Logics and Type Systems

een wetenschappelijke proeve op het gebied van de
wiskunde en informatica

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Katholieke Universiteit Nijmegen,
volgens besluit van het College van Decanen
in het openbaar te verdedigen
op dinsdag 14 september 1993,
des namiddags te 3.30 uur precies

door

JAN HERMAN GEUVERS

geboren 19 mei 1964 te Deventer

druk: Universiteitsdrukkerij Nijmegen

Promotor: Professor dr. H. P. Barendregt

Logics and Type Systems

Herman Geuvers

Cover design: Jean Bernard Koeman

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Geuvers, Jan Herman

Logics and type systems / Jan Herman Geuvers. - [S.l. :
s.n.] (Nijmegen : Universiteitsdrukkerij Nijmegen)
Proefschrift Nijmegen. - Met lit. opg.,reg.
ISBN 90-9006352-8
Trefw.: logica voor de informatica.

Contents

Acknowledgements

First of all I would like to thank my supervisor Henk Barendregt, not only for creating a stimulating research environment during the last four and a half years but also for letting me find my own way in the jungle of interesting subjects of research. But maybe most of all I should thank him for sharing his knowledge with me.

I am also very grateful to all those other researchers that I have been able to talk to and to listen to. Especially the contact with (in reverse alphabetical order) Benjamin Werner, Marco Swaen, Thomas Streicher, Randy Pollack, Christine Paulin, Mark-Jan Nederhof, James McKinna, Zhaohui Luo, Bart Jacobs, Philippa Gardner, Gilles Dowek, Thierry Coquand, Stefano Berardi, Bert van Benthem Jutting, Erik Barendsen and Thorsten Altenkirch has been both very pleasant and very fruitful. In an earlier stage the contact with Wim Veldman has been very important: his lectures have guided me into the field of logic and have stimulated my interest in foundational issues.

In particular with respect to the contents of this thesis, I would furthermore like to thank the manuscript committee, consisting of Rob Nederpelt, Jan-Willem Klop and Thierry Coquand, for their judgement. Special thanks to Rob Nederpelt for his detailed comments on part of an earlier version of this work and to James McKinna for his valuable comments on English, contents and typos. Erik Barendsen deserves a very special thanks: without his knowledge of LaTeX and his willingness to always answer my technical questions, this thesis would not be as it is now.

A pleasant working environment is very valuable and almost a necessary condition for a good result. I would therefore like to thank the people from our faculty that have made work pleasant, especially those from the research groups 'foundations of computing science' and 'parallelism and computational models'.

Last but not least I would like to thank Monique for her support during the ups and the downs of the work on this thesis.

Propositions

1. Define the mapping $[-]$ from full one-sorted first order predicate logic to higher order proposition logic as follows.

$$\begin{array}{rcl} [x] & = & x, \\ [Rt_1 \ldots t_n] & = & R[t_1] \ldots [t_n], \\ [\varphi \supset \psi] & = & [\varphi] \supset [\psi], \\ [\varphi \mathbin{\&} \psi] & = & [\varphi] \mathbin{\&} [\psi], \\ [\varphi \vee \psi] & = & [\varphi] \vee [\psi], \\ [\neg\varphi] & = & \neg[\varphi], \\ [\forall x.\varphi] & = & \forall x.[\varphi], \\ [\exists x.\varphi] & = & \exists x.[\varphi]. \end{array}$$

So for example $[\forall x.Px \supset Px] = \forall x.Px \supset Px$. (The x on the left is an object variable, the x on the right a propositional variable. Similarly, the R on the left is a relation symbol, the R on the right a higher order variable.) In fact the range of the mapping $[-]$ is a very small extension of second order propositionl logic.
The mapping $[-]$ is sound but not complete.

2. There is no fixed point combinator in the Pure Type System λU. (Thanks to Benjamin Werner.)

3. The system of higher order propositional logic (PROPω) is a conservative extension of second order propositional logic (PROP2). The proof uses the fact that complete Heyting algebras constitute a sound and complete model for PROP2.
If Δ is a set of formulas and φ a formula of PROP2 such that $\Delta \vdash_{\mathrm{PROP}\omega} \varphi$ with derivation Θ, then it is in general not true that the normal form of Θ, which is obtained by eliminating cuts, is a derivation of $\Delta \vdash_{\mathrm{PROP2}} \varphi$.
In typed lambda calculus this corresponds to the following two facts. Let Γ be a context and σ be a type of $\lambda 2$. Then

$$\Gamma \vdash_{\lambda\omega} M : \sigma \quad \not\Rightarrow \quad \Gamma \vdash_{\lambda 2} \mathrm{nf}(M) : \sigma,$$

$$\Gamma \vdash_{\lambda\omega} M : \sigma \quad \Rightarrow \quad \exists N[\Gamma \vdash_{\lambda 2} N : \sigma].$$

Therefore it is not surprising that up to now there is no purely syntactical proof of the conservativity of PROPω over PROP2.

4. The restriction of the typed lambda calculus with recursive types $\lambda\mu$ to the calculus $\lambda\mu^+$, where one only allows μ-abstractions over positive type schemes, is not a real restriction. For every type σ of $\lambda\mu$ one can construct a type σ' of $\lambda\mu^+$ such that $\sigma \approx \sigma'$. Hence all lambda terms can be given a type in $\lambda\mu^+$.

5. The proof of Corollary 15.1.5 in [Barendregt 1984] is not complete: the ($\Rightarrow$)-part, stating

$$M \text{ has a } \beta\text{-nf} \Rightarrow M \text{ has a } \beta\eta\text{-nf}$$

is indeed trivial, but it is not true that η contractions do not create new redexes.

6. It is well-known that it is impossible to prove $0 \neq 1$ in the Calculus of Constructions. (Here, 0 and 1 are the polymorphic Church numerals.) In the inconsistent systems $\lambda\star$, λU^- and λU, the statement $0 \neq 1$ is of course provable, but even with a proof in normal form.

7. Let $\lambda\mathbb{N}$ be the Pure Type System with $\beta\eta$-conversion, defined by

$$\begin{array}{rcl} \mathcal{S} & = & \mathbb{N}, \\ \mathcal{A} & = & \mathbb{N} \times \mathbb{N}, \\ \mathcal{R} & = & \mathbb{N} \times \mathbb{N} \times \mathbb{N} \end{array}$$

If $\lambda\mathbb{N}$ satisfies the Church-Rosser property for $\beta\eta$-reduction ($\mathrm{CR}_{\beta\eta}$), then all Pure Type Systems satisfy $\mathrm{CR}_{\beta\eta}$.

8. The relation $\twoheadrightarrow_d$ with

$$t \twoheadrightarrow_d u \text{ if } t \twoheadrightarrow_\beta t' \text{ and } u \text{ is a domain of } t' \text{ for some } t',$$

is in general not well-founded on the set of well-typed terms of a Pure Type System. (A *domain* of t' is a term that appears in t' as the type of a λ-abstraction.)
This causes a problem when trying to prove confluence of $\beta\eta$-reduction in Pure Type Systems that are not normalizing.

9. Besides the difference in income, the most important difference between AIOs and the old-style research trainees on Dutch universities is that the first, on top of the tasks of the old-style research trainees, also have the duty to follow courses. The AIOs should not demand from the universities that they organise courses as compensation for the financial offer. Instead they should try to keep their duties in terms of courses they have to attend as low as possible.

10. The experience of having deep insight is not the same as having deep insight. The first can be attained by various means, the second only by serious study.

Chapter 1

Introduction

In this thesis we are concerned with systems of logic, systems of types and the relations between them. The systems of types should be understood here as systems of typed lambda calculus, so in fact this thesis takes up the study of the relation between typed lambda calculus and logic. This is not a new subject: a lot of research has been done, most of which is centered around the so called 'formulas-as-types embedding' from a logical system into a typed lambda calculus. This embedding will also be the main topic of this thesis.

The first to describe the formulas-as-types embedding was Howard, who also introduced the terminology 'formulas-as-types', [Howard 1980]. The manuscript of this paper goes back to 1968 and a lot of ideas behind the embedding go back even further, especially to Curry (see [Curry and Feys 1958]), who was the first to note the close connection between minimal proposition logic and combinatory logic. The article of Howard is mainly concerned with giving a formal explanation of the intuitionistic connectives. In this way it is an attempt to formalize the Brouwer-Heyting-Kolmogorov (BHK) interpretation of the intuitionistic connectives, as it can be found in the original work [Kolmogorov 1932] and [Heyting 1934], but also in the recent book [Troelstra and Van Dalen 1988]. In that interpretation a connective is explained in terms of what it means to have a proof of a sentence built up by that connective. Howard gives a formal interpretation of proofs (and hence of connectives) in terms of typed lambda calculus, by giving an interpretation to the introduction and elimination rule of the logic. For $\supset$ and $\forall$, the introduction rule corresponds to λ-abstraction and the elimination rule to application. The ideas in [Howard 1980] were used and extended further by Martin-Löf in his Intuitionistic Theory of Types [Martin-Löf 1975], [Martin-Löf 1984] and by Girard who extended it to higher orders [Girard 1972], [Girard 1986], [Girard et al. 1989]. All this work can be united under the heading of 'proof-theory'.

Another approach was taken in the research project Automath by de Bruijn [de Bruijn 1980], who independently defined a kind of formulas-as-types embedding from logic into typed lambda calculus which is of a different nature and, maybe more important, which has a different purpose. The difference in nature lies in the fact that the typed lambda calculus is not meant to represent one particular system of logic as close as possible, but to serve as a framework for mathematical reasoning in general. The purpose of this work is to clarify and formalize the underlying principles that all mathematicians use and agree on. In a sense this is an attempt to put on stage the part of mathematics that comes 'before logic', the part that every mathematician is informally aware of, such as how to use and give definitions. A practical off-shoot of this program is the possibility of doing mathematics on a computer by implementing the formal system of typed lambda calculus. Let's point out here that the difference between the two approaches is not always as sharp as this discussion might suggest. It is very well possible to use both approaches in one system.

The most interesting part of the various embeddings is not that formulas are interpreted as types, but that proofs are interpreted as terms (which obviously comes as a consequence of 'formulas-as-types', if we understand a type as a set in some weak sense). This makes that the proofs become first class citizens in the type system. On the one hand this provides for a whole world of new options, like the possibility to formalize meta-reasoning (reasoning about proofs) in the system or the possibility to let terms depend on proofs (like a function that extracts from a proof of an existential sentence a 'witnessing object' of the sentence). On the other hand this requires a well-understood notion of what a proof is: if we claim that the terms of some typed lambda calculus represent proofs, this statement implicitly contains a definition of the notion of proof. A workable approximation of the notion of proof is the notion of 'derivation' in a specific formal system of logic.

The formulas-as-types embedding described by Howard goes from first order predicate logic in natural deduction style to an extension of the simply typed lambda calculus. It yields an isomorphism on the level of proofs (derivations), if we identify derivations that only differ in some specific trivial way. The systems described by de Bruijn provide the possibility to embed a large variety of formal logics, hence we can not expect to have an isomorphism on the level of derivations: only some of the proof-terms correspond to a derivation in the logic. In both systems, the interpretation of proofs-as-terms *does* provide an equivalence relation on the proofs, signifying which derivations are to be understood as being equal.

We have already mentioned as a practical application of the formulas-as-types embedding the possibility of doing mathematics on a computer. This was one of the main starting points for de Bruijn in setting up the Automath project. In Automath the computer was mainly used as a proof-checker: the user types in a proof (in the form of a λ-term) and the formula it is supposed to be proving (in the form of a type) and the computer checks whether the proof proves the formula, that is whether the term is of the given type. Later, other research groups enlarged the job of the computer by developing interactive theorem provers. The pioneering work on LCF [Gordon et al. 1976] has been very important here, because it has lead to the interactive meta-language ML. This language is very well suited for implementing a typed lambda calculus that is to be used for interactive theorem proving, because it allows the user to program tactics for proof-search. Important developments in the field are the Calculus of Constructions [Coquand 1985] [Coquand and Huet 1985], [Coquand and Huet 1988] and its recent extension Coq [Dowek et al. 1991], which are implemented in a language closely related to ML. Further we want to mention the work in Edinburgh on ECC (Extended Calculus of Constructions, [Luo 1989] and its implementation in ML 'LEGO' [Luo and Pollack 1992] and the work at Cornell on the system Nuprl [Constable et.al. 1986], which is an implementation of Martin-Löf's type theory. The work on LCF itself grew into the system HOL [Gordon 1988], a proof-assistant for classical higher order logic, which does not use the formulas-as-types embedding but implements Church's simple theory of types [Church 1940].

Another important practical application of the formulas-as-types embedding, in particular the one described by Howard, is the possibility to extract programs from proofs. This conforms to the BHK-interpretation of connectives and proofs in constructive mathematics, according to which, for example, a proof of the sentence $\forall x \in A \exists y \in B \varphi(x, y)$ contains a construction of an element $b_a \in B$ for every $a \in A$ such that $\varphi(a, b_a)$ holds for every $a \in A$. In the formulas-as-types interpretation of Howard, the proof-term contains an algorithm in the form of a λ-term. This was extended to higher order logic by Girard, who also emphasized the consequence of this approach, namely that cut-elimination in the logic corresponds to evaluation of a program. As a calculus for typing the λ-terms that were extracted from the proofs he introduced the systems Fn ($n \geq 2$) and Fω [Girard 1972], which can be seen as very rudimentary programming languages. Also Martin-Löf made contributions to the idea of extracting programs from proofs, not by going to higher orders but by adding an inductive type forming operator [Martin-Löf 1984].

The programs-from-proofs notion has been extended and refined a lot over the years, notably by the Projet Formel group in Paris (Calculus of Constructions and Coq, [Coquand and Huet 1985], [Coquand and Huet 1988], [Mohring 1986] and [Paulin 1989]), the Nuprl project at Cornell [Constable et.al. 1986], the Equipe de Logique group in Paris [Krivine and Parigot 1990], [Parigot 1992] and the research group in Göteborg [Nordström et al. 1990]. The crucial feature of the programs-from-proofs approach is that the proofs are preserved in the formal system in some 'algorithmic' form. If one just wants to do mathematics on a computer this is less important, because it will often be sufficient to know that a formula is provable. Note however that also in the latter case it can be an advantage to preserve proofs, for example if one wants to set up a library of mathematics which is reproducible in book form.

In this thesis we are mainly concerned with the formulas-as-types embedding itself, with some emphasis on the Howard approach. So we do not for example discuss technical details of the programs-from-proofs notion, nor do we discuss technical problems that arise when trying to set up a library of mathematics. The reader can find a detailed description of the logics that are subject to the formulas-as-types interpretation. These logics are chosen in such a way that we can easily define a collection of typed lambda calculi for which the embedding is an isomorphism on the derivations of the logic (modulo some easy equivalence relation). Then we discuss the two approaches to formulas-as-types by studying some examples. Further we study and prove Strong Normalization and Confluence of the reduction relation in the typed lambda calculi, which are important properties for these systems. Most of the typed lambda calculi that are looked at in this thesis are instances of so called 'Pure Type Systems'. This is a general framework for describing typed lambda calculi that will be discussed in detail here. Most of the meta-theory that one would like to have for the typed lambda calculi can be proved once and for all for the whole collection of Pure Type Systems.

An important issue of the formulas-as-types embedding is its completeness on the level of provability: even if there is no isomorphism on the level of derivations, it would be really undesirable if the typed lambda calculus would prove more sentences than the logic. This issue will be discussed in detail for the Calculus of Constructions. On the one hand the embedding is not complete, but on the other hand this is not so dramatic, because there is a completeness result for sentences of a specific form.

We give a short overview of each of the chapters.

1. Chapter 2 describes the logics in a generic way, from first order predicate logic to higher order predicate logic, and relates them to more standard presentations of these logics. The logics are minimal in the sense that we only have $\supset$ and $\forall$. Also the propositional variants will be described. We discuss the conservativity relations between these systems. The most interesting result in this Chapter is probably the proof of conservativity of higher order propositional logic over second order propositional logic (both classical and intuitionistic.) The proof for the intuitionistic case is given by describing a semantics in terms of complete Heyting algebras. As far as we know this is a new result.

2. Chapter 3 discusses the formulas-as-types embedding. Here we distinguish two approaches, one 'à la Howard' and one 'à la de Bruijn'. We give a detailed description of the embedding of minimal first order predicate logic in a typed lambda calculus (à la Howard) and show completeness on the level of derivations. This means that the embedding constitutes an isomorphism between the derivations in the logic and the terms in the typed lambda calculus. Then we discuss the formulas-as-types embedding (à la de Bruijn) in Automath systems and in LF [Harper et al. 1987].

3. Chapter 4 treats the notion of 'Pure Type System'. We prove a list of meta-theoretic properties and give examples of instances of Pure Type Systems. The properties we prove are the ones that are well- known from [Geuvers and Nederhof 1991], but now extended to Pure Type Systems with $\beta\eta$-reduction.

4. In Chapter 5 we give a proof of Confluence of $\beta\eta$-reduction in normalizing Pure Type Systems. Confluence of β-reduction is quite easy, but Confluence of $\beta\eta$-reduction is remarkably complicated. Confluence in fact states the consistency of the type system as a calculus (in the sense that it shows that two different values are indeed distinguished by the system). The importance of this property lies further in the fact that it is one of the main tools for proving decidability of equality and from that decidability of typing. (Under the formulas-as-types embedding, to decide whether a term is of a certain type is the same as to decide whether a proof proves a certain formula.)

5. In Chapter 6 we discuss the Calculus of Constructions (CC) and its fine structure in the form of the so called 'cube of typed lambda calculi'. We study the formulas-as-types embedding from (subsystems of) higher order predicate logic into (subsystems of) CC. We also look at conservativity with respect to provability between the type systems of the cube. A new result here is the conservativity of Fω over F, which comes as a Corollary of the fact that higher order propositional logic is conservative over second order propositional logic, which result was proved in Chapter 2.

6. In Chapter 7 we give a proof of Strong Normalization of $\beta\eta$-reduction in CC. (Strong) Normalization is the other main tool for proving decidability of equality and from that decidability of typing. It is also the main tool for showing consistency of a type system as a logic (in the sense that not all types are inhabited by a closed term). To be a bit more precise: the consistency of CC itself is quite easy, but if one wants to show the consistency of a context of CC, (Strong) Normalization comes in.

7. In Chapter 8 we briefly discuss some issues that have been left and list some open problems that may be of interest for further study.

Some of the work reported in this thesis has already appeared somewhere or will do so later, notably Chapters 4 and 7, which is can an extension of the work in [Geuvers and Nederhof 1991] to the case that includes η-reduction. (In [Geuvers and Nederhof 1991] we only considered β-reduction). Chapter 6 has appeared in a slightly different form (with some mistakes) as [Geuvers 1992] and both Chapters 4 and 6 contain work that has also been reported in [Geuvers 1990] and [Geuvers 199+].

Chapter 2

Natural Deduction Systems of Logic

2.1. Introduction

In this chapter we want to discuss the logical systems that will be used in the context of the Curry-Howard isomorphism. In the original paper by Howard [Howard 1980] on this formulas-as-types isomorphism, there are interpretations of all the standard connectives of intuitionistic logic. As we are mainly interested in second and higher order systems (in which cases all connectives can be coded in terms of $\supset$ and $\forall$), we shall restrict our attention mainly to $\supset$ and $\forall$. The Curry-Howard isomorphism gives an interpretation of derivations as lambda terms in a typed lambda calculus, but it only does so for derivations in natural deduction style. (As already pointed out, the $\supset$- and $\forall$-introduction rules correspond to λ-abstraction and the $\supset$- and $\forall$-elimination rules correspond to application). Consequently, the representation of our logical systems will also be in natural deduction style.

This doesn't yet settle the whole question of what the precise formulation of the system should be. If we would only be interested in provability the choice for the formalization of the logic should be determined by the questions about provability that we want to tackle. In our case however, we are interested in the formal proofs (derivations) themselves and it depends heavily on the formal presentation that we have chosen, how many distinct derivations of a proposition we have. (This is also a reason for not choosing Gentzen's sequent calculus to describe the formulas-as-types embedding, because in that system distinctions between derivations are often due to an inessential difference in bookkeeping). So our choice for the formal system of logic will be determined by the formulas-as-types interpretations of the proofs in typed lambda calculus that we want to do later.

2.2. The Logics

One issue that we want to stress here is the choice of the so called 'discharge convention' that has to be made. This issue was drawn to our attention by the book of [Troelstra and Van Dalen 1988], where the *crude discharge convention*, CDC, is used throughout the book, except when it comes to the formulas-as-types interpretation. Let's briefly state the problem by an example in minimal implicational propositional logic PROP, which we shall describe in two formats, to be called PROP_A and PROP_B, both natural deduction style. This example also shows how our choice for the formalisation of the logic is determined by the Curry-Howard isomorphism. In fact the isomorphism clearly visualizes the differences between the formalizations.

2.2.1. Definition. The systems PROP_A and PROP_B have as formulas the elements of the set FORM, given in abstract syntax by

$$\mathrm{Form} ::= \mathsf{Var} \mid \mathrm{Form} \supset \mathrm{Form},$$

where Var is a countable set of variables.
The derivation rules of PROP_A are the following. (In the rules, φ and ψ are formulas and Γ is a finite set of formulas).

$$(\mathrm{ax}) \quad \frac{}{\Gamma \vdash \varphi} \text{ if } \varphi \in \Gamma$$

$$(\supset\text{-I}) \quad \frac{\Gamma \cup \{\varphi\} \vdash \psi}{\Gamma \vdash \varphi \supset \psi} \qquad (\supset\text{-E}) \quad \frac{\Gamma \vdash \varphi \quad \Gamma \vdash \varphi \supset \psi}{\Gamma \vdash \psi}$$

The derivation rules of PROP_B are the following. (φ and ψ are formulas).

$$(\supset\text{-I}) \quad \frac{\begin{array}{c}[\varphi] \\ \vdots \\ \psi\end{array}}{\varphi \supset \psi} \qquad (\supset\text{-E}) \quad \frac{\varphi \supset \psi \quad \varphi}{\psi}$$

The formula φ in the $\supset$-I rule is said to be *discharged* (or *cancelled*). The $[\varphi]$ does not refer to one single occurrence of φ, but to arbitrary many (zero or more) φ's. With the derivation rules one can form *deduction trees*, starting from a single formula being the most basic form of a deduction tree. Then we say that $\Gamma \vdash \varphi$ is derivable if there is a derivation tree with root φ and all *open* formulas of the tree in Γ. (A formula is open in a derivation tree if it occurs as a leaf in non discharged form).
In practice the name of the rule will of course not be mentioned explicitly.

In the system PROP_B there is in general no canonical node in a derivation tree to which a specific cancelled formula corresponds. Look for example at the following derivation.

2.2.2. EXAMPLE.

$$\dfrac{\dfrac{[\varphi]}{\varphi \supset \varphi}}{\varphi \supset (\varphi \supset \varphi)}$$

The discharging of φ can ambiguously either belong to the first or to the second use of the $\supset$-I rule. To make the proofs more readable this ambiguity is often solved by writing a number on top of the discharged formula and writing the same number besides the line where the discharging took place. In that case the derivation tree above in fact corresponds to two different derivation trees. One can also solve the ambiguity by using the so called *crude discharge convention* (CDC), which says that at the $\supset$-I rule in the definition of PROP_B all open occurrences of φ are discharged. If we adopt CDC, the derivation tree above is canonical: φ is discharged at the first $\supset$-I rule.

In view of the Curry-Howard isomorphism, it is preferable to choose for the discharge convention which attaches a number to the discharged formula-occurrences and to the rule where the formula has been discharged. This is not for reasons of *soundness* but for the *completeness* of the Curry-Howard embedding. The example above represents two proofs of $\varphi \supset \varphi \supset \varphi$: $\lambda x^{\varphi}.\lambda y^{\varphi}.x$ (the discharged φ corresponds to the second $\supset$-I) and $\lambda x^{\varphi}.\lambda y^{\varphi}.y$ (the discharged φ corresponds to the first $\supset$-I). If the formal logical system has CDC, only the latter term can be obtained as the interpretation of a proof. This is why, in [Troelstra and Van Dalen 1988] CDC is dropped when discussing the formulas-as-types isomorphism.

The system PROP_A already has a sequent-like notation that is familiar from typed lambda calculi, but it is nevertheless more inconvenient then PROP_B for describing the Curry-Howard isomorphism. (And therefore it is even more remarkable that this is the kind of formalization that is often used for describing the isomorphism). The problem lies partly in the fact that the judgements $\Gamma \vdash \varphi$ are not really sequents in the sense of Gentzen, because in that case the Γ would have to be a (ordered) sequence instead of a set. We adopt the example above to the formalism of PROP_A to see what the problem is.

2.2.3. EXAMPLE.

$$\dfrac{\dfrac{\{\varphi\} \vdash \varphi}{\{\varphi\} \vdash \varphi \supset \varphi}}{\vdash \varphi \supset (\varphi \supset \varphi)}$$

The first application of the $\supset$-I rule sort of 'splits' the assumption φ into two copies of φ, reading $\{\varphi\}$ as $\{\varphi\} \cup \{\varphi\}$. It is impossible to recover the two possible proofs of $\varphi \supset \varphi \supset \varphi$ ($\lambda x^{\varphi}.\lambda y^{\varphi}.x$ and $\lambda x^{\varphi}.\lambda y^{\varphi}.y$ in typed lambda calculus format) from the derivation above: one could say that the first version is obtained by letting the φ in the succedent correspond to the 'right copy' of φ and the second version by letting the succedent correspond to the 'left copy' of φ, but this is the type of forced solution (with no motivation at all in the logic) that we want to avoid. Note that replacing the $\supset$-I rule by the two rules

$$(\supset\text{-I}_1)\ \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \supset \psi} \qquad (\supset\text{-I}_2)\ \frac{\Gamma \vdash \psi}{\Gamma \setminus \{\varphi\} \vdash \varphi \supset \psi}$$

to solve this problem is not only very unpleasant but on the other hand doesn't give the general solution. So we conclude that presenting natural deduction in a way similar to PROP_A is not what we are looking for.

In the original paper by Howard [Howard 1980] the defects of PROP_A do not appear because there the format of the natural deduction system uses real sequents, which are of the form $\Gamma \vdash \varphi$ with φ a formula and Γ a finite *sequence* of formulas. The rules of first order propositional logic (we call this version PROP_C) are then as follows.

2.2.4. Definition. The formulas of the system PROP_C are the same as for PROP_A and PROP_B. The derivation rules of PROP_C are the following. (In the rules, φ and ψ are formulas and Γ is a finite sequence of formulas; Γ, Δ is the concatenation of Γ and Δ).

$$(\mathrm{ax})\ \frac{}{\Gamma \vdash \varphi}\ \text{if } \varphi \in \Gamma$$

$$(\supset\text{-I})\ \frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \supset \psi} \qquad (\supset\text{-E})\ \frac{\Gamma \vdash \varphi \quad \Delta \vdash \varphi \supset \psi}{\Gamma, \Delta \vdash \psi}$$

$$(\mathrm{weak})\ \frac{\Gamma \vdash \psi}{\Gamma, \varphi \vdash \psi} \qquad (\mathrm{perm})\ \frac{\Gamma, \varphi, \psi, \Delta \vdash \chi}{\Gamma, \psi, \varphi, \Delta \vdash \chi} \qquad (\mathrm{contr})\ \frac{\Gamma, \varphi, \varphi, \Delta \vdash \chi}{\Gamma, \varphi, \Delta \vdash \chi}$$

It is clear how a derivation in the system PROP_C corresponds to a lambda term (construction in the terminology of [Howard 1980]) of the simply typed lambda calculus. The weakening rule amounts to an extension of the context with one new declaration, the permutation rule does not change anything (the contexts of the simply typed lambda calculus are a kind of 'multisets' of formulas) and the contraction rule amounts to substituting in the lambda term one free variable for another. Now there are many more derivations then there are distinct lambda terms of the corresponding type, due to the structural rules of weakening, permutation and contraction. So we can view the Curry-Howard embedding as splitting up the set of derivations in equivalence classes. (Where two derivations are equivalent if they are mapped onto the same image under the Curry-Howard embedding). In fact the embedding only takes care of the 'computationally interesting' part of the derivation; it extracts the construction from the derivation and in that sense it is a satisfying formal treatment of the BHK-interpretation of proofs-as-constructions. In our case, however, we do not just want to recover the construction behind the proof, but also find a unique (up to certain trivial changes) proof that corresponds to the construction. For that purpose, PROP_C is not so convenient as the following example will illustrate.

2.2.5. Example. Look at the following derivations of $\vdash \varphi \supset \varphi \supset \varphi$ in PROP_C.

$$(1)\quad \dfrac{\dfrac{\dfrac{\varphi \vdash \varphi}{\vdash \varphi \supset \varphi}}{\varphi \vdash \varphi \supset \varphi}}{\vdash \varphi \supset \varphi \supset \varphi} \qquad (2)\quad \dfrac{\dfrac{\dfrac{\varphi \vdash \varphi}{\varphi, \varphi \vdash \varphi}}{\varphi \vdash \varphi \supset \varphi}}{\vdash \varphi \supset \varphi \supset \varphi}$$

$$(3)\quad \dfrac{\dfrac{\dfrac{\dfrac{\varphi \vdash \varphi}{\varphi, \varphi \vdash \varphi}}{\varphi, \varphi \vdash \varphi}}{\varphi \vdash \varphi \supset \varphi}}{\vdash \varphi \supset \varphi \supset \varphi} \qquad (4)\quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\varphi \vdash \varphi}{\varphi, \varphi \vdash \varphi}}{\varphi, \varphi \vdash \varphi}}{\varphi, \varphi, \varphi \vdash \varphi}}{\varphi, \varphi \vdash \varphi}}{\varphi \vdash \varphi \supset \varphi}}{\vdash \varphi \supset \varphi \supset \varphi}$$

From the logical derivations it is not very obvious that the first and the third derivation should be considered equivalent and distinct from the second derivation. The Curry-Howard embedding makes this apparent: (1) and (3) correspond to $\lambda x^{\varphi}.\lambda y^{\varphi}.y$, while (2) corresponds to $\lambda x^{\varphi}.\lambda y^{\varphi}.x$. The situation for derivation (4) is even more complicated: the lambda term it corresponds to depends on which two occurrences of φ in the sequence $\varphi, \varphi, \varphi$ have been contracted in the application of the contraction rule. So, disregarding completeness, even to make the soundness of the embedding work we have to make the contraction rule more explicit, either by annotating in the sequent the formulas that are being contracted or by restricting the contraction to the last two formula occurrences.

From the discussion above it may have become clear that we have a strong preference for the format of the system PROP_B, with annotations to fix the formula occurrences that are being discharged at a specific application of a rule.

2.2.6. DEFINITION. For n a natural number, the system of nth order predicate logic, notation PREDn is defined by first giving the nth order language and then describing the deduction rules for the nth order system as follows.

1. The *domains* are given by

   $$\mathcal{D} ::= \mathcal{B} \mid \mathsf{Prop} \mid (\mathcal{D} \to \mathcal{D}),$$

   where $\mathcal{B}$ is a specific set of *basic domains*.
   We let the brackets associate to the right, so $\mathsf{Prop}\to(\mathsf{Prop}\to\mathsf{Prop})$ will be denoted by $\mathsf{Prop}\to\mathsf{Prop}\to\mathsf{Prop}$ and so every domain can be written as $D_1\to\ldots\to D_p\to D$, with $D_1,\ldots,D_p$ domains and D a basic domain or the domain Prop.

2. The *order of a domain* D, $\mathsf{ord}(D)$, is defined by

   $$\begin{array}{rcl}\mathsf{ord}(B) & = & 1 \text{ for } B \in \mathcal{B},\\ \mathsf{ord}(\mathsf{Prop}) & = & 2,\\ \mathsf{ord}(D_1\to\ldots\to D_p\to B) & = & max\{\mathsf{ord}(D_i) \mid 1 \leq i \leq p\}, \text{ if } B \in \mathcal{B},\\ \mathsf{ord}(D_1\to\ldots\to D_p\to \mathsf{Prop}) & = & max\{\mathsf{ord}(D_i) \mid 1 \leq i \leq p\} + 1.\end{array}$$

   Note that $\mathsf{ord}(D() = 1$ iff D does not contain Prop. So the 'functional' domains (like for example $(B\to B)\to B$) are of order 1, whereas one might expect them to be of a higher order or not being part of any of the logics. This use of the orders confirms however with the formulas-as-types interpretation that will be studied in the following Chapters. The orders are defined in such a way that in n-th order logic one can quantify over domains of order $\leq n$.

3. For n a fixed positive natural number, the terms of the nth order language are defined as follows. (Each term is an element of a specific domain, which relation is denoted by ϵ).

   - There are countably many variables of domain D for any D with $\mathsf{ord}(D) \leq n$,
   - If $M \,\epsilon\, D_2$, x a variable of domain D_1 and $\mathsf{ord}(D_1\to D_2) \leq n$, then $\lambda x \epsilon D_1.M \,\epsilon\, D_1\to D_2$,
   - If $M \,\epsilon\, D_1\to D_2$, $N \,\epsilon\, D_1$, then $MN \,\epsilon\, D_2$,
   - If $\varphi \,\epsilon\, \mathsf{Prop}$, x a variable of domain D with $\mathsf{ord}(D) \leq n$, then $\forall x \epsilon D.\varphi \,\epsilon\, \mathsf{Prop}$.
   - If $\varphi \,\epsilon\, \mathsf{Prop}$ and $\psi \,\epsilon\, \mathsf{Prop}$, then $\varphi \supset \psi \,\epsilon\, \mathsf{Prop}$.

   The system PRED1 is a special case. In addition to the rules above we have as rules

- There are countably many variables of domain D if $\mathsf{ord}(D) = 2$,
- If $M \,\epsilon\, D_2$, x a variable of domain D_1 and $\mathsf{ord}(D_1{\rightarrow}D_2) = 2$, then $\lambda x \epsilon D_1.M \,\epsilon\, D_1{\rightarrow}D_2$.

The first states that we have arbitrary many predicate symbols. The second allows the definition of predicates by λ-abstraction, e.g. $\lambda x \,\epsilon\, B.\varphi \,\epsilon\, B{\rightarrow}\mathsf{Prop}$.

4. On the terms we have the well-known notion of definitional equality by β-conversion. This equality is denoted by $=$. The terms φ for which $\varphi \,\epsilon\, \mathsf{Prop}$ are called *formulas* and Form denotes the set of formulas.

5. For n a specific positive natural number, we now describe the deduction rules of the nth order predicate logic (in natural deduction style) that allow us to build derivations. So in the following let φ and ψ be formulas of the nth order language.

$$(\supset\text{-I})\quad \dfrac{\begin{matrix}[\varphi]^i\\ \vdots\\ \psi\end{matrix}}{\varphi\supset\psi}\,i \qquad\qquad (\supset\text{-E})\quad \dfrac{\varphi\supset\psi \quad \varphi}{\psi}$$

$$(\forall\text{-I})\quad \dfrac{\psi}{\forall x\epsilon D.\psi}(*) \qquad\qquad (\forall\text{-E})\quad \dfrac{\forall x\epsilon D.\psi}{\psi[t/x]}\ \text{if } t \,\epsilon\, D$$

$$(\text{conv})\quad \dfrac{\psi}{\varphi}\ \text{if } \varphi = \psi$$

The formula occurrences that are between brackets ($[-]$) in the $\supset$-I rule are *discharged*. The superscript i in the $\supset$-I rule is taken from a countable set of indices I. The index i uniquely corresponds to one specific application of the $\supset$-I rule, so we do not allow one index to be used more than once. The use of the indexes allows us to fix those formula occurrences that are discharged at a specific application of the $\supset$-I rule.
$(*)$: in the $\forall$-I rule we make the usual restriction that the variable x may not occur free in a non-discharged assumption of the derivation.
For Γ a set of formulas of PREDn and φ a formula of PREDn, we say that φ *is derivable from* Γ *in* PREDn, notation $\Gamma \vdash_{\mathrm{PRED}n} \varphi$, if there is a derivation with root φ and all non-discharged formulas in Γ.

The system of predicate logic of finite order, notation PREDω, is the union of all PREDn. We follow the usual convention of not writing the number in case of a first order system, so for PRED1 we write PRED.

2.2.7. Remark. The choice for the connectives $\supset$ and $\forall$ may seem minimal. It is however a well-known fact that in second and higher order systems, the intuitionistic connectives &, $\vee$, $\neg$ and $\exists$ can be defined in terms of $\supset$ and $\forall$ as follows. (Let φ and ψ be formulas).

$$\begin{aligned}
\varphi \,\&\, \psi &:= \forall \alpha \epsilon \mathsf{Prop}.(\varphi \supset \psi \supset \alpha) \supset \alpha, \\
\varphi \vee \psi &:= \forall \alpha \epsilon \mathsf{Prop}.(\varphi \supset \alpha) \supset (\psi \supset \alpha) \supset \alpha, \\
\bot &:= \forall \alpha \epsilon \mathsf{Prop}.\alpha, \\
\neg\varphi &:= \varphi \supset \bot, \\
\exists x \,\epsilon\, D \varphi &:= \forall \alpha \epsilon \mathsf{Prop}.(\forall x \epsilon D.\varphi \supset \alpha) \supset \alpha.
\end{aligned}$$

Similarly we can define an equality judgement (the β-equality $=$, the *definitional equality of the language*, is purely syntactical) by taking the so called Leibniz equality: for $t, q \,\epsilon\, D$,

$$t =_D q := \forall P \epsilon D \rightarrow \mathsf{Prop}.Pt \supset Pq,$$

which says that two objects are equal if they satisfy the same properties. (It is not difficult to show that $=_D$ is symmetric).

It is not difficult to check that all the standard logical rules hold for & , $\vee, \bot, \neg, \exists$ and $=$. In the following we shall freely use these symbols.

2.2.8. Remark. In each PREDn ($n \geq 2$), the *comprehension* property is satisfied. That is, for all $\varphi(\vec{x}) : \mathsf{Prop}$ with $\vec{x} = x_1, \ldots, x_p$ a sequence of free variables, possibly occurring in φ ($x_i \,\epsilon\, D_i$), we have

$$\exists P \,\epsilon\, D_1 \rightarrow \cdots D_p \rightarrow \mathsf{Prop}.\forall \vec{x} \epsilon \vec{D}(\varphi \leftrightarrow P x_1 \cdots x_p).$$

(Take $P \equiv \lambda x_1 \,\epsilon\, D_1 \ldots . \lambda x_p \,\epsilon\, D_p.\varphi(\vec{x})$.)

The above definition has some peculiarities that we want to bring into the spotlight. We have allowed countably many variables of all domains of order ≤ 2, which includes for example countably many variables of domain Prop. For first order logic it may seem more natural to allow only variables of domains of order 1, but the slight extension we give here doesn't do us any harm. (It is a conservative extension.) We have also forced the possibility of forming new predicates by λ-abstraction in first order predicate logic. This is unusual (in second and higher order cases this feature is called 'comprehension') and it has only been added to make the formulas-as-types embedding complete on the level of the proofs. Finally we do not have constants, but only variables. This may seem strange but it confirms with the feature that we allow variables of domains of order 2 in first order logic: a binary relation on B is represented by a variable of domain $B \rightarrow B \rightarrow \mathsf{Prop}$. That we don't have constants is also related to the fact that in our presentation a logic is not introduced via a similarity type that fixes the language (mainly by declaring of the constants). Instead what we described above is more a general presentation of the logic that captures all of the logics-with-similarity-type.

In paragraph 2.3 we show some easy conservativity results to justify the choice of our 'extended' systems.

2.2.1. Extensionality

The definitional equality on the terms is β-equality. There is no objection to taking $\beta\eta$-equality instead: all the properties remain to hold. In fact it would make a lot of sense to do so, especially for predicates, where we tend to view λ-abstraction as the necessary mechanism to make comprehension work. (And so both $P \,\epsilon\, B{\to}\mathsf{Prop}$ and $\lambda x \,\epsilon\, B.Px$ describe the collection of elements t of domain B for which Pt holds).

This is related to the issue of *extensionality*: terms of domain $D{\to}\mathsf{Prop}$ are to be understood as predicates on D or also as subsets of D (an element t being in the set $P \,\epsilon\, D{\to}\mathsf{Prop}$ if Pt holds). But if we take this set-theoretic understanding serious, we have to identify predicates that are extensionally equal:

$$(\forall \vec{x}.f\vec{x} \supset g\vec{x} \;\&\; g\vec{x} \supset f\vec{x}) \supset f =_D g. \quad (1)$$

Obviously, this formula is in general not provable. However, in the standard models where predicates are interpreted as real sets, the formula is satisfied, so it is an important extension. A difficulty is, that extensionality in the form of (1) is in general not expressible: in PREDn we can not express extensionality for f and g of domain D if $\mathsf{ord}(D) = n$, because $f =_D g$ is not a formula of PREDn (it uses a quantification over $D{\to}\mathsf{Prop}$). This means that we shall have to express extensionality by a schematic rule. The most obvious choice is the following.

$$\frac{\forall \vec{x}.f\vec{x} \supset g\vec{x} \quad \forall \vec{x}.g\vec{x} \supset f\vec{x} \quad \varphi(f)}{\varphi(g)}$$

where f and g are arbitrary terms of the same domain $D_1{\to}\cdots{\to}D_n{\to}\mathsf{Prop}$ and $\varphi(f)$ stands for a formula φ with a specific marked occurrence of f. For reasons to be discussed presently our choice for the scheme will be a different one, namely the one given in the following definition.

