pi x{:}A_1.B : Type_j}$$

   We have $y \not\equiv x$ by lemma 3.2.2. By induction hypothesis, there exist $C \preceq Type_j$ and $D \preceq Type_j$ such that $\Gamma, \Gamma' \vdash A_1 : C$ and $\Gamma, \Gamma', x{:}A_1 \vdash B : D$. By Church-Rosser theorem, $C \simeq K \preceq Type_j$ and $D \simeq K' \preceq Type_j$ for some kinds K and K'. Then, by rule ($\preceq$), $\Gamma, \Gamma' \vdash A_1 : Type_j$ and $\Gamma, \Gamma', x{:}A_1 \vdash B : Type_j$. So, $\Gamma, \Gamma' \vdash \Pi x{:}A_1.B : Type_j$ by rule ($\Pi 2$).

7. (λ): With $\Gamma, y{:}Y, \Gamma' \equiv \Gamma_1$,

$$\frac{\Gamma_1, x{:}A_1 \vdash M_1 : B}{\Gamma_1 \vdash \lambda x{:}A_1.M_1 : \Pi x{:}A_1.B}$$

   We have $y \notin FV(M)$ by lemma 3.2.2. By induction hypothesis, $\Gamma, \Gamma', x{:}A_1 \vdash M_1 : B_1$ for some $B_1 \preceq B$. Applying rule (λ) gives us $\Gamma, \Gamma' \vdash \lambda x{:}A_1.M_1 : \Pi x{:}A_1.B_1$ and $\Pi x{:}A_1.B_1 \preceq \Pi x{:}A_1.B$.

8. (app): With $\Gamma, y{:}Y, \Gamma' \equiv \Gamma_1$, $M \equiv M_1 M_2$ and $A \equiv [M_2/x]B_1$,

$$\frac{\Gamma_1 \vdash M_1 : \Pi x{:}A_1.B_1 \quad \Gamma_1 \vdash M_2 : A_1}{\Gamma_1 \vdash M_1 M_2 : [M_2/x]B_1}$$

   By induction hypothesis, there exist $C \preceq \Pi x{:}A_1.B_1$ and $D \preceq A_1$ such that $\Gamma, \Gamma' \vdash M_1 : C$ and $\Gamma, \Gamma' \vdash M_2 : D$. By lemma 3.1.4, $C \simeq \Pi x{:}A_1'.B_1'$ for some $A_1' \simeq A_1$ and $B_1' \preceq B_1$. By Church-Rosser theorem, $C \rhd \Pi x{:}A_0.B_0$ and $\Pi x{:}A_1'.B_1' \rhd \Pi x{:}A_0.B_0$ for some A_0 and B_0 such that $A_1' \rhd A_0$ and $B_1' \rhd B_0$. So, $A_0 \simeq A_1$, $B_0 \preceq B_1$. By lemma 3.2.2, $y \notin FV(C) \cup FV(D)$, which implies that $y \notin FV(\Pi x{:}A_0.B_0)$. By theorem 3.2.7, theorem 3.2.8 and rule ($\preceq$), $\Gamma, \Gamma' \vdash M_1 : \Pi x{:}A_0.B_0$ and $\Gamma, \Gamma' \vdash M_2 : A_0$. Applying rule ($app$), we have $\Gamma, \Gamma' \vdash M_1 M_2 : [M_2/x]B_0$ and $[M_2/x]B_0 \preceq [M_2/x]B_1$.

9. (Σ): Similar to the case for ($\Pi 2$).

10. (*pair*):[4] With $\Gamma, y{:}Y, \Gamma' \equiv \Gamma_1$, $M \equiv \mathbf{pair}_{\Sigma x:A_1.B_1}(M_1, M_2)$ and $A \equiv \Sigma x{:}A_1.B_1$,

$$\frac{\Gamma_1 \vdash M_1 : A_1 \quad \Gamma_1 \vdash M_2 : [M_1/x]B_1 \quad \Gamma_1, x{:}A_1 \vdash B_1 : Type_j}{\Gamma_1 \vdash \mathbf{pair}_{\Sigma x:A_1.B_1}(M_1, M_2) : \Sigma x{:}A_1.B_1}$$

By induction hypothesis, there exist $A_1' \preceq A_1$, $B_1' \preceq [M_1/x]B_1$ $C \preceq Type_j$ and $D \preceq Type_j$ such that $\Gamma, \Gamma' \vdash M_1 : A_1'$, $\Gamma, \Gamma' \vdash M_2 : B_1'$ and $\Gamma, \Gamma', x{:}A_1 \vdash B_1 : C$. Noticing that $y \notin FV(\Sigma x{:}A_1.B_1)$, we have, by lemma 3.2.1, induction hypothesis and rule $(\preceq)$, $\Gamma, \Gamma' \vdash M_1 : A_1$ and $\Gamma, \Gamma', x{:}A_1 \vdash B_1 : Type_j$. By theorem 3.2.6, $\Gamma, \Gamma' \vdash [M_1/x]B_1 : Type_j$. So, by rule $(\preceq)$, $\Gamma, \Gamma' \vdash M_2 : [M_1/x]B_1$. Hence, applying (*pair*) yields $\Gamma, \Gamma' \vdash M : A$.

11. $(\pi 1)(\pi 2)$: With $\Gamma, y{:}Y, \Gamma' \equiv \Gamma_1$ and $M \equiv \pi_i(M_1)$ $(i = 1, 2)$ and $A \equiv \begin{cases} A' & \text{for } (\pi 1) \\ [\pi_1(M_1)/x]B & \text{for } (\pi 2) \end{cases}$,

$$\frac{\Gamma_1 \vdash M_1 : \Sigma x{:}A'.B}{\Gamma_1 \vdash \pi_1(M_1) : A'}$$

$$\frac{\Gamma_1 \vdash M_1 : \Sigma x{:}A'.B}{\Gamma_1 \vdash \pi_2(M_1) : [\pi_1(M_1)/x]B}$$

By induction hypothesis, $\Gamma, \Gamma' \vdash M_1 : C$ for some $C \preceq \Sigma x{:}A'.B$. By lemma 3.1.4, $C \simeq \Sigma x{:}A_1.B_1$ for some $A_1 \preceq A'$ and $B_1 \preceq B$. By Church-Rosser theorem, $C \rhd \Sigma x{:}A_0.B_0$ for some $A_0 \simeq A_1 \preceq A'$ and $B_0 \simeq B_1 \preceq B$. By lemma 3.2.2, $y \notin FV(C)$ and hence $y \notin FV(\Sigma x{:}A_0.B_0)$. By theorem 3.2.7, theorem 3.2.8 and rule $(\preceq)$, $\Gamma, \Gamma' \vdash M_1 : \Sigma x{:}A_0.B_0$. So, for $(\pi 1)$, $\Gamma, \Gamma' \vdash \pi_1(M_1) : A_0$ by rule $(\pi 1)$ and $A_0 \preceq A'$; for $(\pi 2)$, $\Gamma, \Gamma' \vdash \pi_2(M_1) : [\pi_1(M_1)/x]B_0$ by rule $(\pi 2)$ and $[\pi_1(M_1)/x]B_0 \preceq [\pi_1(M_1)/x]B$.

This completes the proof of the lemma. □

Remark Strengthening is the dual of weakening (lemma 3.2.4). It shows that removing redundant assumptions preserves derivability. That it holds for Constructions-like calculi and the idea of proving a stronger statement as shown in the above proof were recognized by the author and presented in [Luo88b]. It is interesting to note that, in an implementation of a proof refinement system based on Constructions (*e.g.*, [CH85][LPT89]), such a lemma is indeed (maybe unconsciously) used (*e.g.*, to implement the Discharge command). □

3.3 Principal Types

Because we have type inclusions induced by type universes, type uniqueness up to conversion fails for **ECC**. However, we show that **ECC** has a simple notion of *principal type* which characterizes the set of types of a well-typed term.

[4] Thanks to Moggi for pointing out a simpler way of proving this case, as presented here, after reading the draft of this thesis.

First, we show that the cumulativity relation characterizes the type cumulativity (or type inclusions) in the calculus.

Lemma 3.3.1 (type cumulativity) *Let A and B be Γ-types. Then, $A \preceq B$ if and only if $\Gamma, x{:}A \vdash x : B$, where $x \notin FV(\Gamma)$.*

Proof The sufficiency is by induction on derivations of $\Gamma, x{:}A \vdash x : B$. The necessity is by rules $(\preceq)(C)(var)$ and lemma 3.2.5. $\Box$

Corollary 3.3.2 *Let A and B be Γ-types. If $A \preceq B$, then, for any term M, $\Gamma \vdash M : A$ implies $\Gamma \vdash M : B$.*

Proof By lemma 3.3.1, $\Gamma, x{:}A \vdash x : B$, where $x \notin FV(\Gamma)$. By theorem 3.2.6, $\Gamma \vdash M : [M/x]B$; *i.e.*, $\Gamma \vdash M : B$, as x does not occur free in Γ-type B by lemma 3.2.2. $\Box$

Remark The converse of this corollary is *not* true as A might be empty (not inhabited by any term) under Γ. $\Box$

Lemma 3.3.3 (diamond property of $\preceq$) *If $\Gamma \vdash M : A$ and $\Gamma \vdash M : B$, then there exists a term C such that $C \preceq A$, $C \preceq B$ and $\Gamma \vdash M : C$.*

Proof By induction on the sum of the lengths of derivations of $\Gamma \vdash M : A$ and $\Gamma \vdash M : B$. Here, we only consider the case when both derivations use (*app*) as the last rule. The other cases are easy. Suppose $M \equiv M_1M_2$, $A \equiv [M_2/x]B_1$, $B \equiv [M_2/x]B_2$ and, for $i = 1, 2$,

$$\frac{\Gamma \vdash M_1 : \Pi x{:}A_i.B_i \quad \Gamma \vdash M_2 : A_i}{\Gamma \vdash M_1M_2 : [M_2/x]B_i}$$

By induction hypothesis, there exists C such that $\Gamma \vdash M_1 : C$ and $C \preceq \Pi x{:}A_i.B_i$ $(i = 1, 2)$. By lemma 3.1.4 and Church-Rosser theorem, $C \rhd \Pi x{:}A_0.B_0$ for some $A_0 \simeq A_i$ and $B_0 \preceq B_i$ $(i = 1, 2)$. By theorems 3.2.7 and 3.2.8, $\Pi x{:}A_0.B_0$ and A_0 are Γ-types. So by rule $(\preceq)$, we have $\Gamma \vdash M_1 : \Pi x{:}A_0.B_0$ and $\Gamma \vdash M_2 : A_0$. Hence, $\Gamma \vdash M_1M_2 : [M_2/x]B_0$. Noticing that $[M_2/x]B_0 \preceq [M_2/x]B_i$ $(i = 1, 2)$, we have the required result. $\Box$

Remark This lemma implies that, if A and B are types of M (under Γ), then $A \approx B$ (see the remark after corollary 3.1.7). It is a sort of 'Church-Rosser property' for types concerned about $\approx$. $\Box$

An immediate consequence of the above diamond property (and the well-foundedness (corollary 3.1.8)) of the cumulativity relation is that every Γ-term has a *minimum type* (under Γ) with respect to the order $\preceq$.

Lemma 3.3.4 (existence of minimum type) *Let M be a Γ-term and $T = \{A \mid \Gamma \vdash M : A\}$. Then, there exists $A \in T$ such that $A \preceq A'$ for all $A' \in T$.*

Proof T is not empty as M is a Γ-term. Let A be a minimal element in T (A exists by corollary 3.1.8). Then, by lemma 3.3.3, for any $A' \in T$, there exists $B \in T$ such that $B \preceq A$ and $B \preceq A'$. Since $B \not\prec A$, we have $A \simeq B \preceq A'$. □

The minimum type of a Γ-term is obviously unique up to conversion. We now show that the minimum type is indeed the *most general* one (principal type).

Definition 3.3.5 (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 3.3.6 (existence of principal type) *Every Γ-term M has a principal type (under Γ); it is the minimum type of M (under Γ) with respect to $\preceq$.*

Proof Let A be the minimum type of M (under Γ) with respect to $\preceq$ (A exists by lemma 3.3.4). Then, $\Gamma \vdash M : A$. For any A' such that $\Gamma \vdash M : A'$, we have $A \preceq A'$ and A' is a Γ-type by theorem 3.2.7. Suppose A' is a Γ-type such that $A \preceq A'$. By corollary 3.3.2, $\Gamma \vdash M : A'$. □

Notation We use $T_\Gamma(M)$ to denote the principal type (being more precise, the set of principal types) of Γ-term M under Γ. □

Remark The above notion of principal type is a nice property of the calculus and is indeed the 'best' one can have when one has type inclusions in a type theory. It yields also a simple and straightforward type inference algorithm as we shall show in section 6.2. Original formulations of universe inclusions by Martin-Löf [ML84] and Coquand [Coq86a] do not lead to such a simple notion of principal type. □

Chapter 4

Quasi Normalization

This chapter is devoted to a proof-theoretic understanding of the predicativity of the type universes $Type_j$ in **ECC**. We prove a *Quasi Normalization* theorem which shows that any well-typed term can be reduced to some *quasi-normal form* which does not contain any σ-redex or any β-redex whose major term has a non-propositional principal type. Besides gaining a better understanding of the calculus, this result has a consequence that every well-typed type can be reduced to some head normal form and allows us to assign a complexity measure to the well-typed types which makes explicit the predicativity (non-circularity) of formations of the non-propositional types. This complexity measure also provides us an important basis to apply Girard-Tait's reducibility method to prove the strong normalization theorem (see chapter 5).

The notion of predicativity dates back to Russell's opinion that logical paradoxes in naive set theory originate from a vicious circle and paradoxes should be eliminated by applying the so-called 'vicious-circle principle': 'Whatever involves all of a collection must not be one of the collection'. This principle in particular prevents from quantification over a collection to form an object of the collection.

In Martin-Löf's type theory, the predicativity seems to be apparent from the very formulation of the type systems [ML73,84]. In particular, there is no way one can form an object of a type (say a universe) by quantifying over the type itself. It is the predicativity of the type theories that enables Martin-Löf to claim the consistency of his type theories in a 'simple-minded' way [ML84].

The polymorphic λ-calculus [Gir72][Rey74] is an impredicative system, in which one is allowed to quantify over type variables to form a new type. It is well-known that a consistency argument for such an impredicative system needs stronger induction principles [Gir72]. The calculus of constructions [CH88][Coq85] extends the second-order λ-calculus to incorporate non-propositional types as well as dependent types. As remarked by Girard [Gir86], 'all attempts to strengthen this system, in particular to temper with the fourth level, should be considered very cautiously'. For example, adding another impredicative level to the calculus of constructions would meet inconsistency [Coq86a].

This last remark implies that the non-propositional types in the calculus of constructions are predicative. Yes, this is obvious. Any non-propositional type in the calculus of constructions is of the form $\Pi x_1{:}A_1 ... x_n{:}A_n.Prop$. In other words, there are no non-propositional type-valued functions and any type of the form MN is a proposition.[1] Therefore, similar to the simple type theory (*c.f.*, [Chu40]), there is a straightforward complexity measure β of types by assigning $\beta(P) = 0$ for proposition P, $\beta(Prop) = 1$ and $\beta(\Pi x{:}A.B) = max\{\beta(A) + 1, \beta(B)\}$ for non-propositional type $\Pi x{:}A.B$. This ranking shows that the formation of non-propositional types depends only on those types with lower ranks; that is, this complexity measure makes explicit that there is no circularity in formations of non-propositional types. As noted by Coquand [Coq86b], the existence of such a complexity measure is essential for the logical consistency of the calculus of constructions and it is impossible to have such a measure for the inconsistent type theory of Martin-Löf with Type:Type [ML71]. Indeed, it is this complexity measure that enables Coquand to succeed in applying Girard-Tait's reducibility method [Gir72][Tai75] to prove the (strong) normalization property of the calculus of constructions [Coq86b].

However, in **ECC** more than one universe exists and the richer type structure makes the predicativity of non-propositional types not so obvious as in the calculus of constructions. That is because there are now functions which have non-propositional types as values. For example, we have $\vdash \lambda x{:}Type_j.x : Type_j \to Type_j$. As a consequence, terms of the form MN (or $\pi_i(M)$) may also be non-propositional types and we do not have the obvious complexity measure shown above for the calculus of constructions. Furthermore, propositions in **ECC** are lifted to higher-level types which allows judgements like $\vdash \Pi x{:}Type_j \Pi B{:}Type_j \to Prop.Bx : Type_j$ to be derivable, although this is only because $\Pi x{:}Type_j \Pi B{:}Type_j \to Prop.Bx$ is a proposition. One may naturally doubt about the predicativity and ask the question: are we sure that there is no circularity in formations of non-propositional types? This raises a problem: How do we show the predicativity of the non-propositional type hierarchy?

Our aim of this chapter is to show that the universes $Type_j$ are still predicative and the formations of non-propositional types are essentially non-circular. This is done by proving a *quasi normalization theorem* (theorem 4.3.13) which implies that every type can be reduced to some head-normal form (corollary 4.3.14) and allows us to define a two-dimensional complexity measure (definition 4.4.2) to make explicit the predicativity of the non-propositional types (lemma 4.4.4).

Section 4.1 introduces a notion of environment, which is a nice tool to deal with type dependency developed by Pottinger [Pot87]. Section 4.2 defines levels of types, which constitute the first dimension of the complexity measure, and studies their properties. The quasi normalization theorem is proved in Section 4.3. Section 4.4 defines the complexity measure.

[1] Note that propositions are not lifted as higher-level types in the calculus of constructions.

4.1 Environment

Because of the nature of dependent types, one needs some tool to deal with the variable bindings occurring in the system. In particular, in a proof of normalization, an infinite 'universal' context is called for and proves to be very useful. In their proofs of (strong) normalization of the calculus of constructions, Coquand [Coq86b] uses a notion of environment of constants and Pottinger [Pot87] a notion of environment of infinite variable bindings.

We follow the idea of Pottinger to introduce below a notion of environment for **ECC** and show that the notions and results relative to valid contexts like those for principal types can all be extended to environments.

Definition 4.1.1 (Environment) *An* environment $\mathcal{E}$ *is an infinite sequence*

$$\mathcal{E} \equiv e_1{:}E_1, e_2{:}E_2, ...$$

where e_i is a variable and E_i is a term, such that, for any $i \in \omega$,

1. *$\mathcal{E}^i \equiv e_1{:}E_1, ..., e_i{:}E_i$ is a valid context, and*

2. *for any $\mathcal{E}^i$-type A, there are infinitely many k such that $E_k \equiv A$.* □

Lemma 4.1.2 (existence of environment) *There exists an environment.*

Proof We construct an environment $\mathcal{E}$ as follows. Assume that we are given a canonical enumeration of variables and a canonical enumeration of derivations in **ECC**. Define $\mathcal{E}^n$ by induction on $n \in \omega$ and, define at the same time a diagonal enumeration $p^n = (p_1^n, p_2^n)$ with the property $p_1^n + p_2^n \leq n$ as follows:

1. $\mathcal{E}^0 =_{\mathrm{df}} \langle\rangle$ (the empty context) and $p^0 = (0, 0)$.

2. Supposing that $\mathcal{E}^i$ for $i < n$ have been defined, define $\mathcal{E}^n$ and p^n as follows. Let T^i ($i < n$) be the sub-sequence of the canonical enumeration of derivations consisting of the derivations of the judgements of the form $\mathcal{E}^i \vdash A : K$ (where K is a kind). If the kth element of T^i is $\mathcal{E}^i \vdash A : K$, we write $T_k^i \equiv A$. (Note that T^i is infinite.) Then, we define $\mathcal{E}^n =_{\mathrm{df}} \mathcal{E}^{n-1}, x{:}T_{p_2^{n-1}}^{p_1^{n-1}}$, where x is the first variable in the canonical enumeration of variables such that $x \notin FV(\mathcal{E}^{n-1})$; and $p^n =_{\mathrm{df}} (p_2^{n-1} + 1, 0)$ if $p_1^{n-1} = 0$, $p^n =_{\mathrm{df}} (p_1^{n-1} - 1, p_2^{n-1} + 1)$ if $p_1^{n-1} \neq 0$. $\mathcal{E}^n$ thus defined is a valid context.

By lemma 3.2.4, it is easy to show that every $\mathcal{E}^n$-type occurs in $\mathcal{E}$ infinitely many times. □

Remark In fact, as shown in [Pot87], one may similarly prove a stronger result which says that every valid context can be extended to an environment. However, the above lemma is enough for our purpose. □

Notation From now on, if not explicitly stated otherwise, $\mathcal{E}$ will stand for a fixed arbitrary environment $e_1{:}E_1, e_2{:}E_2, ...$; that is, $\mathcal{E}_i \equiv e_i{:}E_i$ is the ith component of $\mathcal{E}$ and $\mathcal{E}^i \equiv e_1{:}E_1, ..., e_i{:}E_i$ is the valid context consisting of the first i components of $\mathcal{E}$. □

Most of the notions relative to valid contexts defined before can be similarly defined for environments. First of all, we will write $\mathcal{E} \vdash M : N$ for '$\mathcal{E}^i \vdash M : N$ for some $i \in \omega$'. A term M is called an *$\mathcal{E}$-term*, *$\mathcal{E}$-type*, *$\mathcal{E}$-proposition*, *non-propositional* (or *proper*) *$\mathcal{E}$-type* and *$\mathcal{E}$-proof* if and only if M is an $\mathcal{E}^i$-term, $\mathcal{E}^i$-type, $\mathcal{E}^i$-proposition, non-propositional $\mathcal{E}^i$-type and $\mathcal{E}^i$-proof for some $i \in \omega$, respectively. It is obvious from the definition of environments and lemma 3.2.4 that, if $\mathcal{E}^i \vdash M : A$, then $\mathcal{E}^k \vdash M : A$ for all $k \geq i$; if $Qx{:}M.N$ is an $\mathcal{E}$-term, where $Q \in \{\lambda, \Pi, \Sigma\}$, then there exist $\mathcal{E}$-terms x' and N' such that $Qx{:}M.N \equiv Qx'{:}M.N'$.

The notion of principal type (definition 3.3.5) and its existence (theorem 3.3.6) can also be extended to environment. The notion of principal type under environment $\mathcal{E}$ is defined as in definition 3.3.5 by replacing Γ by $\mathcal{E}$. Corollary 3.3.2, lemma 3.3.3 and lemma 3.3.4 can be proved for environment, and so is theorem 3.3.6.[2] The principal type of an $\mathcal{E}$-term M (under $\mathcal{E}$) is denoted as $T_{\mathcal{E}}(M)$.

4.2 Levels of Types

Now, we define the notion of levels of $\mathcal{E}$-types which will be the first dimension of our complexity measure to be defined in section 4.4. Intuitively, that the level of $\mathcal{E}$-type A is j means that $Type_j$ ($Prop$ when $j = -1$) is the lowest universe in which A resides up to conversion.

Definition 4.2.1 (levels of $\mathcal{E}$-types) *The* level *of an $\mathcal{E}$-type A, $\mathcal{L}(A)$, is defined as follows:*

- *If A is an $\mathcal{E}$-proposition, then $\mathcal{L}(A) =_{\mathrm{df}} -1$.*
- *If A is not an $\mathcal{E}$-proposition, then $\mathcal{L}(A) =_{\mathrm{df}} \mu j.(\exists B.\ B \simeq A \wedge \mathcal{E} \vdash B : Type_j)$, i.e., the minimum $j \in \omega$ such that $\mathcal{E} \vdash B : Type_j$ for some $B \simeq A$.*

□

Remark We have, for every $\mathcal{E}$-type A, $\mathcal{L}(A) = j$ for exactly one $j \in \omega \cup \{-1\}$. $\mathcal{L}(A) \geq 0$ ($\mathcal{L}(A) = -1$) if and only if A is a non-propositional $\mathcal{E}$-type ($\mathcal{E}$-proposition). □

Some properties about levels of $\mathcal{E}$-types are stated as the lemmas below.

[2] In fact, unlike the situation of finite contexts, the inverse of corollary 3.3.2 is also true for environment, because every $\mathcal{E}$-type is inhabited under $\mathcal{E}$ and there are infinite variables inhabiting it.

Lemma 4.2.2 *Let A and B be $\mathcal{E}$-types.*

1. *If $A \simeq B$, then $\mathcal{L}(A) = \mathcal{L}(B)$.*
2. *If $A \preceq B$, then $\mathcal{L}(A) \leq \mathcal{L}(B)$.*

Proof The first statement is obvious from the definition of levels. The second is proved by induction on i for $A \preceq_i B$, using the inductive definition 3.1.3 of $\preceq$. If $A \preceq_0 B$, it is obvious from the definition of levels and the first statement. Consider $A \preceq_{i+1} B$. There are three cases.

1. $A \preceq_i B$;
2. $A \simeq \Pi x{:}A_1.A_2$ and $B \simeq \Pi x{:}B_1.B_2$ for some $A_1 \simeq B_1$ and $A_2 \preceq_i B_2$; or
3. $A \simeq \Sigma x{:}A_1.A_2$ and $B \simeq \Sigma x{:}B_1.B_2$ for some $A_1 \preceq_i B_1$ and $A_2 \preceq_i B_2$.

For the first case, $\mathcal{L}(A) \leq \mathcal{L}(B)$ by induction hypothesis. For the second, we have by the first statement and induction hypothesis that $\mathcal{L}(A_1) = \mathcal{L}(B_1)$ and $\mathcal{L}(A_2) \leq \mathcal{L}(B_2)$. Noticing that, for any $\mathcal{E}$-type $\Pi x{:}C.D$,

$$\mathcal{L}(\Pi x{:}C.D) = \begin{cases} -1 & \text{if } D \text{ is an } \mathcal{E}\text{-proposition} \\ max\{\mathcal{L}(C), \mathcal{L}(D)\} & \text{otherwise} \end{cases}$$

we have $\mathcal{L}(A) = \mathcal{L}(\Pi x{:}A_1.A_2) \leq \mathcal{L}(\Pi x{:}B_1.B_2) = \mathcal{L}(B)$. For the third case, it is similar by noticing that $\mathcal{L}(\Sigma x{:}C.D) = max\{\mathcal{L}(C), \mathcal{L}(D), 0\}$. □

Since convertible $\mathcal{E}$-types have the same level, we use $\mathcal{L}(T_\mathcal{E}(M))$ to denote the level of the principal type of M under $\mathcal{E}$.

Lemma 4.2.3 *If $\mathcal{E}^k \vdash N : E_{k+1}$ and B is an $\mathcal{E}^{k+1}$-type, then $\mathcal{L}([N/e_{k+1}]B) \leq \mathcal{L}(B)$.*

Proof Suppose $\mathcal{L}(B) < \mathcal{L}([N/e_{k+1}]B) = j$. Then, there is B' convertible to B such that $\mathcal{E} \vdash B' : K$ for some kind $K \prec Type_j$. By Church-Rosser theorem, theorem 3.2.8 and lemma 3.2.9, we may assume $\mathcal{E}^{k+1} \vdash B' : K$. But then, by theorem 3.2.6, $\mathcal{E} \vdash [N/e_{k+1}]B' : K$ which implies $\mathcal{L}([N/e_{k+1}]B) < j$ as $[N/e_{k+1}]B \simeq [N/e_{k+1}]B'$, contradicting the assumption. So, $\mathcal{L}([N/e_{k+1}]B) \leq \mathcal{L}(B)$. □

Remark The above lemma shows that type-preserving substitution does not increase the level of an $\mathcal{E}$-type. In particular, for an $\mathcal{E}$-type $\Pi x{:}A.B$ or $\Sigma x{:}A.B$, if $\mathcal{E} \vdash N : A$, then we can always choose x to be e_{k+1} for some k such that $E_{k+1} \equiv A$, $\mathcal{E}^k \vdash N : A$ and B is an $\mathcal{E}^{k+1}$-type, and hence $\mathcal{L}([N/x]B) \leq \mathcal{L}(B)$. □

Lemma 4.2.4 *If $\mathcal{E}$-term R is of the form MN or $\pi_i(M)$ $(i = 1, 2)$, then $\mathcal{L}(T_\mathcal{E}(R)) \leq \mathcal{L}(T_\mathcal{E}(M))$.*

Proof We prove for the case $R \equiv MN$. The other two cases are similar. As $R \equiv MN$ is an $\mathcal{E}$-term, one of the principal types of M has the form $\Pi x{:}A.B$. Then, we have $\mathcal{E} \vdash R : [N/x]B$. By lemma 4.2.2(2) and lemma 4.2.3, $\mathcal{L}(T_{\mathcal{E}}(R)) \leq \mathcal{L}([N/x]B) \leq \mathcal{L}(B) \leq \mathcal{L}(\Pi x{:}A.B) = \mathcal{L}(T_{\mathcal{E}}M)$. □

Remark The above lemma implies that the level of the principal type of the major term of a redex is not less than that of the principal type of the redex. □

Lemma 4.2.5 *Let $A \equiv \Sigma x{:}A_1.A_2$ ($\Pi x{:}A_1.A_2$) be a non-propositional $\mathcal{E}$-type. Then, $\mathcal{L}(A) = j \in \omega$ if and only if*

1. *$\mathcal{L}(A_1) \leq j$ and $\mathcal{L}(A_2) \leq j$ (for Π-case, also $\mathcal{L}(A_2) \geq 0$), and*
2. *either $\mathcal{L}(A_1) = j$ or $\mathcal{L}(A_2) = j$;*

Proof We prove for $A \equiv \Pi x{:}A_1.A_2$. The case for Σ is similar.