2.2.9. Definition. The *extensionality scheme*, (EXT), is

$$(\mathrm{EXT})\frac{f\vec{x} \supset g\vec{x} \quad g\vec{x} \supset f\vec{x} \quad \varphi(f)}{\varphi(g)}\,(*)$$

where f and g are arbitrary terms of the same domain $D_1{\to}\cdots{\to}D_n{\to}\mathsf{Prop}$ and $\varphi(f)$ stands for a formula φ with a specific marked occurrence of f. $(*)$ signifies

the usual restriction that the variables of $\vec{x}$ may not occur free in a non-discharged assumption of the derivations of $f\vec{x} \supset g\vec{x}$ and of $g\vec{x} \supset f\vec{x}$.
The extension of a system with the rule (EXT) will be denoted by adding the prefix E-, so E-PREDn is extensional nth order predicate logic.

NOTATION. For $f, g \in D = D_1 \to \cdots \to D_n \to \mathsf{Prop}$, if quantification over $D_1, \ldots, D_n$ is allowed in the system we can compress the first two premises in the rule (EXT) to $\forall \vec{x}. f\vec{x} \supset g\vec{x} \mathbin{\&} g\vec{x} \supset f\vec{x}$. For convenience this will also be denoted by $f \sim_D g$, so

$$f \sim_D g := \forall \vec{x}. f\vec{x} \supset g\vec{x} \mathbin{\&} g\vec{x} \supset f\vec{x},$$

where the D will usually be omitted if it is clear from the context.

2.2.10. LEMMA. *The extensionality scheme for $D = \mathsf{Prop}$ is admissible in any of the predicate or propositional logics, i.e.*

$$\varphi \supset \psi, \psi \supset \varphi, \chi(\varphi) \vdash \chi(\psi)$$

is always provable.

PROOF. By an easy induction on the structure of χ. ⊠

Of course there is also a scheme for extensionlity of functions:

$$\frac{f\vec{x} =_B g\vec{x} \quad \varphi(f)}{\varphi(g)}\ (*)$$

where f and g are arbitrary terms of the same domain $D_1 \to \cdots \to D_n \to B$ ($B \in \mathcal{B}$) and further as in Definition 2.2.9. We shall not be working with this scheme and hence not introduce it as a new definition. (Note that, if $\vdash f\vec{x} = g\vec{x}$, then $f =_{\beta\eta} g$).

2.2.2. Some useful variants of the systems

For the systems PREDn of Definition 2.2.6, the scheme (EXT) is equivalent to the scheme that we gave just before Definition 2.2.9. The reason for taking the more general scheme lies in the fact that for reasons of semantics we want to look at slight extensions of the systems in which the two versions of the scheme are not equivalent. these extensions come into consideration quite naturally when one notices that the term language of each of the PREDn is a subsystem of the simply typed lambda calculus, found by restricting to terms below a certain order. So for an interpretation of the term language one is tempted to take a model of the full simply typed lambda calculus. (The interpretation of the logic is then given by describing a binary relation between sets of formulas and formulas.) The syntactical analogue is to allow the term language to be the full simply typed lambda calculus and to put the order-restriction only on quantifications. Then we can show that there is no problem with this extension by establishing a conservativity result between the two systems.

2.2.11. DEFINITION. For L one of our logical systems, say of order n, *L based on the full simply typed lambda calculus*, notation L^τ, is obtained by taking as description of the term language of Definition 2.2.6 the following.

- There are countably many variables of domain D for any $D \in \mathcal{D}$,
- There are countably many constants of domain D for any $D \in \mathcal{D}$,
- If $M \epsilon D_2$, x a variable of domain D_1, then $\lambda x \epsilon D_1.M \epsilon D_1 \to D_2$,
- If $M \epsilon D_1 \to D_2$, $N \epsilon D_1$, then $MN \epsilon D_2$,
- If $\varphi \epsilon \mathsf{Prop}$, x a variable of domain D with $\mathsf{ord}(D) \leq n$, then $\forall x \epsilon D.\varphi \epsilon \mathsf{Prop}$.
- If $\varphi \epsilon \mathsf{Prop}$ and $\psi \epsilon \mathsf{Prop}$, then $\varphi \supset \psi \epsilon \mathsf{Prop}$.

One can now do without the last two cases by taking (for D with $\mathsf{ord}(D) \leq n$) a special fixed constant $\forall_D \epsilon (D \to \mathsf{Prop}) \to \mathsf{Prop}$ and similarly a special fixed constant $\supset \epsilon \mathsf{Prop} \to \mathsf{Prop} \to \mathsf{Prop}$. We do not feel that this is useful thing to do, so we don't do it.

By an easy restriction we define nth order propositional logic from nth order predicate logic.

2.2.12. DEFINITION. For n a natural number, the *nth order propositional logic*, notation PROPn, is defined by removing in the definition of the nth order predicate logic, the set of basic domains $\mathcal{B}$.

2.2.13. LEMMA. *The rule (EXT) implies (conv$_{\beta\eta}$) in propositional logic, i.e. in* E-PROPn,

$$\varphi =_{\beta\eta} \psi \Rightarrow \vdash \varphi \supset \psi.$$

PROOF. We only have to show that if $\varphi \longrightarrow_\eta \psi$, then $\vdash \varphi \supset \psi$. (This is so because of CR for $\beta\eta$ for the term language and the fact that $\varphi = \psi$ implies $\vdash \varphi \supset \psi$. Now let $\varphi \longrightarrow_\eta \psi$, say $\varphi \equiv C[\lambda x \epsilon D.Mx] \longrightarrow_\eta C[M] \equiv \psi$. Now $M \epsilon D \to \cdots \to \mathsf{Prop}$ and $M\vec{x} \supset (\lambda x \epsilon D.Mx)\vec{x}$ and vice versa by the (conv) rule, so $\vdash C[\lambda x \epsilon D.Mx] \supset C[M]$ by (EXT). ⊠

The first order predicate and propositional logics are very minimal: they do not have a connective for negation. (The second order logics do not either but in that case intuitionistic negation can be defined by letting $\bot := \forall \alpha \epsilon \mathsf{Prop}.\alpha$ and $\neg\varphi := \varphi \supset \bot$). This implies that we can not specialize PROP or PRED to a classical variant. Therefore, to define classical first order logic, we have to add negation to the system. (Because of the ideological completeness of $\{\supset, \bot\}$ in classical logic, this is sufficient for a treatment of the full classical proposition and predicate logic. For the intuitionistic case, the extension with just $\bot$ is still quite minimal).

2.2.14. Definition. *First order propositional and predicate logic with negation*, notation $\mathrm{PROP}^{\perp}$ and $\mathrm{PRED}^{\perp}$ are defined by adding to PROP and PRED the following.

1. A fixed constant $\perp \epsilon \mathsf{Prop}$,
2. The derivation rule
$$(\perp)\ \frac{\perp}{\varphi}$$

The classical variants of the logics can be defined in several ways, by adding a rule or an axiom. We choose for a rule in the first order case and an axiom in the higher order case.

2.2.15. Definition. The *classical systems of proposition and predicate logic* are defined by adding the following.

1. For $\mathrm{PROP}^{\perp}$ and $\mathrm{PRED}^{\perp}$ by adding the rule
$$(\neg\neg)\ \frac{\neg\neg\varphi}{\varphi}$$
2. For the other systems PROPn and PREDn by adding the axiom
$$\forall \alpha \epsilon \mathsf{Prop}. \neg\neg\alpha \supset \alpha.$$

Notation. the classical variants of the systems will be denoted by addin g a subscript c. So for example $\mathrm{PROP}^{\perp}_{c}$, $\mathrm{PRED}^{\perp}_{c}$, PROP$n_c$ and PREDn_c. They also have extensional variants, which are defined by adding the scheme (EXT) and which are denoted by adding the prefix E-.

Just as in the first order case there is a faithful translation of the systems of classical higher order logic into the systems of intuitionistic higher order logic. This extends the Gödel translation. The definition we give is the one in [Coquand and Herbelin 1992], where it was described more generally in the form of a so called 'A-translation' in a typed lambda calculus framework.

Let in the following L be one of the intuitionistic logics defined in Definitions 2.2.6, 2.2.12 and 2.2.14, but not one of the minimal systems PROP or PRED, and let L_c be the classical variant of L, as defined in Definition 2.2.15.

2.2.16. Definition. The *Gödel translation* $(-)^{\neg}$ from the terms of L to itself is defined inductively by

$$\begin{array}{rcl}
(x)^{\neg} & = & x, \text{ for } x \text{ a variable or the constant } \perp, \\
(PQ)^{\neg} & = & (P)^{\neg}(Q)^{\neg}, \\
(\lambda x \epsilon D.P)^{\neg} & = & \lambda x \epsilon D.(P)^{\neg}, \\
(\varphi \supset \psi)^{\neg} & = & \neg\neg(\varphi)^{\neg} \supset \neg\neg(\psi)^{\neg}, \\
(\forall x \epsilon D.\varphi)^{\neg} & = & \forall x \epsilon D.\neg\neg(\varphi)^{\neg}.
\end{array}$$

This mapping extends straightforwardly to sets of formulas.

So, for example in the higher order systems $(\bot)^\neg \equiv \forall\alpha.\neg\neg\alpha$, which is logically equivalent to $\bot$. In the first order systems we have $(\bot)^\neg \equiv (\bot \supset \bot) \supset \bot$, which is also logically equivalent to $\bot$. Further it is convenient to remark that $(\neg\neg\varphi)^\neg$ is logically equivalent to $\neg\neg(\varphi)^\neg$

2.2.17. LEMMA. *We have the following properties for $(-)^\neg$. Let t and q be terms, x a variable and D a domain.*

1. $t \,\epsilon\, D \Rightarrow (t)^\neg \,\epsilon\, D$.
2. $(t[q/x])^\neg \equiv (t)^\neg[(q)^\neg/x]$.
3. $t =_\beta q \Rightarrow (t)^\neg =_\beta (q)^\neg$.

PROOF. The first two by an easy induction on the structure of terms. The third by showing the statement for a one step β-reduction and applying the Church-Rosser property. ⊠

2.2.18. THEOREM. *For φ a formula of L, Γ a set of formulas of L,*

$$\Gamma \vdash_{L_c} \varphi \Leftrightarrow \neg\neg(\Gamma)^\neg \vdash_L \neg\neg(\varphi)^\neg.$$

PROOF. From right to left is easy by the fact that $(\varphi)^\neg$ is logically equivalent to φ in classical logic.
From left to right is by induction on the derivation, using Lemma 2.2.17. One also uses the general facts

$$\neg\neg(\varphi \supset \psi) \vdash_L \varphi \supset \neg\neg\psi$$

and

$$\neg\neg(\forall x\epsilon D.\varphi) \vdash_L \forall x\epsilon D.\neg\neg\varphi.$$

Further one has to note that the rule $(\neg\neg)$ is sound in L for formulas of the form $\neg\neg(\ldots)$ (if L is first order) and that $(\forall\alpha.\neg\neg\alpha \supset \alpha)^\neg$ is provable in L (if L is higher order). ⊠

2.3. Some easy conservativity results

This paragraph contains a number of syntactic proofs of conservativity results. The results are relatively easy and not surprising. Most of the work therefore lies in a precise formulation of the notions. First we show that (E)-PREDn^τ (see Definition 2.2.11) is conservative over (E)-PREDn. This means that the extension of the logical language of order n to the full simply typed lambda calculus does not affect the provability.

Furthermore we show that our first order predicate logic with all function domains is conservative over the system that has only function constants (which is more standard). This system, in which it is still possible to define predicates by λ-abstraction, is again conservative over the 'standard' system, where one has only basic predicates. The proof of the latter result will only be outlined. In section 6.5.3 we give a precise proof in terms of typed lambda calculi, using the formulas-as-types embedding. In order to achieve our goals, we first have to give some definitions, writing PRED^{-f} for PRED without function domains and PRED^{-fr} for PRED without function domains and definable predicates. So PRED^{-fr} is the standard minimal first order predicate logic, which has only function constants and predicate constants.

We first turn to the conservativity of (E)-PREDn^τ over (E)-PREDn. We define a mapping from (E)-PREDn^τ to (E)-PREDn which preserves provability.

2.3.1. Definition. Let $n \in \mathbb{N}$. The mapping $(-)^*$ on (E)-PREDn^τ is defined by substituting in a term of (E)-PREDn^τ for all free variables and constants of a domain D of order $> n$, the fixed closed term d_D of domain D, where for $D = D_1 \to \cdots D_m \to \mathsf{Prop}$, d_D is defined by

$$d_D := \lambda x_1 \epsilon D_1 . \ldots \lambda x_m \epsilon D_m . \bot .$$

The image of a term of (E)-PREDn^τ will only contain free variables and constants of domains of order $\leq n$. Furthermore, if $t \epsilon D$, then $t^* \epsilon D$. We now want to take β-normal forms and long-$\beta\eta$-normal forms. Recall that a long-$\beta\eta$-normal form is obtained by first taking the β-normal form and then doing all η-expansions, where $C[q]$ η-expands to $C[\lambda x \epsilon D.qx]$ if $x \notin \mathrm{FV}(q)$ and this does not create a β-redex. (This is well-defined by normalization of β and the fact that if $C[q]$ η-expands to $C[\lambda x \epsilon D.qx]$, we can not expand on q or $\lambda x \epsilon D.qx$ anymore.) The long-$\beta\eta$-normal form of M is denoted by long-$\beta\eta$-nf(M).

2.3.2. Lemma. *If $t \in$ (E)-PREDn^τ with $t \epsilon D$ and $\mathsf{ord}(D) \leq$, then β-nf(t^*) and long-$\beta\eta$-nf(t^*) are in* (E)-PREDn.

Proof. By induction on the structure of β-nf(t^*), respectively the structure of long-$\beta\eta$-nf(t^*). We only treat the proof of the statement that β-nf(t^*) is in (E)-PREDn. t^* contains no free variables or constants of domains of order $> n$. So

$$\beta\text{-nf}(t^*) \equiv \lambda x_1 \epsilon D_1 . \ldots \lambda x_m \epsilon D_m . p Q_1 \cdots Q_r$$

with p a constant, a free variable or one of the x_i. Now, all the domains $D_1, \ldots, D_m$ are of order $\leq n$, so the domain of p is of order $\leq n$. By IH, the terms $Q_1, \ldots, Q_r$ are in (E)-PREDn, and hence β-nf(t^*) is. ⊠

2.3.3. Proposition. *For $n \in \mathbb{N}$ or $n = \omega$ we have the following.*

$$\Gamma \vdash_{\mathrm{PRED}n^\tau} \varphi \;\Rightarrow\; \beta\text{-}\mathsf{nf}(\Gamma^*) \vdash_{\mathrm{PRED}n} \beta\text{-}\mathsf{nf}(\varphi^*),$$
$$\Gamma \vdash_{\text{E-PRED}n^\tau} \varphi \;\Rightarrow\; \textit{long-}\beta\eta\textit{-nf}(\Gamma^*) \vdash_{\text{E-PRED}n} \textit{long-}\beta\eta\textit{-nf}(\varphi^*).$$

PROOF. By induction on the derivation. First remark that $\varphi = \psi \Rightarrow \varphi^* = \psi^*$ and $\varphi^*[P^*/x] \equiv (\varphi[P/x])^*$. Then all cases are easy except for the case when the last rule is (EXT). So say we have

$$\frac{f\vec{x} \supset g\vec{x} \quad g\vec{x} \supset f\vec{x} \quad \varphi(f)}{\varphi(g)}$$

as the last step in the proof. By IH we have

$$\begin{aligned}
\text{long-}\beta\eta\text{-nf}(\Gamma^*) &\vdash \text{long-}\beta\eta\text{-nf}((f\vec{x})^*) \supset \text{long-}\beta\eta\text{-nf}((g\vec{x})^*),\\
\text{long-}\beta\eta\text{-nf}(\Gamma^*) &\vdash \text{long-}\beta\eta\text{-nf}((g\vec{x})^*) \supset \text{long-}\beta\eta\text{-nf}((f\vec{x})^*),\\
\text{long-}\beta\eta\text{-nf}(\Gamma^*) &\vdash \text{long-}\beta\eta\text{-nf}((\varphi(f))^*).
\end{aligned}$$

Now we take a fresh variable z of the same domain as f and g and replace f by z in $\varphi(f)$. We look at the term $\varphi^*(z)$, which is the same as $(\varphi(z))^*$ except for the possible substitution of a term for z, which is not performed. Now

$$(\text{long-}\beta\eta\text{-nf}(\varphi^*(z)))[f^*/z] =_{\beta\eta} \varphi^*(z)[f^*/z] \equiv (\varphi(f))^* =_{\beta\eta} \text{long-}\beta\eta\text{-nf}((\varphi(f))^*).$$

So the third part of the IH can be read as

$$\text{long-}\beta\eta\text{-nf}(\Gamma^*) \vdash (\text{long-}\beta\eta\text{-nf}(\varphi^*(z)))[f^*/z]$$

and we are done if we prove

$$\text{long-}\beta\eta\text{-nf}(\Gamma^*) \vdash (\text{long-}\beta\eta\text{-nf}(\varphi^*(z)))[g^*/z]$$

All occurrences of z in long-$\beta\eta$-nf$(\varphi^*(z))$ are of the form $zq_1 \cdots q_p$ with $zq_1 \cdots q_p \in$ Prop. We have extensionality on the level of Prop (Lemma 2.2.10, so

$$\frac{f^*\vec{q} \supset g^*\vec{q} \quad g^*\vec{q} \supset f^*\vec{q} \quad \psi(f^*\vec{q})}{\psi(g^*\vec{q})} \quad (1)$$

Now, for each occurrence of z in long-$\beta\eta$-nf$(\varphi^*(z))$, the first two premises of (1) are satisfied by IH. So all occurrences of f^* in $(\text{long-}\beta\eta\text{-nf}(\varphi^*(z)))[f^*/z]$ can be replaced by g^* by consecutive applications of rule (1). As conclusion we obtain that $(\text{long-}\beta\eta\text{-nf}(\varphi^*(z)))[g^*/z]$ holds. ⊠

2.3.4. COROLLARY. *For all* $n \in \mathbb{N} \cup \{\omega\}$, (E)-PRED$n^\tau$ *is conservative over* (E)-PREDn.

2.3.5. REMARK. The Proposition and Corollary remain to hold if we replace PRED by PROP everywhere.

2.3.6. COROLLARY. *If* (E)-PROP$(n+1)^\tau$ *is conservative over* (E)-PROPn^τ *then* (E)-PROP$(n+1)$ *is conservative over* (E)-PROPn.

We now turn to the issue of the functional domains and define a subsystem of first order predicate logic (PRED) that only has the simplest domains for functions. (Usually these domains are called 'first order' but this conflicts with our terminology, so we shall refrain from using that term.)

2.3.7. DEFINITION. The language of the system PRED$-f$ is defined as follows.

1. The *domains* are given by
$$\mathcal{D} ::= \mathcal{B} \mid \mathsf{Prop} \mid \mathcal{D} \to \cdots \to \mathcal{D} \to \mathsf{Prop}.$$
So there are basic domains (the ones in B) and predicate domains (the ones that contain Prop).
2. The *functional domains* are given by
$$\mathcal{F} ::= \mathcal{B} \to \cdots \to \mathcal{B},$$
(We assume every functional domain to be built up from at least two basic domains.) Note that $\mathcal{F} \not\subseteq \mathcal{D}$.
3. The *order of a domain* D, $\mathsf{ord}(D)$, is defined as it is done for PRED in 2.2.6. (So the functional domains have no order, which confirms with the intention that in PRED^{-f} there is no quantification over functional domains.)
4. There are countably many function-constants c_i^F for every function domain $F \in \mathcal{F}$ in PRED^{-f}.
5. The terms of the language of PRED^{-f} are described as follows.
   - There are countably many variables of each domain D,
   - If c_i^F is a function constant of domain $F \equiv B_1 \to \cdots \to B_{p+1}$ and $t_i \in B_i$ for $1 \leq i \leq p$, then $c_i^F t_1, \ldots, t_p \in B_{p+1}$,
   - If $t \in D_2$, x a variable of domain D_1 and $\mathsf{ord}(D_1 \to D_2) = 2$, then $\lambda x \epsilon D_1.t \in D_1 \to D_2$,
   - If $t \in D_1 \to D_2$, $q \in D_1$, then $tq \in D_2$,
   - If $\varphi \in \mathsf{Prop}$, x a variable of domain D with $\mathsf{ord}(D) = 1$, then $\forall x \epsilon D.\varphi \in \mathsf{Prop}$.
   - If $\varphi \in \mathsf{Prop}$ and $\psi \in \mathsf{Prop}$, then $\varphi \supset \psi \in \mathsf{Prop}$.

The derivation rules of PRED^{-f} are the same as for PRED, so the quantification is restricted to the domains of order 1 (the $D \in \mathcal{B}$).

It is convenient to let PRED also have constants c_i^F for functional domains F, because then PRED^{-f} is formally a subsystem of PRED. We have the following.

2.3.8. PROPOSITION. PRED *is conservative over* PRED^{-f}, *that is, for* Γ *a set of formulas and* φ *a formula of* PRED^{-f},

$$\Gamma \vdash_{\mathrm{PRED}} \varphi \ \Rightarrow\ \Gamma \vdash_{\mathrm{PRED}^{-f}} \varphi.$$

PROOF. The proof is by cut-elimination and normalization. The notion of cut-elimination will only be discussed in section 3.2.3, so we can only sketch this proof. One can show that, if Γ is a set of formulas and φ a formula of PRED^{-f} such that $\Gamma \vdash_{\mathrm{PRED}} \varphi$ is derivable with derivation Θ, then the derivation Θ', which is obtained from Θ by cut-elimination and normalization of all first order expressions, is a derivation of $\Gamma \vdash \varphi$ in PRED^{-f}. In section 6.5.3 we discuss two typed lambda calculi that correspond to PRED respectively PRED^{-f} by the formulas-as-types embedding. The proof of Proposition 6.5.28 can therefore be seen as a detailed proof of this Proposition. ⊠

This is not yet the end of the story: in the usual first order system one can not define predicates by λ-abstraction, so we want to show that this extension is conservative too.

2.3.9. DEFINITION. The system PRED^{-fr} is PRED^{-f} minus the clause
'If $M \in D_2$, x a variable of domain D_1 and $\mathsf{ord}(D_1 \to D_2) = 2$, then $\lambda x \epsilon D_1.M \in D_1 \to D_2$'
in the term formation rules, and the clause
'If $t \in D_1 \to D_2$, $q \in D_1$, then $tq \in D_2$',
replaced by
'If $t \in D_1 \to \cdots \to D_p \to \mathsf{Prop}$, $q_i \in D_i$ for $1 \leq i \leq p$, then $tq_1 \cdots q_p \in \mathsf{Prop}$'.

In PRED^{-fr} there are no more λ-abstractions. It is the 'usual' system of minimal first order predicate logic: the set of terms of the object language is inductively defined from variables and constants by function application, and the set of formulas is inductively defined from the basic formulas by applying connectives. (Where the basic formulas are of the form $x^D t_1 \cdots t_p$, with t_i terms of the object language, and allowing for $p = 0$). The conservativity of PRED^{-f} over PRED^{-fr} is now proved by normalizing out all λ-abstractions, just like we normalized out all relevant λ-abstractions in the proof of conservativity of PRED over PRED^{-f}.

2.3.10. PROPOSITION. *For* Γ *a set of formula and* φ *a formula of* PRED^{-f},

$$\Gamma \vdash_{\mathrm{PRED}^{-f}} \varphi \Rightarrow \mathit{nf}(\Gamma) \vdash_{\mathrm{PRED}^{-fr}} \mathit{nf}(\varphi).$$

PROOF. Easy induction on the derivation. ⊠

2.3.11. COROLLARY. *For* Γ *a set of formulas and* φ *a formula of* PRED^{-fr},

$$\Gamma \vdash_{\mathrm{PRED}^{-f}} \varphi \Rightarrow \Gamma \vdash_{\mathrm{PRED}^{-fr}} \varphi.$$

PROOF. By the fact that for φ a formula of PRED^{-fr}, $\varphi \equiv \mathrm{nf}(\varphi)$. ⊠

2.4. Conservativity between the logics

Having justified the systems PREDn in relation to more standard presentations of predicate logic, we now want to say something about the conservativity relations between the systems themselves. This gives a better understanding of the logics while at the same time these results will be useful later for reference when we discuss the conservativity relations between systems of typed lambda calculus in Chapter 6.1. So this paragraph may be skipped for now if one is merely interested in the typed lambda calculi. The conservativity relations betweem the logics can be collected in the following diagram.

$$\begin{tikzcd}
& \mathrm{PROP}\omega \arrow[r, dotted] & \mathrm{PROP}\omega_c & & \mathrm{PRED}\omega \arrow[r, dotted] & \mathrm{PRED}\omega_c \\
& \mathrm{PROP}3 \arrow[u] \arrow[r, dotted] & \mathrm{PROP}3_c \arrow[u] & & \mathrm{PRED}3 \arrow[u, dotted] \arrow[r, dotted] & \mathrm{PRED}3_c \arrow[u, dotted] \\
& \mathrm{PROP}2 \arrow[u] \arrow[r, dotted] & \mathrm{PROP}2_c \arrow[u] & & \mathrm{PRED}2 \arrow[u, dotted] \arrow[r, dotted] & \mathrm{PRED}2_c \arrow[u, dotted] \\
& \mathrm{PROP}^{\perp} \arrow[u] \arrow[r, dotted] & \mathrm{PROP}^{\perp}_c \arrow[u] & & \mathrm{PRED}^{\perp} \arrow[u, dotted] \arrow[r, dotted] & \mathrm{PRED}^{\perp}_c \arrow[u, dotted] \\
\mathrm{PROP} \arrow[uur] \arrow[ur] \arrow[urr, dotted] & & & \mathrm{PRED} \arrow[uur] \arrow[ur] \arrow[urr, dotted] & &
\end{tikzcd}$$

where a dotted arrow depicts a non-conservative inclusion and an ordinary arrow depicts a conservative inclusion. The (non-)conservativities between predicate logic and propositional logic follow by the fact that any predicate logic on the right is conservative over its propositional variant on the left, and further by transitivity of conservativity and the fact that if L_2 is not conservative overL_1 and $L_2 \subset L_3$, then L_3 is not conservative overL_1.

We do not present this diagram as a theorem, because for some of the depicted arrows we have no proof. In this section, only a small part of the diagram above will be proved formally. One of the things we do not prove is the whole tower of vertical arrows in the propositional part. We only prove the conservativities for extensional versions of the systems. This implies the conservativity of PROPn over PROP2 for any $n \geq 2$ (and similarly for the classical variants).

Also the vertical tower of arrows in the predicate part of the diagram will not be proved. For $n \geq 2$, we believe that non-conservativity can be proved by looking at a structure for Arithmetic in each of the logics. Then one obtains nth order Heyting Arithmetic on the left side and nth order Peano Arithmetic on the right side. Then Gödel's Second Incompleteness Theorem says that each of those systems can not prove its own consistency. Then the non-conservativity can be established by showing that $(n+1)$th order Arithmetic can prove the consistency of nth order Arithmetic.

A similar method should apply to the systems PRED2 and PRED$^\bot$, respectively PRED2$_c$ and PRED$^\bot_c$. For the classical variants this is straightforward: PRED$^\bot_c$ may seem minimal, but due to classical logic, all connectives can be defined in terms of $\supset$, $\forall$ and $\bot$. Hence we can look at Robinson's system Q for Arithmetic, for which Gödel's Second Incompleteness Theorem already applies. The non-conservativity of PRED2 over PRED$^\bot$ can then be derived from the non-conservativity of PRED2$_c$ over PRED$^\bot_c$ by applying a version of the Gödel's double negation translation. This is a faithful mapping from PRED2 respectively PRED$^\bot$ to PRED2$_c$ respectively PRED$^\bot_c$. (See section 6.5.3.)

The conservativity of PROP2 over PROP and of PRED2 over PRED will be discussed later when we look at typed lambda calculus versions of the systems. Then we shall describe mappings from the larger system to the smaller one that also take into account the proofs. From the conservativity of PROP2 over PROP and of PRED2 over PRED it immediately follows that PROP$^\bot$ is conservative over PROP and that PRED$^\bot$ is conservative over PRED.

The non-conservativity of PROP$^\bot_c$ over PROP is easy: $((\alpha\to\beta)\to\alpha)\to\alpha$ is provable in PROP$^\bot_c$, but not in PROP. A derivation of it in PROP$^\bot_c$ is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\alpha\supset\beta]\quad(\alpha\supset\beta)\supset\alpha}{\alpha}\qquad[\neg\alpha]}{\bot}}{\neg(\alpha\supset\beta)}\qquad\dfrac{\dfrac{\dfrac{[\neg\alpha]\;[\alpha]}{\bot}}{\beta}}{\alpha\supset\beta}}{\bot}}{\neg\neg\alpha}}{\alpha}$$

It can easily be seen that $((\alpha\to\beta)\to\alpha)\to\alpha$ is not provable in PROP by noticing that there is no closed term of type $((\alpha\to\beta)\to\alpha)\to\alpha$ in the simply typed lambda calculus (which is saying the same as 'there exists no cut-free proof of $((\alpha\to\beta)\to\alpha)\to\alpha$ in PROP'). The example $((\alpha\to\beta)\to\alpha)\to\alpha$ also applies for showing the non-conservativity of PRED$^\bot_c$ over PRED.

It is obvious that the conservativity of the classical version of the logic over the intuitionistic version never holds, hence the dotted arrows from left to right in the diagram.

Note further that any predicate logic is conservative over its propositional version. This is easily seen by defining a mapping $[-]$ from formulas of the predicate logic to the propositional logic that preserves derivability and is the identity on the propositional logic. It can be defined as follows.

2.4.1. DEFINITION. Let L_2 be a system of predicate logic and L_1 its propositional variant. The mapping $[-]$ is defined on predicate domains of L_2 (the ones of the form $\cdots\to\mathsf{Prop}$) by just removing all the basic domains, so for example $[(B\to\mathsf{Prop})\to\mathsf{Prop}] = \mathsf{Prop}\to\mathsf{Prop}$. Then $[-] : \mathrm{Form}(L_2) \to \mathrm{Form}(L_1)$ is defined as follows.

$$\begin{array}{rcl}
[x_i^D] & = & x_i^{[D]},\\
[\varphi \supset \psi] & = & [\varphi] \supset [\psi],\\
[\forall x\epsilon A.\varphi] & = & \forall x\epsilon[A].[\varphi] \text{ if } A \equiv \cdots\to\mathsf{Prop},\\
 & = & [\varphi] \text{ else},\\
[\lambda x\epsilon A.M] & = & \lambda x\epsilon[A].[M] \text{ if } A \equiv \cdots\to\mathsf{Prop},\\
 & = & [M] \text{ else},\\
[PM] & = & [P][M] \text{ if } M{:}A \equiv \cdots\to\mathsf{Prop},\\
 & = & [P] \text{ else},
\end{array}$$

This map is very similar to the one in Definition 6.5.23, which shows the conservativity of dependent typed lambda calculus over non-dependendent typed lambda calculi.

It is easily shown that this map satisfies the requirements.

The proof of conservativity of extensional PROP$(n+1)$ over extensional PROPn is given by semantical methods. We give a notion of model in terms of complete Heyting algebras that is sound and complete for each of the E-PROPn. We shall also describe a Kripke semantics for PROPn (non-extensional). We had hoped to prove the conservativity of PROP$(n+1)$ over PROPn by using this semantics. However, although we have a sound and complete model notion for each of the PROPn, we haven't been able to derive conservativity because a Kripke model of PROPn is not immediately a Kripke model of PROP$(n+1)$.

The proof of conservativity of E-PROP$(n+1)_c$ over E-PROPn_c follows directly from the proof of conservativity of E-PROP$(n+1)$ over E-PROPn. (Just add the axiom $\forall\alpha\alpha\vee\neg\alpha$ everywhere). Nevertheless we also describe a truth table semantics for E-PROPn_c, because it is the basic semantics for classical propositional logics. Further it shows not only the conservativity of E-PROP$(n+1)_c$ over E-PROPn_c, but also the decidability of E-PROPn_c (for any $n \geq 2$). This should be contrasted with intuitionistic versions of propositional logic: all the systems

PROPn ($n \geq 2$), extensional or not, are undecidable. This is a consequence of the undecidabilty of PROP2, shown by [Löb 1976], and the conservativity of (extensional) PROPn over PROP2 for all $n \geq 2$.

2.4.1. Truth table semantics for classical propositional logics

The method of deciding the validity of a judgement $\Gamma \vdash \varphi$ in classical logic by using truth tables immediately extends to the second order case by letting the value of α vary through $\{0,1\}$ in the interpretation of φ. For higher orders we have to be a bit more careful. The straightforward thing to do is to let for example the value of variables of domain $\mathsf{Prop}{\rightarrow}\mathsf{Prop}$ vary through the set of functions from $\{0,1\}$ to $\{0,1\}$. This, however, gives a model that is not complete, because it is too extensional compared with the syntax, in the sense that e.g. for all $f, g \,\epsilon\, \mathsf{Prop}{\rightarrow}\mathsf{Prop}$,

$$(\forall \alpha \epsilon \mathsf{Prop}. f\alpha \supset g\alpha \;\&\; g\alpha \supset f\alpha) \supset (f =_{\mathsf{Prop}\rightarrow\mathsf{Prop}} g)$$

is satisfied in it. (The equality is the definable Leibniz' equality). We shall show that the truth table model is complete for the extensional version of the logic.

Extensionality is not derivable in any of the logics. This can for example be seen from the fact that if

$$\vdash_{\mathrm{PROP4}_c} \forall f, g \epsilon \mathsf{Prop}{\rightarrow}\mathsf{Prop}. (f \sim g) \supset f = g,$$

then (for P a variable of the appropriate domain)

$$P(\lambda\alpha.\alpha \supset \alpha \supset \alpha), \neg P(\lambda\alpha.\alpha \supset \alpha) \vdash_{\mathrm{PROP4}_c} \bot$$

by the fact that $\lambda\alpha.\alpha \supset \alpha \supset \alpha$ and $\lambda\alpha.\alpha \supset \alpha$ satisfy the assumption for f and g in the extensionality. Now by applying the Gödel's $\neg\neg$-translation of Definition 2.2.16, we obtain

$$\neg\neg P(\lambda\alpha.\neg\neg\alpha \supset \neg\neg(\neg\neg\alpha \supset \neg\neg\alpha)), \neg P(\lambda\alpha.\neg\neg\alpha \supset \neg\neg\alpha) \vdash_{\mathrm{PROP4}} \bot.$$

This, however can only be the case if $\lambda\alpha.\neg\neg\alpha \supset \neg\neg(\neg\neg\alpha \supset \neg\neg\alpha) =_\beta \lambda\alpha.\neg\neg\alpha \supset \neg\neg\alpha$, which is clearly not the case.

2.4.2. Definition. For every domain D we define the set V_D of possible values for the terms of domain D as follows.

$$\begin{array}{rcl} V_{\mathsf{Prop}} & = & \{0,1\}, \\ V_{D_1 \rightarrow D_2} & = & V_{D_1} \rightarrow V_{D_2}, \text{the set of functions from } V_{D_1} \text{ to } V_{D_2}. \end{array}$$

The interpretation of terms as values (modulo a valuation of the free variables) is now straightforward, given the following definitions.

2.4.3. Definition. Any valuation v that maps variables to values of the appropriate set extends immediately to an interpretation v on all terms as follows.

$$\begin{array}{rcl} v(\lambda x\epsilon D.P) & = & \boldsymbol{\lambda} a \in V_D.v[x := a](P), \\ v(PQ) & = & v(P)v(Q), \\ v(\varphi \supset \psi) & = & 0 \text{ if } v(\varphi) = 1 \text{ and } v(\psi) = 0, \\ & = & 1 \text{ otherwise}, \\ v(\forall x\epsilon D.\varphi) & = & 1 \text{ if for all } a \in V_D,\ v[x := a](\varphi) = 1, \\ & = & 0 \text{ otherwise.} \end{array}$$

Here $v[x := a]$ denotes the valuation with $v[x := a](x) = a$ and $v[x := a](y) = v(y)$ if $x \neq y$.

As was to be expected, the value of a closed term does not depend on the particular choice for v and values are stable under $\beta\eta$-equality.

2.4.4. Definition. For Γ a set of formulas and φ a formula of any of the propositional logics, we define

$$\Gamma \models \varphi := \text{for all valuations } v,\ v(\Gamma) = 1 \Rightarrow v(\varphi) = 1,$$

where $v(\Gamma) = 1$ if $v(\psi) = 1$ for all $\psi \in \Gamma$.
We say that φ *is true* if $\models \varphi$.
(The subscripts will usually be omitted).

2.4.5. Proposition (Soundness). *For Γ a set of formulas and φ a formula of* E-PROPn_c^τ,

$$\Gamma \vdash_{\text{E-PROP}n_c^\tau} \varphi \Rightarrow \Gamma \models \varphi.$$

Proof. By an easy induction on the derivation. ⊠

2.4.6. Lemma. *For any domain D, all values of V_D are* λ-definable *in* E-PROPn_c^τ. *That is, for all $F \in V_D$ there is a closed term t of domain D in* E-PROPn_c^τ *such that*

$$v(t) = F$$

(for any valuation v).

Proof. By induction on the structure of D. The proof uses the fact that, due to the extensionality, one can define a function by cases in the logic. For example the value in $(\{0,1\} \rightarrow \{0,1\}) \rightarrow \{0,1\}$ that maps the identity and the swop function to 0 and the two constant functions to 1 can be defined in the syntax by

$$\lambda f\epsilon \mathsf{Prop}{\rightarrow}\mathsf{Prop}.(f \sim \lambda\alpha.\alpha \vee f \sim \lambda\alpha.\neg\alpha) \supset \bot \ \&\ (f \sim \lambda\alpha.\bot \vee f \sim \lambda\alpha.\top) \supset \top.$$

In general, a function $F : V_{D_1}\rightarrow\cdots\rightarrow V_{D_p}\rightarrow\{0,1\}$ can be described in the format

$$\begin{aligned} F v_1,\cdots v_p &= 0 \text{ if } v_1 = t_1 \text{ and } \ldots \text{ and } v_p = t_p, \\ &= 1 \text{ if } \cdots, \\ &= \cdots, \end{aligned}$$

where we just go through all the possible input values. By IH we know how to λ-define all the elements of $V_{D_1},\ldots,V_{D_p}$, so we can translate the format for F into a λ-term by replacing the t_i by its defining element and $=$ by $\simeq$, where $x \simeq y$ for x and y of domain $D'_1\rightarrow\cdots\rightarrow D'_q\rightarrow\mathsf{Prop}$ is defined by $\bigwedge(x\vec{t} \sim y\vec{t})$ with $\bigwedge$ the finite generalised conjunction that lets $\vec{t}$ vary through the sequences of defining elements of $D'_1,\ldots,D'_q$. ⊠

For example 0 can be defined by $\perp$ and 1 by $\top$.

Due to the previous lemma we can internalize a valuation v in the syntax. This is done by substituting for the free variable x the term that λ-defines $v(x)$. We introduce the following notation.

NOTATION. For v a valuation, the substitution that replaces a free variable x by the closed term that λ-defines $v(x)$, will be denoted by Σ_v. (So, for example, for v with $v(\alpha^{\mathsf{Prop}}) = 0$, Σ_v substitutes $\perp$ for α).

The lemma also states that any V_D can be summed up by closed terms, i.e. we can always write $V_D = \{v(t_1), v(t_2),\ldots, v(t_p)\}$, for some closed terms $t_1, t_2,\ldots,t_p$, where v is totally arbitrary. This fact can even be proved inside the logic.

2.4.7. LEMMA. *In* E-PROPn_c, *if* $\mathsf{ord}(D) < n$ *and* $V_D = \{v(t_1), v(t_2),\ldots, v(t_p)\}$, *then*

$$\vdash \forall f\epsilon D. f = t_1 \vee f = t_2 \vee \cdots \vee f = t_p.$$

PROOF. By induction on the structure of D, by proving

$$f \neq t_1 \supset f \neq t_2 \supset \cdots \supset f \neq t_{p-1} \supset f = t_p.$$

The proof uses extensionality in the form of

$$f \neq t_i \vdash \exists\vec{x}.(f\vec{x} \mathbin{\&} \neg t_i\vec{x}) \vee (\neg f\vec{x} \mathbin{\&} t_i\vec{x})$$

which is provable from the extensionality axioms.

The reason that the lemma does not hold for all domains of the logic is simply because for domains of order n the formula

$$\vdash \forall f\epsilon D. f = t_1 \vee f = t_2 \vee \cdots \vee f = t_p$$

is not in the language of E-PROPn_c. ⊠

The lemma says, among other things, that $\vdash \forall\alpha\epsilon\mathsf{Prop}.(\alpha = \top \vee \alpha = \bot)$ is provable in E-PROP3$_c$. Let's shortly digress on how one proves this fact as an illustration of the proof. Extensionality in E-PROP3$_c$ implies the following axiom.

$$\forall\alpha,\beta\epsilon\mathsf{Prop}.(\alpha \sim \beta) \supset \alpha = \beta$$

Now $\alpha \vdash \alpha \sim \top$ and $\neg\alpha \vdash \alpha \sim \bot$, hence $\alpha \vee \neg\alpha \vdash \alpha = \top \vee \alpha = \bot$ by extensionality, and so $\vdash \forall\alpha\epsilon\mathsf{Prop}.(\alpha = \top \vee \alpha = \bot)$.