Sufficiency. By condition 1, $\mathcal{L}(A) \leq j$ by applying rule ($\Pi 2$). Suppose $\mathcal{L}(A) < j$. Then $\mathcal{E} \vdash A' : K$ for some $A' \simeq A$ and kind $K \prec Type_j$. By Church-Rosser theorem and theorem 3.2.8, we may assume that $A' \equiv \Pi x{:}A_1'.A_2'$. So, we have $\mathcal{E} \vdash A_i' : K$, and hence, $\mathcal{L}(A_i) = \mathcal{L}(A_i') < j$ ($i = 1, 2$), contradicting condition 2. So, $\mathcal{L}(A) = j \in \omega$.

Necessity. Suppose $\mathcal{L}(A) = j \in \omega$. We have $\mathcal{L}(A_2) \geq 0$ for otherwise, A is an $\mathcal{E}$-proposition. $\mathcal{L}(A_i) \leq j$ ($i = 1, 2$) for otherwise, there would be no A' convertible to A to be typed by $Type_j$. If both $\mathcal{L}(A_i) < j$, there would be an A' convertible to A to be typed by some kind $K \prec Type_j$. □

4.3 The Quasi Normalization Theorem

The ultimate goal of proving the quasi-normalization theorem is to make explicit the predicativity of formations of the non-propositional types. The basic idea to achieve this is to proceed as follows:

1. Quasi normalization: every $\mathcal{E}$-term can be reduced to some term which does not contain any σ-redex or any β-redex whose major term has a non-propositional principal type; and this implies
2. every $\mathcal{E}$-type can be reduced to some head-normal form; and this allows us to define
3. the degrees of $\mathcal{E}$-types which serve as the second dimension of the complexity measure to be defined.

However, it turns out that the quasi-normalization result can not be directly proved without the help of the notion of degrees of types. This problem can be solved by considering the subsystems $\mathbf{ECC}^n$ of $\mathbf{ECC}$. Roughly speaking, $\mathbf{ECC}^n$ is the type system got from $\mathbf{ECC}$ by 'cutting off' the infinite universes at the nth level. It can be readily proved that the non-propositional types at the highest level (*i.e.*, nth level) of $\mathbf{ECC}^n$ have head-normal forms (lemma 4.3.2); and then their degrees can be defined. Based on this, we can prove the quasi normalization result for $\mathbf{ECC}^n$ by induction from n to 0, as shown in subsection 4.3.2. Then, the quasi-normalization theorem for **ECC** follows by a global induction on $n \in \omega$.

Before proceeding to give the inductive proof, we first introduce a notion of base term, which is one of the basic forms of the head normal forms mentioned above and is also used in the definition of saturated sets and the proof of the strong normalization in the next chapter.

Definition 4.3.1 (base terms) Base terms *and the* key variable *of a base term are inductively defined on the structure of terms as follows:*

- *A variable is a base term and is the key variable of itself;*
- *If M is a base term, so are MN, $\pi_1(M)$ and $\pi_2(M)$, and their key variable is that of M.* □

Examples of base terms are: x, $xM_1...M_n$, $\pi_i(xM_1...M_n)$, $\pi_i(\pi_k(x)M)N$, *etc.*. x is the key variable of the base terms in these examples.

Remark Note that base terms have the following properties, which can be readily proved by induction on the structure of base terms:

1. If M is a base term and $M \rhd M'$, then M' is a base term, too.
2. If variable y is different from the key variable of a base term M, then $[N/y]M$ is also a base term, where N is an arbitrary term. □

4.3.1 $\mathbf{ECC}^n$

$\mathbf{ECC}^n$ is defined as follows. The underlying term calculus of $\mathbf{ECC}^n$ is the same as that of **ECC** except that the constants $Type_{n+k+1}$ ($k \in \omega$) are removed. The inference rules of $\mathbf{ECC}^n$ are the same as those for **ECC** except that we add the following side conditions:

- $0 \leq j \leq n$ for rules $(C)(\Pi 2)(\Sigma)(pair)(\preceq)$,
- $0 \leq j < n$ for rule (T), and
- $B \not\equiv Type_n$ for rule (λ).

and a new rule for lifting types at lower levels to the nth level:

$$(Type_n) \qquad \frac{\Gamma \vdash M : K}{\Gamma \vdash M : Type_n} \quad (K \prec Type_n \textit{ is a kind})$$

In particular, $\mathbf{ECC}^0$ is the calculus of constructions extended by Σ-types and the inclusion of propositions as types [Luo88a]. Informally, we can describe the relationship of $\mathbf{ECC}^n$ with $\mathbf{ECC}$ as follows:

$$\mathbf{ECC} = \bigcup_{n\in\omega} \mathbf{ECC}^n$$

As any derivation is finite, it can easily be proved by induction on derivations that a sequence of judgements is a derivation in $\mathbf{ECC}$ if and only if it is a derivation in $\mathbf{ECC}^n$ for some $n \in \omega$. All of the notions we have defined for $\mathbf{ECC}$ before are defined in the same way for $\mathbf{ECC}^n$.

Remark It is easy to show that, in $\mathbf{ECC}^n$, if $\Gamma \vdash M : A$, then $Type_n$ does not occur in M or Γ, and either $A \equiv Type_n$ or $Type_n$ does not occur in A. (Note that $Type_n$ is not a Γ-type or $\mathcal{E}$-type in $\mathbf{ECC}^n$.) All of the theorems and lemmas in chapter 3 can be similarly proved except that theorem 3.2.7 for $\mathbf{ECC}^n$ has an assumption that $A \not\equiv Type_n$. The notion of principal type for $\mathbf{ECC}^n$ is defined by changing the second clause of definition 3.3.5 to the following clause:

- for any term A', $\Gamma \vdash M : A'$ if and only if $A \preceq A'$ and either A' is a Γ-type or $A' \equiv Type_n$. $\Box$

The following lemma shows that the top-level $\mathcal{E}$-types in $\mathbf{ECC}^n$ have head-normal forms which will enable us to establish the basis of the induction proof of quasi normalization.

Lemma 4.3.2 *Let A be an $\mathcal{E}$-type in* $\mathbf{ECC}^n$ *and $\mathcal{L}(A) = n$. Then, either $A \equiv Type_{n-1}$ (Prop when $n = 0$) or A has the form of $\Pi x{:}A_1.A_2$ or $\Sigma x{:}A_1.A_2$.*

Proof By induction on derivations of $\mathcal{E}^i \vdash A : Type_n$ in $\mathbf{ECC}^n$. $\Box$

4.3.2 An inductive proof of quasi normalization

Now, we are ready to prove the quasi-normalization by considering $\mathbf{ECC^n}$ for an arbitrary $\mathbf{n} \in \omega$. In the rest of this section, $\mathbf{n}$ stands for a fixed (arbitrary) natural number for which $\mathbf{ECC^n}$ is under consideration.

The quasi-normalization result for $\mathbf{ECC^n}$ is proved by induction from $\mathbf{n}$ and downwards. In other words, the following definitions, lemmas and theorems in the rest of this section are inductively defined and proved for $\mathbf{j} = \mathbf{n}, \mathbf{n}-1, ..., 0$. The general steps are summarized as follows:

1. **j**=**n**:

   (a) Define the **n**-degree $\mathcal{D}_\mathbf{n}$ of $\mathcal{E}$-types (definition 4.3.3 for **j**=**n**), which is well-defined by lemma 4.3.2 and Church-Rosser theorem, and prove properties about $\mathcal{D}_\mathbf{n}$ (lemma 4.3.5 and lemma 4.3.6 for **j**=**n**);

(b) Define measures $\delta_{\mathbf{n}}$, $\gamma_{\mathbf{n}}$ and the notion of $\mathbf{n}$-quasi-normal term (definition 4.3.7 for $\mathbf{j}$=$\mathbf{n}$), and prove another two measure properties (lemma 4.3.8 and lemma 4.3.9 for $\mathbf{j}$=$\mathbf{n}$);

(c) Prove the quasi-normalization result for the $\mathbf{n}$th level (theorem 4.3.10 and corollary 4.3.11 for $\mathbf{j} = \mathbf{n}$).

2. $\mathbf{j} = k < \mathbf{n}$:

(a) Define the k-degree $\mathcal{D}_k$ of $\mathcal{E}$-types (definition 4.3.3 for $\mathbf{j} = k$), which is well-defined by theorem 4.3.10 and corollary 4.3.11 for $\mathbf{j} = k + 1$ and Church-Rosser theorem, and prove properties about $\mathcal{D}_k$ (lemma 4.3.5 and lemma 4.3.6 for $\mathbf{j} = k$);

(b) Define measures δ_k, γ_k and the notion of k-quasi-normal term (definition 4.3.7 for $\mathbf{j}$=k), and prove another two measure properties (lemma 4.3.8 and lemma 4.3.9 for $\mathbf{j}$=k);

(c) Prove the quasi-normalization result for the kth level (theorem 4.3.10 and corollary 4.3.11 for $\mathbf{j} = k$).

Definition 4.3.3 (j-degree $\mathcal{D}_{\mathbf{j}}$ of $\mathcal{E}$-types in $\mathbf{ECC^n}$) *The* $\mathbf{j}$-degree $\mathcal{D}_{\mathbf{j}}(A)$ *of an $\mathcal{E}$-type A in* $\mathbf{ECC^n}$ *is defined as follows.*

- $\mathcal{D}_{\mathbf{j}}$ *for the $\mathcal{E}$-types A°, which are i-quasi-normal for $i > \mathbf{j}$, is defined as follows:*
  1. *If* $\mathcal{L}(A^\circ) \neq \mathbf{j}$, *then* $\mathcal{D}_{\mathbf{j}}(A^\circ) =_{\mathrm{df}} 0$*;*
  2. *If* $A^\circ \equiv Type_{\mathbf{j}-1}$ *(Prop when* $\mathbf{j} = 0$*), then* $\mathcal{D}_{\mathbf{j}}(A^\circ) =_{\mathrm{df}} 1$*;*
  3. *If* $\mathcal{L}(A^\circ) = \mathbf{j}$ *and A° is a base term, then* $\mathcal{D}_{\mathbf{j}}(A^\circ) =_{\mathrm{df}} 1$*;*
  4. *If* $\mathcal{L}(A^\circ) = \mathbf{j}$, *and* $A^\circ \equiv Qx{:}A_1^\circ.A_2^\circ$, *where* $Q \in \{\Pi, \Sigma\}$, *then* $\mathcal{D}_{\mathbf{j}}(A^\circ) =_{\mathrm{df}} max\{\mathcal{D}_{\mathbf{j}}(A_1^\circ), \mathcal{D}_{\mathbf{j}}(A_2^\circ)\} + 1$.
- *If $\mathcal{E}$-type A is not k-quasi-normal for some $k > \mathbf{j}$, then, letting A° be some i-quasi-normal term for $i > \mathbf{j}$ such that $A \rhd A^\circ$, define* $\mathcal{D}_{\mathbf{j}}(A) =_{\mathrm{df}} \mathcal{D}_{\mathbf{j}}(A^\circ)$.

□

Note that the definition above is a 'two-step' definition. As quasi-normal forms are in general not unique, we must show that the definition is well-defined, *i.e.*, the arbitrary choice of A° in the second part of the above definition gives unique degree value. When $\mathbf{j} = \mathbf{n}$, it is well-defined by lemma 4.3.2 and Church-Rosser theorem. For $\mathbf{j} < \mathbf{n}$, it is well-defined by theorem 4.3.10 (for $\mathbf{j} + 1$), corollary 4.3.11 (for $\mathbf{j} + 1$) and Church-Rosser theorem.

Lemma 4.3.4 (well-definedness of j-degree) $\mathcal{D}_{\mathbf{j}}$ *is a function from the $\mathcal{E}$-types of* $\mathbf{ECC^n}$ *to natural numbers and respects conversion, i.e.,* $\mathcal{D}_{\mathbf{j}}A = \mathcal{D}_{\mathbf{j}}B$ *if* $A \simeq B$ *are $\mathcal{E}$-types.*

Proof We consider two cases.

- $\mathbf{j} = \mathbf{n}$. $\mathcal{D}_\mathbf{n}$ is well-defined by lemma 4.3.2.

  Suppose $A \simeq B$ are $\mathcal{E}$-types. Then, $\mathcal{L}(A) = \mathcal{L}(B) = k$ for some k, by lemma 4.2.2. If $k < \mathbf{n}$, then $\mathcal{D}_\mathbf{j}A = \mathcal{D}_\mathbf{j}B = 0$. If $k = \mathbf{n}$, we show $\mathcal{D}_\mathbf{n}A = \mathcal{D}_\mathbf{n}B$ by induction on the structure of A and B. By lemma 4.3.2 and Church-Rosser theorem, either $A \equiv B \equiv Type_{\mathbf{n}-1}$ ($Prop$ when $\mathbf{n} = 0$) or $A \equiv Qx{:}A_1.A_2$ and $B \equiv Qx{:}B_1.B_2$ for some $A_i \simeq B_i$ $(i = 1, 2)$, where $Q \in \{\Pi, \Sigma\}$. The former case is obvious. For the latter case, as $\mathcal{L}(A_i) = \mathcal{L}(B_i)$, $\mathcal{D}_\mathbf{n}(A_i) = \mathcal{D}_\mathbf{n}(B_i)$ by induction hypothesis. Hence, $\mathcal{D}_\mathbf{n}(A) = max\{\mathcal{D}_\mathbf{n}(A_1), \mathcal{D}_\mathbf{n}(A_2)\} + 1 = max\{\mathcal{D}_\mathbf{n}(B_1), \mathcal{D}_\mathbf{n}(B_2)\} + 1 = \mathcal{D}_\mathbf{n}B$.

- $\mathbf{j} < \mathbf{n}$. We consider the following two cases in the sequel.

  1. First, we consider $\mathcal{E}$-types A° which are i-quasi-normal for $i > \mathbf{j}$. $\mathcal{D}_\mathbf{j}(A^\circ)$ is well-defined by theorem 4.3.10 (for $\mathbf{j}+1$) and corollary 4.3.11 (for $\mathbf{j}+1$).

     Suppose $A^\circ \simeq B^\circ$ are $\mathcal{E}$-types which are i-quasi-normal for $i > \mathbf{j}$. Then, $\mathcal{L}(A^\circ) = \mathcal{L}(B^\circ) = k$ for some k, by lemma 4.2.2. If $k \neq \mathbf{j}$, then $\mathcal{D}_\mathbf{j}(A^\circ) = \mathcal{D}_\mathbf{j}(B^\circ) = 0$. If $k = \mathbf{j}$, we show $\mathcal{D}_\mathbf{j}(A^\circ) = \mathcal{D}_\mathbf{j}(B^\circ)$ by induction on the structure of A° and B°. By theorem 4.3.10 (for $\mathbf{j}+1$), corollary 4.3.11 (for $\mathbf{j}+1$) and Church-Rosser theorem, either $A^\circ \equiv B^\circ \equiv Type_{\mathbf{j}-1}$ ($Prop$ when $\mathbf{j} = 0$), or both A° and B° are base terms, or $A^\circ \equiv Qx{:}A_1^\circ.A_2^\circ$ and $B^\circ \equiv Qx{:}B_1^\circ.B_2^\circ$ for some $A_i^\circ \simeq B_i^\circ$ $(i = 1, 2)$, where $Q \in \{\Pi, \Sigma\}$. The former two cases are obvious. For the latter case, as $\mathcal{L}(A_i^\circ) = \mathcal{L}(B_i^\circ)$, $\mathcal{D}_\mathbf{j}(A_i^\circ) = \mathcal{D}_\mathbf{j}(B_i^\circ)$ by induction hypothesis. Hence, $\mathcal{D}_\mathbf{j}(A^\circ) = max\{\mathcal{D}_\mathbf{j}(A_1^\circ), \mathcal{D}_\mathbf{j}(A_2^\circ)\} + 1 = max\{\mathcal{D}_\mathbf{j}(B_1^\circ), \mathcal{D}_\mathbf{j}(B_2^\circ)\} + 1 = \mathcal{D}_\mathbf{j}(B^\circ)$.

  2. Now, consider $\mathcal{E}$-types A which are not k-quasi-normal for some $k > \mathbf{j}$. If $A \rhd A^\circ$ and $A \rhd A^{\circ\prime}$, where A° and $A^{\circ\prime}$ are i-quasi-normal for $i > \mathbf{j}$, then, by the result above, $\mathcal{D}_\mathbf{j}(A^\circ) = \mathcal{D}_\mathbf{j}(A^{\circ\prime})$ as $A^\circ \simeq A^{\circ\prime}$. So, $\mathcal{D}_\mathbf{j}(A)$ is uniquely well-defined.

     Suppose $A \simeq B$ are $\mathcal{E}$-types. Then, by theorem 4.3.10 (for $\mathbf{j}+1$), $A \rhd A^\circ$ and $B \rhd B^\circ$ for some A° and B° which are i-quasi-normal for $i > \mathbf{j}$. Then, by the above result, $\mathcal{D}_\mathbf{j}(A) = \mathcal{D}_\mathbf{j}(A^\circ) = \mathcal{D}_\mathbf{j}(B^\circ) = \mathcal{D}_\mathbf{j}(B)$.

This completes the proof of the lemma. □

Remark Note that Church-Rosser theorem is used to show the well-definedness of degrees. One may understand this in the following way: although there may be different A_1° and A_2° which are i-quasi-normal for $i > \mathbf{j}$ such that $A \rhd A_k^\circ$, there is another A° which is i-quasi-normal for $i > \mathbf{j}$ such that $A_k^\circ \rhd A^\circ$. □

Lemma 4.3.5 *Let A and B be $\mathcal{E}$-types.*

1. *$\mathcal{L}(A) = \mathbf{j}$ if and only if $\mathcal{D}_{\mathbf{j}}A \geq 1$.*

2. *If $A \preceq B$, then either $\mathcal{L}(A) < \mathcal{L}(B)$, or $\mathcal{L}(A) = \mathcal{L}(B)$ and $\mathcal{D}_{\mathbf{j}}A \leq D_{\mathbf{j}}B$.*

Proof The first statement is obvious from the definition of degrees, lemma 4.2.2 and lemma 4.3.4. For the second, by lemma 4.2.2 and the definition of $\mathcal{D}_{\mathbf{j}}$, we only have to show, $\mathcal{D}_{\mathbf{j}}A \leq \mathcal{D}_{\mathbf{j}}B$ if $A \preceq B$ and $\mathcal{L}(A) = \mathcal{L}(B) = \mathbf{j}$.

First consider the case when both A and B are i-quasi-normal for $i > \mathbf{j}$. We prove by induction on the structure of A and B. By corollary 4.3.11 (for $\mathbf{j}+1$), lemma 3.1.4 and Church-Rosser theorem, there are the following possibilities:

1. Both A and B are $Type_{j-1}$ ($Prop$ when $\mathbf{j} = 0$), or both are base terms. Then, $\mathcal{D}_{\mathbf{j}}(A) = \mathcal{D}_{\mathbf{j}}(B) = 1$.

2. $A \equiv \Pi x{:}A_1.A_2$ and $B \equiv \Pi x{:}B_1.B_2$, where $A_1 \simeq B_1$ and $A_2 \preceq B_2$. Then, $\mathcal{D}_{\mathbf{j}}(A_1) = \mathcal{D}_{\mathbf{j}}(B_1)$ by lemma 4.3.4. As $A_2 \preceq B_2$, $\mathcal{L}(A_2) \leq \mathcal{L}(B_2)$ by lemma 4.2.2. Noticing that $\mathcal{L}(B_2) \leq \mathbf{j}$ (because $\mathcal{L}(B) = \mathbf{j}$), we have $\mathcal{D}_{\mathbf{j}}(A_2) \leq \mathcal{D}_{\mathbf{j}}(B_2)$ by the definition of degrees and induction hypothesis. Hence, $\mathcal{D}_{\mathbf{j}}(A) = max\{\mathcal{D}_{\mathbf{j}}(A_1), \mathcal{D}_{\mathbf{j}}(A_2)\} + 1 \leq max\{\mathcal{D}_{\mathbf{j}}(B_1), \mathcal{D}_{\mathbf{j}}(B_2)\} + 1 = \mathcal{D}_{\mathbf{j}}(B)$.

3. $A \equiv \Sigma x{:}A_1.A_2$ and $B \equiv \Sigma x{:}B_1.B_2$, where $A_1 \preceq B_1$ and $A_2 \preceq B_2$. Similar to the above case.

Now, for arbitrary $\mathcal{E}$-types $A \preceq B$, by theorem 4.3.10 (for $\mathbf{j}+1$), there are $\mathcal{E}$-types A° and B° which are i-quasi-normal for $i > \mathbf{j}$ such that $A \rhd A^\circ$ and $B \rhd B^\circ$. As $A^\circ \simeq A \preceq B \simeq B^\circ$, we have, by lemma 4.3.4 and the result above, $\mathcal{D}_{\mathbf{j}}(A) = \mathcal{D}_{\mathbf{j}}(A^\circ) \leq \mathcal{D}_{\mathbf{j}}(B^\circ) = \mathcal{D}_{\mathbf{j}}(B)$. □

Lemma 4.3.6 *Suppose $\mathcal{E}^k \vdash N : E_{k+1}$ and B is an $\mathcal{E}^{k+1}$-type. If $\mathcal{L}(E_{k+1}) \leq \mathbf{j}$ and $\mathcal{L}(B) \leq \mathbf{j}$, then $\mathcal{D}_{\mathbf{j}}([N/e_{k+1}]B) \leq \mathcal{D}_{\mathbf{j}}(B)$.*

Proof By theorem 4.3.10 (for $\mathbf{j}+1$), $B \rhd B'$ for some B' which is i-quasi-normal for $i > \mathbf{j}$ ($B' \equiv B$ when $\mathbf{j} = \mathbf{n}$). As $\mathcal{D}_{\mathbf{j}}B = \mathcal{D}_{\mathbf{j}}B'$ and $\mathcal{D}_{\mathbf{j}}[N/e_{k+1}]B = \mathcal{D}_{\mathbf{j}}[N/e_{k+1}]B'$ by lemma 4.3.4, we only have to show $\mathcal{D}_{\mathbf{j}}[N/e_{k+1}]B' \leq \mathcal{D}_{\mathbf{j}}B'$. We prove this by induction on the structure of B'. By theorem 4.3.11 for $\mathbf{j}+1$ (lemma 4.3.2 when $\mathbf{j} = \mathbf{n}$) and lemma 4.2.3, we only have to consider the following cases assuming $\mathcal{L}([N/e_{k+1}]B') = \mathcal{L}(B') = \mathbf{j}$:

1. $B' \equiv Type_{\mathbf{j}-1}$. Obvious.

2. B' is a base term. Let y be the key variable of B'. If $y \not\equiv e_{k+1}$, then $[N/e_{k+1}]B'$ is also a base term and, by definition $\mathcal{D}_{\mathbf{j}}[N/e_{k+1}]B' = \mathcal{D}_{\mathbf{j}}B' = 1$. But y can not be e_{k+1}, for otherwise, by lemma 4.2.4, $\mathcal{L}(T_{\mathcal{E}}(e_{k+1})) \geq \mathcal{L}(T_{\mathcal{E}}B') = \mathbf{j}+1$, contradicting with the assumption that $\mathcal{L}(E_{k+1}) \leq \mathbf{j}$.

3. B' has the form $\Pi x{:}B_1.B_2$ or $\Sigma x{:}B_1.B_2$. Then, by induction hypothesis, we have $\mathcal{D}_{\mathbf{j}}[N/e_{k+1}]B' = max\{\mathcal{D}_{\mathbf{j}}[N/e_{k+1}]B_1, \mathcal{D}_{\mathbf{j}}[N/e_{k+1}]B_2\} + 1 \leq max\{\mathcal{D}_{\mathbf{j}}B_1, \mathcal{D}_{\mathbf{j}}B_2\} + 1 = \mathcal{D}_{\mathbf{j}}B'$. □

Remark The above lemma shows that type-preserving substitution does not increase the $\mathbf{j}$-degree of an $\mathcal{E}$-type B if the levels of B and the principal type of the substituted variable are not bigger than $\mathbf{j}$. (*c.f.*, remark after lemma 4.2.3.) Note that the condition $\mathcal{L}(E_{k+1}) \leq j$ is necessary and important (*c.f.*, proof of lemma 4.4.4). $\Box$

As convertible $\mathcal{E}$-types have the same $\mathbf{j}$-degree, we use the notation $\mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}(M))$ to express the $\mathbf{j}$-degree of the principal type of an $\mathcal{E}$-term M. Let $\mathcal{E}$-term R be a redex. We define $\delta_{\mathbf{j}}R$ to be the $\mathbf{j}$-degree of the principal type of its major term; that is, if M is the major term of redex R (*i.e.*, $\mathcal{E}$-term R is a redex of the form MN or $\pi_i(M)$),

$$\delta_{\mathbf{j}}R =_{\mathrm{df}} \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}M)$$

For any $\mathcal{E}$-term M, we define $\gamma_{\mathbf{j}}M$ to be the largest $\delta_{\mathbf{j}}$-value of the redexes occurring in M; that is,

$$\gamma_{\mathbf{j}}M =_{\mathrm{df}} max\{\,\delta_{\mathbf{j}}(R) \mid R \textit{ is a redex occurring in } M\,\}$$

These measures are extensions of the measures used by Pottinger and Seldin [Pot87]. They are essentially in the same spirit as that used in [Pra65] for higher-order logic, but more complex.

Definition 4.3.7 (j-quasi-normal $\mathcal{E}$-terms) *An $\mathcal{E}$-term M is* $\mathbf{j}$-quasi-normal *if and only if $\gamma_{\mathbf{j}}M = 0$, i.e., M does not contain any redex such that the level of the principal type of its major term is* $\mathbf{j}$. $\Box$

The aim of quasi-normalization at the $\mathbf{j}$th level is to show that every $\mathcal{E}$-term can be reduced to a term which is i-quasi-normal for every i such that $\mathbf{j} \leq i \leq \mathbf{n}$. We first prove two lemmas about the measures we have defined.

Lemma 4.3.8 *Suppose $\mathcal{E}^k \vdash N : E_{k+1}$, $\mathcal{E}^{k+1} \vdash M : B$ and M is i-quasi-normal for $i > \mathbf{j}$. Then,*

$$\gamma_{\mathbf{j}}([N/e_{k+1}]M) \leq max\{\gamma_{\mathbf{j}}M, \gamma_{\mathbf{j}}N, \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}N)\}$$

Proof By induction on the structure of M.

1. M is a kind or variable. Obvious.

2. M has the form $\Pi y{:}M_1.M_2$, $\lambda y{:}M_1.M_2$ or $\Sigma y{:}M_1.M_2$. Then, by induction hypothesis,

$$\begin{array}{rl}
& \gamma_{\mathbf{j}}([N/e_{k+1}]M) \\
= & max\{\,\delta_{\mathbf{j}}(R) \mid R \text{ is a redex in } [N/e_{k+1}]M)\,\} \\
= & max\{\,\delta_{\mathbf{j}}(R) \mid R \text{ is a redex in } [N/e_{k+1}]M_1 \text{ or } [N/e_{k+1}]M_2\,\} \\
= & max\{\gamma_{\mathbf{j}}([N/e_{k+1}]M_1), \gamma_{\mathbf{j}}([N/e_{k+1}]M_2)\} \\
\leq & max\{max\{\gamma_{\mathbf{j}}(M_1), \gamma_{\mathbf{j}}(N), \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}N)\}, max\{\gamma_{\mathbf{j}}(M_2), \gamma_{\mathbf{j}}(N), \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}N)\}\} \\
= & max\{\gamma_{\mathbf{j}}(M_1), \gamma_{\mathbf{j}}(M_2), \gamma_{\mathbf{j}}(N), \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}N)\} \\
= & max\{\gamma_{\mathbf{j}}M, \gamma_{\mathbf{j}}N, \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}N)\}
\end{array}$$

3. $M \equiv \mathbf{pair}_C(M_1, M_2)$. Then, by induction hypothesis,

$$\begin{array}{rl} & \gamma_{\mathbf{j}}([N/e_{k+1}]M) \\ = & max\{\, \delta_{\mathbf{j}}(R) \mid R \text{ is a redex in } [N/e_{k+1}]M)\,\} \\ = & max\{\, \delta_{\mathbf{j}}(R) \mid R \text{ is a redex in } [N/e_{k+1}]M_1, [N/e_{k+1}]M_2 \text{ or } [N/e_{k+1}]C\,\} \\ = & max\{\gamma_{\mathbf{j}}([N/e_{k+1}]M_1), \gamma_{\mathbf{j}}([N/e_{k+1}]M_2), \gamma_{\mathbf{j}}([N/e_{k+1}]C)\} \\ \leq & max\{max\{\gamma_{\mathbf{j}}(M_1), \gamma_{\mathbf{j}}(N), \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}N)\}, max\{\gamma_{\mathbf{j}}(M_2), \gamma_{\mathbf{j}}(N), \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}N)\}, \\ & \quad max\{\gamma_{\mathbf{j}}(C), \gamma_{\mathbf{j}}(N), \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}N)\}\ \} \\ = & max\{\gamma_{\mathbf{j}}M, \gamma_{\mathbf{j}}N, \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}N)\} \end{array}$$

4. $M \equiv M_1M_2$. If $[N/e_{k+1}]M$ is not a β-redex such that $\delta_{\mathbf{j}}([N/e_{k+1}]M) > 0$, then a similar argument as above cases suffices. Suppose $[N/e_{k+1}]M$ is a β-redex such that $\delta_{\mathbf{j}}([N/e_{k+1}]M) > 0$. Then, by induction hypothesis, we only have to show

$$\delta_{\mathbf{j}}[N/e_{k+1}]M = \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}[N/e_{k+1}]M_1) \leq max\{\gamma_{\mathbf{j}}M, \gamma_{\mathbf{j}}N, \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}N)\}$$

   As $[N/e_{k+1}]M$ is a β-redex, there are two cases to consider:

   (a) $M_1 \equiv e_{k+1}$ and N is of λ-form. Then, $\mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}[N/e_{k+1}]M_1) = \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}N)$.

   (b) M_1 is of λ-form. We only have to show $\mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}[N/e_{k+1}]M_1) \leq \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}M_1)$, as $\mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}M_1) = \delta_{\mathbf{j}}M \leq \gamma_{\mathbf{j}}M$. By theorem 3.2.6, $\mathcal{E}^k \vdash [N/e_{k+1}]M_1 : [N/e_{k+1}]T_{\mathcal{E}}M_1$. So, $T_{\mathcal{E}}[N/e_{k+1}]M_1 \preceq [N/e_{k+1}]T_{\mathcal{E}}M_1$. Furthermore, by the assumption that $\mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}[N/e_{k+1}]M_1) > 0$, lemma 4.2.2, lemma 4.2.3 and the assumption that M is i-quasi-normal for $i > \mathbf{j}$, $\mathbf{j} = \mathcal{L}(T_{\mathcal{E}}[N/e_{k+1}]M_1) \leq \mathcal{L}([N/e_{k+1}]T_{\mathcal{E}}M_1) \leq \mathcal{L}(T_{\mathcal{E}}M_1) \leq \mathbf{j}$, which implies that $\mathcal{L}(T_{\mathcal{E}}[N/e_{k+1}]M_1) = \mathcal{L}([N/e_{k+1}]T_{\mathcal{E}}M_1)$. Hence, by lemma 4.3.5 and induction hypothesis, $\mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}[N/e_{k+1}]M_1) \leq \mathcal{D}_{\mathbf{j}}[N/e_{k+1}]T_{\mathcal{E}}M_1 \leq \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}M_1)$.

5. M has the form $\pi_i(M_1)$ $(i = 1, 2)$. Similar to the above case. □

Lemma 4.3.9 *Let $\mathcal{E}$-term M be a redex and M' be its contractum. If M is i-quasi-normal for $i > \mathbf{j}$ and M is the only redex in M whose $\delta_{\mathbf{j}}$-value equals $\gamma_{\mathbf{j}}M > 0$, then*

1. $\gamma_{\mathbf{j}}M' < \gamma_{\mathbf{j}}M$, *and*

2. $\mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}M') < \gamma_{\mathbf{j}}M$, *if M' is of λ-form or* **pair**-*form.*

Proof Proof of 1. If M is a σ-redex, it is obvious. If $M \equiv (\lambda x{:}A.M_1)N \rhd_1 [N/x]M_1 \equiv M'$, we have by assumptions, $\gamma_{\mathbf{j}}M_1 < \gamma_{\mathbf{j}}M$, $\gamma_{\mathbf{j}}N < \gamma_{\mathbf{j}}M$, and $\mathcal{D}_{\mathbf{j}}A < \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}(\lambda x{:}A.M_1)) = \gamma_{\mathbf{j}}M$. By lemma 4.2.2 and the assumption that M is i-quasi-normal for $i > \mathbf{j}$ and $\gamma_{\mathbf{j}}M > 0$ $\mathcal{L}(T_{\mathcal{E}}N) \leq \mathcal{L}(A) \leq \mathbf{j}$. So, $\mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}N) \leq \mathcal{D}_{\mathbf{j}}A$. So, by lemma 4.3.8, $\gamma_{\mathbf{j}}M' < \gamma_{\mathbf{j}}M$.

Proof of 2. If $M \equiv \pi_i(\mathbf{pair}_C(M_1, M_2))$ is a σ-redex, Then, $\mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}M') = \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}M_i) < \mathcal{D}_{\mathbf{j}}C = \gamma_{\mathbf{j}}M$. If $M \equiv (\lambda x{:}A.M_1)N$ is a β-redex and M' is of λ-form (or **pair**-form), we have two possibilities:

1. $M_1 \equiv x$. Then $\mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}M') = \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}N) \leq \mathcal{D}_{\mathbf{j}}A < \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}(\lambda x{:}A.M_1)) = \gamma_{\mathbf{j}}M$.

2. M_1 is of λ-form (or **pair**-form). Then, by lemma 4.3.5, lemma 4.3.6 and the assumption that $\gamma_{\mathbf{j}}M > 0$, $\mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}M') = \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}[N/x]M_1) \leq \mathcal{D}_{\mathbf{j}}([N/x]T_{\mathcal{E}}M_1) \leq \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}M_1) < \mathcal{D}_{\mathbf{j}}(T_{\mathcal{E}}(\lambda x{:}A.M_1)) = \gamma_{\mathbf{j}}M$. □

Now, we prove the quasi normalization theorem at the **j**th level for $\mathbf{ECC^n}$.

Theorem 4.3.10 *Every $\mathcal{E}$-term in* $\mathbf{ECC^n}$ *can be reduced to some $\mathcal{E}$-term which is i-quasi-normal for every i such that* $\mathbf{j} \leq i \leq \mathbf{n}$.

Proof By our global induction hypothesis (on $\mathbf{j}$), we only have to show that if $\mathcal{E}$-term M is i-quasi-normal for all i such that $\mathbf{j} < i \leq \mathbf{n}$, then $M \rhd N$ for some N which is i-quasi-normal for all i such that $\mathbf{j} \leq i \leq \mathbf{n}$. So, it is enough to prove the following two points:

1. Any $\mathcal{E}$-term M can be reduced to a $\mathbf{j}$-quasi-normal term by contracting σ-redexes and non-proof β-redexes.

2. Reducing σ-redexes and non-proof β-redexes in an $\mathcal{E}$-term preserves i-quasi-normalness for $i > \mathbf{j}$.

The first can be proved by double induction on $\gamma_{\mathbf{j}}M$ and the number of redexes occurring in M whose $\delta_{\mathbf{j}}$-values are equal to $\gamma_{\mathbf{j}}M > 0$. Given an $\mathcal{E}$-term M, take any redex in the term whose $\delta_{\mathbf{j}}$-value is $\gamma_{\mathbf{j}}M > 0$ and whose proper subterms do not contain any redex whose $\delta_{\mathbf{j}}$-value is $\gamma_{\mathbf{j}}M$. (Note that such a redex is not an $\mathcal{E}$-proof if it is a β-redex.) By lemma 4.3.9 above, reducing M by contracting the redex thus selected decreases by one the number of redexes whose $\delta_{\mathbf{j}}$-values are equal to $\gamma_{\mathbf{j}}M > 0$ and, if it is the only redex whose $\delta_{\mathbf{j}}$-value is $\gamma_{\mathbf{j}}M$, $\gamma_{\mathbf{j}}M$ is decreased by one or more.

When $\mathbf{j} = \mathbf{n}$, the second point is trivial. We now prove it for $\mathbf{j} < \mathbf{n}$. By our global induction on $\mathbf{j}$, we only have to show that, if M is i-quasi-normal for $i > \mathbf{j}$ and $M \rhd_1 N$ by contracting a σ-redex or non-proof β-redex, then N is $(\mathbf{j}+1)$-quasi-normal. We prove this by induction on the structure of M.

1. M is not a variable or a kind.

2. M is of the form $\Pi x{:}A.B$, $\lambda x{:}A.B$, $\Sigma x{:}A.B$ or $\mathbf{pair}_A(B, C)$. By induction hypothesis.

3. $M \equiv M_1M_2$. Consider the following three subcases:

   (a) $M \equiv M_1M_2 \rhd_1 M_1N_2 \equiv N$. By induction hypothesis.

   (b) $M \equiv M_1M_2 \rhd_1 N_1M_2 \equiv N$. By induction hypothesis, if N is not $(\mathbf{j}+1)$-quasi-normal, N must be a β-redex ($N_1 \equiv \lambda x{:}X.Y$) such that $\delta_{\mathbf{j}+1}N > 0$. This is impossible as the following shows:

- $M_1 \equiv \lambda x{:}X_1.Y_1 \rhd_1 \lambda x{:}X.Y \equiv N_1$. But then, as $T_{\mathcal{E}}N_1 \preceq T_{\mathcal{E}}M_1$, by lemma 4.3.5(2), either $\mathcal{L}(T_{\mathcal{E}}N_1) \leq \mathcal{L}(T_{\mathcal{E}}M_1) < \mathbf{j}{+}1$ or $\mathcal{D}_{\mathbf{j+1}}(T_{\mathcal{E}}N_1) \leq \mathcal{D}_{\mathbf{j+1}}(T_{\mathcal{E}}M_1) = 0$.
- $M_1 \equiv (\lambda x{:}X_1.Y_1)Z_1 \rhd_1 [Z_1/x]Y_1 \equiv N_1$. Then, either $Y_1 \equiv x$ and $Z_1 \equiv N_1$, or $Y_1 \equiv \lambda y{:}Y_1'.Y_1''$. In either case, by lemma 4.3.5, $\delta_i M_1 > 0$ for some $i \geq \mathbf{j}+1$, which is impossible.
- $M_1 \equiv \pi_i(\mathbf{pair}_A(X_1, X_2) \rhd_1 X_i \equiv N_1$. Then, either $\mathcal{L}(T_{\mathcal{E}}N_1) < \mathcal{L}(A)$ or $\mathcal{D}_{\mathbf{j+1}}(T_{\mathcal{E}}N_1) \leq \mathcal{D}_{\mathbf{j+1}}A = 0$.

(c) $M \equiv M_1M_2 \equiv (\lambda x{:}X.Y)M_2 \rhd_1 [M_2/x]Y \equiv N$. By lemma 4.3.8, $\gamma_{\mathbf{j+1}}N \leq max\{\gamma_{\mathbf{j+1}}Y, \gamma_{\mathbf{j+1}}M_2, \mathcal{D}_{\mathbf{j+1}}(T_{\mathcal{E}}M_2)\}$. By induction hypothesis, if N is not $(\mathbf{j}+1)$-quasi-normal, M_2 must be the major term of a new-created redex in N such that $\mathcal{D}_{\mathbf{j+1}}(T_{\mathcal{E}}M_2) > 0$. However, as $T_{\mathcal{E}}M_2 \preceq X$ and M is not an $\mathcal{E}$-proof , we would have $\delta_i M = \mathcal{D}_i(T_{\mathcal{E}}M_1) \geq \mathcal{D}_i X > 0$ for some $i \geq \mathbf{j}+1$.

4. $M \equiv \pi_i(M')$. Consider the following two subcases:

(a) $M \equiv \pi_i(M') \equiv \pi_i(\mathbf{pair}_A(X_1, X_2)) \rhd_1 X_i \equiv N$. Obvious.

(b) $M \equiv \pi_i(M') \rhd_1 \pi_i(N') \equiv N$. By induction hypothesis, if N is not $(\mathbf{j}+1)$-quasi-normal, it must be the case that $N' \equiv \mathbf{pair}_A(X, Y)$ and $\delta_{\mathbf{j+1}}N = \mathcal{D}_{\mathbf{j+1}}A > 0$. There are only three possibilities:

- $M' \equiv \mathbf{pair}_{A_1}(X_1, Y_1)$. Then, $\delta_{\mathbf{j+1}}N = \mathcal{D}_{\mathbf{j+1}}A = \mathcal{D}_{\mathbf{j+1}}A_1 = \delta_{\mathbf{j+1}}M = 0$.
- $M' \equiv M_0 \equiv (\lambda x{:}X_0.Y_0)Z_0 \rhd_1 [Z_0/x]Y_0 \equiv N'$. Then either (1) $Y_0 \equiv x$ and $Z_0 \equiv N_0$, or (2) $Y_0 \equiv \mathbf{pair}_{A_1}(X_1, Y_1)$. In either case, we have $\delta_i M_0 > 0$ for some $i \geq \mathbf{j}+1$, which is impossible.
- $M' \equiv M_0 \equiv \pi_i(\mathbf{pair}_{A_1}(X_1, X_2)) \rhd_1 X_i \equiv N'$. Then, either $\mathcal{L}(A) \leq \mathcal{L}(A_1) < \mathbf{j}+1$ or $\delta_{\mathbf{j+1}}M' = \mathcal{D}_{\mathbf{j+1}}(A) \leq \mathcal{D}_{\mathbf{j+1}}(A_1) = 0$.

This completes the proof of the theorem. □

Remark In the above proof, the condition that a β-redex to be reduced is not an $\mathcal{E}$-proof is important. Reducing a proof β-redex may create a new redex destroying the quasi-normalness of a term. For example, if $\mathcal{E} \vdash P : Prop$, $\mathcal{E} \vdash z : B \to P$ and $\mathcal{E} \vdash y : A$, we can have $M_0 \equiv (\lambda x{:}A \to B.z(xy))X_0 \rightsquigarrow_\beta z(X_0y)$. As M_0 is a proof, its δ_i-value is always 0. However, X_0 can be of λ-form the level of whose type $A \to B$ may be $i \geq \mathbf{j}$. □

Corollary 4.3.11 *Let A be an $\mathcal{E}$-type which is i-quasi-normal for every i such that $\mathbf{j} \leq i \leq \mathbf{n}$ and $\mathcal{L}(A) = \mathbf{j}-1$. Then, A has the form of*

$$Type_{\mathbf{j}-2} \text{ (Prop when } \mathbf{j}=1), \quad \text{a base term}, \quad \Pi x{:}A_1.A_2 \quad \text{or} \quad \Sigma x{:}A_1.A_2$$

Proof By induction on the structure of A.

1. A is a kind. Then $\mathbf{j} \geq 1$. We have $A \equiv Type_{\mathbf{j}-2}$ ($Prop$ when $\mathbf{j} = 1$) as it is the only kind whose level is $\mathbf{j} - 1$.

2. A is a variable or of Π/Σ-form. It is as required.

3. A can not be of λ-form or **pair**-form, as A is an $\mathcal{E}$-type.

4. $A \equiv A_1 A_2$ or $\pi_i(A_1)$. We show that A is a base term, *i.e.*, A_1 is a base term. A_1 can not be a kind or of λ-form or **pair**-form. (If it is of λ-form or **pair**-form, by lemma 4.2.4, $\mathcal{L}(T_{\mathcal{E}}A_1) \geq \mathcal{L}(T_{\mathcal{E}}A) > \mathcal{L}(A) = \mathbf{j} - 1$, which implies by lemma 4.3.5 that $\delta_i(A) > 0$ for some $i \geq \mathbf{j}$, contradicting with the assumption.) So, A_1 can only be of the form x, $A_{11}A_{12}$ or $\pi_i(A_{11})$. If it is a variable, then A is a base term. If it is of one of the latter two forms, we repeat this argument to prove A_{11} is a base term, ... This will obviously end with a variable case from which we conclude that A is a base term.

$\square$

4.3.3 Quasi normalization: a summary

We summarize the result of quasi-normalization for **ECC** as follows.

Definition 4.3.12 (quasi-normal terms) *An $\mathcal{E}$-term is* quasi-normal *if and only if it does not contain any σ-redex or any β-redex whose major term has a non-propositional principal type.* $\square$

Remark An $\mathcal{E}$-term may have different quasi-normal forms. Arbitrary reduction does not in general preserve quasi-normalness (*c.f.*, the remark after theorem 4.3.10). $\square$

By the inductive proof in section 4.3.2, we have

Theorem 4.3.13 (quasi normalization of ECC) *Every $\mathcal{E}$-term can be reduced to some quasi-normal form.* $\square$

Corollary 4.3.14 (forms of quasi-normal $\mathcal{E}$-types) *Every $\mathcal{E}$-type of* **ECC** *can be reduced to some quasi-normal term of one of the following forms:*

$$\textit{a kind } K, \quad \textit{a base term}, \quad \Pi x{:}A.B, \textit{ or } \Sigma x{:}A.B.$$

Therefore, every $\mathcal{E}$-type can be reduced to a term of the form

$$(*) \qquad Q_1 x_1{:}A_1 \ldots Q_n x_n{:}A_n.B$$

where $n \in \omega$, Q_i is either Π or Σ, B is either a kind or a base term, and A_i is of the same form as $(*)$ above. $\square$

4.4 A Complexity Measure of Types

The quasi-normalization theorem 4.3.13 and its corollary 4.3.14 allow us to define a two-dimensional complexity measure of $\mathcal{E}$-types. First, we define the j-degree of $\mathcal{E}$-types in **ECC**.

Definition 4.4.1 (j-degree $\mathcal{D}_j$) *Let A be an $\mathcal{E}$-type and A° be a quasi normal term such that $A \rhd A^\circ$. Define the* j-degree *of A for $j \in \omega$, $\mathcal{D}_j A$, as follows:*

- *If $\mathcal{L}(A^\circ) \neq j$, then $\mathcal{D}_j A =_{\mathrm{df}} 0$;*
- *If $A^\circ \equiv Type_{j-1}$ (Prop when $j = 0$), then $\mathcal{D}_j A =_{\mathrm{df}} 1$;*
- *If A° is a base term and $\mathcal{L}(A^\circ) = j$, then $\mathcal{D}_j A =_{\mathrm{df}} 1$;*
- *If $A^\circ \equiv \Pi x{:}A_1.A_2$ or $A^\circ \equiv \Sigma x{:}A_1.A_2$, and $\mathcal{L}(A^\circ) = j$, then $\mathcal{D}_j A =_{\mathrm{df}} max\{\mathcal{D}_j A_1, \mathcal{D}_j A_2\} + 1$.*

We also define $\mathcal{D}_{-1} A =_{\mathrm{df}} 0$ for every $\mathcal{E}$-type A. □

Remark The above definition of degrees is well-defined by theorem 4.3.13, corollary 4.3.14 and Church-Rosser theorem. For $j \in \omega$, it is the same as defined in definition 4.3.3; the properties of degrees proved in section 4.3.2 (lemmas 4.3.4, 4.3.5 and 4.3.6 in particular) hold and the proofs are the same using theorem 4.3.13 and corollary 4.3.14. □

Definition 4.4.2 (complexity of $\mathcal{E}$-types, β) *Let A be an $\mathcal{E}$-type. Then define the complexity of A, βA, as follows:*

$$\beta A \ =_{\mathrm{df}} \ (\mathcal{L}(A) + 1,\ \mathcal{D}_{\mathcal{L}(A)} A)$$

where $\mathcal{L}(A)$ is the level of A and $\mathcal{D}_j A$ is the j-degree of A. β-values of $\mathcal{E}$-types are ordered by the lexicographic ordering. □

Lemma 4.4.3 *Let A and B be $\mathcal{E}$-types.*

1. *If $A \simeq B$, then $\beta(A) = \beta(B)$.*
2. *If $A \preceq B$, then $\beta(A) \leq \beta(B)$.*

Proof By lemma 4.2.2 and lemma 4.3.5. □

Lemma 4.4.4 *Let A be a non-propositional $\mathcal{E}$-type. Then, if A reduces to a quasi-normal $\mathcal{E}$-type of the form $\Pi x{:}A_1.A_2$ or $\Sigma x{:}A_1.A_2$, we have*

1. *$\beta(A_1) < \beta(A)$, and*
2. *$\beta([N/x]A_2) < \beta(A)$ for every N such that $\mathcal{E} \vdash N{:}A_1$.*

Proof As $\mathcal{L}(A_1) \leq \mathcal{L}(A)$ and $\mathcal{L}(A) \geq 0$, either $\mathcal{L}(A_1) < \mathcal{L}(A)$ or $\mathcal{L}(A_1) = \mathcal{L}(A)$ and $\mathcal{D}_{\mathcal{L}(A_1)}A_1 < \mathcal{D}_{\mathcal{L}(A)}A$. So, by definition, $\beta(A_1) < \mathcal{L}(A)$.

Suppose $\mathcal{E} \vdash N{:}A_1$. If $\mathcal{L}([N/x]A_2) = -1$, then $\beta([N/x]A_2) = (0,0) < (1,0) \leq \beta A$. If $\mathcal{L}([N/x]A_2) = j \geq 0$, then $\mathcal{L}(A_2) \geq \mathcal{L}([N/x]A_2) = j$ by lemma 4.2.4. There are two cases to consider:

1. $\mathcal{L}(A_2) > \mathcal{L}([N/x]A_2) = j$. Then, $\beta([N/x]A_2) = (j+1, \mathcal{D}_j([N/x]A_2)) < (\mathcal{L}(A_2)+1, \mathcal{D}_{\mathcal{L}(A_2)}A_2) \leq (\mathcal{L}(A)+1, \mathcal{D}_{\mathcal{L}(A)}A) = \beta A$.

2. $\mathcal{L}(A_2) = \mathcal{L}([N/x]A_2) = j$. Then, by lemma 4.3.6, either $\mathcal{L}(A_1) > j$ or $\mathcal{L}(A_1) \leq j$ and $\mathcal{D}_j([N/x]A_2) \leq \mathcal{D}_j A_2$. For the former case, $\beta([N/x]A_2) = (j+1, \mathcal{D}_j([N/x]A_2)) < \beta A_1 < \beta A$; for the latter case, by lemma 4.3.6 and the fact $\mathcal{L}(A) = \mathcal{L}(A_2) = j \geq \mathcal{L}(A_1)$, $\beta([N/x]A_2) = (j+1, \mathcal{D}_j([N/x]A_2)) \leq (\mathcal{L}(A_2)+1, \mathcal{D}_j A_2) = \beta A_2 < \beta A$.

This completes the proof of the lemma. □

Remark The existence of the complexity measure β with the above property shows that the formations of the non-propositional types are essentially non-circular and that the type universes $Type_j$ are predicative. In other words, the types can be ranked in such a way (by β) that the existence of any non-propositional type depends essentially only on those types with lower ranks. This is one of the key properties used to prove strong normalization theorem for **ECC** (see the next chapter). Note that only *non-propositional* types can be stratified to have the above property. For propositions, there is no way one can define such a measure to stratify them because formations of propositions are impredicative (circular). □

Chapter 5

Strong Normalization

In this chapter, we prove the strong normalization theorem for **ECC**:

- *every well-typed term is strongly normalizable*

i.e., every reduction sequence starting from a well-typed term is finite. (Computationally, every program is terminating.) This is the most important property, which implies many important properties of the calculus, including the following:

1. Logical consistency (theorem 6.1.5),
2. Decidability of conversion and the cumulativity relation for well-typed terms (lemma 6.2.1),
3. Decidability of type inference and type-checking (theorem 6.2.3 and corollary 6.2.4), and
4. Equality reflection (theorem 9.1.1).

The strong normalization theorem will be proved by using Girard-Tait's reducibility method [Gir72,89][Tai75]. The proof is based on Coquand's method of proving normalization of the calculus of constructions [Coq86b] and the quasi normalization result proved in the previous chapter.

Section 5.1 gives a general discussion of Girard-Tait's reducibility method. The strong normalization theorem is proved in section 5.2.

In this chapter, $\mathcal{E}$ still stands for a fixed environment as in the previous chapter and the notational conventions for $\mathcal{E}$ introduced in section 4.1 apply.

5.1 Girard-Tait's Reducibility Method

We first discuss in general Girard-Tait's method for normalization proofs. We explain why it is difficult to prove (strong) normalization for type theories with more complicated type structures like the calculus of constructions. In particular, we discuss why the predicativity of higher universes is essential to apply Girard-Tait's method to prove strong normalization for Constructions with more than one universe. Consideration of Σ-types leads us to slightly generalize the key notion of saturated sets into a more transparent definition which we feel would give us a better understanding of the reducibility method.[1]

Girard-Tait's reducibility method [Gir72,89][Tai75] has been well-known and widely used to prove (strong) normalization property of various type systems including the polymorphic λ-calculus [Gir72][Gir89] and the calculus of constructions [Coq85][Coq86b][Pot87]. One can find a nice and rather detailed explication of the method for proving strong normalization of the second-order λ-calculus in [Gal89].

The basic idea of the method came from the fact that a proof of normalization by straightforward induction on term structure fails because β-reduction may result in a term with larger size. A stronger induction method was invented by Tait [Tai67] and generalized to higher-order systems by Girard [Gir71,72][Tai75]; it is very adaptable for different type systems. The general steps of the reducibility method can be analyzed as follows:[2]

1. Define a notion of *saturated sets* or *candidates of reducibility*.

2. Define an interpretation of types A, $Eval_\rho A$, with respect to type variable assignment ρ.

3. Prove that $Eval_\rho A$ is a saturated set (or candidate of reducibility) for every type A.

4. Prove the soundness of the interpretation $Eval$, *i.e.*, if M is of type A, then M is in $Eval_\rho A$.

As every term in a saturated set (or candidate of reducibility) is (strongly) normalizable, by the very definition of saturated sets (or candidates of reducibility), we conclude that every well-typed term is (strongly) normalizable.

The above outline of the reducibility method is rather informal but is enough for understanding our following discussions and also gives a guideline to understand our proof of strong normalization in section 5.2. We now discuss several points we feel interesting when applying the method to richer type systems.

5.1.1 Saturated sets and candidates of reducibility

The core notion of Girard-Tait's reducibility method is that of saturated sets [Tai75] (or candidates of reducibility [Gir71,72]) which are assigned to types of typed λ-calculi in their term model constructions.

[1] The discussion in this section was given in [Luo89c].

[2] We only consider the typed version of reducibility method here. We remark that the erasing trick used in the untyped version of the reducibility method [Tai75][Mit86] does not seem to apply to the calculus of constructions or richer calculi as it is based on *separation* of type reduction and term reduction which may not be done when type-valued functions exist.

Instead of giving the ordinary definition of saturated sets (in which people only consider λ-terms), we give a slightly more general definition using the notions of base term (definition 4.3.1) and key redex (definition 5.1.1 below) which easily incorporates terms for Σ-types (and products as a special case). Our definition makes more explicit the idea behind the notion of saturated sets.

Definition 5.1.1 (key redex) *The notion of* key redex *of a term M is defined as follows:*

1. *If M is a redex, then M* has key redex *and it is the* key redex *of itself.*
2. *If M has key redex, then so do MN, $\pi_1(M)$ and $\pi_2(M)$, and their key redexes are that of M.*

(Thus, a term has at most one key redex.) If M has key redex, we write $\mathrm{red}_{\mathbf{k}}(M)$ *for the term obtained from M by contracting the key redex of M.* □

For example, the redexes $(\lambda x{:}A.M)N$ and $\pi_1(\mathbf{pair}_A(M,N))$ are the key redexes of $(\lambda x{:}A.M)NN_1...N_m$ and $\pi_1(\mathbf{pair}_A(M,N))N_1...N_m$, respectively. The intuitive idea behind the notion of key redex is that every reduction sequence, starting from a term with key redex and ending with a normal form, will necessarily contract the key redex of the term (possibly after contracting some redexes in subterms of the key redex).

Notation Let A be an $\mathcal{E}$-type. Then, $SN(A)$ is the set of strongly normalizable terms M such that $\mathcal{E} \vdash M : A$. □

Definition 5.1.2 (saturated sets) *Let A be an $\mathcal{E}$-type. S is an A-*saturated set *if and only if*

(S1) *$S \subseteq SN(A)$;*

(S2) *if $M \in SN(A)$ is a base term, then $M \in S$;*

(S3) *if $M \in SN(A)$ has key redex and $\mathrm{red}_{\mathbf{k}}(M) \in S$, then $M \in S$.*

$Sat(A)$ is defined to be the set of A-saturated sets. □

Remark $Sat(A)$ is not empty. In fact, $SN(A) \in Sat(A)$. To show the generality of the above definition, we remark that $(S2)$ has the following as special cases:

- If $M \equiv xM_1...M_n \in SN(A)$, then $M \in S$.
- If $M \equiv \pi_{i_1}(...\pi_{i_j}(x)) \in SN(A)$, then $M \in S$, where $i_k \in \{1,2\}$.

and $(S3)$ has the following as special cases:

- if $M \equiv (\lambda x{:}B.M')NN_1...N_m \in SN(A)$ and $([N/x]M')N_1...N_m \in S$, then $M \in S$;
- if $M \equiv \pi_{i_1}(...\pi_{i_j}(\mathbf{pair}_B(M_1, M_2))) \in SN(A)$ and $\pi_{i_1}(...\pi_{i_{j-1}}(M_{i_j})) \in S$, then $M \in S$, where $i_k \in \{1, 2\}$. □

The above definition of saturated sets will be used in this chapter to prove strong normalization of **ECC**. As a digression, before we proceed to discuss the next steps of the reducibility method, we would like to compare the notion of saturated sets with the notion of candidates of reducibility of Girard [Gir72,89] and show that the conditions for the latter are stronger.