We have a version of Lemma 2.4.7 for domains of order n in E-PROPn_c. It is strong enough for our purposes.

2.4.8. Lemma. *In* E-PROPn_c, *if* $V_D = \{v(t_1), \ldots, v(t_p)\}$, *then*

$$\forall f\epsilon D.f \sim t_1 \vee \cdots \vee f \sim t_p.$$

Proof. For domains of order $< n$ the lemma follows immediately from the previous one (Lemma 2.4.7). For domains D of order n we have to do a case analysis and use the previous Lemma. What one really proves is

$$\vdash \forall f\epsilon D.(\exists\vec{x}.f\vec{x} \neq t_1\vec{x}) \supset \cdots \supset (\exists\vec{x}.f\vec{x} \neq t_{p-1}\vec{x}) \supset (\forall\vec{x}.f\vec{x} = t_p\vec{x})$$

which is sufficient. We give some details for the case of the domain $\mathsf{Prop}{\rightarrow}\mathsf{Prop}$ in E-PROP3$_c$. We have to prove

$$\vdash \forall f\epsilon\mathsf{Prop}{\rightarrow}\mathsf{Prop}.(\exists\alpha.f\alpha \neq \alpha) \supset (\exists\alpha.f\alpha \neq \neg\alpha) \supset (\exists\alpha.f\alpha \neq \top) \supset (\forall\alpha.f\alpha = \bot).$$

This is easily done by deriving a contradiction from $\exists\alpha.f\alpha \neq \alpha$, $\exists\alpha.f\alpha \neq \neg\alpha$, $\exists\alpha.f\alpha \neq \top$ and $(f\top = \top) \vee (f\bot = \top)$. ⊠

2.4.9. Proposition. *In* E-PROPn_c, *for* v *a valuation,*

$$\begin{aligned} v(\varphi) = 1 &\;\Rightarrow\; \vdash \Sigma_v(\varphi),\\ v(\varphi) = 0 &\;\Rightarrow\; \vdash \neg\Sigma_v(\varphi), \end{aligned}$$

Proof. Simultaneously, by induction on the structure of the normal form of φ. For $\varphi \equiv \forall x\epsilon D.\psi$ we distinguish two subcases: $\mathsf{ord}(D) = n$ and $\mathsf{ord}(D) < n$. We treat both subcases for $v(\varphi) = 1$.

Suppose $v(\forall x\epsilon D.\psi) = 1$ and $\mathsf{ord}(D) < n$. Then $v[x := F](\psi) = 1$ for all $F \in V_D$. Say $V_D = \{v(t_1), \ldots, v(t_p)\}$ (which is justified by Lemma 2.4.6). Then by IH

$$\vdash \Sigma_v[x := t_i](\psi)$$

for all t_i $(1 \leq i \leq p)$. By Lemma 2.4.7 we know that

$$\vdash x = t_1 \vee \cdots \vee x = t_p,$$

so we can do a case analysis to find

$$\vdash \Sigma_v(\psi).$$

Now Σ_v does not substitute anything for x, so x is still free in $\Sigma_v(\psi)$. We may conclude

$$\vdash \Sigma_v(\forall x \epsilon D.\psi).$$

Suppose now that $v(\forall x \epsilon D.\psi) = 1$ with $\mathsf{ord}(D) = n$. Then again $v[x := F](\psi) = 1$ for all $F \in V_D$. (Say $V_D = \{v(t_1), \dots, v(t_p)\}$.) Again by IH

$$\vdash \Sigma_v[x := t_i](\psi)$$

for all t_i $(1 \leq i \leq p)$. By Lemma 2.4.8 we know that

$$\vdash x \sim t_1 \vee \cdots \vee x \sim t_p.$$

This is not as strong as what we had in the first case, but it still suffices because we may assume that in ψ all occurrences of x appear in the form $(xq_1 \cdots q_r)$ with $xq_1 \cdots q_r \,\epsilon\, \mathsf{Prop}$, i.e. x occurs only as a real function. (If ψ is not yet of this shape we η-expand it). We can do a case analysis to find

$$\vdash \Sigma_v(\psi).$$

Again x is free in $\Sigma_v(\psi)$ and we can conclude

$$\vdash \Sigma_v(\forall x \epsilon D.\psi). \boxtimes$$

2.4.10. Corollary (Completeness). *In* E-PROPn_c, *for* φ *a formula*

$$\models \varphi \;\Rightarrow \vdash \varphi.$$

Proof. $\models \varphi$ means $\forall v.v(\varphi) = 1$, so by the Proposition $\vdash \Sigma_v(\varphi)$ for any valuation v. Hence $\vdash \varphi$ because we can make all the necessary case distinctions by Lemma 2.4.7 and Lemma 2.4.8. $\boxtimes$

2.4.11. Corollary. *All* E-PROPn_c *are decidable.*

Proof. Immediate from the previous Corollary and the Soundness (Proposition 2.4.5) by the fact that the validity of a formula can always be checked in a finite part of the truth table model. $\boxtimes$

2.4.12. Proposition. E-PROP$(n+1)_c$ *is conservative over* E-PROPn_c *($n \neq \omega$), and hence* E-PROPω_c *is conservative over each of the* E-PROPn_c.

Proof. By the fact that the truth table model is a model for all the E-PROPn_c. $\boxtimes$

2.4.13. Corollary. PROPn_c *is conservative over* PROP2_c *for each* n.

Proof. Immediate from the fact that PROP2_c and E-PROP2_c are the same system. (By Lemma 2.2.10). $\boxtimes$

2.4.2. Algebraic semantics for intuitionistic propositional logics

In this section we describe a semantics for our systems of intuitionistic propositional logic in terms of Heyting algebras. It is well-known how this is done for the full first order propositional logic, giving rise to a completeness result. For second and higher order propositional logic we need to refine the notion of Heyting algebra to also allow interpretations for the universal quantifier. It is easily seen that *complete* Heyting algebras are strong enough to satisfy our purpose: complete Heyting algebras have arbitrary meets and joins, so for example $\forall f \in \mathsf{Prop}{\rightarrow}\mathsf{Prop}.\varphi$ can be interpreted as $\bigwedge\{[\![\varphi]\!]_{[f:=F]} \mid F \in A{\rightarrow}A\}$. It is however not so easy to show the completeness of complete Heyting algebras over E-PROPn (for any n), because the Lindenbaum algebra defined from E-PROPn is not a complete Heyting algebra. The way out was suggested by Theorem 13.6.13 of [Troelstra and Van Dalen 1988], stating that any Heyting algebra can be embedded in a complete Heyting algebra such that $\supset$, $\perp$ and all existing $\bigvee$ and $\bigwedge$ are preserved (and hence the ordering is preserved). The embedding i that is constructed in the proof is also faithful with respect to the ordering, that is, if $i(a) \leq i(b)$ in the image, then $a \leq b$ in the original Heyting algebra. All this implies completeness of complete Heyting algebras with respect to E-PROPn, for any n. Hence we have conservativity of E-PROP$(n+1)$ over E-PROPn.

In fact the argument that we use gives a completeness result for the systems E-PROPn^τ, which is E-PROPn based on the language of the full simply typed lambda calculus. This is only done to make things slightly easier and it does not have any effect on the results. (See also Remark 2.3.5).

At this point we do not know how (if at all possible) to conclude the conservativity of PROP$(n+1)$ over PROPn from the conservativity of E-PROP$(n+1)$ over E-PROPn. However, we do have the conservativity of PROPn over PROP2 for any n, because PROP2 and E-PROP2 are the same system.

It is obvious that extensionality is required in the syntax because the model notion is extensional: if, for example, $F, G : A{\rightarrow}A$ (where A is the carrier set of the algebra) and $F(a) = G(a)$ for all $a \in A$, then $F = G$.

The method of showing conservativity by semantical means seems to be quite essential here. Most of the other conservativity proofs in this chapter use mappings from the 'larger' system to the 'smaller' system that are the identity on the smaller system. These mappings also constitute a mapping from derivations to derivations that is the identity on derivations of the smaller system. For the case of intuitionistic propositional logics, this method seems to be essentially impossible: there are formulas of PROP2 that have more and more cut-free derivations when we go higher in the hierarchy of propositional logics.

2.4.14. Definition. A *Heyting algebra* (or just Ha) is a tuple $(A, \wedge, \vee, \perp, \supset)$ such that $(A, \wedge, \vee)$ is a lattice with least element $\perp$ and $\supset$ is a binary operation with

$$a \wedge b \leq c \Leftrightarrow a \leq b \supset c.$$

Remember that $(A, \wedge, \vee)$ is a lattice if the binary operations $\wedge$ and $\vee$ satisfy the following requirements.

$$\begin{array}{rclrcl} a \wedge a & = & a, & a \vee a & = & a, \\ a \wedge b & = & b \wedge a, & a \vee b & = & b \vee a, \\ a \wedge (b \wedge c) & = & (a \wedge b) \wedge c, & a \vee (b \vee c) & = & (a \vee b) \vee c, \\ a \vee (a \wedge b) & = & a, & a \wedge (a \vee b) & = & a. \end{array}$$

Another way of defining the notion of lattice is by saying that it is a poset $(A, \leq)$ with the property that each pair of elements $a, b \in A$ has a least upperbound (denoted by $a \vee b$) and a greatest lowerbound (denoted by $a \wedge b$). By defining $a \leq b := a \wedge b = a$ we can then show the equivalence of the two definitions of lattice.

2.4.15. Definition. A *complete Heyting algebra* (cHa) is a tuple $(A, \bigwedge, \bigvee, \bot, \supset)$ such that $(A, \bigwedge, \bigvee)$ is a complete lattice and $(A, \wedge, \vee, \bot, \supset)$ is a Heyting algebra. (So $\bigvee$ and $\bigwedge$ are mappings from $\wp(A)$ to A such that for $X \subset A$, $\bigvee X$ is the least upperbound of X and $\bigwedge X$ is the greatest lower bound of X. The binary operations $\wedge$ and $\vee$ are defined by (for $a, b \in A$) $a \wedge b := \bigwedge\{a, b\}$ and $a \vee b := \bigvee\{a, b\}$).

An important feature of Heyting algebras which is forced upon by the presence of the binary operation $\supset$, is that they satisfy the infinitary distributive law:

$$(D) \quad a \wedge \bigvee X = \bigvee\{a \wedge b \mid b \in X\}, \text{ if } \bigvee X \text{ exists.}$$

(The inclusion $\supseteq$ holds in any lattice; for the inclusion $\subseteq$ it is enough to show that $a \wedge c \subseteq \bigvee\{a \wedge b \mid b \in X\}$ for any $c \in X$, due to the properties of $\supset$).

Two other important facts are the following.

2.4.16. Fact. 1. If a complete lattice satisfies the infinitary distributive law (D), it can be turned into a cHa by defining

$$b \supset c := \bigvee\{d \mid d \wedge b \leq c\}.$$

2. Any Heyting algebra is distributive, i.e. any Ha satisfies

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c).$$

For the first statement one has to show that $a \wedge b \leq c \Leftrightarrow a \leq \bigvee\{d \mid d \wedge b \leq c\}$. From left to right is easy; from right to left, notice that if $a \leq \bigvee\{d \mid d \wedge b \leq c\}$, then $a \wedge b \leq b \wedge \bigvee\{d \mid d \wedge b \leq c\}$ and the latter is (by D) equal to $\bigvee\{b \wedge d \mid d \wedge b \leq c\}$, which is just c. The second is easily verified.

We are now ready to give the algebraic semantics for the systems E-PROPn^τ. (A logical system L^τ is based on the full simply typed lambda calculus, see Definition 2.2.11). Let in the following $(A, \bigwedge, \bigvee, \bot, \supset)$ be a cHa. We shall freely use the notions $\vee$ and $\wedge$, as they were given in Definition 2.4.15. The interpretation of the terms of E-PROPn will be in A and its higher order function spaces. We therefore let $\lceil - \rceil$ be the mapping that associates the right function space to a domain D, so

$$\begin{array}{rcl}\lceil \mathsf{Prop} \rceil & = & A, \\ \lceil D_1 \to D_2 \rceil & = & \lceil D_1 \rceil \to \lceil D_2 \rceil,\end{array}$$

where the second $\to$ describes function space. In the following we shall freely speak of the 'interpretation of E-PROPn^τ in $(A, \bigwedge, \bigvee, \bot, \supset)$', where of course this interpretation includes the mapping of higher order terms into the appropriate higher order function space based on A.

2.4.17. Definition. Let $n \in \mathbb{N} \cup \{\omega\}$. An *algebraic model of* E-PROPn^τ is a pair $(\Theta, \mathcal{C})$, with Θ a cHa and $\mathcal{C}$ a valuation of the constants in Θ such that, if c is a constant of domain D, then $\mathcal{C}(c) \in \lceil D \rceil$.

2.4.18. Definition. The interpretation of E-PROPn^τ in the algebraic model $((A, \bigwedge, \bigvee, \bot, \supset), \mathcal{C})$, $[\![-]\!]$, is defined modulo a valuation ρ for free variables that maps variables of domain D into $\lceil D \rceil$. So let ρ be a valuation. Then $[\![-]\!]_\rho$ is defined inductively as follows.

$$\begin{array}{rcl}[\![c]\!]_\rho & = & \mathcal{C}(c), \text{ for } c \text{ a constant}, \\ [\![\alpha]\!]_\rho & = & \rho(\alpha), \text{ for } \alpha \text{ a variable}, \\ [\![PQ]\!]_\rho & = & [\![P]\!]_\rho [\![Q]\!]_\rho, \\ [\![\lambda x \epsilon D.Q]\!]_\rho & = & \boldsymbol{\lambda} t \in \lceil D \rceil . [\![Q]\!]_{\rho(x:=t)}, \\ [\![\varphi \supset \psi]\!]_\rho & = & [\![\varphi]\!]_\rho \supset [\![\psi]\!]_\rho, \\ [\![\forall x \epsilon D.\varphi]\!]_\rho & = & \bigwedge\{[\![\varphi]\!]_{\rho(x:=t)} \mid t \in \lceil D \rceil\}.\end{array}$$

It is easily seen that $[\![-]\!]_\rho$ satisfies the usual substitution property and that interpretations are stable under $\beta\eta$-equality, i.e.

$$[\![P]\!]_{\rho(x:=[\![Q]\!]_\rho)} = [\![P[Q/x]]\!]_\rho$$

and

$$P =_{\beta\eta} Q \Rightarrow [\![P]\!]_\rho = [\![Q]\!]_\rho.$$

2.4.19. Definition. For Γ a finite set of formulas of E-PROPn^τ, φ a formula of E-PROPn^τ and $(\Theta, \mathcal{C})$ an algebraic model, φ *is* $(\Theta, \mathcal{C})$*-valid in* Γ, notation $\Gamma \models_{(\Theta,\mathcal{C})} \varphi$, if for all valuations ρ,

$$\bigwedge\{[\![\psi]\!]_\rho \mid \psi \in \Gamma\} \leq [\![\varphi]\!]_\rho.$$

If Γ is empty we say that φ *is* $(\Theta, \mathcal{C})$*-valid* if $\models_{(\Theta,\mathcal{C})} \varphi$.

Note that $\bigwedge\{[\![\psi]\!]_\rho \mid \psi \in \Gamma\}$ exists, beacuse Γ is finite. In the following we just write $[\![\Gamma]\!]_\rho$ for $\bigwedge\{[\![\psi]\!]_\rho \mid \psi \in \Gamma\}$.

Our definition is a bit different from the one in [Troelstra and Van Dalen 1988], where $\Gamma \models_{(\Theta,\mathcal{C})} \varphi$ is defined by

$$\forall \psi \in \Gamma[[\![\psi]\!]_\rho = \top] \;\Rightarrow\; [\![\varphi]\!]_\rho = \top.$$

Our notion implies the one above, but not the other way around. However, they are the same if $\Gamma = \emptyset$ and they also yield the same consequence relation. One disadvantage of our notion is that we have to restrict to finite Γ. This is easily overcome by putting

$$\Gamma \models_{(\Theta,\mathcal{C})} \varphi \text{ if for all finite } \Gamma' \subseteq \Gamma, \Gamma' \models_{(\Theta,\mathcal{C})} \varphi.$$

2.4.20. Definition. Let Γ be a (finite) set of formulas of E-PROPn^τ and φ a formula of E-PROPn^τ. We say that φ *is a consequence of* Γ, notation $\Gamma \models \varphi$, if $\Gamma \models_{(\Theta,\mathcal{C})} \varphi$ for all algebraic models $(\Theta,\mathcal{C})$.

2.4.21. Proposition (Soundness). *For* Γ *a finite set of formulas of* E-PROPn^τ *and* φ *a formula of* E-PROPn^τ*,*

$$\Gamma \vdash_{\text{E-PROP}n^\tau} \varphi \Rightarrow \Gamma \models \varphi.$$

Proof. Let $(\Theta,\mathcal{C})$ be a model. By induction on the derivation of $\Gamma \vdash \varphi$ we show that for all valuations ρ, $[\![\Gamma]\!]_\rho \leq [\![\varphi]\!]_\rho$. None of the six cases is difficult. We treat the cases for the last rule being ($\supset$ -E) and ($\forall$-I).

($\supset$ -E) Say φ has been derived from $\psi \supset \varphi$ and ψ. Let ρ be valuation. Then by IH $[\![\Gamma]\!]_\rho \leq [\![\psi]\!]_\rho$ and $[\![\Gamma]\!]_\rho \leq [\![\psi \supset \varphi]\!]_\rho$. The second implies $[\![\Gamma]\!]_\rho \wedge [\![\psi]\!]_\rho \leq [\![\varphi]\!]_\rho$. So, by $[\![\Gamma]\!]_\rho \leq [\![\psi]\!]_\rho$ we conclude $[\![\Gamma]\!]_\rho \leq [\![\varphi]\!]_\rho$.

($\forall$-I) Say $\varphi \equiv \forall f \epsilon D.\psi$ and $\Gamma' \subseteq \Gamma$ is the finite set of non-discharged formulas of the derivation with conclusion ψ. Then by IH, $\forall \rho[[\![\Gamma']\!]_\rho \leq [\![\psi]\!]_\rho]$, so $\forall \rho \forall F \in \lceil D \rceil[[\![\Gamma']\!]_\rho \leq [\![\psi]\!]_{\rho(f:=F)}]$, because $f \notin \mathrm{FV}(\Gamma')$. This immediately implies that $[\![\Gamma]\!]_\rho \leq [\![\forall f \;\epsilon\; D.\psi]\!]_\rho$. $\boxtimes$

To show completeness we first construct the Lindenbaum algebra for E-PROPn^τ. This is a Ha but not yet a cHa. The construction in [Troelstra and Van Dalen 1988] tels us how to turn it into a cHa which has all the desired properties.

2.4.22. Definition. For $n \in \mathbb{N} \cup \{\omega\}$, we define the *Lindenbaum algebra for* E-PROPn, $\mathcal{L}_n$. First we define the equivalence relation $\sim$ on Sent(E-PROPn^τ) by

$$\varphi \sim \psi \;:= \vdash_{\text{E-PROP}n^\tau} \varphi \supset \psi \;\&\; \psi \supset \varphi.$$

We denote the equivalence class of φ under $\sim$ by $[\varphi]$. $\mathcal{L}_n$ is now defined as the Ha $(A, \wedge, \vee, \perp, \supset)$ where

$$\begin{aligned} A &= (\mathrm{Sent}(\text{E-PROP}n^\tau))_\sim, \\ [\varphi] \wedge [\psi] &= [\varphi \,\&\, \psi], \\ [\varphi] \vee [\psi] &= [\varphi \vee \psi], \\ [\varphi] \supset [\psi] &= [\varphi \supset \psi], \\ [\perp] &= [\perp]. \end{aligned}$$

Note that the &, $\vee$, $\supset$ and $\perp$ on the right of the = are the logical connectives: $\supset$ is basic and the others were defined in Remark 2.2.7 by

$$\begin{aligned} \varphi \,\&\, \psi &:= \forall \alpha \epsilon \mathsf{Prop}(\varphi \supset \psi \supset \alpha) \supset \alpha, \\ \varphi \vee \psi &:= \forall \alpha \epsilon \mathsf{Prop}(\varphi \supset \alpha) \supset (\psi \supset \alpha) \supset \alpha, \\ \perp &:= \forall \alpha \epsilon \mathsf{Prop} \alpha. \end{aligned}$$

Each $\mathcal{L}_n$ is obviously a Ha: $[\varphi] \leq [\psi]$ iff $\varphi \vdash_{\text{E-PROP}n^\tau} \psi$. Further each $\mathcal{L}_n$ can trivially be turned into a model by taking as valuation of the constants $\mathcal{C}$ the mapping that associates to a constant its equivalence class. We shall not distinguish between the Lindenbaum algebra $\mathcal{L}_n$ and the model $(\mathcal{L}_n, \mathcal{C})$.

2.4.23. Lemma. *For* Γ *a finite set of sentences of* E-PROPn^τ *and* φ *a sentence of* E-PROPn^τ*,*

$$\Gamma \vdash_{\text{E-PROP}n^\tau} \varphi \;\Leftrightarrow\; \Gamma \leq \varphi \textit{ in } \mathcal{L}_n).$$

Proof. Immediate by the construction of $\mathcal{L}_n$. ⊠

2.4.24. Theorem ([Troelstra and Van Dalen 1988]). *Each Ha* Θ *can be embedded into a cHa* $c\Theta$ *such that* $\wedge$, $\vee$, $\perp$, $\supset$ *and existing* $\bigwedge$ *and* $\bigvee$ *are preserved and* $\leq$ *is reflected.*

Proof. Let $\Theta = (A, \wedge, \vee, \perp, \supset)$ be a Ha. A *complete ideal of* Θ, or just *c-ideal*, is a subset $I \subset A$ that satisfies the following properties.

1. $\perp \in I$,
2. I is *downward closed* (i.e. if $b \in I$ and $a \leq b$, then $a \in I$),
3. I is *closed under existing sups* (i.e. if $X \subset I$ and $\bigvee X$ exists, then $\bigvee X \in I$).

Now define $c\Theta$ to be the lattice of c-ideals, ordered by inclusion. Then $c\Theta$ is a complete lattice that satisfies the infinitary distributive law D, and hence $c\Theta$ is a cHa by defining

$$I \supset J := \bigvee \{K \mid K \wedge I \subset J\}.$$

To verify this note the following.

- $c\Theta$ has infs defined by $\bigwedge_{q\in Q} I_q = \bigcap_{q\in Q} I_q$.

- $c\Theta$ has sups defined by $\bigvee_{q\in Q} I_q = \{\bigvee X \mid X \subset \bigcup_{q\in Q} I_q, \bigvee X \text{ exists}\}$: the set $\{\bigvee X \mid X \subset \bigcup_{q\in Q} I_q, \bigvee X \text{ exists}\}$ is indeed a c-ideal and it is also the least c-ideal containing all I_q.

- $I \cap \bigvee_{q\in Q} I_q = \bigvee\{I \cap I_q \mid q \in Q\}$ and so D holds.

The embedding i from Θ to $c\Theta$ is now defined by

$$i(a) = \{x \in A \mid x \leq a\}.$$

The embedding preserves $\perp$, $\supset$ and all existing $\bigwedge$, $\bigvee$. For the preserving of $\bigvee$, let $X \subset A$ such that $\bigvee X$ exists in Θ. We have to show that $i(\bigvee X) = \bigvee_{x\in X} i(x)$, i.e. show that

$$\{y \in A \mid y \leq \bigvee X\} = \{\bigvee Y \mid Y \subset \bigcup_{x\in X} i(x), \bigvee Y \text{exists}\}.$$

For the inclusion from left to right, note that $X \subset \{y \in A \mid \exists x \in X[y \leq x]\}$ and so $X \subset \bigcup_{x\in X} i(x)$. This implies that $\bigvee X \in \{\bigvee Y \mid Y \subset \bigcup_{x\in X} i(x), \bigvee Y \text{exists}\}$ and so we are done because the latter is a c-ideal. For the inclusion from right to left, let $z = \bigvee Y_0$ with $Y_0 \subset \bigcup_{x\in X} i(x)$. Then $z \leq \bigvee X$ so we are done.

Finally, the embedding i reflects the ordering, i.e.

$$i(a) \subset i(b) \Rightarrow a \leq b. \boxtimes$$

2.4.25. Corollary (Completeness). *For Γ a finite set of sentences of* E-PROPn^τ *and φ a sentence of* E-PROPn^τ*,*

$$\Gamma \models \varphi \Rightarrow \Gamma \vdash_{\text{E-PROP}n^\tau} \varphi.$$

Proof. Following the Theorem, we embed the Lindenbaum algebra of E-PROPn^τ, $\mathcal{L}_n$, in the cHa $c\mathcal{L}_n$. This cHa $c\mathcal{L}_n$ is then turned into an algebraic model of E-PROPn^τ by taking as valuation of the constants, $\mathcal{C}$, just the embedding of the equivalence classes of constants in $c\mathcal{L}_n$. This algebraic model $(c\mathcal{L}_n, \mathcal{C})$ is complete with respect to the logic: for Γ a finite set of sentences and φ a sentence of E-PROPn^τ, we have

$$\Gamma \models_{(c\mathcal{L}_n,\mathcal{C})} \varphi \;\Rightarrow\; \Gamma \leq \varphi \text{ in } \mathcal{L}_n \;\Rightarrow\; \Gamma \vdash_{\text{E-PROP}n^\tau} \varphi. \boxtimes$$

2.4.26. Corollary (Conservativity). *For any $n \geq 2$,* E-PROP$(n+1)$ *is conservative over* E-PROPn*, and hence* E-PROPω *is conservative over* E-PROPn*.*

PROOF. By Corollary 2.3.6, it suffices to show the conservativity of E-PROP$(n+1)^\tau$ over E-PROPn^τ. For Γ a finite set of sentences and φ a sentence of E-PROPn^τ,

$$\Gamma \vdash_{\text{E-PROP}(n+1)^\tau} \varphi \Rightarrow \Gamma \models \varphi \Rightarrow \Gamma \vdash_{\text{E-PROP}n^\tau} \varphi$$

by soundness and completeness of the algebraic models for any of the E-PROPn^τ.

The conservativity of E-PROPω over E-PROPn is now immediate: any derivation in E-PROPω is a derivation in E-PROPm for some $m \in \mathbb{N}$. ⊠

2.4.27. COROLLARY. *For any $n \in \mathbb{N} \cup \{\omega\}$,* PROP$n$ *is conservative over* PROP2.

PROOF. By the fact that PROPn is a subsystem of E-PROPn and the fact that PROP2 and E-PROP2 are the same system. ⊠

2.4.3. Kripke semantics for intuitionistic propositional logics

In the previous section we saw an algebraic semantics for the systems E-PROPn^τ (which is at the same time a semantics for the systems E-PROPn). In this paragraph we want to give a Kripke semantics for the systems PROPn, so without extensionality. In fact this was our first starting point for the research into the conservativity of PROP$(n+1)$ over PROPn. However, as it did not seem to work for our purpose, we considered using an algebraic semantics instead. This, as the previous paragraph shows, works only for the extensional case. So, although we do not know how to use the Kripke semantics for solving the conservativity problem, we do want to describe it here, because it gives a complete model notion for the PROPn. For convenience we describe the models as a semantics for PROPn^τ, but we know that there is no problem in that slight extension.

The exposition we give here owes much to [Smorynski 1973], where extensions of Kripke models to higher orders are suggested.

The basis of a Kripke model is a partial order, which is in practice usually a well-founded tree, $<K, \sqsubseteq>$, whose elements are called *nodes*. There is a relation $\Vdash$ between the set of nodes and the set of formulas of the propositional logic, such that certain conditions are satisfied. (Roughly that 'knowledge' grows with the increasing of the order and that $\perp$ is not satisfied at any of the nodes). Now, if one adds first order quantification to the logic, the partial order $<K, \sqsubseteq>$ has to be extended with a function W that assigns to every node k a set $W(k)$ (the 'world' at node k) such that W is monotone. (Our knowledge of the world grows). The case for many-sorted logics is not really different; in that case we have a number of monotone functions W_i, as many as we have sorts in the logic.

For second order propositional logic the situation is not very different from that for first order predicate logic, except that now the domain of quantification is the set of closed formulas, Sent, and so $W : K \to \text{Sent}$. Higher order propositional logic can now just be treated in a 'many-sorted' way: for every domain D in the logic we have a function $W_D : K \to \tilde{D}$, where $\tilde{D}$ is in fact just obtained by replacing Prop by Sent everywhere in D. So we see that the sets over which is quantified in the model are just sets of syntactic objects of the same domain. It is a bit peculiar to let the sets that one quantifies over in the model only be a subset of the set of all syntactic objects of the corresponding domain: shouldn't $W_{\mathsf{Prop}}(k)$ be Sent for all $k \in K$? (All formulas are known to us at any specific node). It turns out that this is the right choice: it conforms with the Kripke semantics for higher order predicate logic and, more importantly, this is the way to get a notion of model that is sound and complete with respect to the logics.

It is obvious that the kind of model that we get by this construction is very syntactical. Moreover it doesn't seem to use the partial order structure of the Kripke model in an essential way. One way to make it a bit less syntactical is by letting the world not be Sent at any point but an arbitrary model of the language of PROPn^τ, that is an arbitrary model of the simply typed lambda calculus. We shall not follow this possibility here because at the one hand it doesn't seem to give us a lot of extras while at the other hand it will be quite obvious from our definitions how to do it.

2.4.28. Definition. To every domain D of PROP$_\omega$ we associate a set of terms, $\tilde{D}$, which is just $\{t | t \in D, t \text{ is closed}\}$.

So, for example $\tilde{\mathsf{Prop}} = \text{Sent}$. The definition is very trivial, but we want to be specific about this, because it is easy to confuse the object language of the logic and the language of the model.

2.4.29. Definition. A Kripke model for PROPn^τ is a triple $<K, \sqsubseteq, \Vdash>$, where $<K, \sqsubseteq>$ is a partial order and $\Vdash$ is a binary relation between elements of K and sentences that satisfies

$$\begin{aligned}
k \Vdash \varphi \mathbin{\&} \varphi =_\beta \psi &\quad\Rightarrow\quad k \Vdash \psi,\\
k \Vdash \varphi \mathbin{\&} l \sqsupseteq k &\quad\Rightarrow\quad l \Vdash \varphi,\\
k \Vdash \varphi \supset \psi &\quad\Leftrightarrow\quad \forall l \sqsupseteq k[l \Vdash \varphi \Rightarrow l \Vdash \psi],\\
k \Vdash \forall x^D \varphi &\quad\Leftrightarrow\quad \forall t \in \tilde{D}[k \Vdash \varphi[t/x]],
\end{aligned}$$

where the l and k range over the nodes (the set K), φ and ψ are formulas and D is a domain over which quantification is allowed in PROPn^τ.

Note that the $\forall l$ and $\forall t$ in the definition are in the meta-language of the model.

2.4.30. Remark. As condition on the relation $\Vdash$ with respect to the $\forall$ connective, one usually finds

$$k \Vdash \forall x^D \varphi \Leftrightarrow \forall l \sqsupseteq k \forall t \in \tilde{D}[l \Vdash \varphi[t/x]],$$

but as the range of quantification in the model does not grow with the increasing of the ordering $\sqsubseteq$, this is equivalent to the second condition in Definition 2.4.29.

In some definitions of Kripke model (like the one in [van Dalen 1983]) the relation $\Vdash$ is between the nodes and the atomic formulas. As the systems we are considering are all impredicative this method does not work here.

To interpret formulas we have to close them by substituting closed terms for the free variables. We denote such a substitution by * and we always assume that for all variables it substitutes a closed term of the right domain.

2.4.31. Definition. Let φ be a formula of PROPn^τ and Γ a set of formulas of PROPn^τ.

1. For $<K, \sqsubseteq, \Vdash>$ a Kripke model for PROPn^τ, we say that φ *is* $<K, \sqsubseteq, \Vdash>$-*valid in* Γ, notation $\Gamma \Vdash_{<K,\sqsubseteq,\Vdash>} \varphi$, if

$$\text{for all substitutions }^*, \forall k \in K[k \Vdash (\Gamma)^* \Rightarrow k \Vdash (\varphi)^*],$$

   where $k \Vdash (\Gamma)^*$ obviously means that $k \Vdash \psi$ for all $\psi \in (\Gamma)^*$.

2. We say that φ *is valid in* Γ, notation $\Gamma \models \varphi$, if

$$\Gamma \Vdash_{<K,\sqsubseteq,\Vdash>} \varphi \text{ for all Kripke models } <K, \sqsubseteq, \Vdash> \text{ of PROP}n^\tau.$$

2.4.32. Proposition (Soundness). *For* Γ *a set of formulas of* PROPn^τ *and* φ *a formula of* PROPn^τ,

$$\Gamma \vdash_{\text{PROP}n^\tau} \varphi \Rightarrow \Gamma \models \varphi.$$

Proof. Let $<K, \sqsubseteq, \Vdash>$ be a Kripke model for PROPn^τ. By induction on the derivation of $\Gamma \vdash_{\text{PROP}n^\tau} \varphi$ we prove

$$\Gamma \vdash_{\text{PROP}n^\tau} \varphi \Rightarrow \Gamma \Vdash_{<K,\sqsubseteq,\Vdash>} \varphi.$$

If the last rule is (conv), or if $\varphi \in \Gamma$, we are immediately done.

($\supset$-I) Say $\varphi \equiv \chi \supset \psi$. Then by IH $\Gamma, \chi \Vdash \psi$, i.e. for all substitutions * we have $\forall k \in K[k \Vdash \Gamma^*, \chi^* \Rightarrow k \Vdash \psi^*]$. Now let * be a substitution and let $l \in K$ with $l \Vdash \Gamma^*$ and $m \sqsupseteq l$ with $m \Vdash \chi^*$. Then $m \Vdash \Gamma^*, \chi^*$ and hence by IH $m \Vdash \psi^*$, so we are done.

($\supset$-E) Say φ has been derived from $\psi \supset \varphi$ and ψ, so we have as IH $\Gamma \Vdash \psi \supset \varphi$ and $\Gamma \Vdash \psi$. Now let * be a substitution and let $k \in K$ with $k \Vdash \Gamma^*$. Then by IH $k \Vdash \psi^*$ and $\forall l \sqsupseteq k[l \Vdash \psi^* \Rightarrow l \Vdash \varphi^*]$. Because $k \sqsupseteq k$ we find that $k \Vdash \varphi^*$ and we are done.

($\forall$-I) Say $\varphi \equiv \forall x \epsilon D.\psi$, so we have as IH $\Gamma \Vdash \psi$. That is, for all substitutions * we have $\forall k \in K[k \Vdash \Gamma^* \Rightarrow k \Vdash \psi^*]$. Now x^D does not occur free in Γ, so we know that for all substitutions * and all $t \in \tilde{D}$, $\forall k \in K[k \Vdash \Gamma^* \Rightarrow k \Vdash (\psi[t/x])^*]$. Hence $\Gamma \Vdash \forall x \epsilon D.\psi$.

($\forall$-E) Say $\varphi \equiv \psi[P/x]$, which has been derived from $\forall x \epsilon D.\psi$. Then by IH $\Gamma \Vdash \forall x \epsilon D.\psi$. Now let * be a substitution and $k \in K$ such that $k \Vdash \Gamma^*$. Then for all $t \in \tilde{D}$ $k \Vdash (\psi[t/x])^*$ and hence $k \Vdash (\psi[P^*/x])^*$, i.e. $k \Vdash (\psi[P/x])^*$. $\boxtimes$

2.4.33. Proposition (Completeness). *For* Γ *a set of sentences of* PROPn^τ *and* φ *a sentence of* PROPn^τ*,*

$$\Gamma \models \varphi \Rightarrow \Gamma \vdash_{\mathrm{PROP}n^\tau} \varphi.$$

Proof. The proof is by contraposition, so we suppose $\Gamma \nvdash_{\mathrm{PROP}n^\tau} \varphi$ and construct a Kripke model $<K, \sqsubseteq, \Vdash>$ of PROPn^τ in which $\Gamma \nVdash \varphi$. (Our construction of the counter-model is a direct generalisation of the standard construction of a counter-model for showing completeness of Kripke models with respect to first order intuitionistic predicate logic, as it is given for example in [van Dalen 1983]). Before giving the model we introduce one extra notion: for Δ a set of sentences, we write $\overline{\Delta}$ for the closure of Δ under derivability in PROPn^τ. Now the model is defined as follows.

- $K := \mathbb{N}^*$, the set of finite sequences of natural numbers,
- $\vec{p} \sqsubseteq \vec{m} := \exists \vec{a}[\vec{p} \star \vec{a} = \vec{m}]$, where $\star$ is the concatenation operation,
- For every $\vec{m} \in \mathbb{N}^*$ we define a set of sentences of PROPn^τ, $\Sigma(\vec{m})$, by induction on the length of $\vec{m}$, as follows.
  - $\Sigma(<>) = \overline{\Gamma}$,
  - For $\Sigma(\vec{m})$ defined, consider an enumeration of sentences $\varphi_0, \varphi_1, \ldots$ such that $\Sigma(\vec{m}) \cup \{\varphi_i\}$ is consistent for all i. Now define

$$\Sigma(\vec{m} \star i) := \overline{\Sigma(\vec{m}) \cup \{\varphi_i\}}.$$

  The relation $\Vdash$ is now defined by

$$\vec{m} \Vdash \psi := \psi \in \Sigma(\vec{m}) (\Leftrightarrow \Sigma(\vec{m}) \vdash \psi).$$

We now only have to verify the following two facts

1. $<\mathbb{N}^*, \sqsubseteq, \Vdash>$ is a Kripke model of PROPn^τ,
2. In the model we have $<> \Vdash \Gamma$, $<> \nVdash \varphi$ and hence $\Gamma \nVdash \varphi$.

The second follows immediately from the construction of the model. The first is slightly more work: we have to check the four cases of Definition 2.4.29. The first two cases are trivial; we give detailed proofs of the third and fourth case.

- $\vec{m} \Vdash \varphi \supset \psi \Leftrightarrow \forall \vec{p} \sqsupseteq \vec{m}[\vec{p} \Vdash \varphi \Rightarrow \vec{p} \Vdash \psi]$: for $(\Rightarrow)$, let $\vec{p} \sqsupseteq \vec{m}$, $\vec{p} \Vdash \varphi$. Then $\Sigma(\vec{p}) \vdash \varphi$ and $\Sigma(\vec{p}) \vdash \varphi \supset \psi$, so $\Sigma(\vec{p}) \vdash \psi$, so $\vec{p} \Vdash \psi$. For $(\Leftarrow)$, let $\vec{m}$ be a finite sequence. From the assumption we know that $\forall \vec{p} \sqsupseteq \vec{m}.\Sigma(\vec{p}) \vdash \varphi \Rightarrow \Sigma(\vec{p}) \vdash \psi$. We distinguish two cases according to whether $\Sigma(\vec{m}) \cup \{\varphi\}$ is consistent or not. If $\Sigma(\vec{m}) \cup \{\varphi\}$ is inconsistent, then trivially $\Sigma(\vec{m}) \cup \{\varphi\} \vdash \psi$ and so $\Sigma(\vec{m}) \vdash \varphi \supset \psi$ and hence $\vec{m} \Vdash \varphi \supset \psi$. If $\Sigma(\vec{m}) \cup \{\varphi\}$ is consistent, then $\overline{\Sigma(\vec{m}) \cup \{\varphi\}} = \Sigma(\vec{m} * i)$ for some i, and hence $\Sigma(\vec{m} * i) \vdash \psi$ by the assumption. But then $\Sigma(\vec{m} * i) \cup \{\varphi\} \vdash \psi$, so $\Sigma(\vec{m}) \vdash \varphi \supset \psi$ and hence $\vec{m} \Vdash \varphi \supset \psi$.

- $\vec{m} \Vdash \forall x \epsilon D \varphi \Leftrightarrow \forall t \in \tilde{D}[\vec{m} \Vdash \varphi[t/x]]$: for $\Rightarrow$, let $t \in \tilde{D}$. Now $\Sigma(\vec{m}) \vdash \forall x \epsilon D \varphi$ and hence $\Sigma(\vec{m}) \vdash \varphi[t/x]$. For $\Leftarrow$, from the assumption we know that $\vec{m} \Vdash \varphi[c/x]$ for all constants c, i.e. $\Sigma(\vec{m}) \vdash \varphi[c/x]$ for all constants and so $\Sigma(\vec{m}) \vdash \forall x \epsilon D.\varphi$. ⊠

Technically, the reason that we can not get conservativity from this model notion is that a model of PROPn^τ is in general not a model of PROP$(n+1)^\tau$. In less technical terms the reason seems to be that the model notion is too syntactical, especially in the clause for the universal quantifier, where the ordering $\sqsubseteq$ doesn't play any role at all.