Definition 5.1.3 (candidates of reducibility) *([Gir89]) Let A be an $\mathcal{E}$-type. S is an A-*candidate of reducibility *if and only if*

(CR1) *$S \subseteq SN(A)$;*

(CR2) *if $M \in S$ and $M \rhd_1 N$, then $N \in S$;*

(CR3) *if $\mathcal{E} \vdash M : A$, M is simple (i.e., M is of the form x, M_1M_2 or $\pi_i(M')$), and $N \in S$ for every N such that $M \rhd_1 N$, then $M \in S$.*

$CR(A)$ is defined to be the set of A-candidates of reducibility. □

We have the following relationship between saturated sets and Girard's candidates of reducibility.

Proposition $CR(A) \subseteq SAT(A)$, *i.e.*, every A-candidate of reducibility is an A-saturated set.

Proof Suppose that S is an A-candidate of reducibility. We show that S satisfies (S1)(S2)(S3). Notice that, for every $\mathcal{E}$-term M, M is simple if and only if M is a base term or has key redex. We use this fact below tacitly.

(S1) By definition.

(S2) We show that every base term M in $SN(A)$ is in S by induction on the height $h(M)$ of the reduction tree of M. If $h(M) = 0$, *i.e.*, M is strongly normalizable, then $M \in S$ by (CR3). If $h(M) = n + 1$, then, for every N such that $M \rhd_1 N$, $N \in SN(A)$ is a base term and $h(N) < h(M)$; and hence $N \in S$ by induction hypothesis. Therefore, $M \in S$ by (CR3).

(S3) We show, by induction on the height $h(M)$ of the reduction tree of M, that $M \in S$ for every M in $SN(A)$ which has key redex and whose key reduct $\mathrm{red}_{\mathbf{k}}(M)$ is in S. If $h(M) = 1$, then if $M \rhd_1 N$, we have $N \equiv \mathrm{red}_{\mathbf{k}}(M) \in S$. So, $M \in S$ by (CR3). If $h(M) = n + 1$, then if $M \rhd_1 N$, we have either $N \equiv \mathrm{red}_{\mathbf{k}}(M) \in S$ or N has key redex and $\mathrm{red}_{\mathbf{k}}(M) \rhd \mathrm{red}_{\mathbf{k}}(N)$. In the former case, $N \in S$ by assumption. In the latter case, as $\mathrm{red}_{\mathbf{k}}(M) \in S$, $\mathrm{red}_{\mathbf{k}}(N) \in S$ by (CR2); and by induction hypothesis, we have $N \in S$ as $h(N) < h(M)$. Therefore, $M \in S$ by (CR3). □

Remark The condition (CR2) is necessary to prove the above proposition. The converse of the proposition is not true; some saturated sets are not candidates of reducibility because they do not satisfy (CR2). The above relationship between saturated sets and candidates of reducibility has also been noticed by Gallier [Gal89]. □

5.1.2 Separability of types v.s. type-valued functions

Now, we discuss how to define an interpretation $Eval_\rho A$ such that every type is interpreted as a saturated set. This can be done by induction on type structures for simpler systems like the simply typed λ-calculus [Tai67] and the second-order λ-calculus [Gir72][Mit86], because in these systems types are essentially separated from the other objects and there are no type-valued function terms. However, for richer systems like the calculus of constructions [CH88][Coq85], types are mixed up with other terms and can not be simply separated. In particular, there are *type-valued λ-terms* or intuitively type-valued functions. For example, in the calculus of constructions, one has $\vdash \lambda x{:}Prop.x : Prop \to Prop$. Therefore, a term of the form MN may be a type too. Therefore, a problem is: how to define the interpretation of types of the form MN?

Coquand [Coq86b][Coq85] gives a nice solution to this problem: not only types are interpreted, but the other terms too. Then, in order to show that the interpretation defined by induction on term structure is well-defined and does interpret every type as a saturated set, he makes a substantial use of the fact that there is a complexity measure of non-propositional types in the calculus of constructions (as we mentioned at the beginning of chapter 4). This straightforward measure for non-propositional types exists simply because that there is only one real universe in the calculus of constructions and there is no non-propositional type-valued function.

Things become different when we have more than one universe like in **ECC**. Now there are functions with non-propositional types as values, say $\lambda x{:}Type_0.x$. Therefore, a term of the form MN (or $\pi_i(M)$) may be a non-propositional type. We need to clarify the forms of the non-propositional types and have a complexity measure to make explicit the predicativity of them. This is the main reason that we spend a lot of energy to prove the quasi normalization theorem and find the complexity measure in the last chapter. Coquand's solution is the clue that motivates our work on quasi normalization. As we have succeeded in proving the quasi-normalization theorem and finding the complexity measure β in the previous chapter, we are now ready to apply Girard-Tait's method to prove strong normalization.

5.2 The Strong Normalization Theorem

We now apply the reducibility method to prove strong normalization for **ECC**. The central theme is to define a term model (interpretation) in which types are interpreted as saturated sets (see definition 5.1.2 in the previous section), and then prove the soundness of the interpretation, which implies the strong normalization theorem.

5.2.1 Possible values of terms

We first define a notion of *value-sets* which indicates the possible values of a term in the term model (subject to some variable assignment). In particular, the possible values of an $\mathcal{E}$-type A are the A-saturated sets.

Definition 5.2.1 (value-sets of $\mathcal{E}$-terms) *The* set of (possible) values *of an $\mathcal{E}$-term M, $V(M)$, is defined by considering the form of its principal type $T_{\mathcal{E}}(M)$, which is assumed to be in quasi-normal form, and by induction on the complexity measure $\beta(T_{\mathcal{E}}(M))$:*

1. *If $T_{\mathcal{E}}(M)$ is a kind, i.e., M is an $\mathcal{E}$-type, then*

$$V(M) =_{\mathrm{df}} Sat(M).$$

2. *If $T_{\mathcal{E}}(M)$ is an $\mathcal{E}$-proposition, i.e., M is an $\mathcal{E}$-proof, then*

$$V(M) =_{\mathrm{df}} \{\theta\}.$$

   where θ is a fixed arbitrary symbol.

3. *If $T_{\mathcal{E}}(M)$ is a base term, then*

$$V(M) =_{\mathrm{df}} \{\theta\}.$$

4. *If $T_{\mathcal{E}}(M) \equiv \Pi x{:}A_1.A_2$ is a non-propositional $\mathcal{E}$-type, then define $V(M)$ as the set consisting of the functions f such that*
   - *the domain of f, $dom(f) = \{\,(N, v) \mid \mathcal{E} \vdash N : A_1, v \in V(N)\,\}$,*
   - *$f(N, v) \in V(MN)$ for $(N, v) \in dom(f)$, and*
   - *$f(N, v) = f(N', v)$ for $(N, v), (N', v) \in dom(f)$ such that $N \simeq N'$.*

5. *if $T_{\mathcal{E}}(M) \equiv \Sigma x{:}A_1.A_2$, then*

$$V(M) =_{\mathrm{df}} \{\,(v_1, v_2) \mid v_1 \in V(\pi_1(M)), v_2 \in V(\pi_2(M))\,\}.$$

$\square$

Remark The above definition is well defined by theorem 4.3.13, corollary 4.3.14, lemma 4.4.3, lemma 4.4.4, and Church-Rosser theorem. Note that the quasi-normalization theorem and its corollary are essential for the definition to work and the properties of the complexity measure β are also important. For example, when non-propositional $\mathcal{E}$-type $\Pi x{:}A_1.A_2$ is the principal type of M, we know that, for $\mathcal{E} \vdash N : A_1$, $V(N)$ and $V(MN)$ are already defined because $\beta(T_{\mathcal{E}}(N)) \leq \beta(A_1) < \beta(\Pi x{:}A_1.A_2)$ and $\beta(T_{\mathcal{E}}(MN)) \leq \beta([N/x]A_2) < \beta(\Pi x{:}A_1.A_2)$ by lemma 4.4.3 and lemma 4.4.4. $\square$

Convertible terms have the same possible values, as the following lemma shows.

Lemma 5.2.2 *Let M and N be $\mathcal{E}$-terms. If $M \simeq N$, then $V(M) = V(N)$.*

Proof We prove the lemma by the same induction as used in definition 5.2.1. Note that, as $M \simeq N$, $T_{\mathcal{E}}(M) \approx T_{\mathcal{E}}(N)$ (see the remark after lemma 3.1.4 for definition of $\approx$) have the same sort of forms up to conversion.

1. If M is an $\mathcal{E}$-type, so is N. So, by definition of saturated sets, $V(M) = Sat(M) = Sat(N) = V(N)$.
2. If $T_{\mathcal{E}}(M)$ is an $\mathcal{E}$-proposition, so is $T_{\mathcal{E}}(N)$. So, $V(M) = \{\theta\} = V(N)$.
3. If $T_{\mathcal{E}}(M)$ reduces to a quasi-normal base term, so is $T_{\mathcal{E}}(N)$. So, $V(M) = \{\theta\} = V(N)$.
4. If $T_{\mathcal{E}}(M)$ reduces to a quasi-normal non-propositional $\mathcal{E}$-type $\Pi x{:}A_1.A_2$, then $T_{\mathcal{E}}(N)$ reduces to some quasi-normal non-propositional $\mathcal{E}$-type $\Pi x{:}A_1'.A_2'$ and $A_1' \simeq A_1$. By induction hypothesis, $V(MN_0) = V(NN_0)$ for every N_0 such that $\mathcal{E} \vdash N_0 : A_1$. By definition of value-sets, $V(M) = V(N)$.
5. If $T_{\mathcal{E}}(M)$ reduces to a quasi-normal term $\Sigma x{:}A_1.A_2$, then $T_{\mathcal{E}}(N)$ reduces to some quasi-normal term $\Sigma x{:}A_1'.A_2'$. By induction hypothesis, $V(\pi_i(M)) = V(\pi_i(N))$ for $i = 1, 2$. By definition of value-sets, $V(M) = V(N)$. $\square$

Every $\mathcal{E}$-term has at least one possible value. In fact, the proof of the following lemma defines a 'canonical value' for each $\mathcal{E}$-term.

Lemma 5.2.3 (canonical value of $\mathcal{E}$-terms) *For every $\mathcal{E}$-term M, $V(M)$ is not empty.*

Proof The following definition gives every $\mathcal{E}$-term M a 'canonical' value $v_M \in V(M)$, by the same induction as used in definition 5.2.1.

1. If M is an $\mathcal{E}$-type, then $v_M =_{\mathrm{df}} SN(M)$.
2. If M is an $\mathcal{E}$-proof, then $v_M =_{\mathrm{df}} \theta$.

3. If $T_{\mathcal{E}}(M)$ reduces to a quasi-normal base term, then $v_M =_{\mathrm{df}} \theta$.

4. If $T_{\mathcal{E}}(M)$ reduces to a quasi-normal non-propositional $\mathcal{E}$-type $\Pi x{:}A_1.A_2$, then v_M is defined to be the function $f \in V(M)$ such that $f(N,v) = v_{MN}$ for all $(N,v) \in dom(f)$.

5. If $T_{\mathcal{E}}(M)$ reduces to a quasi-normal term $\Sigma x{:}A_1.A_2$, then $v_M =_{\mathrm{df}} (v_{\pi_1(M)}, v_{\pi_2(M)})$. □

5.2.2 Assignments and valuations

In this section, variable assignments and valuations are defined and studied. We first introduce a notation for simultaneous substitution.

Notation (simultaneous substitution) We write $[N_1,...,N_n/x_1,...,x_n]M$ for (the resulting term by) the *simultaneous substitution* of terms N_i for the free occurrences of variables x_i $(i = 1,...,n)$ in M. □

Lemma 5.2.4 (simultaneous substitution) *If* $\mathcal{E}^k \vdash M : A$ *and, for all* $i \leq k$, $\mathcal{E} \vdash N_i : [N_1,...,N_{i-1}/e_1,...,e_{i-1}]E_i$, *then*

$$\mathcal{E} \vdash [N_1,...,N_k/e_1,...,e_k]M : [N_1,...,N_k/e_1,...,e_k]A.$$

Proof As $\mathcal{E}^k \vdash M : A$, by repeated applications of rule (λ),

$$\mathcal{E} \vdash \lambda e_1{:}E_1...\lambda e_k{:}E_k.M : \Pi e_1{:}E_1...\Pi e_k{:}E_k.A$$

Then, by repeated applications of rule (app) and assuming that the bound variables e_i above are not in $\bigcup_{1\leq i\leq k} FV(N_i)$, we have

$$\mathcal{E} \vdash (\lambda e_1{:}E_1...\lambda e_k{:}E_k.M)N_1...N_k : [N_1,...,N_k/e_1,...,e_k]A$$

The result then follows from theorem 3.2.8. □

We now introduce the notion of assignment and valuation.

Definition 5.2.5 ($\mathcal{E}$-assignment and $\mathcal{E}$-valuation) *An* $\mathcal{E}$-assignment *is a function* $\phi : FV(\mathcal{E}^k) \to \mathcal{T}$ *for some* $k \in \omega$ *such that* $\mathcal{E} \vdash \phi(e_i) : \phi(E_i)$ *for each* $1 \leq i \leq k$, *where* $\mathcal{E}_i \equiv e_i{:}E_i$. *(We also write* ϕ *for the simultaneous substitution determined by* ϕ, *i.e.,* $[\phi(e_1),...,\phi(e_k)/e_1,...,e_k]$.*)*

An $\mathcal{E}$-valuation *is a pair of functions* $\rho = (\phi, val)$ *such that* ϕ *is an* $\mathcal{E}$-*assignment and* val *is a function with* $dom(\phi)$ *as its domain such that, for each* $e_i \in dom(\phi)$, $val(e_i) \in V(\phi(e_i))$. *The domain of* ρ *is the domain of* ϕ.

An $\mathcal{E}$-*valuation* ρ *with domain* $FV(\mathcal{E}^k)$ covers *an* $\mathcal{E}$-*term* M *if and only if* $\mathcal{E}^k \vdash M : A$ *for some* A. □

Lemma 5.2.6 (extensibility of $\mathcal{E}$-valuations) *Let $A \equiv E_m$ be an $\mathcal{E}^k$-type, where $m \geq k$. If $\mathcal{E} \vdash N_j : [N_1, ..., N_{j-1}/e_1, ..., e_{j-1}]E_j$ for $1 \leq j \leq k$ and $\mathcal{E} \vdash N : [N_1, ..., N_k/e_1, ..., e_k]A$, then there exist variables $y_{k+1}, ..., y_{m-1}$ such that*

$$\mathcal{E} \vdash y_{k+i} : [N_1, ..., N_k, y_{k+1}, ..., y_{k+i-1}/e_1, ..., e_{k+i-1}]E_{k+i} \quad (i = 1, ..., m-k-1)$$

$$\mathcal{E} \vdash N : [N_1, ..., N_k, y_{k+1}, ..., y_{m-1}/e_1, ..., e_{m-1}]A$$

Proof By lemma 3.2.9, for $i = 1, ..., m-k-1$, E_{k+i} are $\mathcal{E}^{k+i-1}$-types, and so $\Pi e_1{:}E_1 ... \Pi e_{k+i-1}{:}E_{k+i-1}.E_{k+i}$ are $\mathcal{E}$-types and there exist variables z_i such that $\mathcal{E} \vdash z_i : \Pi e_1{:}E_1 ... \Pi e_{k+i-1}{:}E_{k+i-1}.E_{k+i}$. Hence, by induction on $i = 1, ..., m-k-1$, we have

$$\mathcal{E} \vdash z_i N_1 ... N_k y_{k+1} ... y_{k+i-1} : [N_1, ..., N_k, y_{k+1}, ..., y_{k+i-1}/e_1, ..., e_{k+i-1}]E_{k+i}$$

So, by theorem 3.2.7, $[N_1, ..., N_k, y_{k+1}, ..., y_{k+i-1}/e_1, ..., e_{k+i-1}]E_{k+i}$ is an $\mathcal{E}$-type. Hence, there exists y_{k+i} satisfying the requirement. As $FV(A) \subseteq \{e_1, ..., e_k\}$, $[N_1, ..., N_k, y_{k+1}, ..., y_{m-1}/e_1, ..., e_{m-1}]A \equiv [N_1, ..., N_k/e_1, ..., e_k]A$; hence, $\mathcal{E} \vdash N : [N_1, ..., N_k, y_{k+1}, ..., y_{m-1}/e_1, ..., e_{m-1}]A$. □

Remark The above lemma shows that, if $\rho = (\phi, val)$ is an $\mathcal{E}$-valuation which covers N and A, and $\mathcal{E} \vdash N : \phi(A)$, then ρ can be extended to an $\mathcal{E}$-valuation $\rho' = (\phi', val')$ such that $\phi'(x) = N$ for some variable $x \notin dom(\rho)$. □

5.2.3 Interpretation of terms

Now, we define the interpretation of $\mathcal{E}$-terms. Every $\mathcal{E}$-term is given a unique value in its value-set, subject to an $\mathcal{E}$-valuation.

Definition 5.2.7 (Evaluation $Eval_\rho$) *Let $\rho = (\phi, val)$ be an $\mathcal{E}$-valuation. The* evaluation function *$Eval_\rho$ of $\mathcal{E}$-terms which are covered by ρ are defined as follows:*

1. *If M is an $\mathcal{E}$-proof, then $Eval_\rho(M) =_{\mathrm{df}} \theta$.*

2. *If M is not an $\mathcal{E}$-proof, $Eval_\rho(M)$ is defined by induction on the structure of M:*

   (a) *M is a kind. Then $Eval_\rho(M) =_{\mathrm{df}} SN(M)$.*

   (b) *M is a variable. Then $Eval_\rho(M) =_{\mathrm{df}} val(M)$.*

   (c) *$M \equiv \Pi x{:}M_1.M_2$. We may assume that $x \notin dom(\rho)$. Then, $Eval_\rho(M)$ is defined to be the set of the terms F such that*

      i. *$\mathcal{E} \vdash F : \phi(M)$, and*

      ii. *$FN \in Eval_{\rho'}(M_2)$ for every $\mathcal{E}$-valuation $\rho' = (\phi', val')$ which extends ρ such that $\phi'(x) = N \in Eval_\rho(M_1)$.*

(d) $M \equiv \lambda x{:}M_1.M_2$. *We may assume that* $x \notin dom(\rho)$. *Then,* $Eval_\rho(M)$ *is defined to be the function* f *such that*

   i. $dom(f) = \{ (N, v) \mid \mathcal{E} \vdash N : \phi(M_1), v \in V(N) \}$, *and*

   ii. $f(N, v) = Eval_{\rho'}(M_2)$ *for* $(N, v) \in dom(f)$, *where* ρ' *extends* ρ *such that* $\rho'(x) = (N, v)$.

(e) $M \equiv M_1M_2$. *Then* $Eval_\rho(M) =_{\mathrm{df}} Eval_\rho(M_1)(\phi(M_2), Eval_\rho(M_2))$.

(f) $M \equiv \Sigma x{:}M_1.M_2$. *We may assume that* $x \notin dom(\rho)$. *Then,* $Eval_\rho(M)$ *is defined to be the set of the terms* P *such that*

   i. $\mathcal{E} \vdash P : \phi(M)$, *and*

   ii. $\pi_1(P) \in Eval_\rho(M_1)$ *and* $\pi_2(P) \in Eval_{\rho'}(M_2)$ *for every* $\mathcal{E}$*-valuation* $\rho' = (\phi', val')$ *which extends* ρ *such that* $\phi'(x) = \pi_1(P)$.

(g) $M \equiv \mathbf{pair}_A(M_1, M_2)$. *Then,* $Eval_\rho(M) =_{\mathrm{df}} (Eval_\rho(M_1), Eval_\rho(M_2))$.

(h) $M \equiv \pi_i(M')$. *Then,* $Eval_\rho(M) =_{\mathrm{df}} v_i$, *if* $Eval_\rho(M') = (v_1, v_2)$, *where* $i \in \{1, 2\}$.

$\square$

The following lemma guarantees the well-definedness of the interpretation. In particular, it implies that every $\mathcal{E}$-type is interpreted as a saturated set.

Lemma 5.2.8 (well definedness of Eval) *Let* $\rho = (\phi, val)$ *be an* $\mathcal{E}$*-valuation which covers* $\mathcal{E}$*-term* M.

1. *If* $\mathcal{E}$*-valuation* $\rho' = (\phi', val')$ *covers* M, *and* $\phi(x) \simeq \phi'(x)$ *and* $val(x) = val'(x)$ *for every* $x \in FV(M)$, *then* $Eval_\rho M = Eval_{\rho'} M$.

2. $Eval_\rho(M) \in V(\phi(M))$.

Proof If M is an $\mathcal{E}$-proof, then so are $\phi(M)$ and $\phi'(M)$, and we have

$$Eval_\rho M = Eval_{\rho'} M = \theta \in \{\theta\} = V(\phi(M)) = V(\phi'(M))$$

If M is not an $\mathcal{E}$-proof, we prove the two statements of the lemma by mutual induction on the structure of M.[3]

Proof of the first statement.

1. M is a kind. $Eval_\rho M = SN(M) = Eval_{\rho'} M$.

2. M is a variable. $Eval_\rho M = val(M) = val'(M) = Eval_{\rho'} M$.

[3]By mutual induction, we mean to prove the two statements *simultaneously* but just write the proofs separately.

3. $M \equiv \Pi x{:}M_1.M_2$. We show $Eval_\rho M \subseteq Eval_{\rho'} M$ and the other direction is the same. Suppose $F \in Eval_\rho M$. Then, $\mathcal{E} \vdash F : \phi'(M)$ as $\phi(M) \simeq \phi'(M)$. For any extension $\rho'_1 = (\phi'_1, val'_1)$ of ρ' such that $\phi'_1(x) = N \in Eval_{\rho'} M_1$, we can also find an extension $\rho_1 = (\phi_1, val_1)$ of ρ such that $\phi_1(x) = N \in Eval_\rho M_1$ by lemma 5.2.6 as $Eval_\rho M_1 = Eval_{\rho'} M_1$ by induction hypothesis. Hence, $FN \in Eval_{\rho_1} M_2 = Eval_{\rho'_1} M_2$ by induction hypothesis.

4. $M \equiv \lambda x{:}M_1.M_2$. We have $dom(Eval_\rho M) = dom(Eval_{\rho'} M)$ as $\phi(M_1) \simeq \phi'(M_1)$, and $Eval_\rho M(N, v) = Eval_{\rho_1} M_2 = Eval_{\rho'_1} M_2 = Eval_{\rho'_1} M$ by induction hypothesis, where ρ_1 and ρ'_1 extend ρ and ρ' respectively as in the definition of $Eval$.

5. $M \equiv M_1 M_2$. As $\phi(M_2) \simeq \phi'(M_2)$ and $Eval_{\rho'} M_1 \in V(\phi'(M_1))$ by (mutual) induction hypothesis, we have by induction hypothesis

$$\begin{aligned} Eval_\rho M &= Eval_\rho M_1(\phi(M_2), Eval_\rho M_2) \\ &= Eval_{\rho'} M_1(\phi(M_2), Eval_{\rho'} M_2) \\ &= Eval_{\rho'} M_1(\phi'(M_2), Eval_{\rho'} M_2) \\ &= Eval_{\rho'} M \end{aligned}$$

6. $M \equiv \Sigma x{:}M_1.M_2$. Similar to the case of Π-form.

7. $M \equiv \mathbf{pair}_A(M_1, M_2)$. By induction hypothesis.

8. $M \equiv \pi_i(M')$. By induction hypothesis.

Proof of the second statement.

1. M is a kind. $Eval_\rho M = SN(M) \in Sat(M) = V(M) = V(\phi(M))$.

2. M is a variable. $Eval_\rho M = val(M) \in V(\phi(M))$.

3. $M \equiv \Pi x{:}M_1.M_2$. We have to show that $Eval_\rho M$ is a $\phi(M)$-saturated set.

   (S1) Suppose $F \in Eval_\rho M$. We only have to show that F is strongly normalizable. Take a variable y such that $\mathcal{E} \vdash y : \phi(M_1)$. As $Eval_\rho M_1 \in V(\phi(M_1)) = Sat(\phi(M_1))$ is a saturated set by induction hypothesis, $y \in Eval_\rho M_1$. Let ρ' be an extension of ρ such that $\rho'(x) = (y, v_y)$, where v_y is the canonical value of y. Then, by induction hypothesis, $Eval_{\rho'} M_2 \in V(\phi'(M_2)) = Sat(\phi'(M_2))$ is a saturated set. So, $Fy \in Eval_{\rho'} M_2$ is strongly normalizable, which implies that F is strongly normalizable.

   (S2) Suppose $M_0 \in SN(\phi(M))$ is a base term. We only have to show $M_0 N \in Eval_{\rho'} M_2$ for any extension $\rho' = (\phi', val')$ of ρ such that $\phi'(x) = N \in Eval_\rho M_1$. This follows from that $Eval_{\rho'} M_2$ is a $\phi'(M_2)$-saturated set (by induction hypothesis), $M_0 N$ is a base term and $\mathcal{E} \vdash M_0 N : \phi'(M_2)$ (as $\phi'(M_2) \equiv [N/x]\phi(M_2)$).

(S3) Suppose $M_0 \in SN(\phi(M))$ has key redex and $\mathrm{red}_{\mathbf{k}}(M_0) \in Eval_\rho M$. We only have to show $M_0 N \in Eval_{\rho'} M_2$ for any extension $\rho' = (\phi', val')$ of ρ such that $\phi'(x) = N \in Eval_\rho M_1$. By induction hypothesis, $Eval_{\rho'} M_2 \in Sat(\phi'(M_2))$ is a saturated set. As $\mathrm{red}_{\mathbf{k}}(M_0) \in Eval_\rho M$, $\mathrm{red}_{\mathbf{k}}(M_0)N \in Eval_{\rho'} M_2$ is strongly normalizable which implies that $M_0 N$ is strongly normalizable as M_0 is a strongly normalizable. Noticing that $\mathcal{E} \vdash M_0 N : \phi'(M_2)$ (as $\phi'(M_2) \equiv [N/x]\phi(M_2)$) and $M_0 N$ has the same key redex as M_0, we have $M_0 N \in Eval_{\rho'} M_2$.

4. $M \equiv \lambda x{:}M_1.M_2$. For any extension $\rho' = (\phi', val')$ of ρ such that $\rho'(x) = (N, v) \in dom(Eval_\rho M)$, by induction hypothesis,

   $$Eval_\rho M(N, v) = Eval_{\rho'}(M_2) \in V(\phi'(M_2)) = V([N/x]\phi(M_2)) = V(\phi(M)N)$$

   If $\mathcal{E} \vdash N' : \phi(M_1)$ and $N' \simeq N$, we have, by (mutual) induction hypothesis,

   $$Eval_\rho(M)(N, v) = Eval_{\rho'}(M_2) = Eval_{\rho'_{N'}}(M_2) = Eval_\rho(M)(N', v)$$

   where $\rho'_{N'}$ extends ρ such that $\rho'_{N'}(x) = (N', v)$.

5. $M \equiv M_1 M_2$. As $T_{\mathcal{E}}(\phi(M_1))$ has the form $\Pi x{:}A_1.A_2$, $Eval_\rho M_1 \in V(\phi(M_1))$ is a function f such that $f(N, v) \in V(\phi(M_1)N)$ for any N such that $\mathcal{E} \vdash N : A_1$ and any $v \in V(N)$. Noticing that $\mathcal{E} \vdash \phi(M_2) : A_1$ (as $M_1 M_2$ is an $\mathcal{E}$-term) and $Eval_\rho M_2 \in V(\phi(M_2))$, we have $Eval_\rho M = Eval_\rho M_1(\phi(M_2), Eval_\rho M_2) \in V(\phi(M_1)\phi(M_2)) = V(\phi(M))$.

6. $M \equiv \Sigma x{:}M_1.M_2$. We have to show that $Eval_\rho M$ is a $\phi(M)$-saturated set.

   (S1) Suppose $P \in Eval_\rho M$. We only have to show that P is strongly normalizable. By induction hypothesis, $\pi_1(P) \in Eval_\rho M_1 \in V(\phi(M_1)) = Sat(\phi(M_1))$ is strongly normalizable, so is P.

   (S2) Suppose $M_0 \in SN(\phi(M))$ is a base term. Then, $\pi_1(M_0) \in SN(\phi(M_1))$ is also a base term, so $\pi_1(M_0)$ is in $\phi(M_1)$-saturated set $Eval_\rho M_1$. For any $\mathcal{E}$-valuation $\rho' = (\phi', val')$ which extends ρ such that $\phi'(x) = \pi_1(P)$ (assuming $x \notin dom(\rho)$), $\pi_2(M_0) \in SN([\pi_1(M_0)/x]\phi(M_2)) = SN(\phi'(M_2))$ is also a base term, and so $\pi_2(M_0)$ is in $\phi'(M_2)$-saturated set $Eval_{\rho'} M_2$. Hence, $M_0 \in Eval_\rho M$.

   (S3) Suppose $M_0 \in SN(\phi(M))$ has key redex and $\mathrm{red}_{\mathbf{k}}(M_0) \in Eval_\rho M$. Then, $\pi_1(M_0) \in SN(\phi(M_1))$ has the same key redex and $\pi_1(\mathrm{red}_{\mathbf{k}}(M_0)) \in Eval_\rho M_1$, so $\pi_1(M_0)$ is in $\phi(M_1)$-saturated set $Eval_\rho M_1$. For any $\mathcal{E}$-valuation $\rho' = (\phi', val')$ which extends ρ such that $\phi'(x) = \pi_1(P)$ (assuming $x \notin dom(\rho)$), $\pi_2(M_0) \in SN([\pi_1(M_0)/x]\phi(M_2)) = SN(\phi'(M_2))$ has the same key redex and $\pi_2(\mathrm{red}_{\mathbf{k}}(M_0)) \in Eval_{\rho'} M_2$, so $\pi_2(M_0)$ is in $\phi'(M_2)$-saturated set $Eval_{\rho'} M_2$. Hence, $M_0 \in Eval_\rho M$.

7. $M \equiv \mathbf{pair}_A(M_1, M_2)$. By induction hypothesis and lemma 5.2.2.

8. $M \equiv \pi_i(M')$. As $T_{\mathcal{E}}(M')$ has Σ-form, $Eval_\rho M' = (v_1, v_2) \in V(\phi(M'))$ such that $v_i \in V(\pi_i(\phi(M')))$, where $i = 1, 2$. So, $Eval_\rho M = v_i \in V(\pi_i(\phi(M'))) = V(\phi(M))$. $\square$

Corollary 5.2.9 *If A is an $\mathcal{E}$-type and $\rho = (\phi, val)$ is an $\mathcal{E}$-valuation which covers A, then $Eval_\rho A$ is a $\phi(A)$-saturated set.*

Proof By lemma 5.2.8 and the definition of value-sets. $\square$

5.2.4 Soundness of the interpretation

We prove the interpretation $Eval_\rho$ is sound in the following sense:

1. It respects the conversion relation by equality and the cumulativity relation by inclusion (lemma 5.2.11);

2. If M has type A, then, under a suitable variable assignment, M is an element of the interpretation of A (theorem 5.2.12).

To prove these results, the following substitution property has to be proved first.

Lemma 5.2.10 (substitution property) *Suppose $\rho = (\phi, val)$ is an $\mathcal{E}$-valuation which covers N and $[N/x]M$, where $x \notin dom(\rho)$, and $\rho' = (\phi', val')$ is an extension of ρ which covers M such that $\rho'(x) = (\phi(N), Eval_\rho(N))$. Then, $Eval_\rho([N/x]M) = Eval_{\rho'}(M)$.*

Proof If M is an $\mathcal{E}$-proof, so is $[N/x]M$; then $Eval_\rho([N/x]M) = Eval_\rho(M) = \theta$. If none of M and $[N/x]M$ is an $\mathcal{E}$-proof, we prove the lemma by induction on the structure of M.

1. M is a kind. $Eval_\rho([N/x]M) = Eval_\rho(M) = SN(M) = Eval_{\rho'}(M)$.

2. M is a variable. If $M \not\equiv x$, then $Eval_\rho([N/x]M) = Eval_\rho(M) = val(M) = val'(M) = Eval_{\rho'}(M)$. If $M \equiv x$, then $Eval_\rho([N/x]M) = Eval_\rho(N) = val'(M) = Eval_{\rho'}(M)$.

3. $M \equiv \Pi y{:}M_1.M_2$. We may assume $y \notin dom(\rho')$. We have

   - $\phi([N/x]M) \equiv [\phi(N)/z]\phi([z/x]M) \equiv \phi'(M)$, by suitably choosing variable z;
   - $Eval_\rho([N/x]M_1) = Eval_{\rho'}(M_1)$, by induction hypothesis; and
   - by lemma 5.2.6, for any $\mathcal{E}$-valuation $\rho_1 = (\phi_1, val_1)$ which extends ρ such that $\phi_1(y) = N_1 \in Eval_\rho M_1$, we can find an $\mathcal{E}$-valuation $\rho'_1 = (\phi'_1, val'_1)$ which extends ρ' such that $\phi'_1(y) = N_1 \in Eval_\rho M_1$, (and vice versa); furthermore, by lemma 5.2.8, $Eval_{\rho_1} M_2 = Eval_{\rho'_1} M_2$.

   From these, by definition of $Eval$, $Eval_\rho([N/x]M) = Eval_{\rho'}(M)$.

4. $M \equiv \lambda y{:}M_1.M_2$. We may assume $y \notin dom(\rho')$. We have
   - $dom(Eval_\rho[N/x]M) = dom(Eval_{\rho'}M)$, as $\phi([N/x]M_1) \equiv \phi([z/x]M_1) \equiv \phi'(M_1)$, by suitably choosing variable z;
   - for any $(N_1, v_1) \in dom(Eval_\rho[N/x]M)$, by lemma 5.2.8, $Eval_\rho[N/x]M(N_1, v_1) = Eval_{\rho_1}M_2 = Eval_{\rho'_1}M_2 = Eval_{\rho'}M(N_1, v_1)$, where ρ_1 and ρ'_1 extend ρ and ρ' such that $\rho_1(y) = \rho'_1(y) = (N_1, v_1)$.

   Hence, by definition of $Eval$, $Eval_\rho([N/x]M) = Eval_{\rho'}(M)$.

5. $M \equiv M_1M_2$. As $\phi([N/x]M_2) \equiv [\phi(N)/z]\phi([z/x]M_2) \equiv \phi'(M_2)$ (by suitably choosing variable z), we have by induction hypothesis

$$\begin{aligned} Eval_\rho([N/x]M) &= Eval_\rho([N/x]M_1)(\phi([N/x]M_2), Eval_\rho([N/x]M_2)) \\ &= Eval_{\rho'}(M_1)(\phi'(M_2), Eval_{\rho'}(M_2)) \\ &= Eval_{\rho'}(M) \end{aligned}$$

6. $M \equiv \Sigma y{:}M_1.M_2$. Similar to the Π-case.

7. $M \equiv \mathbf{pair}_A(M_1, M_2)$. By induction hypothesis.

8. $M \equiv \pi_i(M')$, $i = 1, 2$. By induction hypothesis, $Eval_\rho[N/x]M' = Eval_{\rho'}M' = (v_1, v_2)$. So, $Eval_\rho[N/x]M = Eval_\rho\pi_i([N/x]M') = v_i = Eval_{\rho'}M$.

□

Lemma 5.2.11 *Let $\rho = (\phi, val)$ be an $\mathcal{E}$-valuation.*

1. *If M and N are convertible $\mathcal{E}$-terms covered by ρ, then $Eval_\rho(M) = Eval_\rho(N)$.*
2. *If M and N are $\mathcal{E}$-types covered by ρ and $M \preceq N$, then $Eval_\rho(M) \subseteq Eval_\rho(N)$.*

Proof By induction on the structure of M.

Proof of the first statement. By Church-Rosser theorem, we only have to prove the statement for $M \rhd_1 N$. Then, M can not be a kind or variable. For the cases M is of Π-form, λ-form, Σ-form or **pair**-form, it is true by induction hypothesis. The cases that $M \equiv \pi_i(M')$ can also be readily verified by induction hypothesis and the definition of $Eval$. We consider the case when $M \equiv M_1M_2$. There are two subcases.

1. $M \equiv M_1M_2 \rhd_1 N_1N_2 \equiv N$ with $M_1 \rhd_1 N_1$ or $M_2 \rhd_1 N_2$. Then, $Eval_\rho(M_1) = Eval_\rho(N_1) \in V(\phi(N_1))$, by induction hypothesis and lemma 5.2.8. As $\phi(M_2) \simeq \phi(N_2)$ and $\mathcal{E} \vdash \phi(N_2) : T_{\mathcal{E}}(\phi(M_2))$ by theorem 3.2.8, and

noticing that $T_{\mathcal{E}}(\phi(N_1))$ is of Π-form, we have by induction hypothesis and definition of value-sets,

$$\begin{array}{rcl} Eval_\rho(N) & = & Eval_\rho(N_1)(\phi(N_2), Eval_\rho(N_2)) \\ & = & Eval_\rho(M_1)(\phi(N_2), Eval_\rho(M_2)) \\ & = & Eval_\rho(M_1)(\phi(M_2), Eval_\rho(M_2)) \\ & = & Eval_\rho(M) \end{array}$$

2. $M \equiv M_1 M_2 \equiv (\lambda x{:}X.Y)M_2 \rhd_1 [M_2/x]Y \equiv N$. By lemma 5.2.6, there exists an $\mathcal{E}$-valuation ρ' which extends ρ such that $\rho'(x) = (\phi(M_2), Eval_\rho M_2)$. By lemma 5.2.10,

$$\begin{array}{rcl} Eval_\rho M & = & Eval_\rho(\lambda x{:}X.Y)(\phi(M_2), Eval_\rho M_2) \\ & = & Eval_{\rho'} Y \\ & = & Eval_\rho [M_2/x]Y \\ & = & Eval_\rho N \end{array}$$

Proof of the second statement. By the first statement just proved and lemma 3.1.4, we only have to consider the following cases.

1. $M \preceq N$ are kinds. Then, $Eval_\rho M = SN(M) \subseteq SN(N) = Eval_\rho N$.

2. $M \equiv Qx{:}M_1.M_2 \preceq Qx{:}N_1.N_2 \equiv N$, where $Q \in \{\Pi, \Sigma\}$, and, $M_1 \begin{cases} \simeq N_1 & \text{if } Q \equiv \Pi \\ \preceq N_1 & \text{if } Q \equiv \Sigma \end{cases}$ and $M_2 \preceq N_2$. The result then follows from induction hypothesis.

This completes the proof of the lemma. $\square$

Now, we prove the soundness theorem of the interpretation. As we are dealing with a much richer system than the second-order λ-calculus, the theorem reads more complex than we stated in the outline of the reducibility method in section 5.1.

Theorem 5.2.12 (soundness) *Let $\rho = (\phi, val)$ be an $\mathcal{E}$-valuation with $FV(\mathcal{E}^k)$ as domain such that $\phi(e_i) \in Eval_\rho(E_i)$ for $e_i \in dom(\rho)$. If $\mathcal{E}^k \vdash M : A$, then $\phi(M) \in Eval_\rho A$.*

Proof By induction on the structure of M.

1. M is a kind. Then, $T_{\mathcal{E}}(M) \preceq A$ is convertible to a kind. By lemma 5.2.11, $\phi(M) = M \in SN(T_{\mathcal{E}}(M)) = Eval_\rho T_{\mathcal{E}}(M) \subseteq Eval_\rho A$.

2. $M \equiv e_i$ is a variable. Then, E_i is the principal type of M. By assumption and lemma 5.2.11, $\phi(M) \in Eval_\rho E_i \subseteq Eval_\rho A$.

3. $M \equiv \Pi x{:}M_1.M_2$. Then $A \simeq K$ for some kind K and $Eval_\rho A = Eval_\rho K = SN(K)$ by lemma 5.2.11. We only have to show that $\mathcal{E} \vdash \phi(M) : K$ and $\phi(M)$ is strongly normalizable. As $\mathcal{E} \vdash M : K$ and ϕ is an $\mathcal{E}$-assignment, $\mathcal{E} \vdash \phi(M) : K$ by lemma 5.2.4. By lemma 3.2.9, $\mathcal{E}^k \vdash M_1 : K_1$ for some kind K_1. So, by induction hypothesis, $\phi(M_1) \in Eval_\rho K_1 = SN(K_1)$ is strongly normalizable. We may assume that $x \equiv e_j$ with $E_j \equiv M_1$ for some $j > k$ such that $\mathcal{E}^j \vdash M_2 : K_2$ for some kind K_2. Let $\rho' = (\phi', val')$ be an extension of ρ such that $\phi'(e_{k+i})$ is an variable y_{k+i} such that $\mathcal{E} \vdash y_{k+i} : [\phi(e_1), ..., \phi(e_k), y_{k+1}, ..., y_{k+i-1}/e_1, ..., e_{k+i-1}]E_{k+i}$ and $val'(e_{k+i})$ is any value in $V(y_{k+i})$, where $1 \leq i \leq j-k$. Then, ρ' is an $\mathcal{E}$-valuation and, by induction hypothesis, $\phi'(M_2) \in Eval_{\rho'} K_2 = SN(K_2)$ is strongly normalizable, which implies that $\phi(M_2)$ is strongly normalizable. Therefore, $\phi(M) \equiv \Pi x{:}\phi(M_1).\phi(M_2)$ is strongly normalizable.

4. $M \equiv \lambda x{:}M_1.M_2$. Then, by Church-Rosser theorem and lemma 3.3.3, $A \rhd \Pi x{:}M_1.A_2$ for some $\mathcal{E}$-type A_2.

   (a) As $\mathcal{E} \vdash M : \Pi x{:}M_1.A_2$, $\mathcal{E} \vdash \phi(M) : \phi(\Pi x{:}M_1.A_2)$.

   (b) We may assume that $x \equiv e_j$ for some $j > k$ such that $E_j \equiv x{:}M_1$ and $\mathcal{E}^j \vdash M_2 : A_2$. Suppose $\rho_1 = (\phi_1, val_1)$ is an $\mathcal{E}$-valuation which extends ρ and covers x such that $\phi_1(x) = N \in Eval_\rho M_1$. Then, we can find another $\mathcal{E}$-valuation ρ' with $FV(\mathcal{E}^j)$ as domain which extends ρ in the similar way as in the above case except that $\rho'(x) = \rho_1(x)$. Then, ρ' satisfies the condition required by the theorem. . By induction hypothesis and lemma 5.2.8, $\phi_1(M_2) = \phi'(M_2) \in Eval_{\rho'} A_2 = Eval_{\rho_1} A_2$. As, by lemma 5.2.8, $Eval_{\rho_1} A_2 \in V(\phi_1(T_\mathcal{E}(A_2))) = Sat(\phi_1(T_\mathcal{E}(A_2)))$ is a saturated set, $\phi(M)N$ has key redex, and by contracting the key redex, $\phi(M)N \rhd_1 [\phi_1(x)/x]\phi(M_2) \equiv \phi_1(M_2) \in Eval_{\rho_1} A_2$, we have $\phi(M)N \in Eval_{\rho_1} A_2$.

   So, we have $\phi(M) \in Eval_\rho(\Pi x{:}M_1.A_2)$; hence, $\phi(M) \in Eval_\rho A$, by lemma 5.2.11.

5. $M \equiv M_1 M_2$. Then, $\mathcal{E} \vdash M_1 : \Pi x{:}B_1.B_2$, $\mathcal{E} \vdash M_2 : B_1$ and $[M_2/x]B_2 \preceq A$ for some B_1 and B_2. Let $\rho' = (\phi', val')$ be an $\mathcal{E}$-valuation extending ρ such that $\rho'(x) = (\phi(M_2), Eval_\rho M_2)$. By induction hypothesis, lemmas 5.2.10 and 5.2.11, $\phi(M) \equiv \phi(M_1)\phi'(x) \in Eval_{\rho'} B_2 = Eval_\rho [M_2/x]B_2 \subseteq Eval_\rho A$.

6. $M \equiv \Sigma x{:}M_1.M_2$. Similar to the Π-case.

7. $M \equiv \mathbf{pair}_B(M_1, M_2)$. Then, by Church-Rosser theorem and lemma 3.3.3, $A \rhd \Sigma x{:}A_1.A_2$ for some A_1 and A_2.

   (a) As $\mathcal{E} \vdash M : \Sigma x{:}A_1.A_2$, $\mathcal{E} \vdash \phi(M) : \phi(\Sigma x{:}A_1.A_2)$ by lemma 5.2.4.

(b) Noticing that $Eval_\rho A_1$ is a saturated set, we have $\pi_1(\phi(M)) \in Eval_\rho A_1$, because it is a redex (and hence is the key redex of itself) and its contractum is in $Eval_\rho A_1$ by induction hypothesis. We may assume that $x \equiv e_j$ for some $j > k$. Suppose $\rho_1 = (\phi_1, val_1)$ is an $\mathcal{E}$-valuation extending ρ such that $\phi_1(x) = \pi_1(M)$. Similar to the λ-case, we can find a ρ' which satisfies the condition required in the theorem and $\rho'(x) = \rho_1(x)$. Noticing that $Eval_{\rho'}A_2$ is a saturated set, we have $\pi_1(\phi(M)) \in Eval_\rho A_1$ and $\pi_2(\phi(M)) \in Eval_{\rho'}A_2$, because its contractum is in $Eval_\rho A_1$ by induction hypothesis.

Hence, $\phi(M) \in Eval_\rho \Sigma x{:}A_1.A_2 = Eval_\rho A$.

8. $M \equiv \pi_i(M')$, $i = 1, 2$. Then, $\mathcal{E} \vdash M' : \Sigma x{:}B_1.B_2$ for some B_1 and B_2. By induction hypothesis, $\phi(M') \in Eval_\rho \Sigma x{:}B_1.B_2$.

(a) $i = 1$. Then $B_1 \preceq A$. We have $\phi(M) \equiv \pi_1(\phi(M')) \in Eval_\rho B_1 \subseteq Eval_\rho A$, by definition of $Eval$ and lemma 5.2.11.

(b) $i = 2$. Then $[\pi_1(M')/x]B_2 \preceq A$. We may assume $x \equiv e_j$ for some $j > k$ such that $E_j \equiv B_1$ and let ρ' be an $\mathcal{E}$-valuation extending ρ such that $\rho'(x) = (\pi_1(\phi(M')), Eval_\rho \phi(M'))$. Then, by definition of $Eval$, lemma 5.2.10 and lemma 5.2.11, $\phi(M) \equiv \pi_2(\phi(M')) \in Eval_{\rho'}B_2 = Eval_\rho[\pi_1(M')/x]B_2 \subseteq Eval_\rho A$.

This completes the proof of the theorem. □

5.2.5 The strong normalization theorem

The strong normalization theorem is now a corollary of the above results.

Theorem 5.2.13 (strong normalization) *If $\Gamma \vdash M : A$, then M is strongly normalizable.*

Proof We first show that $\vdash M : A$ implies that M is strongly normalizable. Let $\rho = (\phi, val)$ be any $\mathcal{E}$-valuation. If $\vdash M : A$, then

1. $FV(M) = FV(A) = \emptyset$, by lemma 3.2.2;
2. $\phi(M) \in Eval_\rho(A)$, by theorem 5.2.12;
3. A is an $\mathcal{E}$-type, by theorem 3.2.7; and
4. $Eval_\rho(A) \in V(\phi(A))$, by lemma 5.2.8.

So, we have $M \equiv \phi(M) \in Eval_\rho(A) \in V(\phi(A)) = V(A) = Sat(A)$. By definition of saturated sets, $M \in Eval_\rho(A) \subseteq SN(A)$ is strongly normalizable.

For the arbitrary case, if $\Gamma \vdash M : A$, $\Gamma \equiv x_1{:}A_1, ..., x_m{:}A_m$ by lemma 3.2.2. By applying rule (λ), we have $\vdash \lambda x_1{:}A_1 ... \lambda x_m{:}A_m.M : \Pi x_1{:}A_1 ... \Pi x_m{:}A_m.A$. So, $\lambda x_1{:}A_1 ... \lambda x_m{:}A_m.M$ is strongly normalizable; and this implies that M is strongly normalizable. □

Corollary 5.2.14 *If $x_1{:}A_1, ..., x_n{:}A_n \vdash M : A$, then $A_1, ..., A_n, A$ and M are all strongly normalizable.*

Proof By theorem 5.2.13, theorem 3.2.7 and lemma 3.2.1. $\square$

Chapter 6

Logical Consistency and Decidability

The normalization property of the calculus proved in the previous chapter has several important corollaries, two of which are studied in this chapter.

We first show that, by Curry-Howard correspondence of formulas-as-types, there is a powerful higher-order logic embedded in **ECC** which is consistent by the strong normalization theorem. This gives a sound logical basis of using the theory in applications of, for example, theorem proving and program specification.

Then, we show that the calculus is decidable: (1) the conversion relation and the cumulativity relation are decidable for well-typed terms; and (2) the problems of type inference and type checking are decidable, and we describe algorithms for them and prove their correctness. This provides a direct basis for computer implementations of the theory for development of proofs or programs.

6.1 The Embedded Higher-order Logic

Just as in the correspondence between propositional logic and simply typed λ-calculus, there is an embedded logic in the calculus **ECC** by the Curry-Howard principle of formulas-as-types [CF58][How69]. As the theory provides rich type structures, the embedded logic is a powerful higher-order logic in which one can quantify over arbitrary predicates and functions.

In this section, we follow the idea of formulas as types to describe the embedded logic in **ECC** and prove its consistency. We also briefly discuss an (open) conservativity conjecture which concerns the relationship of the embedded logic with other more standard logics, in particular, intuitionistic higher-order logic.

6.1.1 The embedded logic

A logic can generally be viewed as consisting of a language and a notion of theoremhood. The former is usually given by a notion of (well-formed) formulas and the latter by a notion of proof. These notions of the embedded logic of **ECC** are all relativized to valid contexts. In fact, a valid context can be thought of as a theory in the ordinary sense. Therefore, what we describe below is indeed a 'raw' logical mechanism in which one can build up different theories or even describe different logics.

Definition 6.1.1 (formulas and proofs) *Let Γ be a valid context.*

- *A term P is called a* Γ-formula *if P is a Γ-proposition.*
- *A term M is called a* proof *of a Γ-formula P (in Γ) if $\Gamma \vdash M : P$.*

A Γ-formula P is provable (in Γ) *if there is a proof of P (in Γ).* □

Definition 6.1.2 (functions and predicates) *Let Γ be a valid context. A term F is called an* (n-ary) Γ-function *if for some A_i and A,*

$$\Gamma \vdash F : \Pi x_1{:}A_1 ... \Pi x_n{:}A_n.A$$

Furthermore, if $F(x_1, ..., x_n)$ is a $(\Gamma, x_1{:}A_1, ..., x_n{:}A_n)$-proposition, then F is also called an (n-ary) Γ-predicate (over A_1, ..., A_n). □

Given the above definitions, we can now *define* the ordinary logical operators (and constants), following the well-known idea in higher-order logic (see [Pra65][CH85] for example). Note that sets correspond to types and we are in fact formulating a many-sorted logical mechanism.

Definition 6.1.3 (logical operators) *Let Γ be a valid context, P_1 and P_2 Γ-formulas, A a Γ-type, and P a Γ-predicate over A.*

$$\begin{array}{rcl}
\mathbf{true} & =_{\mathrm{df}} & \Pi x{:}Prop.x \to x \\
\mathbf{false} & =_{\mathrm{df}} & \Pi x{:}Prop.x \\
P_1 \supset P_2 & =_{\mathrm{df}} & P_1 \to P_2 \\
P_1 \,\&\, P_2 & =_{\mathrm{df}} & \Pi R{:}Prop.(P_1 \to P_2 \to R) \to R \\
P_1 \vee P_2 & =_{\mathrm{df}} & \Pi R{:}Prop.(P_1 \to R) \to (P_2 \to R) \to R \\
\neg P_1 & =_{\mathrm{df}} & P_1 \to \mathbf{false} \\
\forall x \in A.P(x) & =_{\mathrm{df}} & \Pi x{:}A.P(x) \\
\exists x \in A.P(x) & =_{\mathrm{df}} & \Pi R{:}Prop.(\Pi x{:}A.(P(x) \to R)) \to R
\end{array}$$

□

It can be verified that all of the ordinary logical inference rules are sound, as the following shows.

1. $\supset$-introduction: *If Γ-formula P_2 is provable under the extra assumption that Γ-formula P_1 is provable (i.e., $\Gamma, p_1{:}P_1 \vdash p_2 : P_2$* for some p_2), *then $P_1 \supset P_2$ is a provable Γ-formula.* This is reflected by rule (λ).

2. $\supset$-elimination (Modus Ponens): *If* $P_1 \supset P_2$ *and* P_2 *are provable* Γ*-formulas, then so is* P_2. By rule (app), $p'p$ is a proof of P_2, if p and p' are proofs of P_1 and $P_1 \supset P_2$, respectively.

3. &-introduction: *If* Γ*-formula* P_1 *and* P_2 *are provable, so is* P_1 & P_2. If p_i is a proof of P_i ($i = 1, 2$), then $\lambda R{:}Prop\lambda h{:}P_1 \to P_1 \to R.hp_1p_2$ is a proof of P_1 & P_2.

4. &-elimination: *If* Γ*-formula* P_1 & P_2 *is provable, so are* P_1 *and* P_2. Suppose h is a proof of P_1 & P_2. Let P'_i ($i = 1, 2$) be the Γ-propositions such that $P'_i \simeq P_i$ and $\Gamma \vdash P'_i : Prop$. Then, $h(P'_i, \lambda p_1{:}P_1\lambda p_2{:}P_2.p_i)$ is a proof of P_i ($i = 1, 2$).

5. $\vee$-introduction: *If* Γ*-formula* P_1 (P_2) *is provable, so is* Γ*-formula* $P_1 \vee P_2$. Suppose p_i is a proof of P_i, where $i \in \{1, 2\}$. Then, $\lambda R{:}Prop\lambda h_1{:}P_1 \to R\lambda h_2{:}P_2 \to R.h_ip_i$ is a proof of $P_1 \vee P_2$.

6. $\vee$-elimination: *If* Γ*-formula* $P_1 \vee P_2$ *is provable, and* Γ*-formula* R_0 *is provable under the extra assumption that* P_i *is provable or* P_2 *is provable, then* R_0 *is provable in* Γ. Suppose h is a proof of $P_1 \vee P_2$ and r_i is a proof of R_0 in $\Gamma, p_i{:}P_i$ ($i = 1, 2$). Then, $h(R'_0, \lambda p_1{:}P_1.r_1, \lambda p_2{:}P_2.r_2)$ is a proof of R_0 in Γ.

7. $\forall$-introduction: *If* $P(x)$ *is provable in* $\Gamma, x{:}A$*, then* $\forall x \in A.P(x)$ *is provable in* Γ. By rule (λ).

8. $\forall$-elimination: *If* Γ*-formula* $\forall x \in A.P(x)$ *is provable and* a *is an element of* A (*i.e.*, $\Gamma \vdash a : A$), *then* $P(a)$ *is provable in* Γ. By rule (app).

9. $\exists$-introduction: *If* $P(a)$ *is provable for some element* a *of* A*, then* $\exists x \in A.P(x)$ *is provable.* Suppose p is a proof of $P(a)$ and a is an element of A. Then $\lambda R{:}Prop\lambda h{:}(\Pi x{:}A.P(x) \to R).h(a, p)$ is a proof of $\exists x \in A.P(x)$.

10. $\exists$-elimination: *If* Γ*-formula* $\exists x \in A.P(x)$ *is provable and* R_0 *is provable in* $\Gamma, x{:}P(a)$*, where* a *is an element of* A*, then* R_0 *is provable in* Γ. Suppose h is a proof of $\exists x \in A.P(x)$ in Γ and r_0 is a proof of R_0 in $\Gamma, x{:}P(a)$. Then, $h(R'_0, a, r_0)$ is a proof of R_0 in Γ, where $R'_0 \simeq R_0$ such that $\Gamma \vdash R'_0 : Prop$.

11. Truth: **true** *is provable in every valid context.* $\lambda x{:}Prop\lambda y{:}x.y$ is a proof of **true**.

12. Absurdity: **false** *implies every formula.* Suppose f is a proof of **false** in Γ. Then, for any Γ-formula P, $f(P')$ is a proof of P, where $P' \simeq P$ such that $\Gamma \vdash P' : Prop$.

We can define a propositional equality following Leibniz's principle by impredicative higher-order definition [Rus03][CH85] as follows.

Definition 6.1.4 (Leibniz's equality) *Let A be a Γ-type. The Leibniz's equality over A, notation $=_A$, is defined as follows:*

$$=_A \ =_{\mathrm{df}} \ \lambda x{:}A\lambda y{:}A\ \Pi P{:}A \to Prop.(Px \to Py)$$

$=_A$ is of type $A \to A \to Prop$ (a binary predicate over A). We shall write $a =_A b$ for $=_A (a, b)$. □

It can be verified that $=_A$ satisfies the laws of identity:

$$\forall x{:}A.\ (x =_A x)$$

$$\forall x{:}A\forall y{:}A.(x =_A y) \supset (y =_A x)$$

$$\forall x{:}A\forall y{:}A\forall z{:}A.\ (x =_A y) \to (y =_A z) \to (x =_A z)$$

For example, the second above (symmetry) is proved by

$$\lambda x{:}A\lambda y{:}A\lambda h{:}(x =_A y).\lambda P{:}A \to Prop\lambda p{:}P(y).\ h(\lambda z{:}A.P(z) \to P(x), \lambda q{:}P(x).q, p)$$

In section 9.1, we shall show that the Leibniz's equality reflects the definitional equality (conversion) and so it provides us a fundamental basis for program specification as well as theorem proving.

6.1.2 Logical consistency

By the strong normalization theorem, the embedded higher-order logic is consistent — there exist unprovable formulas.