Chapter 3

The formulas-as-types embedding

3.1. Introduction

The so called formulas-as-types embedding provides a formalization of the Brouwer-Heyting-Kolmogorov interpretation of proofs as constructions. The first detailed description is in [Howard 1980], where also the terminology 'formulas-as-types' is first used. There it is shown how, in first order logic, types can be associated with formulas and lambda terms with proofs in such a way that there is a one-to-one correspondence between types and formulas and terms and proofs and further that cut-elimination in the logic corresponds to reduction in the term calculus. In view of the last point it would be correct also to associate Tait with the formulas-as-types notion, as his [Tait 1965] 'discovery of the close correspondence between cut-elimination and reduction of lambda terms provided half of the motivation' for [Howard 1980]. Also de Bruijn is often associated to the formulas-as-types notion, because the Automath project which was founded by de Bruijn, was the first to rigorously interpret mathematical structures and propositions as types and objects and proofs as lambda terms. So, from a wider perspective it is certainly justifiable to speak of the Curry-Howard-de Bruijn embedding (also because the earliest developments in Automath took place independent of the work of Howard). Having said this we want to point out that there are essential differences between the two approaches. For example, in the Automath systems the logic is *coded* into the system, so there is in general no reduction relation in the term calculus that corresponds to cut-elimination. Automath systems are intended to serve as a logical framework in which the user can work with any formal systems he or she desires. Application, λ-abstraction and conversion serve as tools for handling the basic mathematical manipulations like function application, function definition and substitution. It is appropriate to remark here that some later systems of the Automath family *do* use the abstraction-application features of the system to interpret logical connectives directly (and hence reduction corresponds to cut-elimination). Later in this section we shall give some examples of Automath systems to clarify these remarks.

We do not go into great detail about the Brouwer-Heyting-Kolmogorov (BHK) interpretation of proofs. [Troelstra and Van Dalen 1988] is a good reference and gives a thorough explanation of the idea. Let's just discuss the connectives $\supset$ and $\forall$ according to the BHK interpretation.

1. A proof of $\varphi \supset \psi$ is a method for constructing a proof of ψ from a proof of φ.

2. A proof of $\forall x \in A.\varphi(x)$ is a method for constructing a proof of $\varphi(a)$ from a proof of $a \in A$.

It is obvious (in retrospect) that the lambda calculus provides the necessary mechanisms for turning the informal interpretations into a formal system. For minimal propositional logic this was already noticed by [Curry and Feys 1958]. For first order predicate logic, [Howard 1980] was the first to give a formalisation of the BHK interpretation using typed lambda calculus. Due to the work of [Church 1940] it was already known that also the language of predicate logic can be presented as a typed lambda calculus. Over the years this has led to the definition of various typed lambda calculi that incorporate the logical language and proofs (in the form of lambda terms) in one system. In this thesis we shall see a variety of those systems.

We do not claim to give an overview of all the possible approaches to the formulas-as-types embedding. In fact we do not even attempt to do this. For example one of the main contributions to the field, the work in Martin-Löf's type theory, will not be treated at all. One of the reasons is that a PhD thesis is not the place to give a detailed technical overview of such a broad field as Type Theory, but another important reason is that the approach of Martin-Löf does not really fit with the framework of logics as we have set it up in the previous chapter. One of the main problems is that, due to the understanding of the existential quantifier in terms of a strong Σ-type, the logic of Martin-Löf is strictly first order (in order to remain consistent). We do not feel that the forced lack of Σ-types in our higher order logics is a big gap, but that is because we feel that the strong Σ-type is not the right way to formalise the intuitionistic existential quantifier. (To be precise: we do not mean to say that Σ-types are not a valid mathematical concept, but only that Σ should not be understood as $\exists$).

Of course there is also a lot to say about systems that we *do* treat and we shall do so at the appropriate places in the text.

3.2. The formulas-as-types notion à la Howard

In this paragraph we look at an interpretation of formulas as types and proofs as terms in the flavour of [Howard 1980], where the interpretation is given for full first order predicate logic. Although in flavour the same, our treatment is quite a bit different from Howard's, as has already been pointed out in the previous chapter. As we are mainly concerned with logics that only use $\supset$ and $\forall$ we shall not treat the full first order predicate logic here but restrict to the system PRED. First order logic based on just $\supset$ and $\forall$ is quite minimal, but it is sufficient to make the general idea sufficiently clear. In our formalisation the logical language will also be presented in a typed lambda calculus manner. This idea of an 'all-in-one presentation' is probably due to de Bruijn and his Automath project, although we are not absolutely sure.

3.2.1. Definition. 1. The set of *functional types* of ΛPRED, Type^f, is described by the following abstract syntax.

$$\mathsf{Type}^f ::= \mathsf{Var}^{ty} \mid \mathsf{Type}^f {\rightarrow} \mathsf{Type}^f,$$

with Var^{ty} a countable set of *type-variables*. The set of *predicate types* of ΛPRED, Type^p, consists of the expressions

$$\sigma_1{\rightarrow}\sigma_2{\rightarrow}\cdots{\rightarrow}\sigma_n{\rightarrow}\mathsf{Prop},$$

with $n \geq 0$ and all σ_i functional types.

2. The *object-terms of the language of* ΛPRED form a subset of the set of *pseudoterms*, T, which is generated by the following abstract syntax.

$$\mathsf{T} ::= \mathsf{Var}^{ob} \mid \mathsf{T}\mathsf{T} \mid \lambda x{:}\mathsf{Type}^f.\mathsf{T} \mid \mathsf{T} \supset \mathsf{T} \mid \forall \mathsf{Var}^{ob}{:}\mathsf{Type}^f.\mathsf{T},$$

with Var^{ob} a countable set of *object-variables*. An object-term is of a certain type only under assumption of specific types (functional or predicative) for the free variables that occur in the term. That the object-term t *is of type* A *if* x_i *is of type* A_i *for* $1 \leq i \leq n$, is denoted by

$$x_1{:}A_1, x_2{:}A_2, \ldots, x_n{:}A_n \vdash t : A.$$

Here $x_1, \ldots, x_n$ are different object-variables and $A_1, \ldots, A_n$ are types. The part $x_1{:}A_1, x_2{:}A_2, \ldots, x_n{:}A_n$ is called an *object-context*. The rules for deriving these typing judgements are the following.

$$(\text{var})\quad \frac{}{\Gamma \vdash x : A} \qquad \text{if } x{:}A \text{ in } \Gamma$$

$$(\lambda\text{-abs})\quad \frac{\Gamma, x{:}A \vdash t : B}{\Gamma \vdash \lambda x{:}A.t : A{\to}B} \qquad \text{if } A \text{ functional}$$

$$(\text{app})\quad \frac{\Gamma \vdash q : A{\to}B \quad \Gamma \vdash t : A}{\Gamma \vdash qt : B}$$

$$(\supset)\quad \frac{\Gamma \vdash \varphi : \mathsf{Prop} \quad \Gamma \vdash \psi : \mathsf{Prop}}{\Gamma \vdash \varphi \supset \psi : \mathsf{Prop}}$$

$$(\forall)\quad \frac{\Gamma, x{:}A \vdash \varphi : \mathsf{Prop}}{\Gamma \vdash \forall x{:}A.\varphi : \mathsf{Prop}} \qquad \text{if } A \text{ a functional type}$$

3. The set of *proof-terms* is a subset of the set of *pseudoproofs*, P, generated by the following abstract syntax.

$$\mathsf{P} ::= \mathsf{Var}^{pr} \mid \mathsf{PP} \mid \mathsf{PT} \mid \lambda x{:}\mathsf{Type}^{f}.\mathsf{P} \mid \lambda x{:}\mathsf{T}.\mathsf{P},$$

where Var^{pr} is the set of proof-variables. The rules for generating statements of the form

$$x_1{:}A_1, \ldots, x_n{:}A_n; p_1{:}\varphi_1, \ldots, p_k{:}\varphi_k \vdash M : A,$$

where the $\vec{x}{:}\vec{A}$ is as in 2, $p_1, \ldots p_k$ are different proof-variables and

$$x_1{:}A_1, \ldots, x_n{:}A_n \vdash \varphi_i : \mathsf{Prop} \text{ for } 1 \leq i \leq k,$$

are the following. (The part $p_1{:}\varphi_1, \ldots, p_k{:}\varphi_k$ is called the *proof-context* as opposed to the object-context.)

$$\text{(axiom)} \quad \frac{}{\Gamma; \Delta \vdash p : \varphi} \qquad \text{if } p{:}\varphi \text{ in } \Delta,$$

$$(\supset\text{-in}) \quad \frac{\Gamma; \Delta, p{:}\varphi \vdash M : \psi}{\Gamma; \Delta \vdash \lambda p{:}\varphi.M : \varphi \supset \psi}$$

$$(\supset\text{-el}) \quad \frac{\Gamma; \Delta \vdash M : \varphi \supset \psi \quad \Gamma; \Delta \vdash N : \varphi}{\Gamma; \Delta \vdash MN : \psi}$$

$$(\forall\text{-in}) \quad \frac{\Gamma, x{:}A; \Delta \vdash M : \varphi}{\Gamma; \Delta \vdash \lambda x{:}A.M : \forall x{:}A.\varphi} \qquad \text{if } x \notin \mathrm{FV}(\Delta), A \text{ a functional type,}$$

$$(\forall\text{-el}) \quad \frac{\Gamma; \Delta \vdash M : \forall x{:}A.\varphi \quad \Gamma \vdash t : A}{\Gamma; \Delta \vdash Mt : \varphi[t/x]}$$

$$\text{(conv)} \quad \frac{\Gamma; \Delta \vdash M : \varphi \quad \Gamma \vdash \psi : \mathsf{Prop}}{\Gamma; \Delta \vdash M : \psi} \qquad \text{if } \varphi =_\beta \psi.$$

The intention of the system should be clear: natural deduction proofs of PRED are interpreted as typed lambda terms in ΛPRED. The language of PRED is also a typed lambda calculus and also that part is formalised in ΛPRED in a typing judgement that is obtained via derivations. Note that the functional types correspond to domains of order 1 (the ones over which quantification is allowed) and the predicative types correspond to domains of order 2. Before describing a formal correspondence between derivations in PRED and proof-terms in ΛPRED, we give two examples.

3.2.2. Examples. 1. From the deduction

$$\frac{\dfrac{\forall x{:}A.(Px \supset Q)}{Px \supset Q} \qquad \dfrac{\forall x{:}A.Px}{Px}}{Q}$$

we obtain the judgement

$$\begin{array}{c} P{:}A{\to}\mathsf{Prop}, Q{:}\mathsf{Prop}, x{:}A; \\ p_1{:}\forall x{:}A.(Px \supset Q), p_2{:}\forall x{:}A.Px \;\;\vdash\;\; p_1x(p_2x) : Q. \end{array}$$

Notice that the declaration of x is essential here for the construction of the proof. (ΛPRED explicitly takes care of the so called *free logic*, where domains are allowed to be empty.)

2. From the deduction

$$\dfrac{\dfrac{\forall x{:}A.(Px \supset Q)}{Px \supset Q} \qquad \dfrac{\forall x{:}A.Px}{Px}}{\dfrac{Q}{\forall x{:}A.Qx}}$$

   we obtain the judgement

$$\begin{array}{c} P{:}A{\to}\mathsf{Prop}, Q{:}\mathsf{Prop};\\ p_1{:}\forall x{:}A.(Px \supset Q), p_2{:}\forall x{:}A.Px \quad \vdash \quad \lambda x{:}A.p_1 x(p_2 x) : \forall x{:}A.Q. \end{array}$$

   Now it is not needed for the construction of the proof to declare x.

We list some of the meta-theoretical properties of λPRED that we shall be using later. They are given without proof: later we shall encounter other (more complicated) typed lambda calculi for which these properties also hold and we prefer to prove them once for all systems together.

3.2.3. Fact. Let $\Gamma; \Delta \vdash M : \varphi$ be derivable in λPRED. We have the following properties.

1. Permutation: if Γ' is a permutation of Γ and Δ' a permutation of Δ, then $\Gamma'; \Delta' \vdash M : \varphi$ is also derivable.

2. Substitution: if Γ contains $x{:}A$ and $\Gamma\backslash(x{:}A) \vdash t : A$ then $\Gamma\backslash(x{:}A); \Delta[t/x] \vdash M[t/x] : \varphi[t/x]$ is also derivable.

3. Thinning: if $\Gamma' \supseteq \Gamma$, Γ' an object-context and $\Delta' \supseteq \Delta$, Δ' a proof-context, then $\Gamma'; \Delta' \vdash M : \varphi$ is also derivable.

4. Closure or Subject-Reduction:iIf $M \twoheadrightarrow_\beta M'$, then $\Gamma; \Delta \vdash M' : \varphi$ is also derivable.

5. Stripping or Generation:

$$\begin{array}{rcl}
\Gamma; \Delta \vdash p : \varphi & \Rightarrow & \varphi = \psi \text{ with } p : \psi \in \Delta \\
(p \text{ a proof variable}) & & \quad\text{for some } \psi, \\
\Gamma; \Delta \vdash \lambda x{:}A.M : \varphi & \Rightarrow & \Gamma, x{:}A; \Delta \vdash M : \psi \text{ with } \varphi = \forall x{:}A.\psi \\
(A \text{ a type}) & & \quad\text{for some } \psi, \\
\Gamma; \Delta \vdash \lambda p{:}\chi.M : \varphi & \Rightarrow & \Gamma; \Delta, p{:}\chi \vdash M : \psi \text{ with } \varphi = \chi \supset \psi \\
(\chi \text{ a proposition}) & & \quad\text{for some } \psi, \\
\Gamma; \Delta \vdash MN : \varphi & \Rightarrow & \Gamma; \Delta \vdash M : \psi \supset \chi, \Gamma; \Delta \vdash N : \psi \text{ with } \varphi = \chi \\
(N \text{ a proof}) & & \quad\text{for some } \psi, \chi, \\
\Gamma; \Delta \vdash Mt : \varphi & \Rightarrow & \Gamma; \Delta \vdash M : \forall x{:}A.\psi, \Gamma \vdash t : A \text{ with } \varphi = \psi[t/x] \\
(t \text{ an object}) & & \quad\text{for some } \psi, A.
\end{array}$$

To a deduction of

$$\varphi_1, \ldots, \varphi_n \vdash \psi$$

in PRED, we are going to associate an object-context Γ and a proof-term M such that

$$\Gamma; p_1{:}\varphi_1, \ldots, p_n{:}\varphi_n \vdash M : \psi$$

in ΛPRED. We want M to be a faithful representaton of the deduction in PRED such that there is a one-to-one correspondence between deductions (in PRED) and proof-terms (in ΛPRED). To achieve this, Γ should assign types to all the free term-variables in the deduction that are not bound by a $\forall$ at any later stage. (What it means for variables to be 'bound by a $\forall$' will be explained later). From the examples it will be clear that sometimes we have to declare a variable x, even though this variable does not occur free in the conclusion or in any of the premises of the derivation. Before giving the translation we have to define two operations on contexts that will be used.

3.2.4. DEFINITION. For Γ_1 and Γ_2 object-context, the *union of* Γ_1 *and* Γ_2 , $\Gamma_1 \cup \Gamma_2$, is Γ_1 followed by Γ_2, with the restriction that if x is declared in both contexts, say $x : A \in \Gamma_1$ and $x : B \in \Gamma_2$, then

$$\begin{array}{lll} A \equiv B & \Rightarrow & x : B \text{ is left out, (so we leave only } x : A \\ A \not\equiv B & \Rightarrow & \text{both } x : A \text{ and } x : B \text{ are left out.} \end{array}$$

For Δ_1 and Δ_2 proof-context, the *disjoint union of* Δ_1 *and* Δ_2 , $\Delta_1 \uplus \Delta_2$, is Δ_1 followed by Δ_2, with the restriction that if p is declared in both contexts, say $p : \varphi \in \Delta_1$ and $p : \psi \in \Delta_2$, then the second p is renamed with a fresh proof-variable q. So, for example

$$(p : \varphi) \uplus (p : \varphi) = p : \varphi, q : \varphi.$$

Note that $\Gamma_1 \cup \Gamma_2$ is always a correct object-context and that, if Δ_1 and Δ_2 are corrects proof-contexts w.r.t. Γ, then $\Delta_1 \uplus \Delta_2$ is a correct proof-context w.r.t. Γ.

3.2.5. DEFINITION. For every term t of the language of PRED we define a context Γ_t such that $\Gamma_t \vdash t : D$ (in ΛPRED) if $t \in D$ (in PRED), as follows.

$$\begin{array}{lllllll} t & \equiv & x^D & \Gamma_t & := & x_D{:}D, \\ t & \equiv & \lambda x \in D.M & \Gamma_t & := & \Gamma_M \setminus (x{:}D), \\ t & \equiv & MN & \Gamma_t & := & \Gamma_M \cup \Gamma_N, \\ t & \equiv & \varphi \supset \psi & \Gamma_t & := & \Gamma_\varphi \cup \Gamma_\psi, \\ t & \equiv & \forall x \in D.\varphi & \Gamma_t & := & \Gamma_\varphi \setminus (x{:}D). \end{array}$$

We now define, by induction, for every deduction in PRED an object-context Γ, a proof-context Δ and a term M such that

$$\Gamma; \Delta \vdash M : \varphi \text{ (in } \lambda\text{PRED)},$$

if φ is the conclusion of the deduction. In fact this establishes a mapping from deductions to λ terms. In the following we shall denote deductions by the capital Greek characters Σ and Θ. To denote explicitly that φ is a conclusion of the deduction φ we shall often use the format

$$\begin{array}{c}\Sigma\\ \varphi\end{array}$$

(So when we write this down we mean that φ is part of the deduction Σ.) For reasons of hygiene we shall assume that in a deduction all bound variables are chosen distinct and different from the free ones.

3.2.6. Definition. We inductively define the mapping $(\![-]\!)$ from deductions of PRED to proof-terms of λPRED. Together with the proof-term we define an object-context and a proof-context in which the proof-term is typable. The double horizontal lines on the right mean that the judgement below is being defined in terms of the judgement above.

$$\psi \quad\longmapsto\quad \Gamma_\psi; p{:}\psi \vdash p : \psi,$$

$$\dfrac{\begin{array}{c}[\varphi]^i\\ \Sigma\\ \psi\end{array}}{\varphi \supset \psi}\,i \quad\longmapsto\quad \dfrac{\Gamma; \Delta \vdash (\![\Sigma]\!) : \psi}{\Gamma \cup \Gamma_\varphi; \Delta \setminus (\vec{p}{:}\varphi) \vdash \lambda p{:}\varphi.(\![\Sigma]\!) : \varphi \supset \psi}$$

$$\dfrac{\begin{array}{cc}\Sigma & \Theta\\ \varphi \supset \psi & \varphi\end{array}}{\psi} \quad\longmapsto\quad \dfrac{\Gamma_1; \Delta_1 \vdash (\![\Sigma]\!) : \varphi \supset \psi \quad \Gamma_2; \Delta_2 \vdash (\![\Theta]\!) : \varphi}{\Gamma_1 \cup \Gamma_2; \Delta_1 \uplus \Delta_2 \vdash (\![\Sigma]\!)(\![\Theta]\!) : \psi}$$

$$\dfrac{\begin{array}{c}\Sigma\\ \psi\end{array}}{\forall x \,\epsilon\, D.\psi} \quad\longmapsto\quad \dfrac{\Gamma; \Delta \vdash (\![\Sigma]\!) : \psi}{\Gamma \setminus (x{:}D); \Delta \vdash \lambda x{:}D.(\![\Sigma]\!) : \forall x{:}D.\psi}$$

$$\dfrac{\begin{array}{c}\Sigma\\ \forall x \,\epsilon\, D.\psi\end{array}}{\psi[t/x]} \quad\longmapsto\quad \dfrac{\Gamma_1; \Delta \vdash (\![\Sigma]\!) : \forall x{:}D.\psi \quad \Gamma_2 \vdash t : D}{\Gamma_1 \cup \Gamma_2; \Delta \vdash (\![\Sigma]\!)t : \psi[t/x]}$$

$$\dfrac{\begin{array}{c}\Sigma\\ \psi\end{array}}{\varphi}\ \text{if } \varphi = \psi \quad\longmapsto\quad \dfrac{\Gamma; \Delta \vdash (\![\Sigma]\!) : \psi \quad \Gamma_\varphi \vdash \varphi : \mathsf{Prop}}{\Gamma \cup \Gamma_\varphi; \Delta \vdash (\![\Sigma]\!) : \varphi}$$

The cases in the definition for the last rule being ($\supset$-I) and ($\supset$-E) need some extra clarification:

($\supset$-I) The $[\varphi]^i$ on top of the deduction signifies a specific set of occurrences of the formula φ as leaves of the deduction tree. As this set may also be empty we have to take the union of Γ with Γ_φ. What happens at an ($\supset$-I) rule is the following:

1. Add a fresh declaration $p{:}\varphi$ to Δ.
2. Remove the declarations $p'{:}\varphi$ that correspond to an occurrence of φ that is being discharged.
3. Substitute p for p' in $([\Sigma])$.
4. Abstract from the last declaration in Δ (which is $p{:}\varphi$).

($\supset$-E) In fact the $([\Theta])$ is not exactly the $([\Theta])$ that is found by induction. Possibly some of the free variables in $([\Theta])$ are renamed. What happens is the following:

1. Consider the proof-context $\Delta_1 \uplus \Delta_2$ and especially the renaming of the declared variables in Δ_2 that has been caused by the operation $\uplus$.
2. Rename the free proof-variables in $([\Theta])$ accordingly, obtaining say, $([\Theta'])$.
3. Apply $([\Sigma])$ to $([\Theta'])$.

(There will in practice be no confusion if we just write $([\Theta])$ instead.)

Of course the intended meaning is that the judgement below the double lines is derivable if the judgement above the lines is. This will be proved later in Theorem 3.2.8. It should be clear at this point however that there is a one-to-one correspondence between the occurrences of φ as a (non-discharged) premise in the deduction and declarations $p{:}\varphi$ in Δ.

NOTATION. If for Σ a deduction in PRED, $\Gamma; \Delta \vdash ([\Sigma]) : \varphi$ is the judgement that we obtain from Σ by Definition 3.2.6 above, we write Γ_Σ for Γ and Δ_Σ for Δ.

Let us state the following trivial facts about the definition.

3.2.7. FACT. 1. For Σ a deduction in PRED there is a one-to-one correspondence between occurrences of non-discharged formulas of Σ and declarations of variables to the same formula in Δ_Σ.

2. In the case for the ($\forall$-I) rule the variable x does not occur free in the proof-context Δ.

3.2.8. THEOREM.

$$\begin{array}{c}\Sigma\\ \varphi\end{array} \quad \textit{in } \mathrm{PRED} \quad \Rightarrow \quad \Gamma_\Sigma ; \Delta_\Sigma \vdash (\![\Sigma]\!) : \varphi \quad \textit{is derivable in } \lambda\mathrm{PRED}$$

PROOF. By induction on the deduction Σ. The proof follows easily by using the meta-theoretical facts of λPRED that were stated in 3.2.3. ⊠

The proof-context Δ_Σ represents precisely the non-discharged assumptions of Σ. The object-context Γ_Σ declares precisely those object-variables that occur in Σ and are not bound later by a $\forall$.

Due to the conversion rule, the context Γ_Σ is not *minimal* with respect to the judgement

$$\Gamma_\Sigma ; \Delta_\Sigma \vdash (\![\Sigma]\!) : \varphi$$

in the sense that there may be a smaller object-context Γ for which

$$\Gamma ; \Delta_\Sigma \vdash (\![\Sigma]\!) : \varphi$$

is derivable. (A proof of the statement 'all declared variables in Γ_Σ occur free in Δ_Σ or $(\![\Sigma]\!)$' breaks down on the conversion rule.) A counterexample to minimality of Γ_Σ is given by the derivation

$$\cfrac{\cfrac{\cfrac{[\varphi]}{\varphi \supset \varphi}}{(\lambda x{:}A.\varphi \supset \varphi)y}}{\varphi \supset \varphi}$$

We have $\Gamma_\Sigma = \varphi{:}\mathsf{Prop}, y{:}A$, $\Delta_\Sigma = \emptyset$, $(\![\Sigma]\!) = \lambda p{:}\varphi.p$, whereas

$$\varphi{:}\mathsf{Prop}; \vdash \lambda p{:}\varphi.p : \varphi \supset \varphi.$$

The conversion rule is also the reason that the embedding $(\![\,]\!)$ is not really one-to-one. The λ-term $(\![\Sigma]\!)$ that we obtain ignores all applications of (conv) in the deduction Σ and, as is easily seen, applications of (conv) can be moved through the tree Σ more or less freely. There is however a one-to-one correspondence between equivalence classes of deductions and λ-terms if we let two deductions be equivalent if one obtains the same tree after removing all applications of (conv). We shall define this equivalence relation more precisely later.

3.2.1. Completeness of the embedding

We now define a mapping back from the proof-terms of λPRED to deductions of PRED.

3.2.9. DEFINITION. For any proof-terms M with $\Gamma;\Delta \vdash M : \varphi$ we define by induction on the structure of M a deduction $\Sigma(M)$ as follows.

$$\Gamma;\Delta \vdash p : \varphi \;\longmapsto\; \frac{\psi}{\varphi} \text{ if } p : \psi \in \Delta$$

$$\Gamma;\Delta \vdash \lambda p{:}\psi.N : \varphi \;\longmapsto\; \cfrac{\cfrac{\begin{array}{c}[\psi]^i\\ \Sigma(N)\\ \chi\end{array}}{\psi \supset \chi}\,i}{\varphi}$$

$$\Gamma;\Delta \vdash \lambda x{:}A.N : \varphi \;\longmapsto\; \cfrac{\cfrac{\begin{array}{c}\Sigma(N)\\ \psi\end{array}}{\forall x{:}D.\psi}}{\varphi}$$

$$\Gamma;\Delta \vdash MN : \varphi \;\longmapsto\; \cfrac{\cfrac{\begin{array}{c}\Sigma(M)\\ \chi \supset \psi\end{array} \qquad \begin{array}{c}\Sigma(N)\\ \chi\end{array}}{\psi}}{\varphi}$$

$$\Gamma;\Delta \vdash Nt : \varphi \;\longmapsto\; \cfrac{\cfrac{\begin{array}{c}\Sigma(N)\\ \forall x \in D.\psi\end{array}}{\psi[t/x]}}{\varphi}$$

For every case, the final rule is always an application of (conv). This can be vacuous if the conclusion that was obtained is already φ.

The Definition is justified by Stripping, which says that the proposition φ is always equal to a proposition of the form we require.

3.2.10. PROPOSITION. *If* $\Gamma;\Delta \vdash M : \varphi$ *in* λPRED, *then*

1. *the conclusion of* $\Sigma(M)$ *is* φ *and all non-discharged assumptions of* $\Sigma(M)$ *are declared in* Δ,
2. $(\![\Sigma(M)]\!) \equiv M$ *and* $\Gamma_{\Sigma(M)} \subseteq \Gamma$, $\Delta_{\Sigma(M)} \subseteq \Delta$.

PROOF. By induction on the structure of M.$\boxtimes$

To be more precise as to what extent there is a bijective correspondence between deductions in PRED and proof-terms in ΛPRED, we define an equivalence relation on deductions of PRED.

3.2.11. Definition. We define a stripping operation $\langle - \rangle$ from deduction trees to labelled finite trees as follows.

$$\varphi \longmapsto \varphi,$$

$$\frac{\begin{matrix}[\varphi]^i\\ \Sigma\\ \psi\end{matrix}}{\varphi \supset \psi}\,{}^i \longmapsto \frac{\langle\Sigma\rangle}{i}$$

$$\frac{\begin{matrix}\Sigma & \Theta\\ \varphi \supset \psi & \varphi\end{matrix}}{\psi} \longmapsto \frac{\langle\Sigma\rangle \quad \langle\Theta\rangle}{\mathrm{MP}}$$

$$\frac{\begin{matrix}\Sigma\\ \psi\end{matrix}}{\forall x \,\epsilon\, D.\psi} \longmapsto \frac{\langle\Sigma\rangle}{x \,\epsilon\, D}$$

$$\frac{\begin{matrix}\Sigma\\ \forall x \,\epsilon\, D.\psi\end{matrix}}{\psi[t/x]} \longmapsto \frac{\langle\Sigma\rangle}{t}$$

$$\frac{\begin{matrix}\Sigma\\ \psi\end{matrix}}{\varphi}\ \text{if } \varphi = \psi \longmapsto \langle\Sigma\rangle$$

Remember that, when writing φ below Σ, we mean that φ is a part of the deduction Σ. So, the mapping $\langle\rangle$ removes all formulas from the tree Σ, except for the leaves. In doing so it leaves just enough information behind to reconstruct which rule has been applied and in which form (like which occurrences of a formula have been discharged, which variable has been abstracted from and which term has been substituted).

3.2.12. Example.

$$\cfrac{\cfrac{\cfrac{\varphi \supset \varphi \supset \psi \quad [\varphi]^i}{\varphi \supset \psi} \qquad [\varphi]^i}{\psi}}{\varphi \supset \psi}\,{}^i \longmapsto \cfrac{\cfrac{\cfrac{\varphi \supset \varphi \supset \psi \quad [\varphi]^i}{\mathrm{MP}} \qquad [\varphi]^i}{\mathrm{MP}}}{i}$$

3.2.13. Definition. The equivalence relation $\sim_{CH}$ on deductions of PRED is defined by

$$\Sigma \sim_{CH} \Theta := \langle\Sigma\rangle = \langle\Theta\rangle.$$

The $\sim_{CH}$ equivalence classes will be denoted by $[-]_{CH}$.

3.2.14. Example. Let $\varphi \longrightarrow_\beta \psi$. The following deductions are equivalent under $\sim_{CH}$:

$$\dfrac{\dfrac{\dfrac{[\varphi]^i}{\psi}}{\varphi \supset \psi}\,i}{\psi \supset \psi} \qquad \dfrac{\dfrac{[\varphi]^i}{\varphi \supset \varphi}\,i}{\psi \supset \varphi} \qquad \dfrac{[\varphi]^i}{\varphi \supset \varphi}\,i$$

and are different from

$$\dfrac{\dfrac{[\psi]^i}{\psi \supset \psi}\,i}{\varphi \supset \varphi}$$

Also in ΛPRED there is a trivial variation on a proof-term that we want to abstract from. The situation occurs already in the definition of $(\!|\,|\!)$, which is not fully fixed, due to the choices of renamings of proof-variables that we have to make. So, what we want to do is consider pairs (Δ, M), where Δ is a proof-context and M a proof-term, and an equivalence relation on these pairs such that (Δ_1, M_1) and (Δ_2, M_2) are equivalent if there is a substitution of proof-variables for proof-variables σ such that $\sigma(\Delta_1) \equiv \Delta_2$ and $\sigma(M_1) \equiv M_2$. If this is the case we call (Δ_1, M_1) and (Δ_2, M_2) *equivalent modulo renaming of proof-variables.*

3.2.15. Proposition. *Let Θ and Θ' be deductions in* PRED*.*

1. *If $\Theta \sim_{CH} \Theta'$, then $(\Delta_\Theta, (\!|\Theta|\!))$ and $(\Delta_{\Theta'}, (\!|\Theta'|\!))$ are equivalent modulo renaming of proof-variables.*
2. *$\Sigma((\!|\Theta|\!)) \sim_{CH} \Theta$.*

Proof. The first by induction on the structure of $\langle\Theta\rangle (= \langle\Theta'\rangle)$. The second by induction on the structure of Θ. $\boxtimes$

The following is now an obvious consequence of Proposition 3.2.15 and Proposition 3.2.10.

3.2.16. Corollary. *The mappings $\Sigma(-)$ and $(\!|-|\!)$ constitute a bijection between $\sim_{CH}$-equivalence classes of deductions in* PRED *and pairs (Δ, M) modulo renaming of proof-variables.*

3.2.2. Comparison with other embeddings

In [Barendregt 1992] a different embedding of logic-in-natural-deduction-style into typed lambda calculus is given. For this system we have no completeness on the level of derivations (and hence the embedding is not an isomorphism on the level of derivations). In Chapter 2.1, paragraph 2.2, we have already pointed out what the problem is: the formalization of the logic is not good; it is somewhere in between a sequent-formulation of natural deduction (as it is used in [Howard 1980]) and a 'real' natural deduction formulation, (like the one in [Prawitz 1965]). As a consequence the proof-terms $\lambda p{:}\varphi.\lambda q{:}\varphi.p$ and $\lambda p{:}\varphi.\lambda q{:}\varphi.q$ will always be mapped to the same derivation-tree of the original logic.

The embedding that was described in [Barendregt 1992] has been studied extensively in [Tonino anf Fujita 1992] for the case of higher order logic. In this paper a completeness result is stated which can not be right, namely Theorem 6.2, saying that the composition of, first the mapping from type system to logic and then the mapping from logic to type system, constitutes the identity on the level of proof-terms. The two proof-terms of the formula $\varphi \supset \varphi \supset \varphi$ as given above present a counter-example.

It will be clear from these remarks that we feel a strong preference for the embedding as described above: there is a clear correspondence between derivation trees and proof-terms. Note also that in [Barendregt 1992] the embedding is done in two steps: first linearize the derivation trees and then embed these as typed lambda terms in a calculus. This calculus (λPRED) is different from our ΛPRED, because it does not distinguish proof-contexts and object-contexts. Our embedding is also done in two steps: Above we have given the interpretation of derivation trees as typed lambda terms in ΛPRED. In Chapter 6.1 it will be shown that our system ΛPRED is the same as the calculus λPRED used in [Barendregt 1992]. We think that the way in which we have split up the embedding is more natural and gives a better insight .

3.2.3. Reduction of derivations and extensions to higher orders

It is well-known that cut-elimination in PRED corresponds to normalization of β-reduction. Let's make this precise by defining a reduction relation on deductions of PRED.

3.2.17. Definition. The reduction relation $\longrightarrow_B$ on deductions of PRED is defined as follows.

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}[\varphi]^i\\ \Sigma\\ \psi\end{array}}{\varphi\supset\psi}\,i}{\varphi'\supset\psi'} \qquad \begin{array}{c}\Theta\\ \varphi'\end{array}}{\psi'} \quad\longrightarrow_B\quad \dfrac{\begin{array}{c}\Theta\\ \dfrac{\varphi'}{\varphi}\\ \Sigma\\ \psi\end{array}}{\psi'}$$

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\Sigma\\ \varphi\end{array}}{\forall x\,\epsilon\, D.\varphi}}{\forall x\,\epsilon\, D.\varphi'}}{\varphi'[t/x]} \quad\longrightarrow_B\quad \dfrac{\begin{array}{c}\Sigma[t/x]\\ \varphi[t/x]\end{array}}{\varphi'[t/x]}$$

The definition of $\Sigma[t/x]$ will be clear and it is easy to check that $\Sigma[t/x]$ is indeed a deduction of $\varphi[t/x]$.

The reduction relation $\longrightarrow_B$ eliminates what is generally known as a 'cut': a redundancy in a proof by first introducing a connective and then immediately eliminating it.

3.2.18. Proposition. *There is a one-to-one correspondence between reduction steps* $\longrightarrow_B$ *in a deduction* Θ *of* PRED *and* β*-reductions in the corresponding proof-term* $([\Theta])$ *of* ΛPRED. *Hence we have:*

$$\begin{array}{rl} & \longrightarrow_B \textit{ is (strongly) normalizing on deductions of } \mathrm{PRED}\\ \Leftrightarrow & \beta\textit{-reduction is (strongly) normalizing on proof-terms of } \Lambda\mathrm{PRED}.\end{array}$$

Proof. Immediate from the one-to-one correspondence between equivalence classes of deductions and proof-terms modulo renaming of proof-variables, as it was stated in Corollary 3.2.16.$\boxtimes$

In [Howard 1980] the formulas-as-types embedding is discussed for the full intuitionistic first order predicate logic. In ΛPRED this amounts to the addition of the connectives $\vee, \&, \neg$ and $\exists$ and the corresponding operators for the introduction and elimination rules. Also these operators come together with reduction rules that correspond to the rules for cut-elimination for the connectives in the full first order predicate logic. [Howard 1980] also discusses the extension to Heyting Arithmetic which amounts to the addition of an induction operator. We do not give details of these extensions. Our exposition for the case of PRED covers all the basic difficulties that one encounters, so the extension is a straightforward one. Moreover we are more interested in giving some details of the extension to second and higher order systems, in which all the extra connectives and induction can be defined.

3.2.19. Definition. The systems λPRED2 and λPREDω are defined by extending λPRED in the following way.

1. For λPRED2, allow quantification over all types, i.e. add the possibility of quantification over predicate types. (The distinction between functional and predicate types is still meaningful, because we do not allow the formation of object-terms by λ-abstraction over predicate types.)

2. For λPREDω, extend the types to

$$\mathsf{Type} ::= \mathsf{Var}^{ty} \mid \mathsf{Prop} \mid \mathsf{Type}{\rightarrow}\mathsf{Type},$$

   and allow quantification and λ-abstraction over all types. (Then there is no need to distinguish between functional and predicate types, but we may still do so, a type being a functional type if it is of the form $A_1 \rightarrow \cdots \rightarrow A_n$ with A_n a type-variable and a predicate type if it is of the form $A_1 \rightarrow \cdots \rightarrow \mathsf{Prop}$.)

The connectives $\vee, \&, \neg$ and $\exists$ can now be defined in terms of $\supset$ and $\forall$ in both λPRED2 and λPREDω. The definitions have already been given in Remark 2.2.7. This means that there are closed proof-terms that correspond to the introduction and elimination rules for the connectives. The correspondence is even stronger in the sense that these closed terms satisfy part of the reduction rules that correspond to cut-elimination. It is not difficult to verify this and we therefore just treat the cases for $\vee$ and $\exists$ as an example. (The terms corresponding to introduction and elimination only satisfy part of the cut-elimination rules, because in the full predicate logic there are also rules that combine an elimination rule for one connective with a rule of another connective. These are not satisfied. See e.g. [Girard et al. 1989] for these type of rules.)

3.2.20. Examples. We work in λPRED2.

1. The connective $\vee$ is defined by $\varphi \vee \psi := \forall \alpha{:}\mathsf{Prop}(\varphi \supset \alpha) \supset (\psi \supset \alpha) \supset \alpha$ and we have the following combinators for $\vee$-introduction and $\vee$-elimination. (For reasons of readability we have omitted some type information.)

$$\begin{array}{lcl}
\vee\text{-I}_1 & : & \varphi \supset \varphi \vee \psi, \\
 & := & \lambda x{:}\varphi.\lambda \alpha g h.gx, \\
\vee\text{-I}_2 & : & \psi \supset \varphi \vee \psi, \\
 & := & \lambda x{:}\psi.\lambda \alpha g h.hx, \\
\vee\text{-E} & : & \forall \alpha{:}\mathsf{Prop}.\varphi \vee \psi \supset (\varphi \supset \alpha) \supset (\psi \supset \alpha) \supset \alpha, \\
 & := & \lambda \alpha.\lambda x{:}\varphi \vee \psi.\lambda g h.x \alpha g h.
\end{array}$$

These combinators satisfy the following reductions.

$$\begin{array}{rcl}\vee\text{-E}\chi(\vee\text{-I}_1 t)gh & \twoheadrightarrow_\beta & gt,\\ \vee\text{-E}\chi(\vee\text{-I}_2 t)gh & \twoheadrightarrow_\beta & ht.\end{array}$$

These reductions correspond in the obvious way to the rewriting of a part of a deduction where we have done an $\vee$-introduction and then immediately a $\vee$-elimination.

2. The connective $\exists$ is defined by $\exists x{:}A.\varphi := \forall\alpha{:}\mathsf{Prop}.(\forall x{:}A.\varphi \supset \alpha) \supset \alpha$. and we have the following combinators for $\exists$-introduction and $\exists$-elimination. (Again we have omitted some type information.)

$$\begin{array}{rcl}\exists\text{-I} & : & \forall x{:}A.(\varphi \supset \exists x{:}A.\varphi),\\ & := & \lambda x{:}A.\lambda h{:}\varphi.\lambda\alpha g.gxh,\\ \exists\text{-E} & : & \forall\alpha{:}\mathsf{Prop}.(\exists x{:}A.\varphi) \supset (\forall x{:}A.\varphi \supset \alpha) \supset \alpha,\\ & := & \lambda\alpha hg.h\alpha g.\end{array}$$

These combinators satisfy the following reduction.

$$\exists\text{-E}\chi(\exists\text{-I}th)g \quad \twoheadrightarrow_\beta \quad gth,$$

which corresponds to the rewriting of a part of a deduction where we have done an $\exists$-introduction and then immediately a $\exists$-elimination.

In a similar way one can also interpret Heyting Arithmetic in λPRED2: starting from a fixed type A and two objects $0 : A$ and $S : A{\to}A$ (declared as variables in the object-context, but in fact treated as constants), one would like to construct a proof-term of type

$$\text{Ind} := \forall P{:}A{\to}\mathsf{Prop}.P0 \supset (\forall y{:}A.Py \supset P(Sy)) \supset (\forall x{:}A.Px).$$

As it is stated now this is of course impossible: nothing tells us that the objects of type A are just the ones built up from 0 and S. We can handle this by relativization. Let $N{:}A{\to}\mathsf{Prop}$ be defined by

$$N := \lambda x{:}A.\forall P{:}A{\to}\mathsf{Prop}.P0 \supset (\forall y{:}A.Py \supset P(Sy)) \supset Px.$$

So Nt is true if t is built up from 0 and S only, i.e.
Nt is true if t is a numeral. We have the following proof-terms

$$\begin{array}{rcl}\text{zero} & : & N0\\ & := & \lambda P{:}A{\to}\mathsf{Prop}.\lambda h_0h_1.h_0,\\ \text{succ} & : & \forall x{:}A.Nx \supset N(Sx)\\ & := & \lambda x{:}A.\lambda q{:}Nx.\lambda P{:}A{\to}\mathsf{Prop}.\lambda h_0h_1.h_1x(qPh_0h_1).\end{array}$$

We can now define induction as follows.

$$\text{Ind} := \forall P{:}A{\to}\mathsf{Prop}.P0 \supset (\forall y{:}A.(Py \,\&\, Ny) \supset P(Sy)) \supset (\forall x{:}A.Nx \supset Px).$$

So Ind states induction for numerals. We can now find a closed term

$$\text{ind} : \text{Ind}$$

that also satisfies the required equality rules. (Compare for example with the scheme for induction in [Howard 1980].)

$$\begin{array}{rcl} \text{ind}Pt_0t_1 0\text{zero} & \twoheadrightarrow_\beta & t_0 : P0, \\ \text{ind}Pt_0t_1(Sn)(\text{succ}nq) & =_\beta & t_1n(\text{ind}Pt_0t_1nq) : P(Sn). \end{array}$$

3.3. The formulas-as-types notion à la de Bruijn

We now want to say something about the work of de Bruijn in the Automath project in relation to the notion of formulas-as-types. Presenting things in this way suggests that there are two totally different approaches, which is not true. (For example in the Automath project many different systems have been introduced and some of them are quite close to systems that we have seen in the previous section.) The reason for separating the two is that both have their own basic underlying ideas that we want to single out. This is also the reason that in this section we restrict our attention mainly to the system AUT-68, which probably covers best those basic ideas of Automath that we want to talk about.