Theorem 6.1.5 (consistency) ECC *is logically consistent in the sense that there exist unprovable formulas; in particular,* **false** *is not provable in the empty context, i.e., $\not\vdash M : \Pi x{:}Prop.x$ for any term M.*

Proof Suppose $\vdash M : \Pi x{:}Prop.x$. By theorem 5.2.13 and theorem 3.2.8, we may assume that M is in normal form. So, by an easy induction on derivations and lemma 3.2.2, M must be of the form $\lambda x{:}Prop.M'$ and $x{:}Prop \vdash M' : x$, where M' is a base term whose key variable is x. We show that this is impossible by induction on the structure of base terms. If $M' \equiv x$, we would have $x{:}Prop \vdash x : x$. By lemma 3.3.3, lemma 3.1.4 and Church-Rosser theorem, we would have $Prop \simeq x$ which is impossible. If $M' \equiv M_1'M_2'$ or $\pi_i(M_1')$, then it must be the case that $x{:}Prop \vdash x : Qy{:}A.B$ for some A and B, where $Q \in \{\Pi, \Sigma\}$. This would imply, by lemma 3.3.3, lemma 3.1.4 and Church-Rosser theorem, that $Prop \simeq Qy{:}A.B$ which is impossible, either. So, we conclude that $\not\vdash M : \Pi x{:}Prop.x$. □

Note that the above theorem says that **false** is not provable in the empty context, while it can be proved in certain (inconsistent) contexts, for example, context x:**false**. This induces a notion of *consistent context*. Viewing valid contexts as theories, a consistent context corresponds to a consistent theory in the traditional sense.

Definition 6.1.6 (consistent contexts) *A valid context* Γ *is* consistent *if and only if not every* Γ*-formula is provable in* Γ*, or equivalently, if and only if* **false** *is not provable in* Γ*.* □

We have the following corollary of the consistency theorem which gives us a way to prove the consistency of certain contexts.

Corollary 6.1.7 *Let* $\Gamma \equiv x_1{:}A_1, ..., x_n{:}A_n$ *be a valid context. If there exist* $M_1, ..., M_n$ *such that, for* $i = 1, ..., n$,

$$\vdash M_i : [M_1, ..., M_{i-1}/x_1, ..., x_{i-1}]A_i,$$

then Γ *is a consistent context.*

Proof Suppose Γ is not consistent, *i.e.*, $\Gamma \vdash M : \mathbf{false}$ for some M. Then, by theorem 3.2.6, $\vdash [M_1, ..., M_n/x_1, ..., x_n]M : \mathbf{false}$, contradicting theorem 6.1.5. So, Γ is consistent. □

Example Consider the following context which assumes that there is an arbitrary equivalence relation over a type A:

$$\begin{array}{rcl}\Gamma_{A,Q} & \equiv & A{:}Type_j,\ Q{:}A \to A \to Prop, \\ & & reflex{:}\forall x{:}A.\ Q(x,x), \\ & & sym{:}\forall x{:}A\forall y{:}A.\ Q(x,y) \to Q(y,x), \\ & & trans{:}\forall x{:}A\forall y{:}A\forall z{:}A.\ Q(x,y) \to Q(y,z) \to Q(x,z)\end{array}$$

$\Gamma_{A,Q}$ is consistent, since we can apply the above corollary by taking, for example, A and Q to be $Prop$ and $=_{Prop}$, respectively. □

Remark Proving the consistency of a valid context is certainly non-trivial, except for some simple classes of contexts, as the above corollary and example show. Sometimes, one may use the normalization theorem to prove the consistency of some more subtle contexts. There are also other interesting properties one may like to associate with contexts which are very useful in applications. We do not expand this discussion here. □

6.1.3 A conservativity conjecture

It is easy to see that the embedded logic described above is very powerful. It is interesting to know its relationship with other more traditional logics, for example, intuitionistic higher-order logic (*c.f.*, [Chu40][Tak75][Sch77]). One of the problems related to this is the conservative extension problem; for example, can the embedded logic in **ECC** be seen as a conservative extension of the intuitionistic higher-order logic? We conjecture that the answer is '*Yes, provided that we choose an appropriate interpretation*'.

Conjecture 6.1.8 *The embedded logic in* **ECC** *is a conservative extension over the (intensional intuitionistic) higher-order logic HOL under some* appropriate interpretation *from HOL to* **ECC**. □

Note that in the above conjecture, we emphasize that the interpretation of HOL in **ECC** must be *appropriate*; more precisely, *the object set OBJ of HOL should be interpreted as a non-propositional type instead of a proposition.* As we conjectured and discussed in the conclusions of [Luo89a,b],

1. if the object set is interpreted as a proposition $Obj{:}Prop$, the interpretation will *not* give a conservative extension of HOL;

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

The intuition behind the first nonconservativity conjecture 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).[1] 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, Geuvers [Geu89] and Berardi [Ber89b] have independently proved that the (pure) calculus of constructions [CH88] is not a conservative extension of higher-order logic; this shows that the first non-conservativity conjecture is true. Note that, in the (pure) calculus of constructions, it seems that the only possible and reasonable way to interpret object sets is to interpret them as propositions (types of type $Prop$), as there are no predicative type universes in the calculus. In [Geu89] and [Ber89b], such an interpretation is adopted and the central parts of their proofs are using the 'double identity' of the object set of being both a logical formula and a set. Therefore, their results do show that the first part (*i.e.*, the nonconservativity part) of our conjecture is right.

As the proofs by Geuvers and Berardi have not been published, we give an outline of the proof by Geuvers [Geu89]. Let $*^s$ be the 'kind' of object sets and $*^p$ the 'kind' of logical formulas in higher-order logic HOL.[2] The conjunction operator is defined, for formulas P_1 and P_2 (of type $*^p$), as

$$P_1 \;\&\; P_2 =_{\rm df} \forall R{:}\,*^p\,.(P_1 \to P_2 \to R) \to R$$

and the existential quantifier is defined, for object set A (of type $*^s$) and formula P (of type $*^p$) possibly with free variable x of type A, as

$$\exists x \in A.P =_{\rm df} \forall R{:}\,*^p\,.(\forall x{:}A.(P \to R)) \to R$$

[1]Set-theoretically, an (arbitrary) non-propositional type can be understood as an arbitrary set, but an (arbitrary) proposition can not (and should not). Object sets are in general viewed as arbitrary sets, and so it does not seem to be adequate to formalize them as propositions. See section 7.5 for a related discussion.

[2]The higher-order logic Geuvers and Berardi considered was formulated as a generalized type system [Bar89b] and called λPREDω in [Geu89] and N$^\omega$JP in [Ber89b].

Now, consider in HOL the following context

$$\Gamma \;\equiv\; Obj{:}*^s,\; c{:}Obj,\; F{:}*^p \to *^p,\; P{:}*^p,\; z{:}F(\exists x \in Obj.P)$$

and the formula

$$Z \equiv \exists R{:}\, *^p\, .F(R \;\&\; P)$$

Using the normalization property of HOL, one can show that Z is *not* provable in Γ in HOL. However, interpreting both object sets and logical formulas as propositions in the calculus of constructions amounts to map both $*^s$ and $*^p$ to *Prop*. By this interpretation, formula Z above is interpreted as the following proposition:

$$Z' \equiv \exists R{:}Prop.F(R \;\&\; P)$$

and context Γ above is interpreted as the following

$$\Gamma' \;\equiv\; Obj{:}Prop, c{:}Obj, F{:}Prop \to Prop, P{:}Prop, z{:}F(\exists x{:}Obj.P)$$

It is easy to show that Z' is provable in Γ' in the calculus of constructions (and **ECC**); In fact, the following gives such a proof:

$$\lambda y{:}Prop\lambda h{:}(\forall R{:}Prop.F(R \;\&\; P) \to y).\; h(Obj, z)$$

Therefore, the above interpretation is not conservative.

Remark Note that the confusion of formulas and object sets made by the interpretation (about Obj in the above) is the essential point of the above argument. The formula Z is not provable from Γ in HOL because object sets and formulas are distinguished (by different kinds $*^s$ and $*^p$), while the formula Z' is provable in Γ' in the calculus of onstructions because Obj is forced (by the interpretation) to be also a logical formula as well as an object set (and therefore h can be applied to Obj in the proof of Z' as shown above). Distinguishing object sets (data types) from logical formulas (propositions) in an interpretation will make the above argument invalid. □

However, our conservativity conjecture is still an open problem which seems to be rather difficult. Instead of considering **ECC**, one may consider a slightly extended system of CC, that is, the calculus of constructions with non-propositional type constants. Then, object sets can be interpreted as non-propositional type constants. If we can prove this simpler calculus is a conservative extension of HOL under the interpretation hinted above, we may then extend the result to **ECC**.

6.2 Decidability

By the strong normalization theorem, the calculus **ECC** is decidable. The conversion relation and the cumulativity relation are both decidable for well-typed terms and there is a simple algorithm for type inference and type checking.

6.2.1 Decidability of conversion and cumulativity

Lemma 6.2.1 (decidability of $\simeq$ and $\preceq$) *It is decidable whether $M \simeq N$ or $M \preceq N$ for arbitrary well-typed terms M and N.*

Proof By Church-Rosser theorem, the normal form of a term is unique (corollary 3.1.2). Therefore, conversion $\simeq$ is decidable for well-typed terms by the strong normalization theorem. The decidability of the cumulativity relation $\preceq$ for well-typed terms follows from that of conversion. □

6.2.2 Decidability of type inference and type checking

Now, we give an algorithm of type inference for **ECC**, *i.e.*, if a term is well-typed in a given context, the algorithm computes its principal type in the context, and otherwise, it returns a symbol indicating that the term is not well-typed in the context. Then, we shall prove that the algorithm is correct, which establishes the decidability of type inference.

Definition 6.2.2 (algorithm of type inference) *The algorithm of type inference $\mathcal{C}(\_;\ \_)$: when given a context $\Gamma \equiv x_1{:}A_1, ..., x_n{:}A_n$ and a term M, it checks whether M is a Γ-term, and if so, $\mathcal{C}(\Gamma;\ M) = T_\Gamma(M)$, the principal type of M under Γ; otherwise, it returns $\bot$. (For the correctness of the algorithm, see the theorem below.)*

$\mathcal{C}(\_;\ \_)$ is defined as follows by induction on the sum of the lengths of the terms A_i's and M and by considering the structure of M. In the following, we use '$\rhd_{\mathbf{nf}}$ ...' to mean 'reduces to normal form ...', $max_{\preceq}$ to denote the maximum of the terms subject to relation $\preceq$, and $\mathcal{K}$ the set of kinds.

1. *Validity of contexts: To see whether Γ is valid (i.e., $\Gamma \vdash Prop : Type_0$), check whether $\mathcal{C}(x_1{:}A_1, ..., x_{n-1}{:}A_{n-1};\ A_n) \rhd_{\mathbf{nf}} K \in \mathcal{K}$.*

   If Γ is not valid, $\mathcal{C}(\Gamma;\ M) = \bot$ for every M. In the following, we assume that the considered contexts are valid.

2. *Constants: M is a kind. Then, $\mathcal{C}(\Gamma;\ M) = \begin{cases} Type_0 & \text{if } M \equiv Prop \\ Type_{j+1} & \text{if } M \equiv Type_j \end{cases}$.*

3. *Variables: M is a variable. Then, $\mathcal{C}(\Gamma;\ M) = \begin{cases} A_i & \text{if } M \equiv x_i \\ \bot & \text{if } M \notin \{x_1, ..., x_n\} \end{cases}$.*

4. *$M \equiv \Pi x{:}M_1.M_2$. Check whether $\mathcal{C}(\Gamma;\ M_1) \rhd_{\mathbf{nf}} K \in \mathcal{K}$ and $\mathcal{C}(\Gamma, x{:}M_1;\ M_2) \rhd_{\mathbf{nf}} K' \in \mathcal{K}$, and if so,*

$$\mathcal{C}(\Gamma;\ M) = \begin{cases} Prop & \text{if } K' \equiv Prop \\ max_{\preceq}\{K, K'\} & \text{if } K' \not\equiv Prop \end{cases};$$

   otherwise, $\mathcal{C}(\Gamma;\ M) = \bot$.

5. $M \equiv \lambda x{:}M_1.M_2$. *Check whether* $\mathcal{C}(\Gamma;\ M_1) \rhd_{\mathrm{nf}} K \in \mathcal{K}$ *and* $\mathcal{C}(\Gamma, x{:}M_1;\ M_2) = B$ *for some* $B \neq \perp$, *and if so,* $\mathcal{C}(\Gamma;\ M) = \Pi x{:}M_1.B$; *otherwise,* $\mathcal{C}(\Gamma;\ M) = \perp$.

6. $M \equiv M_1 M_2$. *Check whether* $\mathcal{C}(\Gamma;\ M_1) \rhd_{\mathrm{nf}} \Pi x{:}A.B$ *for some* A *and* B, *and* $\mathcal{C}(\Gamma;\ M_2) \preceq A$, *and if so,* $\mathcal{C}(\Gamma;\ M) = [M_2/x]B$; *otherwise,* $\mathcal{C}(\Gamma;\ M) = \perp$.

7. $M \equiv \Sigma x{:}M_1.M_2$. *Check whether* $\mathcal{C}(\Gamma;\ M_1) \rhd_{\mathrm{nf}} K \in \mathcal{K}$ *and* $\mathcal{C}(\Gamma, x{:}M_1;\ M_2) \rhd_{\mathrm{nf}} K' \in \mathcal{K}$, *and if so,* $\mathcal{C}(\Gamma;\ M) = max_{\preceq}\{K, K', Type_0\}$; *otherwise,* $\mathcal{C}(\Gamma;\ M) = \perp$.

8. $M \equiv \mathbf{pair}_C(M_1, M_2)$. *Check whether* $C \equiv \Sigma x{:}A.B$ *for some* A *and* B, $\mathcal{C}(\Gamma;\ C) \rhd_{\mathrm{nf}} K \in \mathcal{K}$, $\mathcal{C}(\Gamma;\ M_1) \preceq A$ *and* $\mathcal{C}(\Gamma;\ M_2) \preceq [M_1/x]B$, *and if so,* $\mathcal{C}(\Gamma;\ M) = C$; *otherwise,* $\mathcal{C}(\Gamma;\ M) = \perp$.

9. $M \equiv \pi_i(M')$. *Check whether* $\mathcal{C}(\Gamma;\ M') \rhd_{\mathrm{nf}} \Sigma x{:}A.B$ *for some* A *and* B, *and if so,* $\mathcal{C}(\Gamma;\ M) = \begin{cases} A & \textit{if } i = 1 \\ [\pi_1(M')/x]B & \textit{if } i = 2 \end{cases}$; *otherwise,* $\mathcal{C}(\Gamma;\ M) = \perp$.

This completes the definition of the algorithm. □

Remark The type inference algorithm is simple and easy to implement. This simplicity is due to the full cumulativity of the type hierarchy of **ECC**. For the systems with universes lacking full cumulativity like that in [Coq86a], although strong normalization theorem holds [Luo86b], its notion of principal type becomes more complex and the algorithm for type inference is quite complicated (*c.f.*, [HPol89]). □

Theorem 6.2.3 (correctness of type inference) *The algorithm* $\mathcal{C}(\_;\ \_)$ *is correct, i.e., when given a context* $\Gamma \equiv x_1{:}A_1, ..., x_n{:}A_n$ *and a term* M,

$$\mathcal{C}(\Gamma;\ M) = \begin{cases} T_\Gamma(M) & \textit{if } M \textit{ is a } \Gamma\textit{-term} \\ \perp & \textit{otherwise} \end{cases}$$

Proof By the same induction used in the definition above. The only cases worth mentioning about this proof are when $M \equiv M_1 M_2$ or $\pi_2(M')$. We discuss the former case and the latter is similar.

$M \equiv M_1 M_2$. If either the normal form of $\mathcal{C}(\Gamma;\ M_1)$ is not of the form $\Pi x{:}A.B$ or it is but $\mathcal{C}(\Gamma;\ M_2) \not\preceq A$, then we certainly have by induction hypothesis that M is not a Γ-term. Otherwise, we have $\Gamma \vdash M : [M_2/x]B$ by rules (*app*) and ($\preceq$). We only have to show that $[M_2/x]B$ is the minimum type of M subject to $\preceq$. Suppose $\Gamma \vdash M : B'$ for some $B' \prec [M_2/x]B$. We may assume that the last rule used to derive $\Gamma \vdash M : B'$ is not ($\preceq$), and hence it is (*app*):

$$\frac{\Gamma \vdash M_1 : \Pi x{:}A_1.B_1 \quad \Gamma \vdash M_2 : A_1}{\Gamma \vdash M : [M_2/x]B_1}$$

where $[M_2/x]B_1 \equiv B' \prec [M_2/x]B$. Note that, by induction hypothesis, $\Pi x{:}A.B \simeq \mathcal{C}(\Gamma;\ M_1) \preceq \Pi x{:}A_1.B_1$. So, $B \preceq B_1$ which implies that $[M_2/x]B \preceq [M_2/x]B_1$, contradiction. So, $[M_2/x]B$ is the principal type of M under Γ. $\square$

The decidability of type inference and that of the cumulativity relation implies that the problem of type checking — deciding whether an arbitrary judgement is derivable — is decidable.

Corollary 6.2.4 (decidability of type-checking) ECC *is decidable, i.e., it is decidable whether* $\Gamma \vdash M : A$ *for arbitrary* Γ, M *and* A.

Proof By definition of principal type, to see whether $\Gamma \vdash M : A$, just check whether $\mathcal{C}(\Gamma;\ M) \preceq A$. By theorem 6.2.3 and lemma 6.2.1, this is decidable. $\square$

Remark For a Constructions-like calculus, the problem of type checking is no simpler than that of type inference, as the process of type checking essentially requires type inference. $\square$

Chapter 7

A Set-theoretic Interpretation

We explain in this chapter how the intuitive meanings of the main constructs in **ECC** may be understood set-theoretically. Intuitively, types in a type theory correspond to sets and the colon relation ($M : A$) in a judgement corresponds to the membership relation ($\in$). Then, dependent Π-types stand for dependent products (function spaces) with functions expressed by λ-expressions as their elements; Σ-types stand for dependent sums with pairs as their elements. Such a functional point of view gives us an intuitive understanding of the basic entities in a type system.

However, **ECC** is a very rich type theory which combines the impredicative calculus of constructions and Martin-Löf's predicative type theory. As well-known by the work of Reynolds [Rey84][RP88], the impredicative polymorphism in the second-order λ-calculus F [Gir72][Rey74] does not have non-trivial classical set-theoretic semantics.[1] Since F is a subsystem of **ECC**, we certainly can not expect any non-trivial classical set-theoretic interpretation of **ECC**. This calls for a more elaborate and more careful effort to understand such an impredicative calculus set-theoretically. Furthermore, the question of how type universes can be understood set-theoretically must also be answered in order to model the whole calculus.

As discussed and shown by many authors (*e.g.*, [Gir72][Mog85][LM88][See86] [Pit87][CGW87][Mes88]), the impredicative polymorphism in F can be given constructive set-theoretic interpretations. In particular, the idea of interpreting types as partial equivalence relations [Gir72][Tro73b][Mog85] provides us a nice framework of ω-sets and modest sets [Mog85][Hyl87,82][LM88] in which impredicative polymorphism can be modeled in a satisfactory way. In fact, this idea can also be further developed and applied to understand set-theoretically more complex type theories like the calculus of constructions [Hyl87][Ehr88][Luo88a] and **ECC** [Luo89a,b]. This would give us an intuitive understanding of the calculus in set theory.

We show in this chapter how the intuitive meanings of the constructs in

[1] To be more precise, the standard interpretation of the simply typed λ-calculus can not be extended to a model of the second-order λ-calculus.

the calculus can be captured mathematically in the constructive set-theoretic framework of ω-sets and modest sets [Mog85][Hyl82,87][LM88]. In particular, we explicate how non-propositional Π-types can be understood as set-theoretic products, propositions as 'small products' isomorphic to partial equivalence relations, Σ-types as sets of dependent pairs, and the universes $Type_j$ as corresponding to large set universes. Such a model-theoretic interpretation would give us a better understanding of the calculus and is helpful both in pragmatic applications and theoretical researches.

We shall not give a model semantics in full detail. There is a known problem about defining a model semantics of rich type theories like the calculus of constructions; that is, since there may be more than one derivation of a derivable judgement, a direct inductive definition by induction on derivations is questionable. An attempt to give a full detailed semantics of **ECC** can be found in [Luo89b]. We also refer to Streicher's work [Str88] on a detailed definition of such a semantic model for the (pure) calculus of constructions in Cartmell's framework of contextual categires [Cart78,86]. Further research is needed to gain a nice approach to this problem.

7.1 Understanding the Calculus in the ω–Set Framework

The notions of ω-sets and modest sets are developed by Moggi and Hyland based on the idea of interpreting types as partial equivalence relations. Hyland [Hyl82] studied the general properties of ω-sets (called separate objects) and modest sets (called effective objects). Moggi [Mog85] showed that there is a small internal complete category in the category of ω-sets. People have used these notions to give set-theoretic (categorical) models for the second-order polymorphic λ-calculus [Pit87][LM88][Mes88]. Later, Hyland [Hyl87] defined a stronger notion of completeness which can be used to model the calculus of constructions [Ehr88][Luo88a].

The following is the definition of ω-sets.

Definition 7.1.1 (ω-sets) *An* ω-set *is a pair*

$$A = (|A|, \Vdash_A)$$

that consists of a set $|A|$ and a relation $\Vdash_A \subseteq \omega \times |A|$ which is surjective (i.e., $\forall a \in |A|.\ \exists n \in \omega.\ n \Vdash_A a$). $|A|$ is called the carrier set *of A and $\Vdash_A$ the* realizability relation *of A.*

A morphism *f between two ω-sets A and B is a function $f : |A| \to |B|$ realized by some partial recursive function, i.e., there exist $n \in \omega$ such that*

$$\forall a \in |A|\ \forall m \in \omega.\ m \Vdash_A a \Rightarrow nm \Vdash_B f(a)$$

where nm denotes the result of Kleene application of n to m.

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

Remark The morphisms between ω-sets are 'computable', *i.e.*, 'computed' (or realized) by a partial recursive function. The category ω–**Set** is a concrete locally cartesian closed category [Hyl82][Mog85]. Hence, it provides us structures richer than those needed to interpret second-order λ-calculus. □

We now discuss how the main constructs in **ECC** can and should be understood set-theoretically. The main question 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 (predicatively);
4. $Prop$ is closed under Π (impredicatively for arbitrary products).

Notice that the universe $Prop$ is impredicative and required to be closed under arbitrary (possibly circular) product formation. As we remarked above, this prevents us from working in classical set theory to gain a non-trivial model of the calculus. Furthermore, there is more than one universe which must be understood set-theoretically to satisfy the above requirements.

We show in the following sections that, in the framework of ω-sets and modest sets, a model-theoretic understanding of the main constructs can be given to satisfy these requirements. In particular,

- a valid context is interpreted as an ω-set which consists of the 'tuples' of its components;
- Γ-types are interpreted as families of ω-sets indexed by the interpretation of context Γ; in particular, Γ-propositions are interpreted as families of 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 the full subcategories ω–**Set**(j) of ω–**Set** whose objects have carrier sets residing in the corresponding large set universes.

In general, a well-typed term M of type A in context Γ is interpreted as a morphism in ω–**Set** corresponding to an element of the interpretation of A indexed by the interpretation of Γ.

7.2 Interpretation of Valid Contexts

A valid context intuitively stands for a sequence of assumptions and assumed constants. It is interpreted as an ω-set which consists of the 'tuples' of the components of the context. To specify the interpretation of contexts, we need an ω-set constructor, σ, defined as follows.

Definition 7.2.1 (σ) *Suppose that* Γ *is an* ω*-set and* $A : |\Gamma| \to \omega$–**Set** *is a* $|\Gamma|$*-indexed family of* ω*-sets. Then, the* ω*-set*

$$\sigma(\Gamma, A)$$

is defined as,

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

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

where $\langle \_, \_\rangle$ *is the index for the pair function.* □

The empty context is interpreted as the terminal object of ω–**Set**:

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

Suppose A is a Γ-type interpreted as a $[\![\Gamma]\!]$-indexed family of ω-sets $[\![\Gamma \vdash A : Type_j]\!]$. Then, the valid context $\Gamma, x{:}A$ is interpreted as the ω-set

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

A Γ-term M of type A is in general interpreted as a $[\![\Gamma]\!]$-indexed element of the interpretation of A, *i.e.*, a morphism satisfying the following first projection property.

Definition 7.2.2 (FPP property) *Let* Γ *be an* ω*-set and* $A : |\Gamma| \to \omega$–**Set** *a* $|\Gamma|$*-indexed family of* ω*-sets. 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* $p(\Gamma, A)(\gamma, a) =_{\mathrm{df}} \gamma$*. Intuitively,* $f : \Gamma \to_{FPP} \sigma(\Gamma, A)$ *is a '*Γ*-indexed element of* A*'.* □

Notice that in a Constructions-like calculus, types and objects are mixed up in the sense that types are also objects with kinds as their types. Therefore, a type has a 'double identity' in a model-theoretic interpretation. This is reflected by the correspondence between constant functions to ω–**Set** and a special kind of morphisms in ω–**Set**.

Lemma 7.2.3 *Suppose* $\Gamma \in \omega$–**Set** *and* $K : |\Gamma| \to \omega$–**Set** *is a constant function such that, for some set* X, $K(\gamma) = (X, \omega \times X)$ *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.

Proof The correspondence is given as follows:

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

We have, $f^{\star\circ} = f$ and $g^{\circ\star} = g$. □

Remark By the above lemma, the interpretations of Γ-types (whose types are kinds), which are FPP-morphisms, correspond to $[\![\Gamma]\!]$-indexed family of ω-sets, as kinds in Γ are interpreted as constant functions from $[\![\Gamma]\!]$ to ω–**Set** of the form required in the above lemma (see below). □

7.3 Interpretation of Universes $Type_j$ and Π/Σ-types

Non-propositional types in a context Γ can be interpreted as $[\![\Gamma]\!]$-indexed families of ω-sets. The intuition is that Σ-types correspond to sets of dependent pairs and Π-types to set-theoretic products, which are given by the following two ω-set constructors.

Definition 7.3.1 **(σ_Γ and π_Γ)** *Suppose that* Γ *is an* ω*-set,* $A : |\Gamma| \to \omega$–**Set** *is a* $|\Gamma|$*-indexed family of* ω*-sets, and* $B : |\sigma(\Gamma, A)| \to \omega$–**Set** *is a* $|\sigma(\Gamma, A)|$*-indexed family of* ω*-sets. Then define*

(σ_Γ) *the* $|\Gamma|$*-indexed family of* ω*-sets*

$$\sigma_\Gamma(A, B) : |\Gamma| \to \omega\text{–}\mathbf{Set}$$

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) *the* $|\Gamma|$*-indexed family of* ω*-sets*

$$\pi_\Gamma(A,B) : |\Gamma| \to \omega\text{–}\mathbf{Set}$$

as, for $\gamma \in |\Gamma|$,

$$|\pi_\Gamma(A,B)(\gamma)| =_{\mathrm{df}} \{\, f \in \prod_{a\in|A(\gamma)|} |B(\gamma,a)| \mid \exists n \in \omega.\ n \Vdash_{\pi_\Gamma(A,B)(\gamma)} f \,\}$$

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

where $\prod_{a\in|A(\gamma)|} |B(\gamma,a)|$ *denotes the product of the* $|A(\gamma)|$*-indexed family of sets* $|B(\gamma,a)|$. □

The interpretations of a Σ-type and a Π-type whose principal type is $Type_j$ can then be given as:

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

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

where $[\![\Gamma \vdash A{:}Type_j]\!]$ and $[\![\Gamma, x{:}A \vdash B{:}Type_j]\!]$ are the interpretations of A in Γ and B in $\Gamma, x{:}A$, respectively.

We now explicate how the universes $Type_j$ should be interpreted so that the requirements we gave in section 7.1 can be satisfied. In other words, we interpret the predicative universes in such a way that they satisfy the membership relation $Type_j \in Type_{j+1}$, the inclusion relation $Type_j \subseteq Type_{j+1}$, and the closedness requirement 3 for Π and Σ.

First, large set universes are used to interpret the predicative universes so that the closedness requirement is satisfied.[2] A basic insight here 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[3] $\kappa_0 < \kappa_1 < \kappa_2 < \ldots$ and let V_α be the cumulative hierarchy of sets. Then $Type_j$ corresponds to the following category ω–$\mathbf{Set}(j)$.

Definition 7.3.2 (ω–$\mathbf{Set}(j)$) *Let j be a natural number. ω–$\mathbf{Set}(j)$ is the full subcategory of ω–$\mathbf{Set}$ whose objects are those ω-sets whose carrier sets are in the set universe V_{κ_j}.* □

The categories ω–$\mathbf{Set}(j)$ are locally cartesian closed. More importantly, they are closed under the ω-set constructors σ_Γ and π_Γ, because the set universes V_{κ_j} are models of ZFC set theory.

[2] The idea of interpreting $Type_j$ as large set universes was suggested to the author by Hayashi, Moggi and Coquand.

[3] 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].

Lemma 7.3.3 *σ_Γ and π_Γ are closed for ω–$\mathbf{Set}(j)$, that is, if $A : |\Gamma| \to \omega$–$\mathbf{Set}(j)$ and $B : \sigma(\Gamma, A) \to \omega$–$\mathbf{Set}(j)$, then $\sigma_\Gamma(A, B), \pi_\Gamma(A, B) : |\Gamma| \to \omega$–$\mathbf{Set}(j)$.* □

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

Based on these, we interpret the universe $Type_j$ as the following $|\,[\![\Gamma]\!]\,|$-indexed family of ω-sets, for $\gamma \in |\,[\![\Gamma]\!]\,|$,

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

7.4 Interpretation of Universe *Prop* and Propositions

Propositions are interpreted as a special class of ω-sets which are isomorphic to partial equivalence relations. Here, the notion of modest set [Hyl82,87][Mog85] is essential. The important point is that the category of modest sets $\mathbf{M}$ is closed for arbitrary products and equivalent to the (small) category of partial equivalence relations.

Definition 7.4.1 (modest sets) *A* modest set *is an ω-set A whose realizability relation is a function, i.e.,*

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

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

Remark The category of modest sets is a concrete locally cartesian closed category [Hyl82]. □

Lemma 7.4.2 *π_Γ is closed for the modest sets in the sense that, for any $|\Gamma|$-indexed family of ω-sets $A : |\Gamma| \to \omega$–$\mathbf{Set}$ and any $|\sigma(\Gamma, A)|$-indexed family of modest sets $B : |\sigma(\Gamma, A)| \to \mathbf{M}$, $\pi_\Gamma(A, B)$ is a $|\Gamma|$-indexed family of modest sets, i.e., $\pi_\Gamma(A, B) : |\Gamma| \to \mathbf{M}$.*

Proof We follow [LM88][Hyl87] to prove the lemma. We only have to show that, for $\gamma \in |\Gamma|$, $\Vdash_{\pi_\Gamma(A,B)(\gamma)}$ is a function, *i.e.*, $n \Vdash_{\pi_\Gamma(A,B)(\gamma)} f$ and $n \Vdash_{\pi_\Gamma(A,B)(\gamma)} g$ implies $f = g$. Suppose $a \in |A(\gamma)|$. Take $m \in \omega$ such that $m \Vdash_{A(\gamma)} a$ (m exists as $A(\gamma)$

is an ω-set). Then we have $nm \Vdash_{B(\gamma,a)} f(a)$ and $nm \Vdash_{B(\gamma,a)} g(a)$. As $B(\gamma, a)$ is a modest set, $f(a) = g(a)$. So, $f = g$ as a is arbitrary. $\square$

Although $\mathbf{M}$ 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 $\mathbf{M}$ itself is *not* a small category. If $Prop$ were interpreted as $\mathbf{M}$, there would be no way to justify $Prop \in Type_0$.[4] Fortunately, $\mathbf{M}$ is an *essentially 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. (Recall that R is a partial equivalence relation if R is symmetric and transitive.)

Definition 7.4.3 (PROP) *The category* **PROP** *is the full subcategory of* $\mathbf{M}$ *(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 *and* $\in \subseteq \omega \times Q(R)$ *is the membership relation.* $\square$

Lemma 7.4.4 *There is an equivalence of categories* $\mathbf{back} : \mathbf{M} \to \mathbf{PROP}$ *such that* $\mathbf{back}(A) \cong A$ *for* $A \in \mathrm{Obj}(\mathbf{M})$*, and* $\mathbf{back}(P) = P$ *for* $P \in \mathrm{Obj}(\mathbf{PROP})$.

Proof Define $\mathbf{back} : \mathbf{M} \to \mathbf{PROP}$ 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 $\mathbf{M}$,

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

back is a category equivalence with the inclusion functor $\mathbf{inc} : \mathbf{PROP} \to \mathbf{M}$ 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 $P = (Q(R), \in) \in \mathrm{Obj}(\mathbf{PROP})$, we have

$$\begin{aligned} R_P &= \{ (n, m) \mid \exists [a]_R . m \Vdash_P [a]_R \text{ and } n \Vdash_P [a]_R \} \\ &= \{ (n, m) \mid \exists a \in \omega . (m, n \in [a]_R) \} \\ &= R \end{aligned}$$

[4]This is a little different from the situation in the second-order λ-calculus, where the only universe Type itself does not have a type.

and so $\mathbf{back}(P) = (Q(R_P), \in) = (Q(R), \in) = P$. □

A proposition $\Pi x{:}A.P$ in context Γ is interpreted as a $|\,[\![\Gamma]\!]\,|$-indexed family of objects of **PROP**. The basic idea is that, when $\Pi x{:}A.P$ is a proposition, we first form the product by $\pi_{[\![\Gamma]\!]}$ operator which results in a $|\,[\![\Gamma]\!]\,|$-indexed family of modest sets and then use **back** to 'take it back' into a family of objects in **PROP**.

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

The impredicative universe *Prop* corresponds to the category **PROP**. By lemma 7.4.2 and lemma 7.4.4, the closedness requirement 4 in section 7.1 is satisfied, *i.e.*, *Prop* is closed under arbitrary products. The inclusion requirement $Prop \subseteq Type_0$ is satisfied by the fact that **PROP** is a full subcategory of ω–$\mathbf{Set}(0)$ and, the membership requirement $Prop \in Type_0$ by the fact $\Delta(\mathrm{Obj}(\mathbf{PROP})) \in \mathrm{Obj}(\omega\text{–}\mathbf{Set}(0))$, where $\Delta : \mathbf{Set} \to \omega\text{–}\mathbf{Set}$ is the embedding functor defined in the last subsection.

Based on these, we interpret the universe *Prop* as the following $|\,[\![\Gamma]\!]\,|$-indexed family of ω-sets, for $\gamma \in |\,[\![\Gamma]\!]\,|$,

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

7.5 Discussions

In the last section, we give a sketch of a model-theoretic semantics of the calculus, which explicates how the intuitive understanding of the main constructs in **ECC** can be captured in the ω–**Set** framework. From this, we can gain some further understanding of the calculus. For example, in the interpretation in the framework of ω-sets and modest sets, empty types exist. We can see that the proposition $\Pi x{:}Prop.x$ (the logical constant **false**) is interpreted as the empty ω-set $(\emptyset, \emptyset)$. This conforms with the theorem 6.1.5 of logical consistency that there is no term which inhabits $\Pi x{:}Prop.x$, or putting in another way, $\Pi x{:}Prop.x$ is an empty type. This is an important feature of such a model-theoretic interpretation and one of the reasons that we view such a model as appropriate. There are other possible and reasonable models. For instance, we can give a truth-value model of **ECC** where propositions are interpreted as 0 or 1 (*c.f.*, [Coq89]). Some other models (*e.g.*, domain-theoretic ones) do not capture the essential properties of the calculus like logical consistency. Semantic models are often used to justify the consistency of a logical calculus and to guide and justify new syntactic extensions. Indeed, an ω–**Set** model construction was used to justify the idea of including propositions as types at an earlier stage of development of **ECC** [Luo88c,a].

The set-theoretic flavor of such a semantics makes possible a deeper understanding of the calculus. It 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 behind the main constructs of the formal calculus can be understood set-theoretically as reflected by the model.

Another insight one may gain from the above set-theoretic interpretation is about how to formalize mathematical problems *adequately*. As we explained in section 2.2.3, one of the basic motivations for introducing predicative type universes is to allow formalization of the notion of an *arbitrary set*. Our interpretation of predicative type universes by large set universes supports such an idea from set-theoretic point of view. In an (intuitionistic) set-theoretic model, propositions in an impredicative universe are interpreted as rather small sets instead of arbitrary sets. Girard's paradox gives us a hint that *an arbitrary set should not be formalized as a proposition which may be formed impredicatively*. 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.

Understanding this distinction between data types (sets) and logical formulas (propositions) is very important both in theoretical researches and practical applications. In practice, based on the above view, an adequate formalization should not take a proposition as a representation of an arbitrary set. In theoretical researches, such a view may lead to a better understanding of a formal system. In fact, our conservativity conjecture discussed in section 6.1.3 was originally proposed partly based on the above set-theoretic understanding of the calculus. The recent result of Geuvers and Berardi about non-conservativity discussed in section 6.1.3 gives another support of such a view from another angle. It is obvious that a formal treatment of these is called for and more researches are needed to make it well-understood.

It should be possible to give a full and detailed interpretation of the calculus based on the ideas sketched in this chapter. We remark here that defining such a detailed semantics for a Constructions-like calculus is a very sophisticated work. Because of the existence of the conversion rule, there may be more than one derivation of a derivable judgement. The induction principle used to define the semantics has to be carefully considered and examined. It seems possible to have a unified way to solve these problems in general for rich theories of dependent types, instead of using somehow ad hoc methods; but this needs further research (*c.f.*, [Str88][Luo89b]).

Chapter 8

Theory Abstraction in Proof Development

ECC is a formal calculus which embodies rich structural facilities as well as a strong logical mechanism. One of the pragmatic applications of the theory is to formalize mathematical problems and to be used as a basis for proof development in (interactive) theorem proving.

ECC, like many type theories including the Automath type theory, Martin-Löf's type theory and the calculus of constructions, is a basic calculus which can be used to do (interactive) proof development based on a proof checker (for example, the LEGO system [Pol89][LPT89], which supports **ECC** as well as some other related type systems). However, **ECC** has much stronger structural mechanisms which support more powerful reasoning facilities for proof development. In particular, the Σ-types and type universes provide a nice abstraction mechanism which can be used to do abstract structured reasoning in a desirable way, as we shall discuss below.

In this chapter, we discuss using *theory abstraction* to develop large proofs by structured abstract reasoning. We show how mathematical theories can be formalized by the abstraction and modularization facilities which the calculus provides and how abstract reasoning and structured reasoning can be done in our setting.[1]

8.1 A Notion of Theory

We need a notion of theory in proof development to do large proof development in a structured and modular way, just as modular programming in large program development. Obviously, some good form of theory mechanism is called for to express this intuitive notion of theory in people's mind so that it can provide us a nice approach to proof development.

[1]The idea of structuring theories in proof development was suggested to the author by Burstall and the current presentation also benefits from discussions with Coquand, Taylor and Pollack.

What is a theory? Although people feel there is a rather clear intuition about this, this question can not be answered precisely unless we set up a formal mechanism of theory manipulation. In fact, 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 between *concrete* theories and *abstract* theories. A concrete theory in the calculus, as we have pointed out in section 6.1, is presented as a valid context. Proved theorems of such a concrete theory are then a set of provable formulas in the theory. 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. One might formalize a theory of semigroups as a concrete theory as follows:

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

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.

It is interesting to see that we can express a notion of *abstract theory* as well by using Σ-types and type universes, which provides a good modulization mechanism for abstract and structured reasoning. We first explain the basic idea of using strong sum to express abstract structures and mathematical theories through an example. Instead of formalizing a theory of semigroups as a concrete theory as above, we express an *abstract* theory of semigroups as consisting of two parts:

- an (abstract) *signature presentation* $\mathrm{Sig\_}SG \equiv \Sigma X{:}Type_0.X \to X \to X$;
- the (abstract) *axiom* which is a predicate $\mathrm{Ax\_}SG$ over $\mathrm{Sig\_}SG$ which, when given any structure s of type $\mathrm{Sig\_}SG$, returns the associativity axiom for s.

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

$$\mathrm{Mod\_}SG \equiv \Sigma s{:}\mathrm{Sig\_}SG.\mathrm{Ax\_}SG(s)$$

Then, to postulate an arbitrary semigroup is just to assume a context $sg{:}Mod\_SG$. The projection operators can be used to extract the components of any semigroup (*i.e.*, an object of type $\mathrm{Mod\_}SG$). One can then prove (abstract) theorems about arbitrary semigroups. Such abstract theorems constitute a predicate Thm_SG over Sig_SG, which can in general be expressed as the following form (say, n theorems have been proved):

$$\text{Thm\_}SG \equiv \lambda s{:}\text{Sig\_}SG.\ P_1\ \&\ \ldots\ \&\ P_n$$

Their (abstract) proofs are then a function Prf_SG of type $\Pi sg{:}\text{Mod\_}SG.\ \text{Thm\_}SG(\pi_1(sg))$. That is, given any concrete semigroup structure (*i.e.*, a type and a binary operation over the type which satisfies the associativity axiom), Prf_SG will result in the proofs of the concrete versions of the theorems for the given semigroup structure.

We now generalize the above ideas to the following definition of (abstract) theory.

Definition 8.1.1 (abstract theories) *A presentation of an abstract mathematical theory T in* **ECC** *consists of four components*

$$T = (\text{Sig\_}T,\ \text{Ax\_}T,\ \text{Thm\_}T,\ \text{Prf\_}T)$$

where

- Sig_T *is called the* signature presentation *of T, which is in general a Σ-type;*
- Ax_T *is called the* abstract axioms *of T, which is a predicate over* Sig_T *(typically, of type* Sig_$T \to Prop$*);*
- Thm_T *is called the (proved)* abstract theorems *of T, which is a predicate over* Sig_T *(typically of type* Sig_$T \to Prop$*) and generally of the form* $\lambda s{:}\text{Sig\_}T.\ P_1\ \&\ P_2\ \&\ \ldots\ \&\ P_n$*; and*
- Prf_T *is called the* abstract proofs *of the theorems of T, which is of type* $\Pi t{:}\text{Mod\_}T.\text{Thm\_}T(\pi_1(t))$*, where*

$$\text{Mod\_}T \ \equiv\ \Sigma s{:}\text{Sig\_}T.\text{Ax\_}T(s)$$

is the type of the T-structures satisfying the T-axioms (the models of T).

Somehow abusing the terminology, we often call Mod_T *the abstract theory.* □

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 (*e.g.*, the category of all small categories) in a similar way. Note that predicative universes are important in formalizing abstract theories. □

We shall show below that the notion of (abstract) theory presented above nicely supports an approach to abstract reasoning and structured reasoning.

8.2 Abstract Reasoning

The idea of abstract reasoning[2] 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:

$$\mathrm{Thm\_}SG \equiv \lambda s{:}\mathrm{Sig\_}SG.\ P_1\&...\&P_n$$

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

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[3]

$$\mathrm{Prf\_}Nat\_SG \equiv \mathrm{Prf\_}SG((nat, +), ass\_nat\_plus)$$

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. Based on such a mechanism, one may build up a *theory base* consisting of well-organized (abstract) theories with proved theorems which can be used by users in many different ways. Such a theory base would be very useful for large proof development tasks. □

8.3 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.

[2]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.

[3]From now on, we elide the explicit typing in the pair operator for notational convenience.

8.3.1 Proof inheritance

Proof inheritance between theories through theory morphisms [TL88][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

$$f : \mathrm{Sig\_}T \to \mathrm{Sig\_}T'$$

$$g : \Pi s{:}\mathrm{Sig\_}T.\ \mathrm{Ax\_}T(s) \to \mathrm{Ax\_}T'(f(s))$$

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 $\mathrm{Prf\_}T'$ is the (abstract) proofs of the theorems proved for T' which is of type $\Pi t'{:}T'.\ \mathrm{Thm\_}T'(\pi_1(t'))$. Then, the corresponding (abstract) theorems in T

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

are proved by the following proofs inherited from $\mathrm{Prf\_}T'$:

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

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.

8.3.2 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:

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

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

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 $\mathrm{Ax\_}SG(s)$ and $\mathrm{Ax\_}AG(g)$ are the propositions expressing the axioms of theories for semigroups and abelian groups.

ringGen can then be defined as

$$\begin{array}{rcl} ringGen & \equiv & \lambda X{:}Type_0 \\ & & \lambda *{:}SGwrt(X)\ \lambda p{:}\mathrm{Ax\_}SG(X,*) \\ & & \lambda(+,0,'){:}AGwrt(X)\ \lambda q{:}\mathrm{Ax\_}AG(X,+,0,') \\ & & \lambda d{:}P_{DISTR}.\ ((X,+,0,',*), and\_intro(p,q,d)) \end{array}$$

which is of type

$$\begin{array}{l} \Pi X{:}Type_0 \\ \Pi *{:}SGwrt(X)\ \Pi p{:}\mathrm{Ax\_}SG(X,*) \\ \Pi g{:}AGwrt(X)\ \Pi q{:}\mathrm{Ax\_}AG(X,g) \\ \Pi d{:}P_{DISTR}.\ \mathrm{Mod\_}Ring \end{array}$$

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

Note that *SGwrt* and *AGwrt* are what are often called '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. It is easy to see that such a facility is also useful for structured programming.

Remark There are several different ways to control structure sharing which appear in programming and specification languages ML [HMM86][Mac86], Pebble [LB88] and Clear [BGog80] (see [Bur84] for a simple explanation). Note that, as Thierry Coquand pointed out to the author, propositional equalities (*e.g.*, Leibniz's equality) can *not* be used to express sharing constraints in the style of ML, since by structure sharing people mean that two substructures are *the same* in a quite strong sense which can not be expressed adequately by propositional equalities. For example, the following term

$$\lambda X{:}Type_0 \lambda Y{:}Type_0 \lambda z{:}(X =_{Type_0} Y).\ \forall x{:}X \exists y{:}Y.x =_X y$$

is *not* well-typed because the variable y is not of type X. The required proof z of $X =_{Type_0} Y$ does *not* play a role to indicate that X and Y are the same (*i.e.*, convertible) as a sharing constraint does. □

8.4 Discussions

We have shown above that Σ-types and type universes provide expressive mechanisms to express a notion of (abstract) theory for structured abstract reasoning.

As well-known, existential types (or weak sums) [MP85][Rey83][Pra65] can be defined in the calculus of constructions [CH85], as we defined it in section 6.1.1 — the logical existential quantifier. It is interesting to note that similar constructions can be given at the predicative levels of **ECC** [Luo89a], as we shall show in section 9.2.2, which are useful to express abstract data types in programming [MP85].

However, existential types are *not* useful to express mathematical theories because they 'hide' the proofs: 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. To express mathematical theories as we have shown above, strong sums (Σ-types) are needed. 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 in our formal calculus, which provides a solid basis to guarantee the correctness of using the theory mechanism based on it. There is another approach to theory structuring [SB83][BLuo88][HST89] borrowing ideas from research in algebraic specification languages like Clear [BGog80]. The ideas in [SB83] are used in Cambridge LCF theory mechanism [Pau87]. This latter approach may be called '*theories as meta-values*', as there are theory operations to 'put theories together', which are performed at the meta-level of implementation. In the Automath project, ideas like telescope of organizing mathematical texts through manipulating contexts were considered [dB80][Zuc75]. Further research and experience are needed to show what is necessary and whether it is possible to combine these ideas together.

As a final remark on the theory mechanism we described above, we note that the notion of abstract theories may even be internally formalized in **ECC**.[4] We may describe it as an (abstract) theory, as the following example shows.

Example An internal description of the notion of abstract theory:

- The class of signature presentations SIG can be represented as $Type_1$:

$$\mathrm{SIG} =_{\mathrm{df}} Type_1$$

[4]Thanks to Taylor and Pollack for discussions on this [TL88].

- The signature-parameterized classes of abstract axioms may be represented as:
$$\mathrm{AX} =_{\mathrm{df}} \lambda s{:}\mathrm{SIG}.\ s \rightarrow Prop$$

- The class of abstract theories can be represented by the following Σ-type:
$$\mathrm{ABS} =_{\mathrm{df}} \Sigma s{:}\mathrm{SIG}.\mathrm{AX}(s)$$
Its constructor and destructors are:
$$\mathrm{Abs} =_{\mathrm{df}} \lambda s{:}\mathrm{SIG}\lambda ax{:}\mathrm{AX}(s).(s, ax)$$
$$\mathrm{Sig} =_{\mathrm{df}} \lambda T{:}\mathrm{ABS}.\pi_1(T)$$
$$\mathrm{Ax} =_{\mathrm{df}} \lambda T{:}\mathrm{ABS}.\pi_2(T)$$

- Given an abstract theory T of type ABS, we can represent the set of T-structures satisfying the T-axioms (the models of T) as:
$$\mathrm{Mod}(T) =_{\mathrm{df}} \Sigma s{:}\mathrm{Sig}(T).\mathrm{Ax}(T)(s)$$
Mod is of type ABS $\rightarrow Type_1$.

- Let $P_i : \mathrm{Sig}(T) \rightarrow Prop$ $(i = 1, ..., n)$ be the proved abstract theorems of an abstract theory T : ABS. They can be represented as:
$$\mathrm{Thm}(T) =_{\mathrm{df}} \lambda s{:}\mathrm{Sig}(T).\ P_1(s)\&...\&P_n(s)$$
Thm is of type $\Pi T{:}\mathrm{ABS}.\ \mathrm{Sig}(T) \rightarrow Prop$.

- Let $p_i(s)$ be the proof of $P_i(s)$ $(i = 1, ..., n)$ above. The proofs of the abstract theorems can be represented as:
$$\mathrm{Prf}(T) =_{\mathrm{df}} \lambda t{:}\mathrm{Mod}(T).\ and\_intro(p_1(\pi_1(t)), ..., p_n(\pi_1(t)))$$
where *and_intro* is the proof operator corresponding to the &-introduction for n formulas. Prf is of type $\Pi T{:}\mathrm{ABS}\Pi t{:}\mathrm{Mod}(T).\ \mathrm{Thm}(T)(\pi_1(t))$.

The notion of theory morphisms can also be formalized to represent the idea of proof inheritance.

- The set of morphisms between abstract theories T and T' of type ABS can be represented as:
$$\mathrm{Mor}(T, T') =_{\mathrm{df}} \Sigma f{:}\mathrm{Sig}(T) \rightarrow \mathrm{Sig}(T').\ \Pi s{:}\mathrm{Sig}(T).\mathrm{Ax}(T)(s) \rightarrow \mathrm{Ax}(T')(f(s))$$
Mor is of type ABS $\rightarrow$ ABS $\rightarrow Type_1$.

- Given a morphism $m = (f, g)$ of type $\mathrm{Mor}(T, T')$, the proved abstract theorems $\mathrm{Thm}(T')$ in T' are transformed by the morphism m into the corresponding abstract theorems $\mathrm{ThmTrans}(T, T')(m)$ in T:

$$\mathrm{ThmTrans}(T, T')(m) =_{\mathrm{df}} \lambda s{:}\mathrm{Sig}(T).\ \mathrm{Thm}(T')(\pi_1(m)(s))$$

  ThmTrans is of type $\Pi T{:}\mathrm{ABS}\Pi T'{:}\mathrm{ABS}\Pi m{:}\mathrm{Mor}(T, T').\ \mathrm{Sig}(T) \to \mathit{Prop}$.

- The transformed theorems above are proved by the following (transformed) proofs:

$$\mathrm{PrfTrans}(T, T')(m) =_{\mathrm{df}} \lambda t{:}\mathrm{Mod}(T).\ \mathrm{Prf}(T')(\pi_1(m)(\pi_1(t)), \pi_2(m)(\pi_1(t), \pi_2(t)))$$

  PrfTrans is of type

$$\Pi T{:}\mathrm{ABS}\Pi T'{:}\mathrm{ABS}\Pi m{:}\mathrm{Mor}(T, T')\ \Pi t{:}\mathrm{Mod}(T).\mathrm{Thm}(T')(\pi_1(m)(\pi_1(t)))$$

□

Remark The above example of internalization shows that the calculus **ECC** is very expressive. Besides this, it is interesting to note that, in the internal formalization above, types at the *fourth level* of the type hierarchy (of type $Type_2$) are necessarily used. From this, one may expect that, in some more sophisticated applications, higher type universes are also useful. □

Chapter 9

Some Issues in Program Specification and Programming

In this chapter, we briefly discuss how to view the extended calculus of constructions **ECC** as (a core of) a programming logic. By a programming logic, we mean a formal system which integrates facilities of programming with a consistent logic so that program specifications can be expressed and program development can be discussed in the system. **ECC** may be viewed as a programming logic according to the following:

- There is a powerful higher-order logic embedded in **ECC** (as shown in chapter 6);
- The λ-abstractions and the rich type structures provide core mechanisms to support typeful functional programming;
- Σ-types in **ECC** provide a basic adequate mechanism for program specification and program development.

A full investigation of using **ECC** as a programming logic is out of the range of this thesis. There are many interesting problems in this aspect to be further studied. We shall concentrate on some specific aspects of using **ECC** to do program specification and programming and try to show its potential power as a programming logic.

Particularly, we show that Leibniz's equality (see definition 6.1.4) can be used in program specifications to model computation as it reflects the definitional equality (theorem 9.1.1). This shows that there is no need to add a new extra propositional equality to the theory to reflect definitional equality. Comparisons with Martin-Löf's type theory in this aspect are discussed.

We also discuss how the predicative levels of the calculus provide us with programming facilities. We first show that Leivant's finitely stratified polymorphic λ-calculus can be embedded in the predicative levels of **ECC**, which indicates that the predicative levels of **ECC** provide programming power. Then, we discuss how existential types may be defined at the predicative levels to express abstract data types.

9.1 Program Specification and Equality Reflection

As we have discussed in the introduction, Σ-types provide a basic adequate mechanism for program specifications. Most of the previous research in this aspect is mainly based on Martin-Löf's type theories. Since the calculus of constructions does not have Σ-types, how to use it in program specifications has not been investigated.[1] Particularly, how to take advantage of the logical power provided by impredicativity in applications like program specification has not been paid enough attention to investigate. After showing how program specifications can be expressed in **ECC** using Σ-types, we show that Leibniz's equality can be used to reflect the definitional equality (conversion); this can be seen as an example showing the advantage of having a powerful higher-order logic.

Following the idea that problems (specifications) correspond to types and solutions (programs, implementations) to elements of specifications expressed as types, a program specification in **ECC** can be expressed in the following basic form as a Σ-type:

$$S \equiv \Sigma f{:}A \to B.\ P(f)$$

where A and B are the types of inputs and outputs of the programs to be specified, respectively, and $P(f)$ is a proposition describing the properties that the correct implementations are required to satisfy. An term I which provably inhabits this specification (*i.e.*, $\vdash I : S$) shall be a pair (F, p) constituting a function (program) F and a correctness proof that F satisfies the required properties. It is obvious that our idea of lifting propositions as types so that propositions can be used in Σ-types is essential for such an idea of program specification to be expressed in **ECC**.

However, such a basic structuring mechanism is in fact not quite enough for specifications yet. Something more has to be considered. For example, based on the above idea, a specification of the identity function of type $A \to A$ (supposing A to be a $\langle\rangle$-type) would be the following:

$$ID^? =_{\mathrm{df}} \Sigma f{:}A \to A.\ \Pi x{:}A.f(x) = x$$

Then, one must ask: what is the equality $=$ in the above specification? In fact, this is an important problem we must consider when claiming that program specifications can be expressed in a type theory like **ECC**. When we write down the equality above, we certainly mean that it is modeling the computational equality in our mind ($f(a)$ and a are computationally equal, *i.e.*, they both compute the same value as their results).

When type theories (λ-calculi in general) are viewed as programming languages, computation is modeled by reduction ($\triangleright$) and the computational equal-

[1]The impression that strong sum is inconsistent with impredicativity and the difficulty of adding Σ-types seems to have prevented people from considering this aspect.

ity by the definitional equality conversion ($\simeq$).[2] However, we certainly can not put $\simeq$ to replace $=$ in $ID^?$, as $\simeq$ is not a formal entity directly expressible in the calculus. In other words, we must have a propositional equality which can be used to model the computational equality (*i.e.*, conversion).

In Martin-Löf's type theories, an extra propositional equality (equality type) is used, as we mentioned in the introduction. The weak intensional equality type in [ML73] reflects the definitional equality; the strong extensional equality type in [ML84] is equivalent to the judgemental equality.

In our theory **ECC**, thanks to its strong power, there is no need to add a new propositional equality to model the definitional equality. We show that Leibniz's equality, $=_A$ (see definition 6.1.4), does reflect the conversion relation.

Theorem 9.1.1 (equality reflection) *Suppose* $\vdash a_1 : A$ *and* $\vdash a_2 : A$. *Then,* $a_1 \simeq a_2$ *if and only if* $\vdash M : a_1 =_A a_2$ *for some term* M.

Proof Necessity. If $a_1 \simeq a_2$, we have by the type conversion rule,

$$\vdash \lambda P{:}A \to Prop \lambda x{:}Pa_1.x : (a_1 =_A a_2)$$

Sufficiency. If $\vdash M : (a_1 =_A a_2)$, by strong normalization theorem, we may assume M, A, a_1 and a_2 are all in normal form. So, M must be of the form $\lambda P{:}A \to Prop.\lambda x{:}Pa_1.M'$ such that $P{:}A \to Prop, x{:}Pa_1 \vdash M' : Pa_2$. Since M' is in normal form, it must be a base term. Let y be the key variable of M'. y can not be P. We have $y \equiv x$. Therefore, it must be the case that $M' \equiv x$, for otherwise, $Qz{:}A_1.B_1 \preceq Pa_2$ for some A_1 and B_1, where $Q \in \{\Pi, \Sigma\}$, which is impossible. Noticing that the only rule which can be used to derive $P{:}A \to Prop, x{:}Pa_1 \vdash x : Pa_2$ is the conversion rule, we conclude $a_1 \simeq a_2$ by Church-Rosser theorem. $\Box$

This result of equality reflection gives a justification of the adequacy of using Leibniz's equality in program specifications to model definitional (computational) equality. For example, the specification of identity function can now be given as follows:

$$ID \equiv \Sigma f{:}A \to A.\ \forall x{:}A.f(x) =_A x$$

This specification is adequate. For any implementation (id, p) of ID ($\vdash (id, p) : ID$) and for any object a of type A ($\vdash a : A$), we have by the above theorem of equality reflection,

$$id(a) \simeq a$$

Remark The above reflection result was realized and proved by the author when

[2] Here, we take a simple point of view. One may consider more sophisticated computational equality like those subject to observational equivalence. But, reduction and conversion are the more basic notions which are incorporated in other notions of computation and computational equivalence.

considering the adequacy problem of using Leibniz's equality to reflect computational equality (*c.f.*, [Bur89a]).[3] A nice consequence is that, unlike Martin-Löf's type theories, we no longer need to add a new equality to our calculus when we use Σ-types to do specifications. This is one of the benefits we gain from combining impredicativity with (predicative) Σ-types. □

To close this section, we conclude that Σ-types based on the idea of lifting propositions as types and the result of equality reflection provide a basic adequate mechanism for program specifications in **ECC**.