We do not want to introduce AUT-68 in the original format, but in a format close to the typed lambda calculus λPRED that we have encountered in the previous section. The reason is twofold: first it would take a lot of space to explain AUT-68 in its original format. (Something which has been done quite succesfully in [van Daalen 1973].) Second we want to present it in a format which is close to one that will be used later for describing typed lambda calculi. This means that we ignore some of the features that are inevitable for making the system feasible for man-machine interaction but are inessential for our discussion of formulas-as-types. (Like the definition-mechanism of Automath.)

Our definition of AUT-68 owes a lot to discussions with van Benthem Jutting. In fact it is a derivative of he description he has given of AUT-68 as a Pure Type System.

3.3.1. Definition. AUT-68 is a system for deriving judgements of the form

$$\Gamma \vdash M : B$$

Here Γ is a *context*, i.e. a sequence of *declarations*, which are statements of the form $x : A$, where x is a variable and A a term. The M and B are terms, which are taken from the set of pseudoterms

$$\mathsf{T} ::= \mathsf{Var} \mid \textbf{type} \mid \mathsf{TT} \mid \lambda x{:}\mathsf{T}.\mathsf{T} \mid \Pi x{:}\mathsf{T}.\mathsf{T},$$

on which we have the usual notions of substitution, β- and η-reduction etcetera. The terms are singled out from the set T by the derivation rules that determine which judgements $\Gamma \vdash M : B$ are derivable. The derivation rules are the following.

$$\text{(base)} \quad \emptyset \vdash$$

$$\text{(ctxt)} \quad \frac{\Gamma \vdash A(:\mathbf{type})}{\Gamma, x{:}A \vdash} \text{ if } x \text{ not in } \Gamma$$

$$\text{(ax)} \quad \frac{\Gamma \vdash}{\Gamma \vdash \mathbf{type}}$$

$$\text{(proj)} \quad \frac{\Gamma \vdash}{\Gamma \vdash x : A} \text{ if } x{:}A \in \Gamma$$

$$(\Pi 1) \quad \frac{\Gamma, x{:}\mathbf{type} \vdash B(:\mathbf{type})}{\Gamma \vdash \Pi x{:}\mathbf{type}.B}$$

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

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

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

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

$$\text{(conv)} \quad \frac{\Gamma \vdash M : B \quad \Gamma \vdash A(:\mathbf{type})}{\Gamma \vdash M : A} \; A =_\beta B$$

We use the convention of writing $A{\to}B$ for $\Pi x{:}A.B$ if $x \notin \mathrm{FV}(B)$.

As people familiar with Automath may notice, we have not only changed the presentation of the system, but also the system itself. For example the original system does not contain Π-expressions: λ is used everywhere at places where we have put a Π. We feel that the systems with Πs is more natural and it is certainly more readily understood by people who are familiar with the actual developments in typed lambda calculi. Moreover there is no real difference between the two versions of the system: if we use the formalisation without Π we can always 'recognise' the λs that should be 'read as' Πs. (This is not true for extensions of AUT-68 like AUT-QE, where the identification of Π and λ really extends the system.)

Those not familiar with this kind of calculus may wonder what the use of this system is. We therefore give an example. The general purpose of the system is to provide a logical framework in which a user can work with a formal system of his or her choice. The situation is then that the language of the formal system is declared in a context, which is then fixed. (This part of the context is then used as a kind of 'signature' and the variables declared in it act as constants.)

3.3.2. EXAMPLE. First Order Predicate Logic. The idea is to interpret the domains of the logic as well as the formulas as types, a domain being understood as the type of its elements and a formula being understood as the type of its proofs. Consider the following context.

$$\begin{array}{rcl} \Gamma & := & \bot : \mathbf{type}, \vee : \mathbf{type}{\to}\mathbf{type}{\to}\mathbf{type}, \\ & & \mathrm{abs} : \Pi x{:}\mathbf{type}.\bot{\to}x, \\ & & \mathrm{in}_1 : \Pi x, y{:}\mathbf{type}.x{\to}(x \vee y), \mathrm{in}_2 : \Pi x, y{:}\mathbf{type}.y{\to}(x \vee y), \\ & & \mathrm{out} : \Pi x, y, z{:}\mathbf{type}.(x \vee y){\to}(x{\to}z){\to}(y{\to}z){\to}z, \\ & & \mathrm{cl} : \Pi x{:}\mathbf{type}.x \vee (x{\to}\bot). \end{array}$$

Then, we have for example (abbreviating $A{\to}\bot$ to $\neg A$),

$$\begin{array}{rcl} \Gamma & \vdash & \lambda x{:}\mathbf{type}.\lambda y{:}\neg\neg x.\mathrm{out}xyz(\mathrm{cl}x)(\lambda p{:}x.p)(\lambda q{:}\neg x.\mathrm{abs}x(yq)) \\ & : & \Pi x{:}\mathbf{type}.((x{\to}\bot){\to}\bot){\to}x. \end{array}$$

The universal quantifier is interpreted by the Π:

$$\forall x{:}A.\varphi := \Pi x{:}A.\varphi$$

and we can define the existential quantifier in terms of the universal one (classically) by

$$\exists x{:}A.\varphi := \neg\forall x{:}A.\neg\varphi.$$

The theory of natural numbers can now be developed by adding to Γ

$$\begin{array}{ll} N : \mathbf{type}, & 0 : N, S : N{\to}N, + : N{\to}N{\to}N, = : N{\to}N{\to}N, \\ & \mathrm{comm} : \Pi x, y{:}N.x + y = y + x, \ \text{etc.} \end{array}$$

One of the drawbacks of this kind of interpretation of first order predicate logic is that domains of the logic and formulas are not only treated in the same manner (as types), but even as if they were the same kind of things: the system itself can not distinguish between formulas and domains. This was also recognised by de Bruijn who especially emphasized this drawback in relation to so called 'proof-irrelevance'. This becomes very apparent if we look at situations where proof-terms are subexpressions of the object-terms, for example if we have

$$\mathbb{R} : \mathbf{type}, \mathrm{pos} : \mathbb{R} \to \mathbf{type}, \mathrm{sqrt} : \Pi x{:}\mathbb{R}.\mathrm{pos}(x) \to \mathbb{R},$$

where $\mathbb{R}$ represents the real numbers, pos the predicate that decides whether a number is non-negative and sqrt constructs the square root of a number if that number is non-negative. Although in general we may want to distinguish different proofs of a formula, we obviously want $\mathrm{sqrt}rp$ only to depend on r and on the *fact* that r is non-negative (not on the particular proof: $\mathrm{sqrt}rp$ and $\mathrm{sqrt}rp'$ should represent the same real number). Clearly there is no way to state proof-irelevance in its most general form like 'for all formulas φ all terms of type φ are equal'.

One of the extensions of AUT-68 that has been considered (and is also known under the name AUT-68) is the one which splits **type** into **type** and **prop**. So for **prop** we have the same rules as **type** (but we can now easily make variants of the system that handles **type** and **prop** differently), but we can specify different axioms for **prop** in the context.

There are some other drawbacks to the direct interpretation of formulas as types. Note that the system is essentially first order: we can not quantify over the collection of subsets of a domain. To do this we would have to be able to write down $(\Pi P{:}A \to \mathbf{type}.\varphi) : \mathbf{type}$, which is not allowed. As a consequence we also can not formalise induction in its most general form. It would have to be something like

$$\Pi P{:}N \to \mathbf{type}.P0 \to (\Pi x{:}N.Px \to P(Sx)) \to (\Pi y{:}N.Py).$$

(Note that the fact that we have $\Pi x{:}\mathbf{type}.B$ (for $B : \mathbf{type}$) in the system does not mean that the system is impredicative: $\Pi x{:}\mathbf{type}.B$ itself is not of type **type**.) For the same reason we can not represent the (first order) intuitionistic existential quantifier. Knowing that it can't be defined in terms of $\forall$, the only option is to declare it in the context with its introduction and elimination rules:

$$\exists : \Pi x{:}\mathbf{type}.(x \to \mathbf{type}) \to \mathbf{type},$$

but this is not allowed.

To overcome the drawbacks that we just mentioned, yet another option has been developed by the Automath community, which does not require a change of the system but only a different use of it. The idea is to not let formulas be types themselves but to introduce a fixed type constant prop, representing the names of the formulas, and a kind of lifting operator $T : \mathrm{prop} \to \mathbf{type}$, which maps a name of a formula to the type of its proofs. Although the difference with the first interpretation may seem small at first sight, this is a major improvement. First the system is now really used as a framework: in the previous interpretation some features of the type system were used directly for the logic (like the Π which is used as $\forall$ and $\supset$), whereas now all the quantifiers have to be represented in a context. Further this interpretation gives much more flexibility, allowing one to interpret for example second order and higher order logic in a similar way, but also more exotic formal systems like typed lambda calculus itself. Let's give an example of a formalisation done according to this new point of view.

3.3.3. EXAMPLE. First Order Predicate Logic. We adapt the example that we gave before to the new interpretation.

$$\begin{aligned}
\Gamma \;:=\; & \mathrm{prop} : \mathbf{type}, T : \mathrm{prop}{\to}\mathbf{type},\\
& \bot : \mathrm{prop}, \vee : \mathrm{prop}{\to}\mathrm{prop}{\to}\mathrm{prop},\\
& \supset : \mathrm{prop}{\to}\mathrm{prop}{\to}\mathrm{prop}, \forall : \Pi x{:}\mathbf{type}.(x{\to}\mathrm{prop}){\to}\mathrm{prop},\\
& \mathrm{abs} : \Pi x{:}\mathrm{prop}.T(\bot){\to}T(x),\\
& \mathrm{in}_1 : \Pi x, y{:}\mathrm{prop}.T(x){\to}T(x \vee y), \mathrm{in}_2 : \Pi x, y{:}\mathrm{prop}.T(y){\to}T(x \vee y),\\
& \mathrm{out} : \Pi x, y, z{:}\mathrm{prop}.T(x \vee y){\to}(T(x){\to}T(z)){\to}(T(y){\to}T(z)){\to}T(z),\\
& \forall\text{-I} : \Pi x{:}\mathbf{type}.\Pi P{:}x{\to}\mathrm{prop}.(\Pi z{:}x.T(Pz)){\to}T(\forall x P),\\
& \forall\text{-E} : \Pi x{:}\mathbf{type}.\Pi P{:}x{\to}\mathrm{prop}.T(\forall x P){\to}\Pi z{:}x.T(Pz),\\
& \mathrm{cl} : \Pi x{:}\mathrm{prop}.T(x) \vee T(x \supset \bot),\\
& \text{etcetera.}
\end{aligned}$$

(We have not stated the rules for $\supset$.) Again we have an M such that

$$\Gamma \vdash M : \Pi x{:}\mathrm{prop}.T(((x \supset \bot) \supset \bot) \supset x).$$

The intuitionistic existential quantifier can now also be defined by letting

$$\exists : \Pi x{:}\mathbf{type}.(x{\to}\mathrm{prop}){\to}\mathrm{prop}$$

and adding declarations for the intuitionistic introduction and elimination rule. We can also add induction for the natural numbers by declaring

$$\mathrm{ind} : \Pi P{:}N{\to}\mathrm{prop}.T(P0){\to}T(\forall N(\lambda x{:}N.Px \supset P(Sx))){\to}T(\forall N(\lambda y{:}N.Py)).$$

The flexibility is really an enormous advantage of the system. This was also noticed by researchers in Edinburgh, who defined their system LF ('Logical Framework', [Harper et al. 1987]) based on ideas from Automath. We have again been inspired by LF in the choice for our representation of AUT-68, which is quite close to LF. We shall say something more about LF later. Now we want to treat as an example higher order predicate logic (PREDω) in AUT-68. As one may have noticed in the previous example, the domains of the logic are still types, which may be undesirable if one wants to allow operations on domains that are not allowed on types in AUT-68. (For example cartesian products of domains.) In that case one would like to push the language of the logic one level lower by introducing a type of names of domains 'dom' and an operator $D : \mathrm{dom}{\to}\mathbf{type}$ that maps a name of a domain to the type of its elements. Higher order predicate logic is one example of a system where such an approach is appropriate.

3.3.4. EXAMPLE. We interpret the system PREDω in AUT-68 by introducing the following context.

$$
\begin{aligned}
\Gamma \;:=\; & \mathrm{dom} : \mathbf{type}, D : \mathrm{dom}{\to}\mathbf{type},\\
& \Rightarrow\, : \mathrm{dom}{\to}\mathrm{dom}{\to}\mathrm{dom}, \mathrm{prop} : \mathrm{dom}\\
& = : \Pi d{:}\mathrm{dom}.Dd{\to}Dd{\to}\mathbf{type},\\
& \mathrm{Ap} : \Pi d, e{:}\mathrm{dom}.D(d{\Rightarrow}e){\to}Dd{\to}De,\\
& \mathrm{Abs} : \Pi d, e{:}\mathrm{dom}.(Dd{\to}De){\to}D(d{\Rightarrow}e),\\
& \beta : \Pi d, e{:}\mathrm{dom}.\Pi f{:}Dd{\to}De.\Pi x{:}Dd.\mathrm{Ap}de(\mathrm{Abs}def)x{=}fx,\\
& \xi : \Pi d, e{:}\mathrm{dom}.\Pi f, g{:}Dd{\to}De.(\Pi x{:}Dd.fx{=}gx){\to}(\mathrm{Abs}def{=}\mathrm{Abs}deg),\\
& \mathrm{comp} : \Pi d, e{:}\mathrm{dom}.\Pi f, g{:}D(d{\Rightarrow}e).\Pi x, y{:}Dd.x{=}y{\to}f{=}g{\to}(\mathrm{Ap}fx{=}\mathrm{Ap}gy),\\
& T : D\mathrm{prop}{\to}\mathbf{type},\\
& \supset: D\mathrm{prop}{\to}D\mathrm{prop}{\to}D\mathrm{prop}, \forall : \Pi d{:}\mathrm{dom}.(Dd{\to}D\mathrm{prop}){\to}D\mathrm{prop},\\
& \supset\text{-I} : \Pi x, y{:}D\mathrm{prop}.T(x \supset y){\to}Tx{\to}Ty,\\
& \supset\text{-E} : \Pi x, y{:}D\mathrm{prop}.(Tx{\to}Ty){\to}T(x \supset y),\\
& \forall\text{-I} : \Pi d{:}\mathrm{dom}.\Pi P{:}Dd{\to}D\mathrm{prop}.(\Pi z{:}Dd.T(Pz)){\to}T(\forall dP),\\
& \forall\text{-E} : \Pi d{:}\mathrm{dom}.\Pi P{:}Dd{\to}D\mathrm{prop}.T(\forall dP){\to}\Pi z{:}Dd.T(Pz).
\end{aligned}
$$

By pushing the domains one level lower, all of the higher order language is now coded, but still the substitution and conversion mechanisms of the system take care of substitution and α-conversion in the defined higher order language.

Note that this is not the only possibility: an alternative is to let the domains still be types in which case one would have for example

$$
\begin{aligned}
\Gamma \;:=\; & \mathrm{prop} : \mathbf{type}, T{:}\mathrm{prop}{\to}\mathbf{type},\\
& \supset: \mathrm{prop}{\to}\mathrm{prop}{\to}\mathrm{prop}, \forall : \Pi d{:}\mathbf{type}.(d{\to}\mathrm{prop}){\to}\mathrm{prop},\\
& \supset\text{-I} : \Pi x, y{:}\mathrm{prop}.T(x \supset y){\to}Tx{\to}Ty,\\
& \supset\text{-E} : \Pi x, y{:}\mathrm{prop}.(Tx{\to}Ty){\to}T(x \supset y),\\
& \forall\text{-I} : \Pi d{:}\mathbf{type}.\Pi P{:}d{\to}\mathrm{prop}.(\Pi z{:}d.T(Pz)){\to}T(\forall dP),\\
& \forall\text{-E} : \Pi d{:}\mathbf{type}.\Pi P{:}d{\to}\mathrm{prop}.T(\forall dP){\to}\Pi z{:}d.T(Pz),
\end{aligned}
$$

etcetera.

But this is exactly the same context as we had in Example 3.3.3!

3.3.5. REMARK. The context of Example 3.3.3 represents higher order predicate logic in AUT-68. The $\forall$ quantifier that is declared in the context applies to all types, so it applies to A, $A{\to}\mathrm{prop}$, $(A{\to}\mathrm{prop}){\to}\mathrm{prop}$ etcetera.

Obviously, less coding makes things easier to read and write. However, there is also an important advantage of the approach of Example 3.3.4, which is that adequacy of the interpretation is easier to prove. This issue has not received a lot of attention in the Automath project, but which is of course very relevant: To which extent is the interpretation of the logic in AUT-68 adequate? (Are there sentences that are provable in the interpretation in AUT-68 that were not provable in the original logic?) In the interpretation of higher order predicate logic of Example 3.3.3, the $\forall$ quantifier can range over any type, including types of the form $T\varphi$, with φ : prop. Clearly this is not available in the logic so we really have to do some work to show that this extra feature doesn't provide us any ingenious proof of an unwanted theorem (like $\bot$ for example).

The problem of adequacy of encodings of formal systems has been taken very seriously by those who defined the system LF. See for example [Gardner 1992]. Let's introduce this system and sketch how adequacy proofs are given for the system. (There is no general theorem saying that a specific way of encoding formal systems will always yield an adequate interpretation, but there is a general proof procedure that will usually do the job of proving adequacy.)

3.3.6. Definition. LF [Harper et al. 1987]is a system for deriving judgements of the form

$$\Gamma \vdash M : B$$

where Γ is a *context* and M and B are terms, which are taken from the set of pseudoterms

$$\mathsf{T} ::= \mathsf{Var} \mid \mathbf{type} \mid \mathbf{kind} \mid \mathsf{T}\mathsf{T} \mid \lambda x{:}\mathsf{T}.\mathsf{T} \mid \Pi x{:}\mathsf{T}.\mathsf{T},$$

like in the definition of AUT-68 (Definition 3.3.1). The derivation rules are the following. ($\mathbf{s}$ ranges over $\{\mathbf{type}, \mathbf{kind}\}$.)

$$\text{(base)}\quad \emptyset \vdash \qquad\qquad \text{(ctxt)}\quad \frac{\Gamma \vdash A : \mathbf{s}}{\Gamma, x{:}A \vdash}\ \text{if } x \text{ not in } \Gamma$$

$$\text{(ax)}\quad \frac{\Gamma \vdash}{\Gamma \vdash \mathbf{type} : \mathbf{kind}} \qquad\qquad \text{(proj)}\quad \frac{\Gamma \vdash}{\Gamma \vdash x : A}\ \text{if } x{:}A \in \Gamma$$

$$(\Pi)\quad \frac{\Gamma, x{:}A \vdash B : \mathbf{s} \quad \Gamma \vdash A : \mathbf{type}}{\Gamma \vdash \Pi x{:}A.B : \mathbf{s}}$$

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

$$\text{(conv)}\quad \frac{\Gamma \vdash M : B \quad \Gamma \vdash A : \mathbf{s}}{\Gamma \vdash M : A}\, A =_{\beta\eta} B$$

Again we use the convention of writing $A{\to}B$ for $\Pi x{:}A.B$ if $x \notin \mathrm{FV}(B)$.

In the definition we have ignored one feature of LF, which is the use of so called 'signatures'. These are special contexts in which constants are declared. In our definition a signature is part of the context, to be precise that part in which the language of the formal system is fixed (like the Γ in 3.3.4).

Looking again at the example of higher order predicate logic, we see that only the interpretation of Example 3.3.4 is possible in LF. The second requires a Π-abstraction over **type**, which is not allowed in LF. Apart from conversion, this is in fact the only difference between LF and AUT-68 (in the way it was defined in Definition 3.3.1). If one reads the judgements of AUT-68 that are of the form $\Gamma \vdash B$ as $\Gamma \vdash B : \mathbf{kind}$, the the systems have the same rules, except for the rule($\Pi 1$), which is extra in AUT-68.

The way to prove adequacy of the interpretation is by using so called 'long-$\beta\eta$-normal forms'. We already encountered this notion in the previous chapter. Recall that a long-$\beta\eta$-normal form is obtained by first taking the β-normal form and then doing η-expansion, where a term $C[M]$ in β-normal form η-expands to $C[\lambda x{:}A.Mx]$ only if $x \notin \mathrm{FV}(M)$, $M : \Pi x{:}A.B$ and $C[\lambda x{:}A.Mx]$ is again in β-normal form. We write long-$\beta\eta$-nf(M) for the long-$\beta\eta$-normal form of the term M. The usefulness of this definition depends on the normalization and confluence of $\beta\eta$-reduction in LF. The first property is relatively easy (shown in [Harper et al. 1987]), but the second is surprisingly complicated and was first proved by [Salvesen 1989].

Now one can define an isomorphism between $\beta\eta$-equivalence classes of terms of a specific type in Γ and terms of the corresponding domain in the higher order predicate logic. (It is of course allowed to extend Γ a little bit, but only with variable declarations x:dom or $x{:}D(d)$.) This is done by defining the isomorphism on the long-$\beta\eta$-normal forms, which form a complete set of representants for the $\beta\eta$-equivalence classes. For example all the terms of type Dd correspond to terms of the higher order predicate logic by first taking long-$\beta\eta$-normal forms and then defining the inductive mapping $[\![-]\!]$ by

$$
\begin{aligned}
[\![x]\!] &= x^{\mathsf{Prop}},\\
[\![\varphi \supset \psi]\!] &= [\![\varphi]\!] \supset [\![\psi]\!],\\
[\![\forall d(\lambda x{:}Dd.M)]\!] &= \forall x \in [\![d]\!].[\![M]\!],\\
[\![\mathrm{Ap}MN]\!] &= [\![M]\!][\![N]\!],\\
[\![\mathrm{Abs}(\lambda x{:}Dd.M))]\!] &= \lambda x \in [\![d]\!].[\![M]\!],
\end{aligned}
$$

where the correspondence $[\![-]\!]$ between terms of type dom and domains is obvious. In a similar way one defines a correspondence between terms of type $T\varphi$ in LF and deductions of φ in PREDω, establishing in this way the adequacy of the interpretation.

As pointed out already, LF can be seen as a subsystem of AUT-68, modulo some small changes. And, although the number of rules is limited, LF is very powerful in interpreting a wide variety of formal systems. (See [Harper et al. 1987], [Avron et al. 1987] or [Gardner 1992] for examples.) It is however not minimal yet: We can do without a rule without weakening the power of the system. This is partly due to the way in which the system is being used. (See the example of higher order predicate logic, 3.3.4.) Once the context Γ that represents the formal system has been established, one is only interested in judgements of the form

$$\Gamma \vdash M : A, \text{ with } A \text{ a type.}$$

On the other hand there is no reason to let the context Γ not be in normal form. From these two principles we can show that half of the rule (λ) is superfluous: there is no need to be able to form $\lambda x{:}A.M : \Pi x{:}A.B$ in case $\Pi x{:}A.B : \mathbf{kind}$.

3.3.7. Definition. In LF we split the rule (λ) in two, a (λ_0) and a (λ_P) rule. For convenience we attach a label to the abstraction that we introduce with the rule, so

$$(\lambda_0) \quad \frac{\Gamma, x{:}A \vdash M : B \quad \Gamma \vdash \Pi x{:}A.B : \mathbf{type}}{\Gamma \vdash \lambda_0 x{:}A.M : \Pi x{:}A.B}$$

$$(\lambda_P) \quad \frac{\Gamma, x{:}A \vdash M : B \quad \Gamma \vdash \Pi x{:}A.B : \mathbf{kind}}{\Gamma \vdash \lambda_P x{:}A.M : \Pi x{:}A.B}$$

The system LF without the rule (λ_P) we call LF^-, and we write $\vdash^-$ for judgements in LF^-. On the terms of LF we now distinguish β_0-reduction from β_P-reduction in the obvious way:

$$\begin{array}{rcl} (\lambda_0 x{:}A.M)N & \longrightarrow_{\beta_0} & M[N/x], \\ (\lambda_P x{:}A.M)N & \longrightarrow_{\beta_P} & M[N/x]. \end{array}$$

Similarly we can now talk about β_P-normal forms etcetera.

We can show that a β_P-normal form of a relevant judgement contains no λ_P and that if a judgement contains no λ_P, it can be derived without the rule (λ_P).

3.3.8. Proposition. *1. If $M : \mathbf{type}$ or $M : A : \mathbf{type}$ in LF, then β_P-nf(M) contains no λ_P.*

2. If $\Gamma \vdash M : A$, Γ, M and A contain no λ_P, then $\Gamma \vdash^- M : A$.

Proof. Both by induction on the derivation of $\Gamma \vdash M : A$. Details can be found in [Geuvers 1990]. ⊠

3.3.9. Corollary. *If $\Gamma \vdash M : A(: \mathbf{type})$, all in β_P-normal form, then $\Gamma \vdash^- M : A(: \mathbf{type})$.*

If Γ is an LF context representing some system of logic and A is a type that represents some formula of this logic, then we can assume Γ and A to be in β_P-normal form. Now, when looking for a proof of A in LF, one only has to look at terms that do not contain a λ_P: the (λ_P) rule can totally be ignored.

The previous Proposition says that the only real need for $\Pi x{:}A.B : \mathbf{kind}$ is to be able to declare a variable in it. Even this use is usually of the most simple form where $x \notin \mathrm{FV}(B)$. The standard application of it in both AUT-68 and LF (certainly for logical systems) is the declaration of $T : \mathrm{prop}{\to}\mathbf{type}$, where $\mathrm{prop} : \mathbf{type}$ is another declaration. In practice, this going hence and forth between $\varphi : \mathrm{prop}$ (the name of the formula) and $T\varphi : \mathbf{type}$ (the type of its proofs) can be very inconvenient, as was already noticed by de Bruijn in [de Bruijn 1974]. This was one of the reasons for him to introduce the system AUT-4. In fact it is a family of systems which are obtained by adding to an Automath system the 'fourth level'. In terms of the system AUT-68, as we defined it in Definition 3.3.1, this means that we add **prop** as a new constant of the language with the axiom

$$\mathbf{prop} : \mathbf{type}$$

and all the rules for **prop** to make it into a logic. For the set of rules one allows, [de Bruijn 1974] suggests different possibilities. We give here an extension of AUT-68 to an AUT-4 like system where the set of rules for **prop** is rather minimal but still interesting.

3.3.10. Definition. We define the system AUT-68$^+$ as an AUT-4 like extension to AUT-68, by adding to AUT-68 (Definition 3.3.1) the constant **prop** with the following rules.(**s** stands for **type** or **prop**.)

$$(\mathrm{ax}')\quad \frac{\Gamma \vdash}{\Gamma \vdash \mathbf{prop} : \mathbf{type}} \qquad\qquad (\mathrm{ctxt}')\quad \frac{\Gamma \vdash A : \mathbf{prop}}{\Gamma, x{:}A \vdash}\ \text{if } x \text{ not in } \Gamma$$

$$(\Pi')\quad \frac{\Gamma, x{:}A \vdash B : \mathbf{prop}\ \ \Gamma \vdash A : \mathbf{s}}{\Gamma \vdash \Pi x{:}A.B : \mathbf{prop}} \qquad (\lambda')\quad \frac{\Gamma, x{:}A \vdash M : B\ \ \Gamma \vdash \Pi x{:}A.B : \mathbf{prop}}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B}$$

The example of higher order predicate logic can now be done without any coding at all, by taking **type** for the class of domains, **prop** for the class of propositions and defining

$$\begin{array}{rcl} \varphi \supset \psi & := & \varphi{\to}\psi \text{ for } \varphi, \psi \in \mathsf{Prop}, \\ \forall x \in A.\varphi & := & \Pi x{:}A.\varphi \text{ for } A \text{ a domain and } \varphi \in \mathsf{Prop}. \end{array}$$

Then all introduction and elimination rules are obviously satisfied.

We see that the formulas-as-types interpretation of PREDω in the system AUT-68$^+$ is very 'Howard-like' in the sense that there is no coding and that introduction rules correspond directly to λ-abstractions, elimination rules to applications. We can make this correspondence formal by restricting the rules of AUT-68$^+$ and showing that the system obtained in this way is equivalent to λPREDω as discussed in the previous section. The restriction of AUT-68$^+$ is easily defined; we just remove all rules that have no meaning in higher order predicate logic.

3.3.11. DEFINITION. The system AUT-HOL (Automath for higher order predicate logic) is defined by removing from AUT-68$^+$ the rules (Π1) and (Π2). So we have the following rules. ($\mathbf{s}$ stands for **type** or **prop**.)

$$\text{(base)}\quad \emptyset \vdash \qquad\qquad \text{(ctxt)}\quad \frac{\Gamma \vdash A(:\mathbf{s})}{\Gamma, x{:}A \vdash}\ \text{if } x \text{ not in } \Gamma$$

$$\text{(ax)}\quad \frac{\Gamma \vdash}{\Gamma \vdash \mathbf{type}} \qquad\qquad \text{(ax}'\text{)}\quad \frac{\Gamma \vdash}{\Gamma \vdash \mathbf{prop} : \mathbf{type}}$$

$$\text{(proj)}\quad \frac{\Gamma \vdash}{\Gamma \vdash x : A}\ \text{if } x{:}A \in \Gamma$$

$$(\Pi)\quad \frac{\Gamma, x{:}A \vdash B : \mathbf{type}\quad \Gamma \vdash A : \mathbf{type}}{\Gamma \vdash \Pi x{:}A.B : \mathbf{type}} \qquad (\Pi')\quad \frac{\Gamma, x{:}A \vdash B : \mathbf{prop}\quad \Gamma \vdash A : \mathbf{s}}{\Gamma \vdash \Pi x{:}A.B : \mathbf{prop}}$$

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

$$\text{(conv)}\quad \frac{\Gamma \vdash M : B\quad \Gamma \vdash A : \mathbf{prop}}{\Gamma \vdash M : A}\, A =_\beta B$$

In the definition we have already anticipated towards its properties by restricting the (conv) rule to propositions. We can prove that, if $\Gamma \vdash M : A$ and A contains a redex, then $\Gamma \vdash A : \mathbf{prop}$.

Due to the fact that we have removed the rules (Π1) and (Π2), the system has a nice property that is sometimes called *context separation*. Notice first that there are three ways of adding a variable to the context, namely by declaring it as a variable of A where $A : \mathbf{prop}$ or $A : \mathbf{type}$ or neither of the two, in which case $A \equiv \mathbf{type}$ as is easily seen. So we can speak of *proof-variables* (if $A : \mathbf{prop}$), *object-variables* (if $A : \mathbf{type}$) and *set-variables*. The system has some nice properties.

3.3.12. LEMMA. *In the system AUT-HOL we have the following.*

1. *Strengthening:* $\Gamma_1, x{:}B, \Gamma_2 \vdash M : A$ *with* $x \notin FV(\Gamma_2, M, A)$*, then* $\Gamma_1, \Gamma_2 \vdash M : A$.

2. *Permutation:* $\Gamma_1, x{:}B, y{:}C, \Gamma_2 \vdash M : A$ *with* $x \notin FV(C)$, *then* $\Gamma_1, y{:}C, x{:}B, \Gamma_2 \vdash M : A$.

3. *If* $\Gamma \vdash A : \mathbf{type}$, *then* $A \equiv A_1 \to \cdots \to A_{n-1} \to A_n$ *with* $A_n \equiv \mathbf{prop}$ *or* $A_n \equiv x$ *with* $x{:}\mathbf{type} \in \Gamma$ *and all* A_i *of the same form as* A $(n > 0)$.

4. *If* $\Gamma \vdash M : A(: \mathbf{type})$, *then* M *contains no proof-variables (variables* x *with* $x : \varphi(: \mathbf{prop}) \in \Gamma$*).*

PROOF. The proof is by induction on derivations. ⊠

3.3.13. COROLLARY. *In AUT-HOL we can split up every context* Γ *into three disjoint parts* $\Gamma_1, \Gamma_2, \Gamma_3$, *the first containing the set-variables, the second the object-variables and the third the proof-variables such that*

$$\begin{aligned}
\Gamma \vdash M : A &\;\Rightarrow\; \Gamma_1, \Gamma_2, \Gamma_3 \vdash M : A \text{ with}\\
A \equiv \mathbf{type} &\;\Rightarrow\; \Gamma_1 \vdash M : \mathbf{type},\\
A : \mathbf{type} &\;\Rightarrow\; \Gamma_1, \Gamma_2 \vdash M : A.
\end{aligned}$$

As a consequence of the Lemma and the Corollary we find that AUT-HOL is isomorphic to the system λPREDω of Definition 3.2.19. The isomorphism from AUT-HOL to λPREDω consists of a rearrangement of the context as suggested in the Corollary and replacing set-variables by names for basic domains. Further we have to write $\forall x \,\epsilon\, A.\varphi$ for $\Pi x{:}A.\varphi$ if $A{:}\mathbf{type}$ and $\varphi{:}\mathbf{prop}$ and $\varphi \supset \psi$ for $\varphi \to \psi (\equiv \Pi z{:}\varphi.\psi)$ if $\varphi, \psi{:}\mathbf{prop}$. In the reverse direction we have similar replacements and rewritings.

3.3.14. PROPOSITION. *Let* $\vdash_A$ *denote derivability in AUT-HOL and* $\vdash_L$ *denote derivability in* λPREDω. *If* B *is a domain, then*

$$\Gamma \vdash_A M : B \Leftrightarrow \Gamma_1, \Gamma_2 \vdash_A M : B \Leftrightarrow \Gamma_2 \vdash_L M : B.$$

If B *a proposition, then*

$$\Gamma \vdash_A M : B \Leftrightarrow \Gamma_1, \Gamma_2, \Gamma_3 \vdash_A M : B \Leftrightarrow \Gamma_2, \Gamma_3 \vdash_L M : B.$$

Chapter 4

Pure Type Systems

4.1. Introduction

The framework of Pure Type Systems (PTSs) provides a general discription of a large class of typed lambda calculi and makes it possible to derive a lot of meta theoretic properties in a generic way. We give a list of examples of systems in the form of a PTS and then give a detailed study of the meta theory. The notion of a PTS first appears explicitly in [Geuvers and Nederhof 1991] under the name GTS (Generalised Type System), where it is used to describe the so called 'cube of typed lambda calculi' of Barendregt and its meta theory. The typed lambda calculi that belong to the class of Generalised Type Systems have only one type constructor (the Π and hence the definable $\rightarrow$) and equality rule (just β), and therefore the name 'Pure Type System' was suggested by Thierry Coquand and has been widely adopted since. The situation is that (almost) every typed lambda calculus contains a core PTS, which does of course not mean that the core PTS is in any respect the most essential part, but it gives a good starting point for research.

A notion very similar to that of PTS occurs already in the work of Terlouw ([Terlouw 1989a] and [Terlouw 1989b]), who describes (in Dutch) what he calls a 'Generalised System for Terms and Types'. It is also implicit in the work of Berardi ([Berardi 1988]), who describes various examples of Pure Type Systems without insisting on a general definition. Both have been inspired by the notion of the 'cube of typed lambda calculi', (see [Barendregt 1992]), a first important step towards the notion of PTS. The first coherent study of the meta theory is [Geuvers and Nederhof 1991], which has strongly benefitted from suggestions in [Terlouw 1989a]. The main meta-theoretic results of [Geuvers and Nederhof 1991] can also be found in [Barendregt 1992].

In what follows we give a slight extension of the notion of PTS, with η-equality, to be able to use it also for our study of the Church-Rosser property (CR) for $\beta\eta$-reduction for the Calculus of Constructions with $\beta\eta$-conversion rule. We also do the meta theory for these extended PTSs.

It is well-known that the inclusion of η complicates things quite a bit, because CR for $\beta\eta$ on the set of pseudoterms is false. We therefore describe a very weak form of the Church-Rosser property for $\beta\eta$, which turns out to be provable for the set of pseudoterms. This 'Key Lemma' will do the job in almost all cases where we used CR in the study of the meta theory of PTSs with only β-conversion in [Geuvers and Nederhof 1991]. One important case is missing, which is Subject Reduction for η (SR for η), saying that if $\Gamma \vdash M : A$ and $M \twoheadrightarrow_\eta N$, then $\Gamma \vdash N : A$. It seems that the proof can't be done without having first established a proof of Strengthening:

$$\left.\begin{array}{r}\Gamma_1, x{:}A, \Gamma_2 \vdash M : B \\ x \notin \mathrm{FV}(\Gamma_2, M, B)\end{array}\right\} \Rightarrow \Gamma_1, \Gamma_2 \vdash M : B.$$

In [Geuvers and Nederhof 1991] there is a proof of this rule for a certain subclass of PTSs. The general proof for all PTSs is given in [van Benthem Jutting 199+]). Both proofs use CR in an essential way, i.e. where the Key Lemma doesn't seem to suffice.

The Calculus of Constructions is a relatively 'simple' system for which we can prove Strengthening without having to rely on CR. This situation turns out to occur more generally: We can describe a subclass of PTSs for which Strengthening, and hence SR for η can be proved without having to rely on CR. This will be discussed in Chapter 5.1, in Definition 5.2.7 and Lemma 5.2.10. It will turn out that the Calculus of Constructions belongs to this class of systems, so it satisfies SR for η.

Often the situation is more complicated and it is not clear how to show that SR for η holds in general. This is even more worrying because a proof of the Church-Rosser property on well-typed terms will certainly require SR. So we have no proof of CR for $\beta\eta$ and it seems we are in a deadlock situation. The way out is suggested in the work of Salvesen ([Salvesen1991]) on proving CR for $\beta\eta$-reduction for LF. The trick is to first add Strengthening as a rule to the system. (This was also suggested in [Geuvers 1992] as an alternative; as things stand it is not an alternative method but the only possible one.) Many problems then vanish: The addition of a rule (strengthening) does not complicate the known meta theory and allows to prove SR for η for the extended PTS notion. We can use this, because the system without (strengthening) rule is a subsystem. This does not yet mean that we can prove SR for η and CR for $\beta\eta$ in general for the system without the rule (strengthening). We only have a proof of these two properties for *normalizing* systems. The general problem remains open.

We see that, for our study of PTSs with $\beta\eta$-conversion rule, it is useful to also study the extension of the system with a rule (strengthening). We therefore define three notions of Pure Type System: The original one with only β-conversion, to be denoted by PTS_β, the one with $\beta\eta$-conversion, to be denoted by $\mathsf{PTS}_{\beta\eta}$ and the one with $\beta\eta$-conversion and strengthening rule, to be denoted by $\mathsf{PTS}^s_{\beta\eta}$.

4.2. Definitions

The Pure Type Systems are formal systems for deriving judgements of the form

$$\Gamma \vdash M : A,$$

where both M and A are in the set of so called *pseudoterms*, a set of expressions from which the derivation rules select the ones that are typable. The Γ is a finite sequence of so called declarations, statements of the form $x : B$, where x is a variable and B is a pseudoterm. The idea is of course that a term M can only be of type A ($M : A$) relative to a typing of the free variables that occur in M and A. Before giving the precise definition of Pure Type Systems we define the set of pseudoterms T over a base set $\mathcal{S}$. (The dependency of T on $\mathcal{S}$ is usually ignored.)

4.2.1. Definition. For $\mathcal{S}$ some set, the set of pseudoterms over $\mathcal{S}$, T, is defined by

$$\mathsf{T} ::= \mathcal{S} \mid \mathsf{Var} \mid (\Pi\mathsf{Var}{:}\mathsf{T}.\mathsf{T}) \mid (\lambda\mathsf{Var}{:}\mathsf{T}.\mathsf{T}) \mid \mathsf{T}\mathsf{T},$$

where Var is a countable set of expressions, called variables. Both Π and λ bind variables and hence we have the usual notions of *free variable* and *bound variable.* We adopt the λ-calculus notation of writing $\mathrm{FV}(M)$ for the set of free variables in the pseudoterm M.
On T we have the usual notions of β and η reduction, generated from

$$(\lambda x{:}A.M)P \longrightarrow_\beta M[P/x],$$

where $M[P/x]$ denotes the substitution of P for x in M (done with the usual care to avoid capturing of free variables) and

$$\lambda x{:}A.Mx \longrightarrow_\eta M, \text{ if } x \notin \mathrm{FV}(M)$$

and both compatible with application, λ-abstraction and Π-abstraction. We also adopt from the untyped lambda calculus the conventions of denoting the transitive reflexive closure of $\longrightarrow_\beta$ by $\twoheadrightarrow_\beta$ and the transitive symmetric closure of $\twoheadrightarrow_\beta$ by $=_\beta$ (and similar for $\longrightarrow_\eta$ and $\longrightarrow_{\beta\eta} := \longrightarrow_\beta \cup \longrightarrow_\eta$.)

The typing of terms is done under the assumption of specific types for the free variables that occur in the term.

4.2.2. Definition. 1. A *declaration* is a statement of the form $x : A$, where x is a variable and A a pseudoterm,

2. A *pseudocontext* is a finite sequence of declarations such that, if $x : A$ and $y : B$ are different declarations of the same pseudocontext, then $x \not\equiv y$,

3. If $\Gamma = x_1{:}A_1, \ldots, x_n{:}A_n$ is a pseudocontext, the *domain of* Γ, $\mathsf{dom}(\Gamma)$ is the set $\{x_1, \ldots, x_n\}$; for $x_i \in \mathsf{dom}(\Gamma)$, the *image of* x_i *in* Γ, notation $\mathsf{im}_\Gamma(x_i)$, is the pseudoterm A_i.

4. For Γ a pseudocontext, a variable y is Γ-*fresh* (or just *fresh* if it is clear which Γ we are talking about) if $y \notin \mathsf{dom}(\Gamma)$.

5. For Γ and Γ' pseudocontexts, $\Gamma' \setminus \Gamma$ is the pseudocontext which is obtained by removing from Γ' all declarations $x : A$ for which $x \in \mathsf{dom}(\Gamma)$.

4.2.3. DEFINITION. A *Pure Type System with β-conversion* (PTS_β) is given by a set $\mathcal{S}$, a set $\mathcal{A} \subset \mathcal{S} \times \mathcal{S}$ and a set $\mathcal{R} \subset \mathcal{S} \times \mathcal{S} \times \mathcal{S}$. The PTS that is given by $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$ is denoted by $\lambda_\beta(\mathcal{S}, \mathcal{A}, \mathcal{R})$ and is the typed lambda calculus with the following deduction rules.

$$\text{(sort)} \quad \vdash s_1 : s_2 \qquad \text{if } (s_1, s_2) \in \mathcal{A}$$

$$\text{(var)} \quad \frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A}$$

$$\text{(weak)} \quad \frac{\Gamma \vdash A : s \quad \Gamma \vdash M : C}{\Gamma, x{:}A \vdash M : C}$$

$$(\Pi) \quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A.B : s_3} \qquad \text{if } (s_1, s_2, s_3) \in \mathcal{R}$$

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

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

$$(\text{conv}_\beta) \quad \frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma \vdash M : B} \qquad A =_\beta B$$

In the rules (var) and (weak) it is always assumed that the newly declared variable is fresh, that is, it has not yet been declared in Γ. If $s_2 \equiv s_3$ in a triple $(s_1, s_2, s_3) \in \mathcal{R}$, we write $(s_1, s_2) \in \mathcal{R}$. The equality in the conversion rule (conv_β) is the β-equality on the set of pseudoterms T.
The elements of $\mathcal{S}$ are called *sorts*, the elements of $\mathcal{A}$ (usually written as $s_1 : s_2$) are called *axioms* and the elements of $\mathcal{R}$ are called *rules*.
A *Pure Type System with $\beta\eta$-conversion* ($\mathsf{PTS}_{\beta\eta}$) is also given by a set $\mathcal{S}$, a set $\mathcal{A} \subset \mathcal{S} \times \mathcal{S}$ and a set $\mathcal{R} \subset \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ and now denoted by $\lambda_{\beta\eta}(\mathcal{S}, \mathcal{A}, \mathcal{R})$. The

only difference with a PTS_β is that a $\mathsf{PTS}_{\beta\eta}$ has a $\beta\eta$-conversion rule:

$$(\mathrm{conv}_{\beta\eta})\quad \frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma \vdash M : B}\quad A =_{\beta\eta} B$$

Again the $\beta\eta$-equality in the side condition is an equality on the set of pseudoterms T.
A *Pure Type System with $\beta\eta$-conversion and strengthening* ($\mathsf{PTS}^s_{\beta\eta}$) is also given by a set $\mathcal{S}$, a set $\mathcal{A} \subset \mathcal{S} \times \mathcal{S}$ and a set $\mathcal{R} \subset \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ and now denoted by $\lambda^s_{\beta\eta}(\mathcal{S}, \mathcal{A}, \mathcal{R})$. The difference with a $\mathsf{PTS}_{\beta\eta}$ is that a $\mathsf{PTS}^s_{\beta\eta}$ has a strengthening rule:

$$(\mathrm{streng})\quad \frac{\Gamma_1, x{:}C, \Gamma_2 \vdash M : A}{\Gamma_1, \Gamma_2 \vdash M : A}\quad \text{If } x \notin \mathrm{FV}(\Gamma_2, M, A)$$

In the following, when we use the notion 'PTS' (without subscript), we arbitrarily refer to one of the three notions above.

We see that there is no distinction between types and terms in the sense that the types are formed first and then the terms are formed using the types. The derivation rules above select the typable terms from the pseudoterms, a pseudoterm A being *typable* if there is a context Γ and a pseudoterm B such that $\Gamma \vdash A : B$ or $\Gamma \vdash B : A$ is derivable. For practical reasons we make the following definitions.

4.2.4. Definition. Let $\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$ be a PTS.

1. A pseudoterm A is *typable in* $\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$ if there is a pseudocontext Γ and a pseudoterm B such that $\Gamma \vdash A : B$ or $\Gamma \vdash B : A$ is derivable. The set of typable terms of $\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$ is denoted by $\mathsf{Term}(\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R}))$ (or just Term if the PTS is clear from the context.)

2. A pseudocontext Γ is a *context of* $\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$ ($\Gamma \in \mathsf{Context}(\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$ or just $\Gamma \in \mathsf{Context}$ if there is no ambiguity), if there are pseudoterms A and B such that $\Gamma \vdash A : B$ is derivable,

3. For Γ a context of $\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$ and A a term, A *is typable in* Γ (notation $A \in \mathsf{Term}(\Gamma)$) if $\Gamma \vdash A : B$ or $\Gamma \vdash B : A$ for some B,

4. For Γ a context, s a sort and A a term, A *is an s-term in* Γ (notation $A \in s\text{-}\mathsf{Term}(\Gamma)$) if $\Gamma \vdash A : s$,

5. For Γ a context, s a sort and A a term, A *is an s-element in* Γ (notation $A \in s\text{-}\mathsf{Elt}(\Gamma)$) if $\Gamma \vdash A : B : s$ for some term B,

6. For s a sort, *the set of s-terms* (of $\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$) is defined by $s\text{-}\mathsf{Term} := \cup_{\Gamma \in \mathsf{Context}} s\text{-}\mathsf{Term}(\Gamma)$ and *the set of s-elements/* (of $\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$) is defined by $s\text{-}\mathsf{Elt} := \cup_{\Gamma \in \mathsf{Context}} s\text{-}\mathsf{Elt}(\Gamma)$.

A practical purpose for the use of the PTS framework is that many properties can be proved once and for all for the whole class of PTSs. In paragraph 4.4 we list and prove the most important ones for the three versions of the Pure Type Systems, PTS_β, $\mathsf{PTS}_{\beta\eta}$ and $\mathsf{PTS}^s_{\beta\eta}$. In order to do the meta theory for the latter two versions, we first study the collection of pseudoterms T in a bit more detail and prove a very weak form of Church-Rosser property for $\beta\eta$-reduction on T, just enough to handle most of the cases where we used CR of β-reduction in the meta theory of PTS_β (as it was given in [Geuvers and Nederhof 1991].) We now want to give some examples of type systems that fit in the PTS framework and also say something about mappings between PTSs.

The framework yields a nice tool for describing a specific class of mappings between type systems that we call PTS-*morphisms*. These PTS-morphisms will be described as a subset of a general set of mappings between Pure Type Systems.

4.2.5. Definition. Let $\lambda(\mathcal{S},\mathcal{A},\mathcal{R})$ and $\lambda(\mathcal{S}',\mathcal{A}',\mathcal{R}')$ be PTSs.
A *mapping from* $\lambda(\mathcal{S},\mathcal{A},\mathcal{R})$ *to* $\lambda(\mathcal{S}',\mathcal{A}',\mathcal{R}')$ is a function that assigns *pseudojudgements* of $\lambda(\mathcal{S}',\mathcal{A}',\mathcal{R}')$ to derivable judgements of $\lambda(\mathcal{S},\mathcal{A},\mathcal{R})$, a pseudojudgement being a sequent $\Gamma \vdash M : B$ with Γ a pseudocontext and M, B pseudoterms.
A *morphism from* $\lambda(\mathcal{S},\mathcal{A},\mathcal{R})$ *to* $\lambda(\mathcal{S}',\mathcal{A}',\mathcal{R}')$ is a mapping f from $\mathcal{S}$ to $\mathcal{S}'$ that preserves axioms and rules, that is

$$\begin{aligned} s_1{:}s_2 \in \mathcal{S} &\Rightarrow f(s_1){:}f(s_2) \in \mathcal{S}', \\ (s_1,s_2,s_3) \in \mathcal{R} &\Rightarrow (f(s_1),f(s_2),f(s_3)) \in \mathcal{R}'. \end{aligned}$$

A PTS-morphism f from $\lambda(\mathcal{S},\mathcal{A},\mathcal{R})$ to $\lambda(\mathcal{S}',\mathcal{A}',\mathcal{R}')$ immediately extends to a mapping from the pseudoterms of $\lambda(\mathcal{S},\mathcal{A},\mathcal{R})$ to the pseudoterms of $\lambda(\mathcal{S}',\mathcal{A}',\mathcal{R}')$ and hence to a mapping between the PTSs by induction on the structure of terms. This mapping preserves substitution and $\beta(\eta)$-equality and also derivability.

4.2.6. Lemma. *If f is a* PTS-*morphism from ζ to ζ', then*

$$\Gamma \vdash_\zeta M : A \Rightarrow f(\Gamma) \vdash_{\zeta'} f(M) : f(A).$$

There are certainly many other interesting mappings between Pure Type Systems and we don't want to give the PTS-morphisms any priority. However they have some practical interest because they are easy to describe and share a lot of desirable properties. And of course the Pure Type Systems with the PTS morphisms form a category with products, coproducts and as terminal object the system with $\mathsf{Type} : \mathsf{Type}$, often referred to as $\lambda *$:

$$\begin{aligned} \mathcal{S} &= \mathsf{Type}, \\ \mathcal{A} &= \mathsf{Type} : \mathsf{Type}, \\ \mathcal{R} &= (\mathsf{Type}, \mathsf{Type}). \end{aligned}$$

There are two subclasses of PTSs that have some special interest because the systems belonging to those subclasses share some additional nice properties. Also, most of the known examples of Pure Type Systems belong to both classes.

4.2.7. DEFINITION. A PTS $\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$ is *functional* if the relation $\mathcal{A}$ is a partial function from $\mathcal{S}$ to $\mathcal{S}$ and the relation $\mathcal{R}$ is a partial function from $\mathcal{S} \times \mathcal{S}$ to $\mathcal{S}$. That is,

$$\begin{aligned} s : s', s : s'' \in \mathcal{A} &\Rightarrow s' \equiv s'', \\ (s_1, s_2, s_3), (s_1, s_2, s_3') \in \mathcal{R} &\Rightarrow s_3 \equiv s_3'. \end{aligned}$$

A PTS $\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$ is *injective* if it is functional and the functions $\mathcal{A}$ and $\mathcal{R}$ are also injective. That is,

$$\begin{aligned} s' : s, s'' : s \in \mathcal{A} &\Rightarrow s' \equiv s'', \\ (s_1, s_2, s_3), (s_1', s_2', s_3) \in \mathcal{R} &\Rightarrow s_1 \equiv s_1' \;\&\; s_2 \equiv s_2'. \end{aligned}$$

In [Barendregt 1992], the notion of functional is called 'singly-sorted' and the notion of injective is called 'singly-occupied'.

In [van Benthem Jutting et. al. 1992] there are more definitions of subclasses of Pure Type Systems that are of interest. One of the purposes of that article is to find different sets of rules that generate the same set of derivable judgements, but have easier operational properties. This is especially important for proving the completeness of type checking algorithms. We shall say something more about this in Chapter 6.1. For now we want to describe two of the subclasses of Pure Type Systems that are defined in [van Benthem Jutting et. al. 1992], because they have some importance later in the text.

4.2.8. DEFINITION. 1. A PTS $\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$ is *full* if

$$\forall s_1, s_2 \in \mathcal{S} \exists s_3 \in \mathcal{S}[(s_1, s_2, s_3) \in \mathcal{R}].$$

2. A PTS $\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$ is *semi-full* if

$$\forall s_1, s_2, s_2', s_3 \in \mathcal{S}[(s_1, s_2, s_3) \in \mathcal{R} \Rightarrow \exists s_3'[(s_1, s_2', s_3') \in \mathcal{R}]].$$

The importance of the notion of 'full' PTS lies in the fact that the second premise of the (λ) rule can be replaced by $\forall s \in \mathcal{S}[B \not\equiv s] \;\vee\; \exists s \in \mathcal{S}[B{:}s \in \mathcal{A}]$, which is much easier to handle. The importance of the notion of 'semi-full' will become clear when we study the Church-Rosser property for $\beta\eta$ in $\mathsf{PTS}_{\beta\eta}$.

To end this section we want to mention some subtle variant of the syntax that has some practical use because it allows to prove a very nice meta property. The idea is to devide up the variables in several disjoint countable subsets, one subset for every sort s, which subset will be denoted by V^s. There are some small alterations in the derivation rules given in the following definition.

4.2.9. Definition. The syntax of *Pure Type Systems with sorted variables* has the set of variables Var devided up into countable subsets Var^s for every $s \in \mathcal{S}$ and the following (var) and (weak) rule:

$$\text{(var)} \quad \frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A} \qquad x \in \mathsf{Var}^s$$

$$\text{(weak)} \quad \frac{\Gamma \vdash A : s \quad \Gamma \vdash M : C}{\Gamma, x{:}A \vdash M : C} \quad x \in \mathsf{Var}^s$$

It will turn out that, if we use the syntax with sorted variables in an injective PTS_β, the sets s-Term and s'-Term are disjoint for $s \not\equiv s'$ (and similarly for s-Elt and s'-Elt.) The importance of this fact lies in the possibility of defining a mapping on the well-typed terms of the PTS_β by induction on the structure of terms, without having to mention a specific context in which the term is typed. One only has to distinguish cases according to the sorts that specific subterms are terms or elements of.

4.3. Examples of Pure Type Systems and morphisms

4.3.1. The cube of typed lambda calculi

We first treat the so called 'cube of typed lambda calculi', as presented by Barendregt in [Barendregt 1992]. The cube includes well-known systems like the simply typed and polymorphically typed lambda calculus. To show that the two representations of these systems are in fact the same requires some technical but not difficult work.

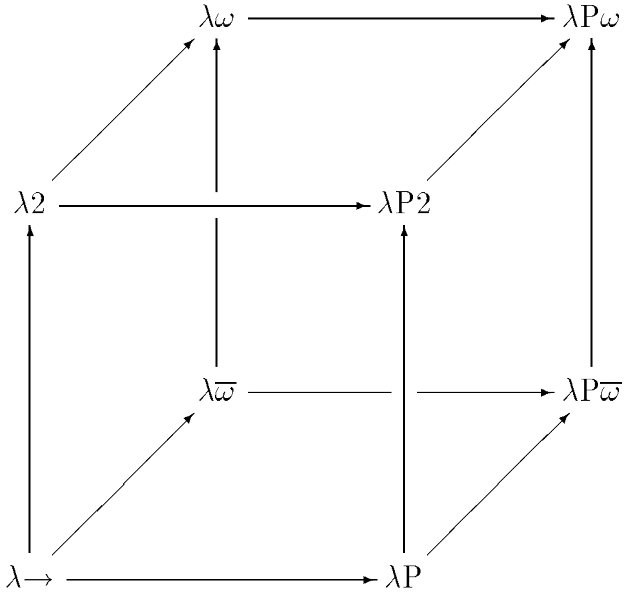
4.3.1. Definition (Barendregt). The *cube of typed lambda calculi* consists of the eight PTS_βs, all of them having as sorts the set $\mathcal{S} := \{\star, \Box\}$ and as axiom $\mathcal{A} := \{\star : \Box\}$ the rules for each system are as follows.

$\lambda\!\rightarrow$	$(\star,\star)$			
$\lambda 2$	$(\star,\star)$	$(\Box,\star)$		
$\lambda\mathrm{P}$	$(\star,\star)$		$(\star,\Box)$	
$\lambda\overline{\omega}$	$(\star,\star)$			$(\Box,\Box)$
$\lambda\omega$	$(\star,\star)$	$(\Box,\star)$		$(\Box,\Box)$
$\lambda\mathrm{P}2$	$(\star,\star)$	$(\Box,\star)$	$(\star,\Box)$	
$\lambda\mathrm{P}\overline{\omega}$	$(\star,\star)$		$(\star,\Box)$	$(\Box,\Box)$
$\lambda\mathrm{P}\omega$	$(\star,\star)$	$(\Box,\star)$	$(\star,\Box)$	$(\Box,\Box)$.

Note that all systems of the cube are injective and hence functional, so they enjoy all the nice properties that hold for these subclasses of PTSs. It is convenient to think of the set of variables Var as being split up into a set $\mathsf{Var}^\star$ and a set $\mathsf{Var}^\Box$, as was suggested in Definition 4.2.9. Te first type of variables will be referred to as *object-variables*, the second as *constructor-variables*.

The systems $\lambda\!\to$ and $\lambda 2$ are also known as the simply typed lambda calculus and the polymorphically typed lambda calculus (due to Girard as system F and Reynolds.) The system $\lambda\omega$ is a higher order version of $\lambda 2$, also known as Girard's system Fω. The presentation of these systems as a PTS is quite different from the original one. If one is just interested in those systems alone it is in general more convenient to study them in their original presentation. The PTS framework is more convenient for systems with *type dependency*, that is the feature that a type $A{:}\star$ may itself contain a term M with $M{:}B{:}\star$. This situation only occurs in the presence of the rule $(\star, \Box)$. In that case there is no other syntax for the systems which is essentially more convenient then the PTS format. The system λP is very close to LF, due to [Harper et al. 1987] (see Definition 3.3.6), in fact LF is the $\mathsf{PTS}_{\beta\eta}$ variant of λP. The system λPω is the Calculus of Constructions, due to [Coquand 1985]. (See also [Coquand and Huet 1988].) The system λP2 was defined under the same name by [Longo and Moggi 1988].

Usually the eight systems of the cube are presented in a picture as follows



where an arrow denotes inclusion of one system in another.

The use of the cube is to give a fine structure for the Calculus of Constructions (λPω), which is the largest system in the cube. It is now possible to understand λPω as built up from the basic constants $\star$ and $\Box$ by allowing three kinds of dependencies, where dependency should be understood as the possibility to abstract over specific terms to form a term of another specific kind. For example if we call the terms of type $\star$ *types* and the terms of type $\Box$ *kinds*, then $(\star,\star)$ means that we can abstract over a type to define a term of a type (e.g. $\lambda x{:}\sigma.x : \sigma\!\to\!\sigma$) and $(\Box,\star)$ means that we can abstract over a kind to define a term of a type (e.g.

$\lambda\alpha{:}\star.\lambda x{:}\alpha.x : \Pi\alpha{:}\star.\alpha\to\alpha$.) An extensive explanation of these dependencies is given in [Barendregt 1992].

As we have already pointed out, the PTS format is not always the most practical if one wants to study a specific system by itself. It is however very convenient if one wants to compare different systems. Applications of this will be given later when studying for example the Strong Normalization for the Calculus of Constructions. One of the features that can come in handy are the PTS-morphisms as defined in Definition 4.2.5. Obviously, all the inclusions inside the cube are PTS-morphisms.

Without a proof we now state the correspondence between the systems $\lambda\!\to$, $\lambda 2$ and $\lambda\omega$ in their original presentation and the PTS-format. Let's therefore define these systems here again in a different format.

4.3.2. Definition. The system $\mathrm{F}\omega$ is defined as follows. The set of *kinds*, K is given in abstract syntax by

$$K ::= \star \mid K\to K.$$

The *constructors* of $\mathrm{F}\omega$ are given by

1. There are countably many variables $\alpha_i^k : k$ for every $k \in K$,
2. If $M : k_1\to k_2$, $N : k_1$, then $MN : k_2$,
3. If $M : k_2$, then $\lambda\alpha_i^{k_1}.M : k_1\to k_2$,
4. If $\sigma : \star$ then $\Pi\alpha_i^k.\sigma : \star$,
5. If $\sigma, \tau : \star$, then $\sigma\to\tau : \star$.

we have the usual notions of bound and free variables, substitution and β-reduction on the set of constructors. An *object-context* is a sequence of declarations $x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n$ with all x_i distinct. Let Γ be an object-context. The derivation rules of Fω are the following

$$\text{(axiom)} \quad \Gamma \vdash x : \sigma \qquad \text{if } x{:}\sigma \text{ in } \Gamma,$$

$$(\rightarrow\text{-in}) \quad \frac{\Gamma, x{:}\sigma \vdash M : \tau}{\Gamma \vdash \lambda x{:}\sigma.M : \sigma\rightarrow\tau}$$

$$(\rightarrow\text{-el}) \quad \frac{\Gamma \vdash M : \sigma\rightarrow\tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}$$

$$(\Pi\text{-in}) \quad \frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \lambda\alpha^k.M : \Pi\alpha^k.\sigma} \qquad \text{if } \alpha \notin \mathrm{FV}(\Gamma),$$

$$(\Pi\text{-el}) \quad \frac{\Gamma \vdash M : \Pi\alpha^k.\sigma}{\Gamma \vdash Mt : \sigma[t/\alpha]} \qquad \text{if } t{:}k$$

$$\text{(conv)} \quad \frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \tau} \qquad \text{if } \sigma =_\beta \tau.$$

We can define the *order of a kind*, $\mathsf{ord}(k)$, just as we defined the order of domains for predicate logic in Definition 2.2.6, as follows.

$$\begin{aligned} \mathsf{ord}(\star) &= 2, \\ \mathsf{ord}(k_1\rightarrow\ldots\rightarrow k_p\rightarrow\star) &= max\{\mathsf{ord}(k_i) \mid 1 \leq i \leq p\} + 1. \end{aligned}$$

Now define for $n \in \mathbb{N}$, Fn by restricting the set of kinds of Fn (and hence the formation of constructors) to those of order $\leq n$. The system F2 will be called F and the systems F0 and F1, which are the same, are just the simply typed lambda calculus and will also be referred to as STλ.

Just as we have defined the systems Fn for $3 \leq n$ as subsystems of Fω that contain the system F, we can also define PTSs λn for all $3 \leq n$ such that

$$\lambda 2 \subset \lambda 3 \subset \cdots \subset \lambda\omega.$$

We shall not do it, because on the one hand it is quite clear what such systems should look like (restrict the formation of kinds to a certain depth) while on the other hand the definition is very involved and doesn't give any real insight. To state the equivalence of Fω and $\lambda\omega$ and of F and $\lambda 2$, we introduce some notation. For Γ a context in $\lambda\omega$ or $\lambda 2$, let $\Gamma^\Box$ be the subcontext that contains only the declarations of constructor-variables, and let $\Gamma^\star$ be the subcontext that contains only the declarations of object-variables.We have the following Lemma. (Something similar would hold for the other systems λn, if we would have defined them.)

4.3.3. LEMMA. *In $\lambda\omega$ and $\lambda 2$ we have.*

$$\begin{array}{rcl} \Gamma \vdash A : \Box & \Rightarrow & A \in K, \textit{ and in } \lambda 2,\ A \equiv \star, \\ \Gamma \vdash M : A(:\Box) & \Rightarrow & \Gamma^{\Box} \vdash M : A, \\ \Gamma \vdash M : A(:\star) & \Rightarrow & \Gamma^{\Box}, \Gamma^{\star} \vdash M : A. \end{array}$$

PROOF. Immediately by induction on derivations. ⊠

Now, if $M : A$ with A a kind in Fω, we have to introduce a context in $\lambda\omega$ to type M in. We denote this context by Γ_M. For every free constructor variable in M, Γ_M contains a declaration of this variable to the kind it has in M. Similarly for $M : A : \star$, $\Gamma_{M,A}$ contains a declaration of each constructor-variable that is free in M or A.

The other way around, if $\Gamma \vdash M : A$ in $\lambda\omega$, we denote by M^+ the term M where each constructor-variable is replaced by a variable of the kind that is given for it in Γ.

We now have the following proposition.

4.3.4. PROPOSITION.

$$\begin{array}{rcl} \Gamma \vdash_{\lambda\omega} M : A(:\Box) & \Rightarrow & M^+ : A(\in K) \textit{ in } F\omega, \\ \Gamma \vdash_{\lambda\omega} M : A(:\star) & \Rightarrow & \Gamma^{\star} \vdash_{F\omega} M : A, \end{array}$$

and the other way around

$$\begin{array}{rcl} M : A(\in K) \textit{ in } F\omega & \Rightarrow & \Gamma_M \vdash_{\lambda\omega} M : A, \\ \Gamma \vdash_{F\omega} M : A(:\star) & \Rightarrow & \Gamma_{M,A}, \Gamma \vdash_{\lambda\omega} M : A. \end{array}$$

PROOF. By induction on derivations or the structure of terms, using the Lemma.⊠

We shall go into more details about the Calculus of Constructions and other systems of the cube later, in Chapter 6.1.

4.3.2. Logics as Pure Type Systems

Other interesting example of PTSs were given by [Berardi 1988], who defined logical systems as PTSs. In Chapter 3.1 we encountered the typed lambda calculi ΛPRED (Definition 3.2.1), ΛPRED2 and ΛPREDω (Definition 3.2.19) that correspond directly to the logical systems PRED, PRED2 and PREDω, as defined in 2.2.6. The correspondence was only verified in full detail for the case of ΛPRED and PRED (see Theorem 3.2.8 and Proposition 3.2.10), but it is not very difficult to extend it to the other cases. We also saw that the correspondence is very strong in the sense that there is a correspondence between proofs and proof-terms. (See Proposition 3.2.15.) The next step is now to define PTSs that correspond to the systems ΛPRED, ΛPRED2 and ΛPREDω. The systems that we are looking for are precisely the systems that were defined by Berardi.

4.3.5. DEFINITION (Berardi). The *cube of logical typed lambda calculi*, also referred to as the *logic cube*, consists of the following eight PTS_βs. Each of them has

$$\begin{aligned} \mathcal{S} &= \{\mathsf{Prop}, \mathsf{Set}, \mathsf{Type}^p, \mathsf{Type}^s\}, \\ \mathcal{A} &= \mathsf{Prop} : \mathsf{Type}^p, \mathsf{Set} : \mathsf{Type}^s. \end{aligned}$$

The rules of each of the systems is given by the following table.

λPROP				
	$(\mathsf{Prop}, \mathsf{Prop})$			
λPROP2				
	$(\mathsf{Prop}, \mathsf{Prop})$		$(\mathsf{Type}^p, \mathsf{Prop})$	
λPROP$\overline{\omega}$				$(\mathsf{Type}^p, \mathsf{Type}^p)$
	$(\mathsf{Prop}, \mathsf{Prop})$			
λPROPω				$(\mathsf{Type}^p, \mathsf{Type}^p)$
	$(\mathsf{Prop}, \mathsf{Prop})$		$(\mathsf{Type}^p, \mathsf{Prop})$	
λPRED	$(\mathsf{Set}, \mathsf{Set})$	$(\mathsf{Set}, \mathsf{Type}^p)$		
	$(\mathsf{Prop}, \mathsf{Prop})$	$(\mathsf{Set}, \mathsf{Prop})$		
λPRED2	$(\mathsf{Set}, \mathsf{Set})$	$(\mathsf{Set}, \mathsf{Type}^p)$		
	$(\mathsf{Prop}, \mathsf{Prop})$	$(\mathsf{Set}, \mathsf{Prop})$	$(\mathsf{Type}^p, \mathsf{Prop})$	
λPRED$\overline{\omega}$	$(\mathsf{Set}, \mathsf{Set})$	$(\mathsf{Set}, \mathsf{Type}^p)$	$(\mathsf{Type}^p, \mathsf{Set})$	$(\mathsf{Type}^p, \mathsf{Type}^p)$
	$(\mathsf{Prop}, \mathsf{Prop})$	$(\mathsf{Set}, \mathsf{Prop})$		
λPREDω	$(\mathsf{Set}, \mathsf{Set})$	$(\mathsf{Set}, \mathsf{Type}^p)$	$(\mathsf{Type}^p, \mathsf{Set})$	$(\mathsf{Type}^p, \mathsf{Type}^p)$
	$(\mathsf{Prop}, \mathsf{Prop})$	$(\mathsf{Set}, \mathsf{Prop})$	$(\mathsf{Type}^p, \mathsf{Prop})$	

The systems are presented in a picture as follows.

$$\begin{tikzcd}
& \lambda\mathrm{PROP}\omega \arrow[rr] & & \lambda\mathrm{PRED}\omega \\
\lambda\mathrm{PROP2} \arrow[rr] \arrow[ur] & & \lambda\mathrm{PRED2} \arrow[ur] & \\
& \lambda\mathrm{PROP}\overline{\omega} \arrow[rr] \arrow[uu] & & \lambda\mathrm{PRED}\overline{\omega} \arrow[uu] \\
\lambda\mathrm{PROP} \arrow[rr] \arrow[uu] \arrow[ur] & & \lambda\mathrm{PRED} \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

where an arrow denotes inclusion of one system in another.

Some intuition is required here; it is probably best to keep ΛPRED and its extensions in mind. The sort Prop is to be understood as the class of propositions. The sorts Set and Type^p together form the universe of domains: Domains of the form $A_1 \to \cdots \to A_n \to \alpha$ with α a variable are of type Set, the functional types, while domains of the form $A_1 \to \cdots \to A_n \to \mathsf{Prop}$ are of type $\mathsf{Type}^p (n \geq 0)$ the predicate types. The sort Type^s allows the introduction of variables of type Set, and that is its only purpose. This should be sufficient to understand the first four rules of $\mathcal{R}$ in λPREDω. The other three correspond to the logical rules in the following sense:

$$\begin{array}{rcl}
(\mathsf{Prop}, \mathsf{Prop}) & \rightsquigarrow & \text{implication } (\varphi \supset \psi), \\
(\mathsf{Set}, \mathsf{Prop}) & \rightsquigarrow & \text{quantification over functional types } (\forall x{:}A.\varphi,\ A : \mathsf{Set}), \\
(\mathsf{Type}^p, \mathsf{Prop}) & \rightsquigarrow & \text{quantification over predicate types } (\forall x{:}A.\varphi,\ A : \mathsf{Type}^p).
\end{array}$$

The systems of first, second and higher order proposition logic are defined by just removing the sorts Set and Type^s. Note that the systems λPROP, λPROP2 and λPROPω that we get in this way are just $\lambda\!\to$, $\lambda 2$ and $\lambda\omega$. The two systems λPRED$\overline{\omega}$ and λPROP$\overline{\omega}$ have just been added to make the whole thing into a cube analoguous to the cube of Defintion 4.3.1. They are in formulas-as-types correspondence with two logical systems that we encountered in Definition 2.2.11, namely λPROP$\overline{\omega}$ corresponds to PROP$^\tau$ and λPRED$\overline{\omega}$ corresponds to PRED$^\tau$. These are logics in which there is no order-restriction on the λ-abstraction, but only on the $\forall$-quantification, so the whole higher order language is available but not the possibility to do higher order quantification.

It is not immediately obvious that we can still see the systems of 4.3.5 as being built up in three stages. (First the domains, then the terms and finally the proofs.) It could well be the case that an object expression contains a proof expression or that a domain expression depends on a term. This is however not the case: The systems λPRED, λPRED2 and λPREDω correspond to ΛPRED, ΛPRED2 respectively ΛPREDω in the similar way as $\lambda 2$ and $\lambda\omega$ correspond to F and Fω. We are not going to state this correspondence explicitly, let alone prove it. It is very similar to the work for $\lambda 2$ and $\lambda\omega$ that we did before. Let's only state the basic property that makes the whole correspondence work. (Compare this Proposition with Lemma 4.3.3.)

4.3.6. PROPOSITION. *In λPREDω we have the following. If $\Gamma \vdash M : A$ then $\Gamma_D, \Gamma_T, \Gamma_P \vdash M : A$ with*

- *$\Gamma_D, \Gamma_T, \Gamma_P$ is a sound permutation of Γ,*
- *Γ_D only contains declarations of the form $x : \mathsf{Set}$,*
- *Γ_T only contains declarations of the form $x : A$ with $\Gamma_D \vdash A : \mathsf{Set}/\mathsf{Type}^p$,*
- *Γ_P only contains declarations of the form $x : \varphi$ with $\Gamma_D, \Gamma_T \vdash \varphi : \mathsf{Prop}$,*
- *if $A \equiv \mathsf{Set}/\mathsf{Type}^p$, then $\Gamma_D \vdash M : A$,*
- *if $\Gamma \vdash A : \mathsf{Set}/\mathsf{Type}^p$, then $\Gamma_D, \Gamma_T \vdash M : A$.*

PROOF. By induction on the derivation.⊠

Similar Propositions hold for λPRED and λPRED2. They demistify these PTSs enough to be able to verify the stated correspondences.

As was noticed by [Barendregt 1992], it is also possible to describe a PTS that corresponds to the subsystem PRED^{-f} of PRED (Definition 2.3.7).

4.3.7. DEFINITION. λPRED^{-f} is the PTS with

$$\begin{array}{ll}
\mathcal{S} & \mathsf{Prop}, \mathsf{Set}, \mathsf{Fun}, \mathsf{Type}^p, \mathsf{Type}^s, \\
\mathcal{A} & \mathsf{Prop} : \mathsf{Type}^p, , \mathsf{Set} : \mathsf{Type}^s, \\
\mathcal{R} & (\mathsf{Set}, \mathsf{Set}, \mathsf{Fun}), (\mathsf{Set}, \mathsf{Fun}, \mathsf{Fun}), (\mathsf{Set}, \mathsf{Type}^p), \\
 & (\mathsf{Prop}, \mathsf{Prop}), (\mathsf{Set}, \mathsf{Prop})
\end{array}$$

The idea is that Set contains only basic domains (B of PRED^{-f}) and Fun contains the functional domains ((F of PRED^{-f}). Quantification is only possible over types in Set. The system λPRED^{-f} is not really a subsystem of λPRED, but only via the morphism that maps Set and Fun to Set. We have a Proposition like 4.3.6 to prove in detail that the formulas-as-types embedding from PRED^{-f} to λPRED^{-f} is an isomorphism.

We have seen that many of the logical systems of Chapter 2.1 are in one-to-one correspondence with a PTS_β. To show such a correspondence one has to make two steps: First define a typed λ calculus 'as close as possible' to the original logic and formalize the formulas-as-types embedding à la Howard. (This has been done in detail for the system PRED in Chapter 3.1, where we defined ΛPRED and the formulas-as-types embedding from PRED to ΛPRED.) Then show that the intermediate typed λ calculus is the same as the PTS_β that we want the logic to correspond with. (This has been done in detail for the intermediate systems F and Fω, that correspond to $\lambda 2$ (= λPROP2), respectively $\lambda\omega$ (= λPROPω)). For the systems PRED2, PREDω and PRED^{-f}, we have only given the corresponding PTS_β without detailed proof, which is very similar to the proof for the other cases. We can depict the correspondences in a picture as follows, where $\simeq$ denotes a correspondence and $\boxed{\simeq}$ denotes a correspondence that we have verified in great detail.

$$
\begin{array}{rclcl}
\text{PRED} & \boxed{\simeq} & \Lambda\text{PRED} & \simeq & \lambda\text{PRED}, \\
\text{PRED2} & \simeq & \Lambda\text{PRED2} & \simeq & \lambda\text{PRED2}, \\
\text{PRED}\omega & \simeq & \Lambda\text{PRED}\omega & \simeq & \lambda\text{PRED}\omega, \\
\text{PROP} & \simeq & \text{ST}\lambda & \simeq & \lambda{\rightarrow}\,(=\lambda\text{PROP}), \\
\text{PROP2} & \simeq & \text{F} & \boxed{\simeq} & \lambda 2\,(=\lambda\text{PROP2}), \\
\text{PROP}\omega & \simeq & \text{F}\omega & \boxed{\simeq} & \lambda\omega\,(=\lambda\text{PROP}\omega), \\
\text{PRED}^{-f} & \simeq & & \lambda\text{PRED}^{-f}. &
\end{array}
$$

For most of the other logical systems of Chapter 2.1 one can also define corresponding PTS_βs. We have not done this here: Most of the times the definition becomes a hack without any intuitive meaning, so we don't see this as a very useful operation.

4.3.3. Morphisms between Pure Type Systems

The reason for introducing the cube of logical Pure Type Systems (Definition 4.3.5) is to formalise the embedding of logics into the typed lambda calculi of the cube, and especially the Calculus of Constructions (λPω.) This was also the original motive for Berardi to define these systems: To formalise the practical use of λPω as a system of higher order predicate logic and to better understand the use of λPω as a higher order predicate logic. We come to speak about λPω and its relation to PREDω in more detail later. At this point we just want to treat the interpretation of logics in the systems of the Barendregt's cube by defining a mapping of the cube of logical systems into the Barendregt's cube. This mapping is sometimes referred to as the formulas-as-types embedding (or even isomorphism), but we feel that it is more appropriate to use that terminology for the transition from 'real' logical systems to typed lambda calculi.

4.3.8. Definition. The *collapsing mapping from the logic cube to the Barendregt's cube* is the PTS-morphism H given by

$$\begin{array}{rcl} H(\mathsf{Prop}) & = & \star, \\ H(\mathsf{Set}) & = & \star, \\ H(\mathsf{Type}^p) & = & \Box, \\ H(\mathsf{Type}^s) & = & \Box. \end{array}$$

It is easy to verify that for each corner of the cube, H is a PTS-morphism from the system in the logic cube to the system in the Barendregt's cube. The question arises whether the mapping is complete, especially with respect to the inhabitation of propositions. One of the nice things of doing logic in for example $\lambda\mathrm{P}\omega$, is that domains of the logic and propositions are treated in exactly the same way. This opens up a wide range of new possibilities (like the possibility to define domains that represent inductive data types.) On the other hand it is not so obvious that all this is still sound. We shall see that in the broadest sense this operation is not sound, i.e. the collapsing mapping is not complete, while in a more narrow sense, things are not that bad. More about this in Chapter 6.1

To end this section we want to give a different Pure Type System that corresponds to PREDω that is more intuitive then λPREDω. It can be seen as a direct formulas-as-types formalisation of PREDω, using the fact that in PREDω there is no reason to distinguish between functional types and predicate types, as was done in λPREDω. (See also Definition 3.2.19.) On the other hand this alternative version can also be obtained by defining the system AUT-HOL in a PTS-format. (AUT-HOL was defined in 3.3.11 by applying ideas from the Automath systems AUT-4 to the system AUT-68.) We already pointed out the correspondence between AUT-HOL and ΛPREDω in Proposition 3.3.14.

4.3.9. Definition. The typed lambda calculus λHOPL is the PTS with

$$\begin{array}{rcl} \mathcal{S} & = & \{\mathsf{Prop}, \mathsf{Type}, \mathsf{Type}'\}, \\ \mathcal{A} & = & \mathsf{Prop} : \mathsf{Type}, \mathsf{Type} : \mathsf{Type}', \\ \mathcal{R} & = & (\mathsf{Type}, \mathsf{Type}), \\ & & (\mathsf{Prop}, \mathsf{Prop}), (\mathsf{Type}, \mathsf{Prop}) \end{array}$$

The meaning of the components of the system should be clear from the intended correspondence with PREDω. Prop is the sort of formulas, Type is the sort of domains and the sort Type$'$ is just there to be able to introduce variables of type Type. (These variables are to be the basic domains of the logic.) There is a heavy overloading of symbols: $\Pi x{:}A.B$ stands for logical implication ($\supset$) if A and B are both propositions (of type Prop), for universal quantification ($\forall_A$) if A is a type and B a proposition ($A{:}\mathsf{Type}, B{:}\mathsf{Prop}$) and it stands for the domain $A{\rightarrow}B$ if both A and B are types (of type Type.) Again it is not immediately obvious that λHOPL can be seen as being built up in three stages. (First the domains, then the objects and finally the proofs.) That this is still the case is stated in the following proposition, which is the λHOPL equivalent of Proposition 4.3.6.

4.3.10. PROPOSITION. *We work in* λHOPL. *If* $\Gamma \vdash M : A$ *then* $\Gamma_D, \Gamma_T, \Gamma_P \vdash M : A$ *with*

- $\Gamma_D, \Gamma_T, \Gamma_P$ *is a sound permutation of* Γ,
- Γ_D *only contains declarations of the form* $x : \mathsf{Type}$,
- Γ_T *only contains declarations of the form* $x : A$ *with* $\Gamma_D \vdash A : \mathsf{Type}$,
- Γ_P *only contains declarations of the form* $x : \varphi$ *with* $\Gamma_D, \Gamma_T \vdash \varphi : \mathsf{Prop}$,
- *if* $A \equiv \mathsf{Type}$ *, then* $\Gamma_D \vdash M : A$,
- *if* $\Gamma \vdash A : \mathsf{Type}$, *then* $\Gamma_D, \Gamma_T \vdash M : A$.