9.2 Programming at Predicative Levels

As **ECC** is a very rich type system, there may be different ways to view it as a programming logic and further research is needed to investigate these possibilities. For example, programming facilities are provided by its underlying λ-calculus, at both the impredicative level and the predicative levels. It is well-known by results about the polymorphic λ-calculus (*c.f.*, [Gir72][Rey74][BB85]) that the impredicative level of **ECC** provides programming power. Theoretically speaking, the class of representable functions in F^ω are exactly those which are provably total in the higher-order arithmetic [Gir73].

On the other hand, the predicative levels of **ECC** also provide programming facilities. Recently, Leivant [Lei89] has studied a stratified variant of the second-order polymorphic λ-calculus (called $S2\lambda^\omega$) and shown that the functions representable in the finitely-stratified λ-calculus are precisely the super-elementary functions.[4] We shall show below that Leivant's finitely-stratified polymorphic λ-calculus can be embedded in the predicative levels of **ECC** through an easy interpretation. This shows that the predicative levels of **ECC** provide us programming power as well.

It is then possible to view **ECC** as a programming logic in the following way:

- the embedded logic resides in the impredicative universe ($Prop$);
- the programming facilities are provided by the predicative universes; and
- the predicative levels also provide structural power for programming and specification.

Such a view has an advantage that there can be a clear conceptual distinction between data types and logical formulas. In our point of view, data types are

[3] After we had formulated and proved the reflection result (and reported it in LICS'89), the author was informed that Martin-Löf had a similar proof of this fact for the first version of his type theory with a type of all types [ML71]. Although the system Martin-Löf considered is inconsistent, his proof is essentially the same as what we give.

[4] Leivant also shows that the functions representable by the stratified polymorphic λ-calculus up to ω^ω are exactly the primitive resursive functions.

represented by non-propositional types. Propositions are just used to stand for logical formulas; they are not data types.

9.2.1 Embedding stratified polymorphism in ECC

Leivant's finitely stratified polymorphic λ-calculus $S2\lambda^\omega$ is similar to the second-order λ-calculus except that the types are classified into levels numbered by natural numbers. We refer to [Lei89] for the original presentation of $S2\lambda^\omega$.

$S2\lambda^\omega$ can be formulated by explicit typing terms as follows, from which it is easy to see that it can be represented at the predicative levels of **ECC**.

- Type expressions and their levels:
  1. Type variables at the jth level ($t{:}Type_j$) are type expressions of level j;
  2. Arrow types: If A and B are type expressions of levels i and j respectively, $A \to B$ is a type expression of level $max\{i, j\}$, *i.e.*, (assuming that all of the free variables in A and B are in Γ, similar below)

     $$\frac{\Gamma \vdash A : Type_i \quad \Gamma \vdash B : Type_j}{\Gamma \vdash A \to B : Type_{max\{i,j\}}}$$

     This is a derivable rule in **ECC**.
  3. Universal quantification: If B is a type expression of level i and t a type variable of level j, then $\forall t{:}Type_j.B$ is a type expression of level $max\{i, j+1\}$, *i.e.*,

     $$\frac{\Gamma, t{:}Type_j \vdash B : Type_i}{\Gamma \vdash \forall t{:}Type_j.B : Type_{max\{i,j+1\}}}$$

     Mapping $\forall$ as Π, this is a derivable rule in **ECC**.
- (Object) expressions and their typings:
  1. Individual variables associated for each type expression A ($x{:}A$) are object expressions whose types are A;
  2. λ-abstraction (of objects): If M is an expression of type B and x is an individual variable of type A, then $\lambda x{:}A.M$ is an expression of type $A \to B$, *i.e.*,

     $$\frac{\Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : A \to B}$$

     Note that an individual variable does not occur in a type expression, so the above rule is derivable in **ECC**.

3. Object application: If M is an expression of type $A \to B$ and N is an expression of type A, then MN is an expression of type B, *i.e.*,

$$\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

This rule is a special case of the rule (*app*) in **ECC**.

4. Λ-abstraction (of types): If M is an expression of type B and t is a type variable of level j, then $\Lambda t{:}Type_j.M$ is an expression of type $\forall t{:}Type_j.B$, *i.e.*,

$$\frac{\Gamma, t{:}Type_j \vdash M : B}{\Gamma \vdash \Lambda t{:}Type_j.M : \forall t{:}Type_j.B}$$

In **ECC**, we do not distinguish type abstraction with object abstraction. Just mapping $\Lambda t{:}Type_j.M$ as $\lambda t{:}Type_j.M$, the above rule is derivable in **ECC**.

5. Type application: If M is an expression of type $\forall t{:}Type_j.B$ and A is a type expression whose level is less than or equal to j, then MA is an expression of type $[A/t]B$, *i.e.*,

$$\frac{\Gamma \vdash M : \forall t{:}Type_j.B \quad \Gamma \vdash A : Type_i}{\Gamma \vdash MA : [A/t]B} \quad (i \leq j)$$

By rules ($\preceq$) and (*app*), the above rule is derivable in **ECC**.

For a type expression A, $Nat(A)$ abbreviates the type expression $A \to (A \to A) \to A$. The nth numeral at the level j is represented by the expression

$$\overline{n}^j =_{\text{df}} \Lambda t{:}Type_j \lambda x{:}t \lambda s{:}t \to t.s^{[n]}x$$

where $s^{[0]}x =_{\text{df}} x$ and $s^{[i+1]}x =_{\text{df}} s(s^{[i]}x)$. The type of $\overline{n}^j$ is $\forall t{:}Type_j.Nat(t)$.

Definition 9.2.1 (slant numerals) *Let A_{i_k} be a type expression of level i_k ($k = 1, ..., m$) and A a type expression of level i. An expression M* represents slantwise *(at levels $(i_1, ..., i_m, i)$) an m-ary recursive function f if, for any natural numbers $n_1, ..., n_m, n$,*

$$f(n_1, ..., n_m) = n \quad \text{if and only if} \quad M\overline{n_1}^{i_1}...\overline{n_m}^{i_m} \simeq \overline{m}^i$$

□

The Grzegorczyk classes $\mathcal{E}_k$ ($k \geq 0$) classify the class of primitive recursive functions [Grz53][Ros84]. $\mathcal{E}_k$ consists of the recursive functions generated by function composition and bounded recursion from zero, successor, the projection functions and the function F_k, where $F_0 =_{\text{df}} suc$ (the successor function) and $F_{i+1}(x) =_{\text{df}} F_k^{[x]}(x)$ with $F^{[x]}$ being the xth interate of F. In particular, $\mathcal{E}_3$ is the class of *elementary* functions and $\mathcal{E}_4$ is called the class of *super-elementary* functions.

Theorem 9.2.2 *(Leivant [Lei89]) The recursive functions representable slantwise in $S2\lambda^\omega$ are exactly the super-elementary functions.* □

By Leivant's result, we have

Corollary 9.2.3 *The super-elementary functions are representable at the predicative levels of* **ECC**. □

9.2.2 Existential types and discussion

Besides the basic programming power of expressing functions, the rich type structures at the predicative levels provide structural mechanisms for modular programming. For example, Σ-types supports a form of module mechanism (*c.f.*, [BLam84][Mac86]) and parameterized sharing [Bur84][BLam84] can be expressed. Furthermore, we would like to show below that existential types can also be defined at the predicative levels of the theory [Luo89a] which can be used to express abstract data types [MP85].

Following the idea of defining the (impredicative) existential quantifier (see section 6.1.1), we can define the ith level existential-type constructor as follows:

$$\begin{aligned} \exists^i \;\; =_{\mathrm{df}} \;\; & \lambda A{:}Type_{i+1}\ \lambda B{:}A \to Type_i. \\ & \Pi C{:}Type_i.(\Pi x{:}A.(B(x) \to C)) \to C \end{aligned}$$

which is of type $\Pi A{:}Type_{i+1}.((A \to Type_i) \to Type_{i+1})$. The introduction and elimination operators rep^i and $\mathrm{abstype}^i$ can be similarly defined as,

$$\begin{aligned} \mathrm{rep}^i \;\; =_{\mathrm{df}} \;\; & \lambda A{:}Type_{i+1}\ \lambda B{:}A \to Type_i \\ & \lambda x{:}A\ \lambda y{:}B(x) \\ & \lambda C{:}Type_i\ \lambda p{:}(\Pi z{:}A.(B(z) \to C)).\ p(x,y) \end{aligned}$$

which is of type

$$\Pi A{:}Type_{i+1} \Pi B{:}A \to Type_i \Pi x{:}A \Pi y{:}B(x).\ \exists^i(A,B)$$

and

$$\begin{aligned} \mathrm{abstype}^i \;\; =_{\mathrm{df}} \;\; & \lambda A{:}Type_{i+1}\ \lambda B{:}A \to Type_i \\ & \lambda M{:}\exists^i(A,B)\ \lambda C{:}Type_i \\ & \lambda N{:}\Pi x{:}A.(B(x) \to C). \\ & M(C,N) \end{aligned}$$

which is of type

$$\Pi A{:}Type_{i+1} \Pi B{:}A \to Type_i \Pi M{:}\exists^i(A,B) \Pi C{:}Type_i \Pi N{:}\Pi x{:}A.(B(x) \to C).\ C$$

According to the notation of [MP85], we may write $\mathrm{rep}^i(A, B, a, b)$ as

$$\mathrm{rep}_{\exists^i x:A.B(x)}(a, b)$$

and $\mathrm{abstype}^i(A, B, M, C, N)$ as

$$\mathrm{abstype}^i\ x{:}A \text{ with } y{:}B(x) \text{ is } M \text{ in } N(x, y)$$

They satisfy the desired properties such as

$$\mathrm{abstype}^i\ x{:}A \text{ with } y{:}B(x) \text{ is } \mathrm{rep}_{\exists^i x:A.B(x)}(a, b) \text{ in } N(x, y) \rhd_\beta [b/y][a/x]N(x, y)$$

Note that, unlike the propositional existential quantifier $\exists$, these 'weak sums' are defined at the *predicative* levels. They can similarly be used to play the role of information hiding and thus of expressing abstract data types for programming. This seems to show that, for expressing abstract data types, the impredicativity is not important. Of course, we do not have these predicative types as values in the strong sense of [MP85]; *e.g.*, $\exists^0 x{:}A.B$ (*i.e.*, $\exists^0(A, B)$) is of type $Type_1$ but *not* of type $Type_0$. These existential types are useful for describing abstract data types in programming.

ECC lacks recursive data types to support the ordinary recursive programming style. Whether the data types like those of natural numbers, lists *etc.* which are definable by coding techniques [BB85][CH85] are suitable for real programming is still to be further investigated. One may extend the calculus with inductive types. For example, it is possible to introduce the types of natural numbers, lists *etc.* as in Martin-Löf's type theories. Another way may be to introduce a general inductive-type constructor μ, as considered by Coquand and Mohring [CM89] and Ore [Ore89], with the following formation rule:

$$\frac{\Gamma, x{:}Type_j \vdash A : Type_j}{\Gamma \vdash \mu x{:}Type_j.A : Type_j} \quad (j \in \omega)$$

where the free occurrence of x in A must be strictly positive, together with other introduction and elimination rules. Then, one can define the usual concrete data types like those of natural numbers, lists, trees, *etc.*. Coquand and Mohring [CM89] studied how inductive types can be extended as predicative types and give a nice account of the issue. Ore [Ore89] studies how **ECC** may be extended with inductive types based on Coquand's idea and how the set-theoretic model for **ECC** may be extended to inductive types. We refer to [CM89] and [Ore89] for further details. This interesting direction of research is in progress.

Chapter 10

Conclusions and Further Research

In this thesis, the Extended Calculus of Constructions **ECC** has been presented and studied as a promising calculus for formalization of mathematics, computer-assisted reasoning and program specification. There are some open problems and further research topics which we feel interesting and summarize as follows.

Concerning about the theory **ECC** itself, we have left some open problems to be solved. The relationships of the embedded higher-order logic with other logical systems are to be investigated. The conservativity conjecture discussed in section 6.1.3 is one of the interesting problems in this aspect. The proof-theoretic power of the theory **ECC** is unknown; it seems to be much smaller than ZF set theory. In the realizability model described in this thesis, large set universes are used to interpret the predicative universes. It may be possible to give a *small* model without using large set universes.

Some possible extensions to the calculus **ECC** may be considered interesting. Inductive types have been mentioned in section 9.2.2 and are useful for applications in program development and theorem-proving. One might consider the problem of including η/π-conversions into the theory, which are useful for some technical reasons; whether they are important in practice is to be seen. For this, we refer to a recent relevant work by Salvesen [Sal89] which considers the Church-Rosser property of LF with η-conversion. Another extension of purely theoretical interest might be to extend the predicative levels to larger ordinals, say ω^ω. This might be interesting when considering the problem of proof-theoretic power of the predicative levels (*c.f.*, [Lei89]).

Further research about semantical models of rich type theories with dependent types needs to be carried out in order to have a disciplined way to define semantics. One may also consider models in a more traditional sense by viewing contexts as (logical) theories. It seems that to achieve this requires a deeper understanding of existing approaches to semantics of type theories as well as the proof-theoretical aspects of type systems.

In the aspect of applications, further practice in theorem-proving and program specification is needed to examine whether the facilities provided by the theory are adequate and strong enough in reality. A good direction would be to use the basic theory mechanism described in chapter 8 to build up theory bases for particular application areas and, based on them, to do practical examples of development of proofs and programs. Such a practice may be done in a proof development system like LEGO [Pol89][LPT89] together with some supporting tools. An implementation of an environment supporting theory development may be directly based on the ideas described in chapter 8 and use ideas from [SB83][BLuo88] to provide theory-building operations. How to combine these nicely is to be further investigated.

Bibliography

[Bar84] H.P Barendregt, *The Lambda Calculus: its Syntax and Semantics*, revised edition, North-Holland.

[Bar89a] H.P Barendregt, *Typed Lambda Calculi*, to appear in Handbook of Logic in Computer Science (eds., S. Abramsky, D. Gabbay and T.S.E. Maibaum), Oxford University Press.

[Bar89b] H.P Barendregt, *'Introduction to Generalized Type Systems'*, to appear in Proc. of the 3rd Italian Conf. on Theoretical Computer Science, Mandera.

[BB85] C. Böhm and A. Beradurcci, *'Automatic Synthesis of Typed λ-programs on Term Algebras'*, Theoretical Computer Science 39.

[Bee85] M.J. Beeson, *Foundations of Constructive Mathematics*, Springer-Verlag.

[Ber89a] S. Berardi, *Type Dependence and Constructive Mathematics*, manuscript, June 1989.

[Ber89b] 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.

[BGeu89] E. Barendsen and H. Geuvers, *'Conservativity of λ**P** over **PRED**'*, manuscript.

[BGog80] R. Burstall and J. Goguen, *'The Semantics of CLEAR, a Specification Language',* Lecture Notes in Computer Science 86.

[Bis67] E. Bishop, *Foundations of Constructive Analysis*, McGraw-Hill.

[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.

[BMS81] R. Burstall, D. MacQueen and D. Sannella, *'HOPE: an Experimental Applicative Language'*, Proc. 1980 LISP Conf., California.

[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.

[Bur89a] R. Burstall, *An Approach to Program Specification and Development in Constructions*, Talk given in Workshop on Programming Logic, Bastad, Sweden, May 1989.

[Bur89b] R. Burstall, *'Computer-assisted Proof for Mathematics: an introduction, using the LEGO proof system'*, to appear in Proc. of the Institute for Applied Math. conf., Brighton Polytechnic.

[Card86] L. Cardelli, *'A Polymorphic λ-calculus with Type:Type'*, manuscript.

[Card89] L. Cardelli, *Typeful Programming*, Lecture notes for the IFIP State of the Art Seminar on Formal Description of Programming Concepts, Rio de Janeiro, Brazil.

[Cart78] J. Cartmell, *Generalized Algebraic Theories and Contextual Category*, Ph.D. Thesis, University of Oxford.

[Cart86] J. Cartmell, *'Generalized Algebraic Theories and Contextual Category'*, 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).

[Con71] R. L. Constable, *'Constructive Mathematics and Automatic Programs Writers'*, Proc. IFIP'71.

[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.

[CM89] Th. Coquand and Ch. Paulin-Mohring, *'Inductively Defined Types',* draft.

[CW85] L. Cardelli and P. Wegner, *'On Understanding Types, Data Abstraction and Polymorphism'*, Computing Surveys 17.

[dB72] N. G. de Bruijn, *'Lambda Calculus Notation with Nameless Dummies: a Tool for Automatic Formula Manipulation with Application to the Church-Rosser Theorem'*, Indag. Mathematics 34.

[dB78] N. G. de Bruijn, *'A Name-free Lambda Calculus with Facilities for Internal Definition of Expressions and Segments'*, Technical Report 78-WSK-03, Eindhoven University of Technology.

[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.

[EFH83] H. Ehrig, W. Fey and H. Hansen, *ACT ONE: an Algebraic Specification Language with Two Levels of Semantics,* Tech. Report 83-03, Technical University of Berlin, Fachbereich Informatik.

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

[Fef79] S. Feferman, *'Constructive Theories of Functions and Classes'*, in Logic Colloquium'78, (eds., M. Boffa, D. van Dalen and K. McAloon) North Holland, Amsterdam.

[FGJM85] K. Futatsugi, J. Goguen, J.-P. Jouannaud and J. Meseguer, *Principles of OBJ2,* Proc. POPL 85.

[Fri77] H. Friedman, *'Set-theoretic Foundations for Constructive Analysis'*, Annals of Mathematics 105.

[Gal89] J.H. Gallier, *On Girard's 'Candidats de Reductibilité',* To appear in Logic and Computer Science (ed. P. Odifreddi), Academic Press.

[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.

[Gir71] J.-Y. Girard, *'Une extension de l'interpretation fonctionelle de Gödel à l'analyse et son application à l'élimination des coupures dans et la thèorie des types'*, Proc. 2nd Scandinavian Logic Symposium.

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

[Gir73] J.-Y. Girard, *'Quelques résultats sur les interprétations fonctionells'*, Lecture Notes in Mathematics 337, Springer.

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

[Gir89] J.-Y. Girard, *Proofs and Types*, Translated by Y. Lafont and P. Taylor, Cambridge University Press.

[GMW79] M.J. Gordon, R. Milner and C.P. Wadsworth, *Edinburgh LCF*, Lecture Notes in Computer Science 78, Springer.

[Gri87] T. Griffin, *An Environment for Formal Systems*, LFCS Report ECS-LFCS-87-34, Dept. of Computer Science, Univ. of Edinburgh.

[Grz53] A. Grzegorczyk, *'Some Classes of Resursive Functions'*, Rozprawy Mate. IV, Warsaw.

[Hay89] S. Hayashi, *'Constructive Mathematics and Computer-assisted Reasoning Systems'*, to appear in Proc. of Heyting'88, Prenum Press.

[Hey71] A. Heyting, *Intuitionism: an Introduction*, North-Holland.

[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.

[HN88] S. Hayashi and H. Nakano, **PX***: a Computational Logic*, The MIT Press, Cambridge, Massachusetts.

[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.

[HS87] J.R. Hindley and J.P. Seldin, *Introduction to Combinators and λ-calculus*, Cambridge University Press.

[Hue87] G. Huet, *'A Calculus with Type:Type'*, unpublished manuscript.

[Hue89] G. Huet (ed.), *The Calculus of Constructions: Documentation and User's Guide,* Technical Report INRIA 110.

[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.

[Jut77] B. Jutting, *Checking Landau's 'Grundlagen' in the Automath System*, Ph.D. thesis, Eindhoven University of Technology, Mathematical Centre Tracts 83.

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

[Kre68] G. Kreisel, *'Functions, Ordinals, Species'*, Logic, Methodology anfd Philosophy of Science III (eds. B. van Rootselaar and J. Staal), North-Holland, Amsterdam.

[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 Technical Notes LFCS-TN-27, Dept. of Computer Science, Edinburgh University.

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

[Luo88c] Zhaohui Luo, *'A Higher-order Calculus and Its ω–**Set** Model',* circulated notes. Jan. 1988.

[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.

[Luo89c] Zhaohui Luo, *On Girard-Tait's Reducibility Method for Strong Normalization Proofs of Type Theories*, Talk given in the 3rd Jumelage meeting on Typed Lambda Calculi, Edinburgh.

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

[McC62] J. McCarthy et al., *Lisp 1.5 Programmer's Manual*, MIT Press, Cambridge, Mass..

[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.

[Mil84] R. Milner, *'A Proposal for Standard ML'*, Proc. Symp. on Lisp and functional Programming, Austin, Texas.

[Mit86] J.C. Mitchell, *'A Type Inference Approach to Reduction Properties and Semantics of Polymorphic Expressions'*, Proc. 1986 ACM Symp. on Lisp and Functional Programming.

[ML71] Per Martin-Löf, *A Theory of Types*, manuscript.

[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.

[ML82] Per Martin-Löf, *'Constructive Mathematics and Computer Programming'*, Logic, Methodology and Philosophy of Science VI (eds., L.J. Cohen et al.). North-Holland, Amsterdam.

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

[MN87] D. Miller and G. Nadathur, *'A Logic Programming Approach to Manipulating Formulas and Programs'*, Proc. IEEE Symp. on Logic Programming, San Francisco.

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

[Moh89] Ch. Paulin-Mohring, *'Extracting F^ω Programs from Proofs in the Calculus of Constructions'*, Proc. POPL 89.

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

[MW71] Z. Manna and R. Waldinger, *'Towards Automatic Program Synthesis'*, Communications of ACM 14.

[Myh75] J. Myhill, *'Constructive Set Theory'*, J. Symbolic Logic 40.

[NP83] B. Nordström and K. Petersson, *'Types and Specifications'*, Proc. IFIP'83, Elsevier.

[NPS89] 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, *Logic and Computation: Interactive Proof with Cambridge LCF,* Cambridge University Press.

[Pau88] L. Paulson, *A Preliminary User's Manual for Isabelle*, Technical Report 133, Computer Laboratory, Cambridge University.

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

[Plo87] G. Plotkin, *'A Search Space for LF'*, Workshop on General Logic, Edinburgh, 1987. in LFCS Report Series, ECS-LFCS-88-52.

[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.

[Ros84] H.E. Rose, *Subrecursion,* Oxford University Press.

[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.

[Rus03] B. Russell, *The Principles of Mathematics, Vol. I*, Cambridge University Press.

[Sal89] A. Salvesen, *'The Church-Rosser Theorem for LF with $\beta\eta$ reduction'*, manuscript.

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

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

[Sco70] D. Scott, *'Constructive Validity'*, Symp. on Automatic Demonstration, Lecture Notes in Mathematics 125.

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

[See86] R.A.G. Seely, *'Categorical Semantics for Higher-order Polymorphic Lambda Calculus'*, J. of Symbolic Logic, vol. 52, no. 4. Springer-Verlag.

[SS88] A. Salvesen and J. Smith, *'The Strength of Subset Type in Martin-Löf's Type Theory'*, Proc. 3rd Ann. Symp. on Logic in Computer Science, Edinburgh.

[ST88] D. Sannella and A. Tarlecki, *Building Specifications in an Arbitrary Institution*, Information and Computation 76(2/3).

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

[Tai67] W.W. Tait, *'Intensional Interpretation of Functionals of Finite Type I',* J. of Symbolic Logic 32.

[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.

[TL88] P. Taylor and Z. Luo, *'Theories, Mathematical Structures and Strong Sums',* manuscript, Dec. 1988.

[Tro73a] A. S. Troelstra, *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis,* Lecture Notes in Mathematics 344.

[Tro73b] A. S. Troelstra, *'Notes on Intuitionistic Second-order Arithmetic'*, Lecture Notes in Mathematics 337.

[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.

Notation and Symbols

The ordinary notation and symbols for meta-level statements like (in)equalities $=, \leq, <, ...$ and meta-level logical operators $\forall, \exists, \Rightarrow, \wedge, \vee, ...$ are *not* included here. We have also used $\not\equiv, \not\leq, \not\vdash, ...$ to express meta-level negation.

The numbers below refer to the pages on which the notation or symbol is introduced or of its first major occurrence.

Miscellaneous

κ_j	the jth inaccessible cardinal	137
$dom(f)$	domain of function f	98
$A \to B$	functions (morphisms, functors) from A to B	101, 134
$A \to_{FPP} B$	FPP-morphisms from A to B	134
$Eval_\rho$	the evaluation function	102
v_M	the canonical value of $\mathcal{E}$-term M	100
$\mathcal{C}(\_;\ \_)$	the algorithm of type inference	126
σ, σ_Γ, π_Γ	ω-set constructors	133, 135
$[\![\_]\!]$	the interpretation of judgement $\_$	134

Terms and judgements

$\Pi x{:}A.B$, $A \to B$	(dependent) product type	22
$\Sigma x{:}A.B$, $A \times B$	(dependent) strong sum type	22
$\lambda x{:}A.M$	Lambda-abstraction	22
MN	functional application	22
$\mathbf{pair}_A(M, N)$	pair	22
$\pi_1(M), \pi_2(M)$	projections of a pair	22
$[N/x]M$	substitution of N for x in M	22
$[N_1, ..., N_m/x_1, ..., x_m]M$	simultaneous substitution	100
$T_\Gamma(M)$	the principal type of M under Γ	62
$T_\mathcal{E}(M)$	the principal type of M under $\mathcal{E}$	69
$\mathrm{red}_\mathbf{k}(M)$	the key-reduct of M	93
$\exists^i(A, B)$	predicative existential type	166
$rep^i, abstype^i$	introduction and elimination operators for $\exists^i(A, B)$	166
$\Gamma \vdash M : A$	judgement form	24
$\mathcal{E} \vdash M : N$	'judgement form' under environment	68

Sets

$\emptyset$	the empty set	114
ω	the set of natural numbers	21
$\mathcal{K}$	the set of kinds	126
$\mathcal{T}$	the set of terms	22
$FV(\_)$	the set of free variables in $\_$	22, 24
V_α	the cumulative hierarchy of sets	137
$CR(A)$	the set of A-candidates of reducibility	94
$Sat(A)$	the set of A-saturated sets	94
$SN(A)$	the set of strongly normalizable terms of type A	93
$V(M)$	the value-set of $\mathcal{E}$-term M	97
$\lvert A\rvert$	the carrier set of ω-set A	131

Relations

$\equiv$	syntactical identity	22
$=_{\mathrm{df}}$	definitional equality	35, 42
$\rhd$	$(\beta\sigma$-)reduction	22
$\rhd_1$	one-step $(\beta\sigma$-)reduction	23
$\simeq$	$(\beta\sigma$-)conversion	22
$\rightsquigarrow_\beta$	β-contraction	22
$\rightsquigarrow_\eta$	η-contraction	40
$\rightsquigarrow_\sigma$	σ-contraction	22
$\rightsquigarrow_\pi$	contraction of surjective pairing	41
$\preceq$	the cumulativity relation	23
$\prec$	the strict cumulativity relation	24
$\preceq_i$	the 'level-i' cumulativity relation	41
$\prec_i$	the strict 'level-i' cumulativity relation	42
$\approx$	cumulativity equivalence	42
$\Vdash_A$	the realizability relation of ω-set A	131

Environment and measures

$\mathcal{E}$	environment	67, 68
$\mathcal{E}^i$	the first i components of environment $\mathcal{E}$	68
$\mathcal{E}_i$	the ith component of environment $\mathcal{E}$	68
e_i	the ith variable of environment $\mathcal{E}$	68
$\mathcal{L}(\_)$	level of type $\_$	69, 70
$\mathcal{D}_j(\_)$	j-degree of type $\_$	76, 80, 87
$\beta(\_)$	the complexity measure of type $\_$	88
$\delta_{\mathbf{j}}R$	a complexity measure	80
$\gamma_{\mathbf{j}}M$	a complexity measure	80

Embedded logic

true	truth	117
false	falsity	117
$\supset$	implication	117
&	conjunction	117
$\vee$	disjunction	117
$\neg$	negation	117
$\forall x{:}A.P(x)$	universally quantified formula	117
$\exists x{:}A.P(x)$	existentially quantified formula	117
$=_A$	Leibniz's equality over type A	119

Categories and functors

ω–**Set**	the category of ω-sets	132
ω–**Set**(j)	the category of ω-sets within V_{κ_j}	137
M	the category of modest sets	138
PROP	the category of 'propositions'	139
Set	the category of sets	137
back	category equivalence from **M** to **PROP**	139
inc	the inclusion functor from **PROP** to **M**	139
Δ	the embedding functor from **Set** to ω–**Set**	137
Obj(_)	the object set of category _	137

Index