The Proposition states (among other things) that the domains (terms of type Type) are just built up from domain-variables using Π, so no object- or proof-variables occur as subterms, so the domains are as in λHOPL. Further it states that the terms of the object-language are formed from the object-variables by λ-abstraction and application and (for terms of type Prop) by Π, so they don't contain proof-variables: $\Pi x{:}\varphi.\psi$ ($\varphi, \psi : \mathsf{Prop}$) denotes $\varphi \supset \psi$, the logical implication.

As an application of the notion of PTS-morphism and also to fully justify the two systems λHOPL and λPREDω in terms of each other, we prove that λPREDω and λHOPL are in a sense the same system.

4.3.11. PROPOSITION. *There is a* PTS*-morphism* G *from* λPREDω *to* λHOPL *and a derivability-preserving map* F *from* λHOPL *to* λPREDω *such that* $F \circ G = \mathrm{Id}$ *and* $G \circ F = \mathrm{Id}$.

PROOF. Take for $G : \lambda\text{PRED}\omega \to \lambda\text{HOPL}$ the PTSmorphism

$$\begin{array}{rcl} G(\mathsf{Prop}) & = & \mathsf{Prop}, \\ G(\mathsf{Set}) & = & \mathsf{Type}, \\ G(\mathsf{Type}^p) & = & \mathsf{Type}, \\ G(\mathsf{Type}^s) & = & \mathsf{Type}'. \end{array}$$

and for $F : \lambda\text{HOPL} \to \lambda\text{PRED}\omega$ first define the mapping F from $\mathsf{Term}(\lambda\text{HOPL}) \setminus \{\mathsf{Type}'\}$ to $\mathsf{Term}(\lambda\text{PRED}\omega)$ by

$$\begin{array}{rcl} F(x) & = & x, (x \text{ a variable}), \\ F(\mathsf{Prop}) & = & \mathsf{Prop}, \\ F(\mathsf{Type}) & = & \mathsf{Set}, \end{array}$$

and further by induction on the structure of the terms. G, being a PTS morphism, preserves derivations. F preserves substitution and β-equality and F extends to contexts straightforwardly by defining

$$F(x_1{:}A_1, \ldots, x_n{:}A_n) := x_1{:}F(A_1), \ldots, x_n{:}F(A_n).$$

(The sort Type' does not appear in a context of λHOPL.) Now we extend F to derivable judgements of λHOPL by defining

$$\begin{array}{rcl}
F(\Gamma \vdash M : A) & = & F(\Gamma) \vdash F(M) : F(A), \text{ if } A \neq \mathsf{Type}, \mathsf{Type}', \\
F(\Gamma \vdash M : \mathsf{Type}) & = & F(\Gamma) \vdash F(M) : \mathsf{Set}, \text{ if } M \equiv \cdots \rightarrow \alpha, (\alpha \text{ a variable}), \\
F(\Gamma \vdash M : \mathsf{Type}) & = & F(\Gamma) \vdash F(M) : \mathsf{Type}^p, \text{ if } M \equiv \cdots \rightarrow \mathsf{Prop}, \\
F(\Gamma \vdash \mathsf{Type} : \mathsf{Type}') & = & F(\Gamma) \vdash \mathsf{Set} : \mathsf{Type}^s.
\end{array}$$

Now F is a PTS mapping in the sense of Definition 4.2.5. By easy induction one proves that F preserves derivations. Also $F(G(\Gamma \vdash M : A)) = \Gamma \vdash M : A$ and $G(F(\Gamma \vdash M : A)) = \Gamma \vdash M : A$. ⊠

We feel that the correspondence between PREDω and λHOPL is more intuitive then the one between λPREDω and PREDω. A disadvantage of presenting higher order predicate logic as λHOPL is that we can not find e.g. second order predicate logic as a subsystem by an easy restriction on the rules: For the rules there is no distinction between the basic domains and the domain Prop. Further λHOPL doesn't allow a straightforward syntactical description of the formulas-as-types embedding of higher order predicate logic into CC. (λPREDω does, as we saw in Definition 4.3.8) In the following we therefore also look at the system λPREDω.

4.3.4. Inconsistent Pure Type Systems

Inconsistency is not really a property of a PTS as such, but it depends on a interpretation that has been given to the different parts of it. One could say that a PTS is inconsistent if all closed types of a specific sort that is intended to be the sort of all formulas, are inhabited by a closed term, but that is not always satisfying. In [Coquand and Herbelin 1992], a restriction is made to so called *logical* $\mathsf{PTS}s$: systems that have two specific sorts Prop and Type with the oproperties that $\mathsf{Prop} : \mathsf{Type}$ is an axiom, $(\mathsf{Type}, \mathsf{Prop})$ is a rule, the system is functional and there are no sorts of type Prop. Usually it is obvious which sort is to be understood as the sort of formulas, so we just speak of 'inconsistent PTSs'. One of the inconsistent PTSs we have seen is $\lambda\star$ (which is not a logical PTS). Other ones are the following.

4.3.12. DEFINITION. The system λU^- is defined as follows.

$$\begin{array}{rcl} \mathcal{S} & = & \mathsf{Prop}, \mathsf{Type}, \mathsf{Type}', \\ \mathcal{A} & = & \mathsf{Prop} : \mathsf{Type}, \mathsf{Type} : \mathsf{Type}', \\ \mathcal{R} & = & (\mathsf{Type}, \mathsf{Type}), (\mathsf{Type}', \mathsf{Type}) \\ & & (\mathsf{Prop}, \mathsf{Prop}), (\mathsf{Type}, \mathsf{Prop}) \end{array}$$

The system λU is defined by extending λU^- with the rule $(\mathsf{Type}', \mathsf{Prop})$.

In [Girard 1971] these systems are discussed as logics. They are obtained by extending PREDω with polymorphic domains (system U^-) and with quantification over all domains (together with the polymorphic domains, this forms the system U). As typed lambda calculi they are extensions of λHOPL: $\lambda \mathrm{U}^-$ is λHOPL with the rule $(\mathsf{Type}', \mathsf{Type})$ (polymorphic domains) and λU is $\lambda \mathrm{U}^-$ with $(\mathsf{Type}, \mathsf{Prop})$ (quantification over all domains). For example in $\lambda \mathrm{U}^-$ one has domains like $\Pi A{:}\mathsf{Type}.A{\to}(A{\to}A){\to}A$ (numerals) and $\Pi A{:}\mathsf{Type}.(A{\to}\mathsf{Prop}){\to}\mathsf{Prop}$. In λU one can write down formulas like $\Pi A{:}\mathsf{Type}.\Pi P{:}A{\to}\mathsf{Prop}.\Pi x{:}A.Px{\to}Px$.

It is not so difficult to see that the extension of higher order predicate logic with just quantification over all domains is consistent and conservative over PREDω.

4.3.13. THEOREM. *Both λU^- and λU are inconsistent, i.e. in both systems there is a term M with*

$$\vdash M : \bot (\equiv \Pi\alpha{:}\mathsf{Prop}.\alpha)$$

PROOF. For λU the proof is in [Girard 1972]. A good explanation of it and a discussion of applications of the proof to other type systems can be found in [Coquand 1986]. This fact has become known as Girard's paradox, especially in its application to the system $\lambda\star$. The proof for λU^- is in [Coquand 199+]. It internalises Reynold's argument that there are no set-theoretic models of the polymorphic lambda calculus. ⊠

Using the meta-theory for Pure Type Systems, it is easy to see that in an inconsistent system there are terms that have no normal form. So the normalization property does not hold for λU, λU^- and $\lambda\star$.

That λU is not such a strange system is shown by the fact that we can separate contexts in the system, just like in λHOPL and other systems. That is, we have the following.

4.3.14. PROPOSITION. *We work in λU. If $\Gamma \vdash M : A$ then $\Gamma_D, \Gamma_T, \Gamma_P \vdash M : A$ with*

- *$\Gamma_D, \Gamma_T, \Gamma_P$ is a sound permutation of Γ,*

- Γ_D *only contains declarations of the form* $x : \mathsf{Type}$,
- Γ_T *only contains declarations of the form* $x : A$ *with* $\Gamma_D \vdash A : \mathsf{Type}$,
- Γ_P *only contains declarations of the form* $x : \varphi$ *with* $\Gamma_D, \Gamma_T \vdash \varphi : \mathsf{Prop}$,
- *if* $A \equiv \mathsf{Type}$, *then* $\Gamma_D \vdash M : A$,
- *if* $\Gamma \vdash A : \mathsf{Type}$, *then* $\Gamma_D, \Gamma_T \vdash M : A$.

In Chapter 6.1 we shall see that, if we are a little bit more careful, it is possible to extend higher order logic with polymorphic domains and still have a consistent system.

4.4. Meta theory of Pure Type Systems

In this section we want to treat the meta theory for our different notions of Pure Type System. For the PTS_βs, most of the results that are listed here have already been treated in [Geuvers and Nederhof 1991]. A lot of the proofs in that paper can immediately be extended to the cases for $\mathsf{PTS}_{\beta\eta}$ and $\mathsf{PTS}^s_{\beta\eta}$, but not all. The essential problem is that the Church-Rosser property for $\beta\eta$-reduction does not hold for T (the set of pseudoterms). This is very problematic, not only because CR on T is *the* tool for proving Subject Reduction and Church-Rosser for the typable terms, but also because it makes the whole system $\mathsf{PTS}_{\beta\eta}$ quite suspect: Think of the possibility that A and B are types with $A =_{\beta\eta} B$, but only by means of an expansion-reduction path which passes through the set of non-typable terms. The conversion rule says that the types A and B still have the same inhabitants, but that is of course not what we want.

Having realised ourselves how problematic the absence of the Church-Rosser property for $\beta\eta$-reduction on T is, we are of course going to look for solutions. It should be remarked here that the solutions given in this thesis have some generality, but can not be the final answer. The fact is that we did manage to prove a general property of $\beta\eta$-equality on T that can in practical situations replace CR. However, using this we only managed to prove CR for $\beta\eta$ on well-typed terms for a restricted class of $\mathsf{PTS}_{\beta\eta}$s: The ones that are functional and normalizing. So we have no proof of CR for $\beta\eta$ for a system like $\lambda\star$, although we very strongly believe that it holds, even more so because there are other $\mathsf{PTS}_{\beta\eta}$s that are not normalizing, for which CR for $\beta\eta$ can easily be proved. (So the lack of normalization doesn't seem to be very essential.) It should be possible to find a general proof which works for all $\mathsf{PTS}_{\beta\eta}$s. Further, the dependency of CR for $\beta\eta$ on normalization implies that CR becomes essentially a higher order property (for example for the Calculus of Constructions, for which a normalization proof can not be done in higher order arithmetic.) We feel that this can not be the case (also because for some non-normalizing $\mathsf{PTS}_{\beta\eta}$s the proof of CR for $\beta\eta$ can be done in first order arithmetic.) Having made all these negative comments on the work, we want to stress that there is still enough generality in the proof, especially the part that analyses $\beta\eta$-equality on pseudoterms, that we think it can be an important contribution to a general proof of CR for $\beta\eta$-reduction for arbitrary Pure Type Systems.

4.4.1. Specifying the notions to be studied

We now want to fix some notions and notations that will be studied in the rest of this thesis.

4.4.1. DEFINITION. Let X be a set of pseudoterms closed under $\beta(\eta)$-reduction. We say that X satisfies the *Church-Rosser property for $\beta(\eta)$-reduction*, notation $X \models \mathrm{CR}_{\beta(\eta)}$, or just X *satisfies $CR_{\beta(\eta)}$*, if

$$\forall M, N, P \in X[N \;{}_{\beta(\eta)}\!\twoheadleftarrow M \twoheadrightarrow_{\beta(\eta)} P \Rightarrow \exists Q \in X[N \twoheadrightarrow_{\beta(\eta)} Q \;{}_{\beta(\eta)}\!\twoheadleftarrow P]].$$

We say that X satisfies *Confluence for $\beta(\eta)$-reduction*, notation $X \models \mathrm{CON}_{\beta(\eta)}$, or just X *satisfies $CON_{\beta(\eta)}$*, if

$$\forall M, N \in X[M =_{\beta(\eta)} N \Rightarrow \exists Q \in X[M \twoheadrightarrow_{\beta(\eta)} Q_{\beta(\eta)} \twoheadleftarrow N]].$$

Obviously, for β-reduction

$$X \models \mathrm{CR}_{\beta} \Leftrightarrow X \models \mathrm{CON}_{\beta},$$

But for $\beta\eta$-reduction this is not the case.

4.4.2. DEFINITION. Let X be a set of pseudoterms closed under $\beta(\eta)$-reduction. We say that X satisfies *Strong Normalization for $\beta(\eta)$-reduction*, notation $X \models \mathrm{SN}_{\beta(\eta)}$, or just X *satisfies $SN_{\beta(\eta)}$*, if there are no infinite $\beta(\eta)$-reduction sequences in X.

We could have formulated this property more positively, for example by saying that for all M in X there is an $n \in \mathbb{N}$ such that n is an upperbound to the length of $\beta(\eta)$-reduction sequences starting from M. We have not done so because the first is a bit easier to work with. Most of the proofs of Strong Normalization in this thesis can be redone with the alternative definition.

4.4.2. Analyzing $\beta\eta$-equality on the pseudoterms

In the proof of Church-Rosser we shall relate the $\beta\eta$-reduction on typed terms to the reductions on untyped lambda terms. Properties of reduction and equality on the untyped terms will be used to obtain results about reduction and equality in T. We therefore define an *erasure* mapping from T to Λ and give some properties for it. With this we can prove the so called Key Lemma about $\beta\eta$-equality in T, which will enable us to prove the important meta theoretical properties like UT (Uniqueness of Types) and SR_β for $\mathsf{PTS}_{\beta\eta}$ and SR_η for $\mathsf{PTS}^s_{\beta\eta}$. But first of all we give a proof of postponement of η-reduction in T, a well-known property of $\beta\eta$-reduction in Λ.

Postponement of η-reduction

We prove the postponement of η-reduction for a set of pseudoterms T by an argument similar to the one used in [Barendregt 1984] (Chapter 15) for the untyped lambda calculus. The idea is to mark η-redexes as superscripts inside the terms (as superscript we take the type of the abstracted variablein the η-redex.) In case one is convinced of the fact that postponement of η-reduction holds for T, this paragraph may be skipped.

4.4.3. Definition. The set of *pseudoterms with markers*, T^+ is defined by abstract syntax as

$$\mathsf{T}^+ ::= \mathcal{S} \mid \mathsf{Var} \mid (\Pi\mathsf{Var}{:}\mathsf{T}^+.\mathsf{T}^+) \mid (\lambda\mathsf{Var}{:}\mathsf{T}^+.\mathsf{T}^+) \mid \mathsf{T}^+\mathsf{T}^+ \mid {\mathsf{T}^+}^{\mathsf{T}^+}.$$

The reduction relation on T^+ is β^+, defined by the basic steps

$$\begin{array}{rcl}(\lambda x{:}A.P)Q & \longrightarrow_{\beta^+} & P[Q/x],\\ P^A Q & \longrightarrow_{\beta^+} & PQ,\end{array}$$

and further by induction on the structure of terms, such that it is compatible with application, λ- and Π-abstraction and the superscript operation.

The intended meaning of P^AQ is $(\lambda x{:}A.Px)Q$, a β-redex, so this should indeedreduce to PQ in T^+. We define the two mappings $|\,|^h$ and φ from T^+ to T, the first erasing the superscripts and the second inserting an η redex for a superscript.

4.4.4. Definition. 1. The mapping $|\,|^h : \mathsf{T}^+ \to \mathsf{T}$ is defined by erasing all superscripts,

2. The mapping $\varphi : \mathsf{T}^+ \to \mathsf{T}$ is defined by

$$\varphi(P^A) = \lambda x{:}\varphi(A).\varphi(P)x \ \text{ (for a fresh } x)$$

and further by induction on the structure of the term.

The following are now easily proved (by induction on the structure of terms.)

4.4.5. Lemma. *For $M, N \in \mathsf{T}^+$,*

1. $\varphi(M[N/x]) \equiv \varphi(M)[\varphi(N)/x]$,
2. $\varphi(M) \twoheadrightarrow_\eta |M|^h$.

The following lemma is a formal justification for the definition of β^+-reduction: It shows that φ preserves (β^+-)reductions and $|\,|^h$ reflects (β^+-) reductions.

4.4.6. LEMMA. *For $P, Q \in \mathsf{T}^+$, $M, M' \in \mathsf{T}$,*

1. $P \longrightarrow_{\beta^+} Q \Rightarrow \varphi(P) \longrightarrow_\beta \varphi(Q)$,
2. $P \mapsto^{|\,|^h} M \longrightarrow_\beta M' \Rightarrow \exists P' \in \mathsf{T}[P \twoheadrightarrow_{\beta^+} P' \mapsto^{|\,|^h} M']$.

PROOF. The proof of the first splits into two cases, depending on the type of redex: $P \equiv C[(\lambda x{:}A.B)C]$ or $P \equiv C[B^A C]$. For both of them the required property is easily proved, using for the first case Lemma 4.4.5(1). The proof of the second is by imitating the reduction from M to M' in T^+. Let $M \equiv C[(\lambda x{:}A.Q)S]$, $M' \equiv C[Q[S/x]]$. Then $P \equiv C^o[((\lambda x{:}B.R)^o T)^o]$, where ^{o} denotes a possible superscript and $|B|^h \equiv A$, $|R|^h \equiv Q$ and $|T|^h \equiv S$. Now $P \twoheadrightarrow_{\beta^+} C^o[((\lambda x{:}A.R)T)^o] \longrightarrow_{\beta^+} C^o[R[T/x]]$. So we are done by taking $P' \equiv C^o[R[T/x]]$. ⊠

4.4.7. LEMMA. *For $Q, M, M' \in \mathsf{T}$,*

$$Q \longrightarrow_\eta M \twoheadrightarrow_\beta M' \Rightarrow \exists Q' \in \mathsf{T}[Q \twoheadrightarrow_\beta Q' \twoheadrightarrow_\eta M']$$

PROOF. Let's say that $Q \equiv C[\lambda x{:}A.Nx]$, $M \equiv C[N]$. Now define $P :\equiv C[N^A]$. Then $\varphi(P) \equiv Q$ and $|P|^h \equiv M$, so, by Lemma 4.4.6(2) we find $P' \in \mathsf{T}^+$ such that $P \twoheadrightarrow_{\beta^+} P' \mapsto^{|\,|^h} M'$. By Lemma 4.4.5(2) we find that also $\varphi(P') \twoheadrightarrow_\eta M'$. By Lemma 4.4.6(1) we find that $(Q \equiv)\varphi(P) \twoheadrightarrow_\beta \varphi(P')$.

We are now done by taking $Q' \equiv \varphi(P')$. ⊠

4.4.8. COROLLARY (Postponement of η-reduction). *For $M, N \in \mathsf{T}$,*

$$M \twoheadrightarrow_{\beta\eta} N \Rightarrow \exists Q \in \mathsf{T}[M \twoheadrightarrow_\beta Q \twoheadrightarrow_\eta N].$$

PROOF. It suffices to prove the following property, which is a slight variation of the Lemma: If $Q \twoheadrightarrow_\eta M \twoheadrightarrow_\beta M'$, then $\exists Q' \in \mathsf{T}[Q \twoheadrightarrow_\beta Q' \twoheadrightarrow_\eta M']$. This property follows immediately from the Lemma itself. ⊠

4.4.9. THEOREM. *For $X \subset \mathsf{T}$, X closed under β-reduction, if $X \models SN_\beta$, then $\downarrow_\eta X \models SN_{\beta\eta}$, where $\downarrow_\eta X$ denotes the closure of X under $\twoheadrightarrow_\eta$.*

PROOF. First remark that $\downarrow_\eta X$ is the same as $\downarrow_{\beta\eta} X$ by the postponement of η. Now, an infinite $\beta\eta$-reduction in $\downarrow_\eta X$ yields an infinite β-reduction in X by postponement of η and the fact that there are no infinite η-reductions. So we are done by $X \models \mathrm{SN}_\beta$. (Note that, if we have an effective bound to the number of β-reduction steps to normal form in X, then we can also compute an effective bound to the number of $\beta\eta$-reduction steps in $\downarrow_\eta X$.) ⊠

The Key Lemma for $\beta\eta$-reduction on T

The counterexample of [Nederpelt 1973] shows that, if one tries to prove $\mathrm{CR}_{\beta\eta}$, there is a problem in the types of the λ-abstracted variables. We call these types *domains*.

4.4.10. DEFINITION. Let $M \in \mathsf{T}$. A subterm A of M is a *domain* if it occurs as $\lambda x{:}A$ in M. (So we are not concerned with Π-abstractions.)

The erasure map removes all domains.

4.4.11. DEFINITION. The *erasure map* $|\,| : \mathsf{T} \to \Lambda^{\Pi}$ is defined by induction on the structure of pseudoterms as follows.

$$\begin{array}{rcl} |x| & := & x, \\ |s| & := & s, \\ |\lambda x{:}A.M| & := & \lambda x.|M|, \\ |\Pi x{:}A.B| & := & \Pi x{:}|A|.|B|, \\ |MN| & := & |M||N|. \end{array}$$

Here, Λ^{Π} is Λ extended with the extra variable binder Π and constants s for each $s \in \mathcal{S}$.

4.4.12. REMARK. All the well-known facts (like $\mathrm{CR}_{\beta\eta}$) about $\beta\eta$-reduction in Λ continue to hold for $\beta\eta$-reduction in Λ^{Π}. This can easily be seen by viewing $\Pi x{:}|A|.|B|$ as $G|A|(\lambda x.|B|)$, with G some fixed constant.

If, for $M, M' \in \mathsf{T}$, $|M| \equiv |M'|$, then M and M' have the same 'structure', apart from the domains that may be very different. We therefore give the following definition.

4.4.13. DEFINITION. Let $M, M' \in \mathsf{T}$. If $|M| \equiv |M'|$ and the respective domains in M and M' are all $\beta\eta$-equal, we say that M and M' are *domain-equal*, notation $M \equiv_d M'$.

We have the following proposition, relating reduction in T to reduction in Λ^{Π}.

4.4.14. PROPOSITION. *For M and M' in T,*

$$(1) \quad M \longrightarrow_{\beta} M' \;\Rightarrow\; |M| \longrightarrow_{\beta} |M'| \vee |M| \equiv |M'|,$$

and similar for $\longrightarrow_{\eta}$ and so for $=_{\beta\eta}$. For $M \in \mathsf{T}$, $Q \in \Lambda^{\Pi}$,

$$(2) \quad |M| \longrightarrow_{\beta} Q \;\Rightarrow\; \exists N[M \longrightarrow_{\beta} N \;\&\; |N| \equiv |Q|].$$

The latter doesn't hold in general for $\longrightarrow_{\eta}$, but we do have (for c a variable or sort)

$$(3) \quad |M| \twoheadrightarrow_{\eta} c \;\Rightarrow\; M \twoheadrightarrow_{\eta} c.$$

PROOF. The first is trivial: If the redex is erased by $|\,|$, then $|M| \equiv |M'|$ and otherwise the same redex can still be done in Λ^{Π}, so $|M| \longrightarrow |M'|$. The second is almost trivial; as $|\,|$ only erases domains, a β-redex in $|M|$ is also a β-redex in M, and by evaluating it we find $N \in \mathsf{T}$ with $M \longrightarrow_{\beta} N$ and $|N| \equiv Q$.
That the second is not valid for η is shown by the taking $M \equiv \lambda x{:}\sigma.y(\lambda z{:}Px.z)x$. (This term can even be well-typed in e.g. the Calculus of Constructions: Take $P \equiv \lambda x{:}\sigma.\tau$, $y{:}(\tau{\rightarrow}\tau){\rightarrow}\sigma{\rightarrow}\sigma$. In Lemma 5.2.3 we see that nevertheless, if M is well-typed in a functional normalizing $\mathsf{PTS}_{\beta\eta}$, and M is in $\beta\eta$-nf, then $|M|$ is in $\beta\eta$-nf.)
The third is a corollary of the following more general lemma. ⊠

4.4.15. LEMMA. *Let M and M' be in* T.

$$|M| \twoheadrightarrow_{\eta} Q, Q \text{ contains no } \lambda s \;\Rightarrow \exists N[M \twoheadrightarrow_{\eta} N \;\&\; |N| \equiv Q].$$

PROOF. By induction on the number of λs in $|M|$. First remark that, as Q contains no λs, all the λs in $|M|$ become the λ of an η-redex at some point in the reduction $|M| \twoheadrightarrow_{\eta} Q$. Further note that the only way in which an η-redex can be created in Λ^{Π} is by $\lambda x.M(\lambda y.xy) \longrightarrow_{\eta} \lambda x.Mx$, which implies that the innermost λ in $|M|$ is always an η-redex in $|M|$. If $|M|$ contains only one λ we are easily done. Now suppose that $|M|$ contains $n+1$ λs and that we are already done for terms containing n λs. Take the innermost η redex of $|M|$, say it is $\lambda x.|P|x$, coming from $\lambda x{:}A.Px$ in M. Then $|P|$ does not contain any λ, for if it would this λ would have to be the λ of a redex, which would make $\lambda x.|P|x$ not innermost. This implies that $\lambda x{:}A.Px$ is also an η-redex in M. So we can apply IH to the term obtained by contracting the η-redex $\lambda x{:}A.Px$ in M and we are done. ⊠

The following is an immediate corollary of the counterexample to $\mathrm{CR}_{\beta\eta}$ on T.

4.4.16. LEMMA (Domain Lemma). *If $C[\lambda x{:}A.M]$ and B are in* T *(i.e. C is a pseudoterm with subterm $\lambda x{:}A.M$), then*

$$C[\lambda x{:}A.M] =_{\beta\eta} C[\lambda x{:}B.M]$$

PROOF.

$$\begin{tikzcd}
 & C[\lambda y{:}B.(\lambda x{:}A.M)y] \arrow[dl, "\eta"'] \arrow[dr, "\beta"] & \\
C[\lambda x{:}A.M] & & C[\lambda x{:}B.M]
\end{tikzcd}$$

where y is some variable not occurring free in A or M. ⊠

First some notation: For $D \in \mathsf{T}$ and $M \in \mathsf{T}$, $M^D \in \mathsf{T}$ is the pseudoterm obtained by replacing all domains in M by D. For $D \in \mathsf{T}$ and $t \in \Lambda^\Pi$, $t^{+D} \in \mathsf{T}$ is the pseudoterm obtained by adding D as domain to every λ abstraction in t. (So for example $(\lambda x.x)^{+D}$ is $\lambda x{:}D.x$.)

4.4.17. Corollary. *For A and B pseudoterms,*

$$|A| =_{\beta\eta} |B| \Rightarrow A =_{\beta\eta} B.$$

Proof. Let $|A| =_{\beta\eta} |B|$, so by Church-Rosser $|A| \downarrow_{\beta\eta} |B|$, say $|A| \twoheadrightarrow_{\beta\eta} t_{\beta\eta} \twoheadleftarrow |B|$. Take for D some closed pseudoterm (or fresh variable), then we have the following diagram. (The $=_{\beta\eta}$ are an immediate consequence of Lemma 4.4.16.)

$$\begin{tikzcd}
A \arrow[r, phantom, "=_{\beta\eta}"] & A^D \arrow[dr, "\beta\eta"'] & & B^D \arrow[dl, "\beta\eta"] \arrow[r, phantom, "=_{\beta\eta}"] & B \\
& & t^{+D} & &
\end{tikzcd}$$

So $A =_{\beta\eta} B$. $\boxtimes$

4.4.18. Lemma (Key Lemma). *Let c be a variable or a sort.*

1. $cP_1 \ldots P_n =_{\beta\eta} Q \Rightarrow Q \twoheadrightarrow_\beta \lambda\vec{y}{:}\vec{A}.cQ_1 \ldots Q_n\vec{R}$, *with* $Q_i =_{\beta\eta} P_i\,(1 \leq i \leq n)$ *and $\vec{R}$ and $\vec{y}$ are of the same length with* $\vec{R} \twoheadrightarrow_\eta \vec{y}$.

2. $\Pi x{:}P_1.P_2 =_{\beta\eta} Q \Rightarrow Q \twoheadrightarrow_\beta \lambda\vec{y}{:}\vec{A}.(\Pi x{:}Q_1.Q_2)\vec{R}$, *with* $P_i =_{\beta\eta} Q_i\,(i = 1, 2)$ *and $\vec{R}$ and $\vec{y}$ are of the same length with* $\vec{R} \twoheadrightarrow_\eta \vec{y}$.

Proof. We only prove the first, since the proof of the second is totally similar. For reasons of readability we adapt here the convention to use capitals for pseudoterms and small characters for elements of Λ^Π.
Let $cP_1 \ldots P_n$ and Q be as in the first case of the lemma. By $\mathrm{CR}_{\beta\eta}$ on Λ^Π we find $t_1, \ldots, t_n \in \Lambda^\Pi$ with $c|P_1| \ldots |P_n| \twoheadrightarrow_{\beta\eta} ct_1 \ldots t_n$ and $|Q| \twoheadrightarrow_{\beta\eta} ct_1 \ldots t_n$. Using postponement of η-reduction, we find that $|Q| \twoheadrightarrow_\beta \lambda\vec{y}.cq_1 \ldots q_n\vec{r} \twoheadrightarrow_\eta ct_1 \ldots t_n$. (Doing as many β-reductions as possible, i.e. we β-reduce all the η-redexes that overlap with a β-redexe. More precisely, if $(\lambda x.Mx)N \longrightarrow_\eta MN$ or $\lambda x.(\lambda z.N)x \longrightarrow_\eta \lambda z.N$ is one of the η-reductions from $\lambda\vec{y}.cq_1 \ldots q_n\vec{r}$ to $ct_1 \ldots t_n$, then we do it already as a β-reduction step.) So $\vec{y}$ and $\vec{r}$ are of the same length. By 4.4.14 we find a term $\lambda\vec{y}{:}\vec{A}.cQ_1 \ldots Q_n\vec{R}$ with $Q \twoheadrightarrow_\beta \lambda\vec{y}{:}\vec{A}.cQ_1 \ldots Q_n\vec{R}$ and $|\lambda\vec{y}{:}\vec{A}.cQ_1 \ldots Q_n\vec{R}| \equiv$

$\lambda\vec{y}.cq_1\ldots q_n\vec{r}$. The situation is as follows.

$$\begin{tikzcd}
cP_1\ldots P_n \arrow[r, mapsto] & c|P_1|\ldots|P_n| \arrow[d, twoheadrightarrow, "\beta\eta"] & {|Q|} \arrow[dl, twoheadrightarrow, "\beta\eta"'] \arrow[d, twoheadrightarrow, "\beta"] & Q \arrow[l, mapsto] \arrow[d, twoheadrightarrow, "\beta"] \\
& ct_1\ldots t_n & \lambda\vec{y}.cq_1\ldots q_n\vec{r} \arrow[l, twoheadrightarrow, "\eta"] & \lambda\vec{y}{:}\vec{A}.cQ_1\ldots Q_n\vec{R} \arrow[l, mapsto]
\end{tikzcd}$$

Now $\vec{R} \longrightarrow_\eta \vec{y}$ follows from $\vec{r} \twoheadrightarrow_\eta \vec{y}$ and Proposition 4.4.14(3). We also have $|Q_i| =_{\beta\eta} |P_i|$ (for $1 \leq i \leq n$), so, by Corollary 4.4.17 we have $Q_i =_{\beta\eta} P_i$ ($1 \leq i \leq n$) and we are done.[1]$\boxtimes$

There is a generalisation of the Key Lemma to include terms that begin with a λ abstraction. We give it for technical completeness.

4.4.19. Lemma (General Key Lemma). *Let c be a variable or a sort.*

1. $\lambda z_1{:}A_1.\ldots\lambda z_p{:}A_p.cP_1\ldots P_n =_{\beta\eta} Q \Rightarrow Q \twoheadrightarrow_\beta \lambda z_1{:}B_1.\ldots\lambda z_q{:}B_q.cQ_1\ldots Q_m$, *with $n+q = m+p$ and $P_1,\ldots,P_n,z_q,\ldots,z_1$ and $Q_1,\ldots,Q_m,z_p,\ldots z_1$ are pairwise $\beta\eta$-convertible.*
2. $\lambda\vec{z}{:}\vec{A}.\Pi x{:}P_1.P_2 =_{\beta\eta} Q \Rightarrow Q \twoheadrightarrow_\beta \lambda\vec{z}{:}\vec{B}.\lambda\vec{y}{:}\vec{C}.(\Pi u{:}Q_1.Q_2)\vec{R}$, *with $P_i =_{\beta\eta} Q_i\,(i=1,2)$ and $\vec{R} \twoheadrightarrow_\eta \vec{y}$.*

Proof. The proof is quite similar to the proof of the Key Lemma. Again we only treat the first case because it is the most difficult one of the two. Using the properties of the untyped labda calculus we now get the following picture. (Notation: $\vec{z}'$ denotes $z_1,\ldots,z_p$, $\vec{z}''$ denotes $z_1,\ldots,z_q$.)

$$\begin{tikzcd}
\lambda\vec{z}'{:}\vec{A}'.cP_1\ldots P_n \arrow[r, mapsto] & \lambda\vec{z}'.c|P_1|\ldots|P_n| \arrow[d, twoheadrightarrow, "\beta\eta"] & {|Q|} \arrow[dl, twoheadrightarrow, "\beta\eta"'] \arrow[d, twoheadrightarrow, "\beta"] & Q \arrow[l, mapsto] \arrow[d, twoheadrightarrow, "\beta"] \\
& \lambda\vec{z}.ct_1\ldots t_r & \lambda\vec{z}''.c\vec{q} \arrow[l, twoheadrightarrow, "\eta"] & \lambda\vec{z}''{:}\vec{B}''.c\vec{Q} \arrow[l, mapsto]
\end{tikzcd}$$

where $\vec{z}$ is $z_1,\ldots z_s$ for some $s \leq p,q$. First, we can conclude from this that $q-s = m-r$ and $p-s = n-r$ and hence $n+q = m+p$. Further, this means that for $r < i \leq n$, $|P_i| \twoheadrightarrow_\eta z_{s+i-r}$ and for $r < i \leq m$, $|Q_i| \twoheadrightarrow_\eta z_{s+i-r}$. Just as in the Key Lemma, we use Corollary 4.4.17 to conclude that $P_1,\ldots,P_n,z_q,\ldots,z_1$ and $Q_1,\ldots,Q_m,z_p,\ldots z_1$ are pairwise $\beta\eta$-equal and we are done. $\boxtimes$

[1] The Lemma can also be proved by induction on the length of the reduction-expansion path from $cP_1\ldots P_n$ to Q, as was suggested to us by B. Werner. This does not change the proof in an essential way; we think that the proof above explains the idea better.

4.4.3. A list of properties for Pure Type Systems

At those points in the text where essential use of specific meta theory is being made, we refer to the relevant lemmas and propositions, so this paragraph may be skipped for now.

In the following we let $\zeta = \lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$ be an arbitrary PTS. If we do not make explicit reference to the PTS, we always refer to this generic system ζ. If the lemma or proposition only holds for a specific notion of PTSs or for a specific subset of the class of all PTSs, this will be explicitly mentioned. So, the generic case is that a lemma or proposition holds for all three notions of PTS and also that the given (sketched) proof works for all three cases.

As remarked, we treat terms modulo α-equivalence, so, for example $\lambda x{:}A.y$ and $\lambda z{:}A.y$ are the same terms (for different x, y and z.) This makes that, for $x \notin \mathrm{FV}(B)$,

$$x{:}A, y{:}B \vdash \lambda x{:}A.y : \Pi x{:}A.B$$

is derivable, whereas it is not without α-conversion. Also variables that are free in a typable term are in a sense bound by a declaration in the context. For those variables we also have a notion of α-conversion that we call 'replacement' and that is provable, as is shown by the following lemma.

4.4.20. Replacement Lemma. For $\Gamma_1, x{:}A, \Gamma_2$ a context, M and B terms and y a fresh variable that is not bound in M or B,

$$\Gamma_1, x{:}A, \Gamma_2 \vdash M : B \;\;\Rightarrow\;\; \Gamma_1, y{:}A, \Gamma_2[y/x] \vdash M[y/x] : B[y/x]$$
by a derivation with the same underlying tree,

where the *underlying tree* of a derivation is the labelled tree that is found by removing from the derivation everything but the names of the applied rules (at every node.)

The lemma says that the names of the declared variables in the context really don't matter and we may assume them to be different from any of the bound variables. The importance of this lemma is illustrated by the fact that now, if we do some proof by induction on the derivation and we want to handle the case that the last rule was (streng), we may take for the variable that has been removed just any fresh variable. (So the lemma implies that the name of the removed variable doesn't matter.)

Proof. By induction on the derivation of $\Gamma_1, x{:}A, \Gamma_2 \vdash M : B$. The only interesting case is when the last rule is (streng) and the variable that has been removed is y, say

$$\frac{\Gamma_1, x{:}A, \Gamma_2, y{:}C, \Gamma_3 \vdash M : B}{\Gamma_1, x{:}A, \Gamma_2, \Gamma_3 \vdash M : B}\; y \notin \mathrm{FV}(\Gamma_3, M, B)$$

Then by IH $\Gamma_1, x{:}A, \Gamma_2, z{:}C, \Gamma_3 \vdash M : B$ is derivable with a derivation with the same underlying tree (for z an arbitrary fresh variable.) So, again by IH, $\Gamma_1, y{:}A, \Gamma_2[y/x], z{:}C[y/x], \Gamma_3[y/x] \vdash M[y/x] : B[y/x]$ is derivable with a derivation with the same underlying tree. Now we are done by one application of the rule (streng) to remove the declaration $z{:}C[y/x]$. ⊠

Another basic property, that is especially important and handy when it comes to proving meta theory and which was first remarked by Randy Pollack is the following.

4.4.21. Lemma (Restricted Weakening). *If $\Gamma \vdash M : A$ is derivable, we may assume the derivation of $\Gamma \vdash M : A$ to contain only applications of the rule (weak) that are of the following form:*

$$(\textit{weak})\ \frac{\Gamma \vdash A : s \quad \Gamma \vdash c : B}{\Gamma, x{:}A \vdash c : B}\ \ \textit{c a variable or a sort, x fresh}$$

i.e. the weakening rule is only applied to typings of variables and sorts.

The proof of the property for PTS_β and $\mathsf{PTS}_{\beta\eta}$ is quite straightforward. We give it below. For $\mathsf{PTS}^s_{\beta\eta}$, the proof is more complicated. For that case the property will be proved later, as a corollary to the more general Sublemma 4.4.25 (that also implies the Thinning Lemma 4.4.24.)

Proof. (For PTS_β and $\mathsf{PTS}_{\beta\eta}$) The proof is by induction on the derivation. All cases except for the last rule being (weak) are easy. In case the last rule is (weak), say

$$(\text{weak})\ \frac{\Gamma \vdash A : s \quad \Gamma \vdash M : B}{\Gamma, x{:}A \vdash M : B}\ \ x \text{ fresh}$$

we find by IH that $\Gamma \vdash A : s$ and $\Gamma \vdash M : B$ are provable with the restricted form of weakening rule as described in the lemma. Now we are going to make some small alterations in the derivation tree of $\Gamma \vdash M : B$ to turn it into a derivation tree of $\Gamma, x{:}A \vdash M : B$ with restricted weakening rule. The alterations are as follows: Go up in the tree to the place where the context Γ is created. So, if $\Gamma \equiv \Gamma', y{:}C$ we go to the places where Γ' is extended to Γ. This is done by a (var) rule or a restricted (weak) rule, so we have either

$$(\text{var})\ \frac{\Gamma' \vdash C : s'}{\Gamma', y{:}C \vdash y : C}$$

or

$$(\text{weak})\ \frac{\Gamma' \vdash C : s' \quad \Gamma \vdash c : E}{\Gamma', y{:}C \vdash c : E}$$

In the first case we change the derivation by inserting

$$\dfrac{\dfrac{\Gamma' \vdash C : s'}{\Gamma', y{:}C \vdash y : C} \quad \Gamma \vdash A : s}{\Gamma, x{:}A \vdash y : C}$$

and replacing Γ by $\Gamma, x{:}A$ downwards. In the second case we change the derivation by inserting

$$\dfrac{\dfrac{\Gamma' \vdash C : s' \quad \Gamma \vdash c : E}{\Gamma', y{:}C \vdash c : E} \quad \Gamma \vdash A : s}{\Gamma, x{:}A \vdash c : E}$$

and replacing Γ by $\Gamma, x{:}A$ downwards. It is easy to see that these alterations satisfy the requirements. ⊠

It is convenient to have some special notation for derivability in a system with a restricted (weak) rule as in the lemma. We therefore introduce the following.

NOTATION. $\Gamma \vdash^w M : A$ denotes the fact that $\Gamma \vdash M : A$ is derivable with a derivation tree with the weakening rule restricted to typings of variables and sorts:

$$\text{(weak)} \quad \dfrac{\Gamma \vdash A : s \quad \Gamma \vdash c : B}{\Gamma, x{:}A \vdash c : B} \quad c \text{ a variable or a sort, } x \text{ fresh}$$

Consequently, if we talk about a *derivation of* $\Gamma \vdash^w M : A$, we refer to a derivation tree with the restricted weakening rule.

4.4.22. LEMMA (Free variables). *For* $\Gamma = x_1{:}A_1, \ldots, x_n{:}A_n$ *and* $\Gamma \vdash M : B$*, then*

1. $FV(M, B) \subset \{x_1, \ldots, x_n\}$,
2. $\forall i, j \leq n[x_i \equiv x_j \Rightarrow i = j]$.

PROOF. By easy induction on the length of the derivation of $\Gamma \vdash M : B$. ⊠

4.4.23. LEMMA. *For* $\Gamma = x_1{:}A_1, \ldots, x_n{:}A_n \in \mathsf{Context}$,

1. $\Gamma \vdash s : s'$ *for all* $s{:}s' \in \mathcal{S}$,
2. $\Gamma \vdash x_i : A_i$ *for all* $i \leq n$,
3. $x_1{:}A_1, \ldots, x_{i-1}{:}A_{i-1} \vdash A_i : s$ *for some* $s \in \mathcal{S}$.

PROOF. All three by an easy induction on the length of the derivation that shows that Γ is a context (i.e. a derivation of a sequent $\Gamma \vdash A : B$ for some A and B.) ⊠

4.4.24. Thinning Lemma. For Γ and Γ' contexts and M and B pseudoterms,

$$\left.\begin{array}{r}\Gamma' \supseteq \Gamma\\ \Gamma \vdash M : B\end{array}\right\} \Rightarrow \Gamma' \vdash M : B.$$

The proof for PTS_β and $\mathsf{PTS}_{\beta\eta}$ is straightforward. Due to the strengthening rule, the proof is quite difficult for $\mathsf{PTS}^s_{\beta\eta}$. It comes as an easy corollary of the Sublemma 4.4.25, which is an induction loading to prove both Thinning and the Lemma on the restricted use of the weakening rule (4.4.21.)[1]

Proof. (For PTS_β and $\mathsf{PTS}_{\beta\eta}$) The proof is by induction on the derivation. We treat the case of the last rule being the (Π) rule, because it has some interest (just as the case of the (λ) rule, which is similar.)
Say

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

and let $\Gamma' \supseteq \Gamma$. We may assume that $x \notin \mathsf{dom}(\Gamma')$ (by Lemma 4.4.20.) By IH $\Gamma' \vdash A : s$ and hence $\Gamma', x{:}A$ is a context. By applying IH to the second premise we find $\Gamma', x{:}A \vdash B : s'$, so by the ($\Pi$) rule: $\Gamma' \vdash \Pi x{:}A.B : s''$ and we are done. ⊠

4.4.25. Sublemma. *For Γ, Γ' and Δ contexts and M and B terms we have the following.*

$$\left.\begin{array}{r}\Gamma' \supseteq \Gamma\\ x \in \mathsf{dom}(\Gamma') \cap \mathsf{dom}(\Delta) \Rightarrow \mathsf{im}_{\Gamma'}(x) \equiv \mathsf{im}_{\Delta}(x)\\ \Gamma \vdash M : B\end{array}\right\} \Rightarrow \Delta, \Gamma' \setminus \Delta \vdash^w M : B.$$

The Sublemma is only interesting for the system $\mathsf{PTS}^s_{\beta\eta}$ because it has as consequences that Thinning and Restricted Weakening hold for $\mathsf{PTS}^s_{\beta\eta}$. Moreover, the Sublemma for PTS_β and $\mathsf{PTS}_{\beta\eta}$ is a very easy consequence of Thinning and

[1] As was pointed out to us by J. McKinna, it is also possible to prove Thinning and Substitution (Proposition 4.4.26) at once by proving the following Lemma.

$$\left.\begin{array}{r}\Gamma \vdash M : B\\ \Delta \vdash \rho(\Gamma)\end{array}\right\} \Rightarrow \Delta \vdash \rho(M) : \rho(B),$$

where ρ is an arbitrary substitution of pseudoterms for variables, which is straightforwardly extended to a mapping from T to T, and $\Delta \vdash \rho(\Gamma)$ means that $\Delta \vdash \rho(x) : \rho(A)$ for every $x{:}A \in \Gamma$. This Lemma can easily be proved if one adapts the rule (streng) as follows

$$(\text{streng}') \quad \frac{\Gamma_1, x{:}C, \Gamma_2 \vdash M : A \quad \Gamma_1 \vdash C : s}{\Gamma_1, \Gamma_2 \vdash M : A} \quad \text{If } x \notin \mathrm{FV}(\Gamma_2, M, A)$$

This rule is equivalent to (streng), as is easily shown by using Lemma 4.4.22.

Restricted Weakening themselves (which have already been proved): The only thing to do is to show that $\Delta, \Gamma' \setminus \Delta$ in the statement of the Sublemma is indeed a context.

PROOF. (For $\mathsf{PTS}^s_{\beta\eta}$) By induction on the derivation of $\Gamma \vdash M : B$. We treat the cases for the last rule being (var), (weak), (Π) and (streng). (The case for (λ) is similar to (Π) and the cases for (sort), (app) and (conv) are easy, like the case for (weak).)

(var) Say

$$\frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A}$$

and $\Gamma' \supseteq \Gamma, x{:}A$ and Δ are contexts satisfying the requirements of the lemma. Now, $\Gamma' \supseteq \Gamma$, so we can apply IH to $\Gamma \vdash A : s$ to obtain $\Delta, \Gamma' \setminus \Delta \vdash^w A : s$. By an argument similar to the proof of the second case of Lemma 4.4.23 one can show that in general, if $\Gamma \vdash^w P : C$ and $x{:}A \in \Gamma$, then $\Gamma \vdash^w x : A$. Now, in the present situation we have that $x{:}A \in \Delta, \Gamma' \setminus \Delta$ and $\Delta, \Gamma' \setminus \Delta \vdash^w A : s$, so we may conclude $\Delta, \Gamma' \setminus \Delta \vdash^w x : A$ and we are done.

(weak) Say

$$\frac{\Gamma \vdash A : s \quad \Gamma \vdash M : B}{\Gamma, x{:}A \vdash M : B}$$

and $\Gamma' \supseteq \Gamma, x{:}A$ and Δ are contexts satisfying the requirements of the lemma. Now, because of $\Gamma' \supseteq \Gamma$ we can apply IH to $\Gamma \vdash M : B$ to obtain $\Delta, \Gamma' \setminus \Delta \vdash^w M : B$ and we are done.

(Π) Say

$$\frac{\Gamma \vdash B : s_1 \quad \Gamma, x : B \vdash C : s_2}{\Gamma \vdash \Pi x{:}B.C : s_3}$$

and $\Gamma' \supseteq \Gamma$ and Δ are contexts satisfying the requirements of the lemma. Then by IH $\Delta, \Gamma' \setminus \Delta \vdash^w B : s_1$, so $\Delta, \Gamma' \setminus \Delta, x{:}B$ is a context. Also $\Delta, \Gamma' \setminus \Delta, x{:}B \supseteq \Gamma, x{:}B$, so we can apply IH to $\Gamma, x{:}B \vdash C : s_2$ to obtain $\Delta, \Gamma' \setminus \Delta, x{:}B \vdash^w C : s_2$ and we can conclude (by an application of (Π)) that $\Delta, \Gamma' \setminus \Delta \vdash^w \Pi x{:}B.C : s_3$.

(streng) Say

$$\frac{\Gamma_1, x{:}A, \Gamma_2 \vdash M : B}{\Gamma_1, \Gamma_2 \vdash M : B}$$

and $\Gamma' \supseteq \Gamma_1, \Gamma_2$ and Δ are contexts satisfying the requirements of the lemma. Then by IH (using the fact that $\Gamma_1, x{:}A, \Gamma_2 \supseteq \Gamma_1, x{:}A, \Gamma_2$) we get that $\Gamma', (\Gamma_1, x{:}A, \Gamma_2) \setminus \Gamma' \vdash^w M : B$ and hence that $\Gamma', x{:}A$ is a context. Also $\Gamma', x{:}A \supseteq \Gamma_1, x{:}A, \Gamma_2$, so we can apply IH again to obtain $\Delta, (\Gamma', x{:}A) \setminus \Delta \vdash^w M : B$. Now, $x \notin \mathrm{FV}(M, B)$, so $\Delta, \Gamma' \setminus \Delta \vdash^w M : B$, by one application of (streng). ⊠

As corollaries we find proofs of Restricted Weakening (Lemma 4.4.21) and Thinning (Lemma 4.4.24) for $\mathsf{PTS}^s_{\beta\eta}$: For the first take $\Delta = \emptyset$, $\Gamma' = \Gamma$ and for the second take $\Delta = \emptyset$.

4.4.26. Proposition (Substitution). *For $\Gamma_1, x{:}A, \Gamma_2$ a context, M, B and N terms,*

$$\left.\begin{array}{r}\Gamma_1, x{:}A, \Gamma_2 \vdash M : B \\ \Gamma_1 \vdash N : A\end{array}\right\} \Rightarrow \Gamma_1, \Gamma_2[N/x] \vdash M[N/x] : B[N/x].$$

Proof. By induction on the length of the derivation of $\Gamma_1, x{:}A, \Gamma_2 \vdash M : B$, assuming that $\Gamma_1 \vdash N : A$ is derivable. The only case that is really interesting is, when the last rule is (streng), i.e. when we are in the system $\mathsf{PTS}^s_{\beta\eta}$. We also treat the case when the last rule is (app), because some attention has to be given to the substitutions.

(streng) Say

$$\text{(streng)}\quad \frac{(\Gamma_1, x{:}A, \Gamma_2)^{y:C} \vdash M : B}{\Gamma_1, x{:}A, \Gamma_2 \vdash M : B} \quad y \notin \mathrm{FV}(\Delta, M, B)$$

where we use the notation $(\Gamma)^{y:C}$ to denote a context from which one obtains the context Γ by removing the declaration $y{:}C$, and Δ is the tail of the context $(\Gamma_1, x{:}A, \Gamma_2)^{y:C}$, relative to the position of $y{:}C$. Now, if $y{:}C$ is a declaration to the right of $x{:}A$ in $(\Gamma_1, x{:}A, \Gamma_2)^{y:C}$, the required consequence follows easily by applying IH to $(\Gamma_1, x{:}A, \Gamma_2)^{y:C} \vdash M : B$ and $\Gamma_1 \vdash N : A$, and then (streng). If the declaration $y{:}C$ is to the left of $x{:}A$, then

$$\text{(streng)}\quad \frac{(\Gamma_1)^{y:C}, x{:}A, \Gamma_2 \vdash M : B}{\Gamma_1, x{:}A, \Gamma_2 \vdash M : B} \quad y \notin \mathrm{FV}(\Delta, M, B)$$

The IH does not immediately apply, but by Thinning (Lemma 4.4.24), we may conclude that $(\Gamma_1)^{y:C} \vdash N : A$ and hence by IH: $(\Gamma_1)^{y:C}, \Gamma_2[N/x] \vdash M[N/x] : B[N/x]$. Note that $y \notin \mathrm{FV}(N)$, so we can apply (streng) to get $\Gamma_1, \Gamma_2[N/x] \vdash M[N/x] : B[N/x]$ and we are done.

(app) Say

$$\text{(app)}\quad \frac{\Gamma_1, x{:}A, \Gamma_2 \vdash M : \Pi y{:}B.C \quad \Gamma_1, x{:}A, \Gamma_2 \vdash P : B}{\Gamma_1, x{:}A, \Gamma_2 \vdash MP : C[P/y]}$$

Now by IH and (app), $\Gamma_1, \Gamma_2[N/x] \vdash M[N/x]P[N/x] : C[N/x][P[N/x]/y]$. We may assume that $y \notin \mathrm{FV}(\Gamma_1, x{:}A, \Gamma_2)$ (a precise justification of this assumption may be found in the Replacement Lemma, 4.4.20.) Hence $y \notin \mathrm{FV}(N)$ and so we can conclude $C[N/x][P[N/x]/y] \equiv (C[P/y])[N/x]$ and we are done. ⊠

4.4.27. Stripping Lemma. For Γ a context, M, N and R terms, we have the following.

$$
\begin{array}{rrcl}
(i) & \Gamma \vdash s : R,\ s \in \mathcal{S} & \Rightarrow & R = s' \text{ with } s : s' \in \mathcal{A} \text{ for some } s' \in \mathcal{S}, \\
(ii) & \Gamma \vdash x : R,\ x \in \mathsf{Var} & \Rightarrow & R = A \text{ with } x : A \in \Gamma \text{ for some term } A, \\
(iii) & \Gamma \vdash \Pi x{:}M.N : R & \Rightarrow & \Gamma \vdash M : s_1, \Gamma, x{:}M \vdash N : s_2 \text{ and } R = s_3 \\
 & & & \text{with } (s_1, s_2, s_3) \in \mathcal{R} \text{ for some } s_1, s_2, s_3 \in \mathcal{S}, \\
 & \text{For } \mathsf{PTS}_{\beta(\eta)} & & \\
(iv) & \Gamma \vdash \lambda x{:}M.N : R & \Rightarrow & \Gamma, x{:}M \vdash N : B, \Gamma \vdash \Pi x{:}M.B : s \text{ and} \\
 & & & R = \Pi x{:}M.B \text{ for some term } B \text{ and } s \in \mathcal{S}, \\
(v) & \Gamma \vdash MN : R & \Rightarrow & \Gamma \vdash M : \Pi x{:}A.B, \Gamma \vdash N : A \text{ with } R = B[N/x] \\
 & & & \text{for some terms } A \text{ and } B, \\
 & \text{For } \mathsf{PTS}^s_{\beta\eta} & & \\
(iv') & \Gamma \vdash \lambda x{:}M.N : R & \Rightarrow & \Gamma', x{:}M \vdash N : B, \Gamma' \vdash \Pi x{:}M.B : s \text{ and} \\
 & & & R = \Pi x{:}M.B \text{ for some } B, s \in \mathcal{S} \text{ and } \Gamma' \supseteq \Gamma, \\
(v') & \Gamma \vdash MN : R & \Rightarrow & \Gamma' \vdash M : \Pi x{:}A.B, \Gamma' \vdash N : A \text{ with } R = B[N/x] \\
 & & & \text{for some terms } A \text{ and } B \text{ and context } \Gamma' \supseteq \Gamma.
\end{array}
$$

In fact the case (iv′) can be strengthened to (iv) for $\mathsf{PTS}^s_{\beta\eta}$, so (iv) holds generally for all three notions of PTS. But we are only in the position to prove this fact after we have proved the Subject Reduction property for β-reduction (Lemma 4.4.30), which in turn uses Stripping (in the weaker version given in the Lemma above.)

Proof. For PTS_β and $\mathsf{PTS}_{\beta\eta}$ the proof is easy: If $\Gamma \vdash P : R$, we may assume the derivation tree of this judgement to have the restricted form of the weakening rule. We can go up in this derivation tree until we reach the point where the term P has been formed. In doing this we only pass through applications of the conversion rule (so the context Γ remains the same; only the type R is replaced by a convertible one.) At the point where the term P has been formed we distinguish the five different cases above, according to the form of P, and we easily check that the conclusions are satisfied.
For $\mathsf{PTS}^s_{\beta\eta}$ the proof is more complicated because, in going up through the derivation tree of a judgement $\Gamma \vdash^w P : R$, we also pass through applications of (streng), which will extend the context Γ to a context Γ'. So the proofs of (iv′), (v′) and (i) are easy: The method described above, going up in the derivation tree until we reach at the point where the term is formed, works for each of the three cases.

For the proof of (ii) we can apply the same method to arrive at a context $\Gamma' \supseteq \Gamma$ whose last declaration is $x : A$ with $A = R$. The context Γ is obtained from Γ' by removing declarations, but $x : A$ can of course not be one of them, so $x : A \in \Gamma$ and we are done. For the proof of (iii) we apply the method to arrive at a context $\Gamma' \supseteq \Gamma$ for which $\Gamma' \vdash M : s_1$, $\Gamma', x{:}M \vdash N : s_2$ and $\Gamma' \vdash \Pi x{:}M.N : s_3$ with $s_3 = R$. Now the domain of Γ' may be larger than that of Γ, but none of the extra variables occurs free in $\Pi x{:}M.N$ (and we may assume all of them to be different from x), so we can conclude that $\Gamma \vdash M : s_1$ and $\Gamma, x{:}M \vdash N : s_2$ and we are done. ⊠

4.4.28. Correctness of Types Lemma. For Γ a context, M and A terms,

$$\Gamma \vdash M : A \Rightarrow \exists s \in \mathcal{S}[A \equiv s \vee \Gamma \vdash A : s].$$

Proof. The proof can be given by analysing the derivation tree of $\Gamma \vdash^w M : A$, like in the proof of 4.4.27, but also by induction on the derivation of $\Gamma \vdash M : A$. We follow the second option, which gives the shortest proof. The only two cases that have some interest are when the last rule is (app) or (streng).

(app)

$$\frac{\Gamma \vdash P : \Pi x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash PN : B[N/x]}$$

Then $\Gamma \vdash \Pi x{:}A.B : s$ by IH and hence by Stripping (Lemma 4.4.27), $\Gamma, x{:}A \vdash B : s'$ for some $s' \in \mathcal{S}$. Now by Substitution (Proposition 4.4.26), we conclude that $\Gamma \vdash B[N/x] : s'$.

(streng)

$$\frac{\Gamma_1, x{:}A, \Gamma_2 \vdash M : B}{\Gamma_1, \Gamma_2 \vdash M : B}$$

Then by IH $B \equiv s$ or $\Gamma_1, x{:}A, \Gamma_2 \vdash B : s$ for some $s \in \mathcal{S}$, so by one application of (streng), $B \equiv s$ or $\Gamma_1, \Gamma_2 \vdash B : s$ for some $s \in \mathcal{S}$. ⊠

4.4.29. Uniqueness of Types Lemma. For functional PTSs, if Γ is a context, M, C and C' are terms we have

$$\left.\begin{array}{r}\Gamma \vdash M : C\\ \Gamma \vdash M : C'\end{array}\right\} \Rightarrow C = C'.$$

Proof. By induction on the structure of the term M, using Stripping. In case M is a sort or a Π-term, we use the functionality. The only interesting cases are when M is an application term or when we are in a $\mathsf{PTS}^s_{\beta\eta}$ and M is a λ-abstraction or an application. We do the latter case, because it covers all the interesting cases. So let $M \equiv PN$. Then we find by Stripping terms A, A', B and B' and contexts $\Gamma' \supseteq \Gamma$ and $\Gamma'' \supseteq \Gamma$ such that

$$\begin{array}{rcl} \Gamma' & \vdash & P : \Pi x{:}A.B, \\ \Gamma'' & \vdash & P : \Pi x{:}A'.B', \end{array}$$

with $C =_{\beta\eta} B[N/x]$, $C' =_{\beta\eta} B'[N/x]$. By the Replacement Lemma we may assume that $\mathsf{dom}(\Gamma' \setminus \Gamma) \cap \mathsf{dom}(\Gamma'' \setminus \Gamma) = \emptyset$. So we can take Δ to be the union of Γ' and Γ'' and we have

$$\begin{array}{rcl} \Delta & \vdash & P : \Pi x{:}A.B, \\ \Delta & \vdash & P : \Pi x{:}A'.B'. \end{array}$$

Now we can apply IH to conclude that $\Pi x{:}A.B =_{\beta\eta} \Pi x{:}A'.B'$. By the Key Lemma we may conclude from this that $B =_{\beta\eta} B'$ and hence $B[N/x] =_{\beta\eta} B'[N/x]$ and we are done. ⊠

4.4.30. Subject Reduction Lemma for beta (SR_β). For Γ, Γ' contexts, P, P' and D terms,

$$\begin{array}{rcl} \Gamma \vdash P : D \;\&\; P \longrightarrow_\beta P' & \Rightarrow & \Gamma \vdash P' : D, \\ \Gamma \vdash P : D \;\&\; \Gamma \longrightarrow_\beta \Gamma' & \Rightarrow & \Gamma' \vdash P : D. \end{array}$$

Proof. We do the proof for $\mathsf{PTS}^s_{\beta\eta}$; for $\mathsf{PTS}_{\beta\eta}$ and PTS_β the proof is slightly easier because of the stronger version of the Stripping Lemma 4.4.27. The proof of the two statements is done simultaneously, by induction on the derivation of $\Gamma \vdash P : D$, distinguishing cases according to the last rule.
Proof of (i): All cases except for the last rule being (app) are immediate, sometimes by using IH. (For (Π) and (λ), use IH on (ii).) If the last rule is (app), we distinguish subcases according to where the reduction takes place.

Subcase 1

$$\frac{\Gamma \vdash M : \Pi x{:}A.C \quad \Gamma \vdash N : A}{\Gamma \vdash MN : C[N/x]}$$

with $P \equiv MN$ and the reduction is inside M or N. Then we are immediately done by IH.

Subcase 2

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

with $P \equiv (\lambda x{:}A.M)N$ and $P' \equiv M[N/x]$. Then by applying Stripping (4.4.27) to the first premise, we find

$$\begin{array}{c} \Gamma', x{:}A \vdash M : C' \quad (1) \\ \Gamma' \vdash \Pi x{:}A.C' : s (\in \mathcal{S}) \\ \Pi x{:}A.C' = \Pi x{:}B.C \\ \text{with } \Gamma' \supseteq \Gamma. \end{array}$$

So, again by Stripping

$$\begin{array}{c} \Gamma' \vdash A : s_1 \quad (2) \\ \Gamma', x{:}A \vdash C' : s_2 \\ \text{for some } s_1, s_2 \in \mathcal{S}. \end{array}$$

By applying Thinning (4.4.24) to the second premise we find

$$\Gamma' \vdash N : B \quad (3)$$

By the Key Lemma (4.4.18), we conclude from $\Pi x{:}A.C' = \Pi x{:}B.C$ that

$$\begin{array}{c} A = B \quad (4) \\ C' = C. \quad (5) \end{array}$$

So, applying (conv) to (2) and (3), using (4), we get

$$\Gamma' \vdash N : A. \quad (6)$$

Applying Substitution (Proposition 4.4.26) to (6) and (1) we get

$$\Gamma' \vdash M[N/x] : C'[N/x]. \quad (7)$$

By applying Correctness of Types (Lemma 4.4.28) to the first premise, we find $\Gamma \vdash \Pi x{:}B.C : s'$ for some $s' \in \mathcal{S}$, hence $\Gamma' \vdash \Pi x{:}B.C : s'$ and hence by Stripping

$$\Gamma', x{:}B \vdash C : s_2' (\in \mathcal{S}). \quad (8)$$

Now apply Substitution to (3) and (8) to get

$$\Gamma' \vdash C[N/x] : s_2'. \quad (9)$$

Apply (conv) to (7) and (9) (using (5)) to conclude

$$\Gamma' \vdash M[N/x] : C[N/x].$$

The variables that are in the set $\mathsf{dom}(\Gamma') \setminus \mathsf{dom}(\Gamma)$ are not free in $M[N/x], C[N/x]$ or Γ, so they can be removed by consecutive applications of (streng) to obtain

$$\Gamma \vdash M[N/x] : C[N/x]$$

and we are done.

Proof of (ii): All cases can be handled easily by applying IH. In case the last rule is (var) or (weak), also use IH on (i). ⊠

4.4.31. COROLLARY.

$$\Gamma \vdash M : C, C \twoheadrightarrow_\beta C' \Rightarrow \Gamma \vdash M : C'$$

PROOF. Immediate, using Correctness of Types (4.4.28). ⊠

4.4.32. SUBJECT REDUCTION LEMMA FOR ETA (SR_η for PTS_β and $\mathsf{PTS}^s_{\beta\eta}$). For Γ, Γ' contexts, P, P' and D terms,

$$\begin{array}{rcl} \Gamma \vdash P : D \;\&\; P \longrightarrow_\eta P' & \Rightarrow & \Gamma \vdash P' : D \\ \Gamma \vdash P : D \;\&\; \Gamma \longrightarrow_\eta \Gamma' & \Rightarrow & \Gamma' \vdash P : D. \end{array}$$

PROOF. We do the prove for $\mathsf{PTS}^s_{\beta\eta}$. The proof for PTS_β is slightly simpler and follows the same lines. (It uses the fact that (streng) is a derived rule, which will only be shown in 4.4.35.) Simultaneously by induction on the derivation of $\Gamma \vdash P : D$. We treat the proof for $\mathsf{PTS}^s_{\beta\eta}$, because it is the most complicated. The only interesting case is when the last rule is the (*lambda*) rule and we are in the following situation

$$\frac{\Gamma, x{:}A \vdash Mx : B \quad \Gamma \vdash \Pi x{:}A.B : s}{\Gamma \vdash \lambda x{:}A.Mx : \Pi x{:}A.B}$$

with $x \notin \mathrm{FV}(M)$. Then by Stripping (4.4.27) we find

$$\begin{array}{rcl} (\Gamma, x{:}A)' & \vdash & M : \Pi y{:}C.E \\ (\Gamma, x{:}A)' & \vdash & x : C \\ E[x/y] & = & B \end{array}$$

with $(\Gamma, x{:}A)' \supseteq \Gamma, x{:}A$. We may conclude that $A = C$ and hence that $\Pi x{:}A.B = \Pi y{:}C.E$. So

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

and by some applications of (streng) we find

$$\Gamma \vdash M : \Pi x{:}A.B$$

and we are done.

For all other cases the proof follows exactly the proof of SR_β. ⊠

4.4.33. COROLLARY. *For* PTS_β *and* $\mathsf{PTS}^s_{\beta\eta}$ *we have*

$$\Gamma \vdash M : C, C \twoheadrightarrow_\eta C' \Rightarrow \Gamma \vdash M : C'$$

PROOF. Immediate, using Correctness of Types (4.4.28). ⊠

4.4.34. SUBLEMMA. *(for proving that (streng) is a derived rule for* PTS_β*) For* PTS_β*, if* $\Gamma_1, x{:}A, \Gamma_2$ *is a context and* M *and* B *are terms, then*

$$\left.\begin{array}{r}\Gamma_1, x{:}A, \Gamma_2 \vdash M : B\\ x \notin FV(\Gamma_2, M)\end{array}\right\} \Rightarrow \exists B'[B \twoheadrightarrow_\beta B' \;\&\; \Gamma_1, \Gamma_2 \vdash M : B'.$$

Although the property seems to be obviously correct, the proof for the general case is remarkably complicated and requires the introduction of many new notions and definitions. For that reason and because the proof is not ours, we omit it here and refer to [van Benthem Jutting 199+] for details (which is the original source.) The idea of using the above Sublemma to prove that (streng) is a derived rule, first appeared in [Luo 1989], who used it for the system ECC. The author and Nederhof used it (in the joint paper [Geuvers and Nederhof 1991]) to give the proof for *functional* PTS_βs. (For this case the situation is easier because we have Uniqueness of Types.) We shortly repeat that proof here.

PROOF. of the Sublemma for functional PTS_βs.
The proof is by induction on the derivation of $\Gamma_1, x{:}A, \Gamma_2 \vdash M : B$, distinguishing cases according to the last rule. The only interesting cases are when the last rule is (λ), (app) or (conv), so we treat those.

(λ) Say $M \equiv \lambda y{:}C.N$, $B \equiv \Pi y{:}C.D$ and

$$\frac{\Gamma_1, x{:}A, \Gamma_2, y{:}C \vdash N : D \quad \Gamma \vdash \Pi y{:}C.D : s}{\Gamma_1, x{:}A, \Gamma_2 \vdash \lambda y{:}C.N : \Pi y{:}C.D}$$

Then by IH $\Gamma_1, \Gamma_2, y{:}C \vdash N : D'$ for some D' with $D \twoheadrightarrow_\beta D'$.
Also, $\Gamma_1, x{:}A, \Gamma_2 \vdash C : s_1$ is a conclusion of a subderivation of the derivation with conclusion $\Gamma_1, x{:}A, \Gamma_2, y{:}C \vdash N : D$, so by IH $\Gamma_1, \Gamma_2 \vdash C : s_1$.
By Correctness of Types we find that $\Gamma_1, \Gamma_2, y{:}C \vdash D' : s_2$ or $D' \equiv s \in \mathcal{S}$.
In the second case too there is an s_2 such that $D'(\equiv s) : s_2$, because for D there is such s_2 and we have SR_β.
Now, by functionality, the s_1 and s_2 are such that $(s_1, s_2, s) \in \mathcal{R}$ ((s_1, s_2, s) is the rule that justified the formation of $\Pi y{:}C.D$), so we can apply (Π) to conclude $\Gamma_1, \Gamma_2 \vdash \Pi y{:}C.D' : s$ and hence $\Gamma_1, \Gamma_2 \vdash \lambda y{:}C.N : \Pi y{:}C.D'$.

(app) Say $M \equiv NP$, $B \equiv D[P/y]$ and

$$\frac{\Gamma_1, x{:}A, \Gamma_2 \vdash N : \Pi y{:}C.D \quad \Gamma \vdash P : C}{\Gamma_1, x{:}A, \Gamma_2 \vdash NP : D[P/y]}$$

Then by IH, $\Gamma_1, \Gamma_2 \vdash N : \Pi y{:}C'.D'$ and $\Gamma_1, \Gamma_2 \vdash N : C''$ with $C \twoheadrightarrow_\beta C', C''$ and $D \twoheadrightarrow_\beta D'$. By Church-Rosser we find a term C''' such that $C', C'' \twoheadrightarrow_\beta C'''$ and hence (by Corollary 4.4.31) $\Gamma_1, \Gamma_2 \vdash N : \Pi y{:}C'''.D'$ and $\Gamma_1, \Gamma_2 \vdash P : C'''$. We may now conclude that $\Gamma_1, \Gamma_2 \vdash NP : D'[P/y]$ and we are done.

(conv) Say

$$\frac{\Gamma_1, x{:}A, \Gamma_2 \vdash M : C \quad \Gamma \vdash D : s}{\Gamma_1, x{:}A, \Gamma_2 \vdash M : D}\ C = D$$

Then by IH $\Gamma_1, \Gamma_2 \vdash M : C'$ for some C' with $C \twoheadrightarrow_\beta C'$. By Church-Rosser there is a C'' such that $C', D \twoheadrightarrow_\beta C''$. Now $\Gamma_1, \Gamma_2 \vdash M : C''$ and we are done. ⊠

The statement of the Sublemma can be weakened a bit by requiring the B' to be convertible with B (and not necessarily a reduct.) This trivializes the case for the last ruel being (conv), but doesn't make the whole proof really easier: We still need Church-Rosser, functionality and the case for the last rule being (λ) becomes a bit more involved. Moreover it is slightly more work to get Strengthening from the Sublemma.

4.4.35. Strengthening Lemma. for PTS_β. For $\Gamma_1, x{:}A, \Gamma_2$ a context and M and B terms,

$$\left.\begin{array}{r}\Gamma_1, x{:}A, \Gamma_2 \vdash M : B \\ x \notin \mathrm{FV}(\Gamma_2, M, B)\end{array}\right\} \Rightarrow \Gamma_1, \Gamma_2 \vdash M : B.$$

Proof. By the Sublemma we find a B' such that $B \twoheadrightarrow_\beta B'$ and

$$\Gamma_1, \Gamma_2 \vdash M : B'.$$

By Correctness of Types there are two possibilities, $\Gamma_1, x{:}A, \Gamma_2 \vdash B : s$ or $B \equiv s \in \mathcal{S}$. In the second case we are immediately done, because $B \equiv B'$. In the first case we can once again apply the Sublemma to

$$\Gamma_1, x{:}A, \Gamma_2 \vdash B : s$$

to find that

$$\Gamma_1, \Gamma_2 \vdash B : s.$$

Now we are done by one application of (conv). ⊠

4.4.36. Strong Permutation Lemma. for PTS_β and $\mathsf{PTS}^s_{\beta\eta}$.
For $\Gamma_1, x{:}A, y{:}B, \Gamma_2$ a context, M and C terms, with $x \notin \mathrm{FV}(B)$,

$$\Gamma_1, x{:}A, y{:}B, \Gamma_2 \vdash M : C \Rightarrow \Gamma_1, y{:}B, x{:}A, \Gamma_2 \vdash M : C.$$

Proof. The only thing to do is to show that $\Gamma_1, y{:}B, x{:}A, \Gamma_2$ is a legal context if $\Gamma_1, x{:}A, y{:}B, \Gamma_2$ is. (Then we are done by Thinning, 4.4.24.) By Lemma 4.4.23 we know that

$$\Gamma_1, x{:}A \vdash B : s$$

for some $s \in \mathcal{S}$. By Strengthening for PTS_β (Lemma 4.4.35) or by the rule (streng) for $\mathsf{PTS}^s_{\beta\eta}$, we conclude that

$$\Gamma_1 \vdash B : s$$

and hence that $\Gamma_1, y{:}B$ is a legal context. So, by one again using Lemma 4.4.23 and Thinning we derive that $\Gamma_1, y{:}B, x{:}A$ is a legal context. We can repeat this operation of applying Lemma 4.4.23 and Thinning for all declarations in Γ_2 and finally conclude that $\Gamma_1, y{:}B, x{:}A, \Gamma_2$ is a legal context. ⊠

A *weak* form of the Permutation Lemma, which holds for all notions of Pure Type System is the following.

$$\left.\begin{array}{r}\Gamma_1, x{:}A, y{:}B, \Gamma_2 \vdash M : C \\ \Gamma_1 \vdash B : s\end{array}\right\} \Rightarrow \Gamma_1, y{:}B, x{:}A, \Gamma_2 \vdash M : C.$$

The proof is the same as for the proof of the Strong Permutation Lemma, except for the fact that one doesn't need Strengthening because of the second assumption in the statement.

Finally we want to prove two properties that use the syntax with *sorted variables* as it was described in Definition 4.2.9. We prove the Lemmas for injective PTS_βs, which is an unpractical restriction, not so much because of the restriction to injectivity but especially because we don't have the Lemma for $\mathsf{PTS}^s_{\beta\eta}$. Therefore we shall look into this matter again in detail when we study the Calculus of Constructions with $\beta\eta$-conversion. Let us remark here that the following Lemmas are not true if we drop the restriction to injective systems; a counterexample can be found in [Geuvers and Nederhof 1991].

4.4.37. Classification Lemma for injective systems. For s, s' sorts, $s \not\equiv s'$,

$$\begin{array}{rcl} s\text{-}\mathsf{Term} \cap s'\text{-}\mathsf{Term} & = & \emptyset, \\ s\text{-}\mathsf{Elt} \cap s'\text{-}\mathsf{Elt} & = & \emptyset. \end{array}$$

Proof. For the first it suffices to prove the following.

$$\Gamma \vdash M : s,\ \Gamma' \vdash M : s' \Rightarrow s \equiv s'.$$

For the second it suffices to prove the following.

$$\Gamma \vdash M : B : s,\ \Gamma' \vdash M : B' : s' \Rightarrow s \equiv s'.$$

We prove these two statements simultaneously by induction on the structure of terms, using the Church-Rosser property, SR_β and Uniqueness of Types. The proof is not really difficult but still a bit tricky and we therefore give it in quite some detail.

var. If $\Gamma \vdash x : s$, $\Gamma' \vdash x : s'$ and $x \in \mathsf{Var}^{s_0}$ then $x : A \in \Gamma$ with $A \twoheadrightarrow_\beta s$ and $x : A' \in \Gamma'$ with $A' \twoheadrightarrow_\beta s'$ for some A, A'. Furthermore $\Gamma \vdash A : s_0$ and $\Gamma \vdash A' : s_0$ and hence $s : s_0$ and $s' : s_0$ are axioms. Now by injectivity, $s \equiv s'$. For the second statement it suffices to show that, if $\Gamma \vdash x : B : s$ with $x \in \mathsf{Var}^{s_0}$, then $s \equiv s_0$. Now, if $\Gamma \vdash x : B$, then $x : A \in \Gamma$ with $\Gamma \vdash A : s_0$ and $A =_\beta B$. Hence by Church-Rosser, SR_β and Uniqueness of Types, $s \equiv s_0$.

Π-abstr. If $\Gamma \vdash \Pi x{:}A.B : s$ and $\Gamma' \vdash \Pi x{:}A.B : s'$, then $\Gamma \vdash A : s_1$ and $\Gamma, x{:}A \vdash B : s_2$ with $(s_1, s_2, s) \in \mathcal{R}$ and at the same time $\Gamma' \vdash A : s_1'$ and $\Gamma', x{:}A \vdash B : s_2'$ with $(s_1', s_2', s') \in \mathcal{R}$. Now, by IH $s_1 \equiv s_1'$ and $s_2 \equiv s_2'$ and hence $s \equiv s'$ because $\mathcal{R} \subseteq (\mathcal{S} \times \mathcal{S}) \times \mathcal{S}$ is a function. For the second statement we are now easily done, because if $\Gamma \vdash \Pi x{:}A.B : C : s$ and $\Gamma' \vdash \Pi x{:}A.B : C' : s'$, then C and C' reduce both to the same fixed $s_0 \in \mathcal{S}$ (which is found by the argument for the first statement.) Hence $s \equiv s'$ by the fact that $\mathcal{A}$ is a function.

λ-abstr. The first statement is trivially satisfied by the fact that a λ-abstraction can not be an s-Term. For the second statement suppose that $\Gamma \vdash \lambda x{:}A.M : B : s$ and $\Gamma' \vdash \lambda x{:}A.M : B' : s'$. Then $B = \Pi x{:}A.C$ with $\Gamma \vdash A : s_1$, $\Gamma, x{:}A \vdash M : C : s_2$ and $\Gamma \vdash \Pi x{:}A.C : s_3$ ($(s_1, s_2, s_3) \in \mathcal{R}$) and at the same time $B' = \Pi x{:}A.C'$ with $\Gamma' \vdash A : s_1'$, $\Gamma', x{:}A \vdash M : C' : s_2'$ and $\Gamma' \vdash \Pi x{:}A.C' : s_3'$ ($(s_1', s_2', s_3') \in \mathcal{R}$). Now, by IH $s_1 \equiv s_1'$ and $s_2 \equiv s_2'$ and hence $s_3 \equiv s_3'$. Further, by Church-Rosser, SR_β and Uniqueness of Types, $s \equiv s_3$ and $s' \equiv s_3'$ and so $s \equiv s'$.

applic. We first prove the second statement, so let $\Gamma \vdash MN : D : s$ and $\Gamma' \vdash MN : D' : s'$. Then $\Gamma \vdash M : \Pi x{:}A.B : s_3$, $\Gamma \vdash N : A : s_1$, $\Gamma \vdash B[N/x] : s_2$ and $B[N/x] =_\beta D$ ($(s_1, s_2, s_3) \in \mathcal{R}$) and at the same time $\Gamma' \vdash M : \Pi x{:}A'.B' : s_3'$, $\Gamma' \vdash N : A' : s_1'$, $\Gamma' \vdash B'[N/x] : s_2'$ and $B'[N/x] = D'$ ($(s_1', s_2', s_3') \in \mathcal{R}$.) Now, by IH $s_1 \equiv s_1'$ and $s_3 \equiv s_3'$ and hence $s_2 \equiv s_2'$ by injectivity. Also, by Church-Rosser, SR_β and Uniqueness of Types, $s \equiv s_2$ and $s' \equiv s_2'$ and so $s \equiv s'$. For the first statement, if $\Gamma \vdash MN : s$ and $\Gamma' \vdash MN : s'$, then we find by the argument for the second statement a fixed sort s_0 such that $s : s_0$ and $s' : s_0$. So, by injectivity, $s \equiv s'$. ⊠

We can specialize this Lemma a bit further by noticing that in a lot of cases the sort s for which $A \in s$-Elt only depends on the 'innermost symbol' of A, which is always a sort or a variable. Let us first define this notion; we call the innermost symbol of A the *heart* of the term A, notation $\mathsf{h}(A)$.

4.4.38. Definition. The *heart of a pseusdoterm* A, $\mathsf{h}(A)$, is defined by induction on the structure of terms as follows.

$$\begin{array}{rcl}\mathsf{h}(s) &:=& s, \text{ for } s \in \mathcal{S},\\ \mathsf{h}(x) &:=& x, \text{ for } x \in \mathsf{Var},\\ \mathsf{h}(\Pi x{:}B.C) &:=& \mathsf{h}(C),\\ \mathsf{h}(\lambda x{:}B.M) &:=& \mathsf{h}(M),\\ \mathsf{h}(MN) &:=& \mathsf{h}(M).\end{array}$$

4.4.39. Lemma. *For an injective* PTS_β *with all rules of the form* (s_1, s_2) *(i.e.* $(s_1, s_2, s_3) \in \mathcal{R} \Rightarrow s_2 \equiv s_3$*) we have*

$$\begin{array}{rcl}M \in s\text{-}\mathsf{Elt} &\Leftrightarrow& \mathsf{h}(M) = x \in \mathsf{Var}^s \vee\\ && \mathsf{h}(M) = s'' \textit{ with } s'' : s' : s \in \mathcal{A} \textit{ for some } s' \in \mathcal{S},\\ M \in s\text{-}\mathsf{Term} &\Rightarrow& \mathsf{h}(M) = x \in \mathsf{Var}^{s'} \textit{ with } s{:}s' \in \mathcal{A} \vee\\ && \mathsf{h}(M) = s' \textit{ with } s'{:}s \in \mathcal{A}.\end{array}$$

Proof. By induction on the structure of M. For the first part of the Lemma: The reverse implication uses the Classification Lemma in case $M \equiv x$. All other cases follow straightforward from IH and the restrictions on the rules and axioms. For the second part of the Lemma, all cases follow easily from IH and the restrictions, except when M is an application term, in which case we need the first part of the Lemma. We do this case in detail.

$M \equiv PN$, say $\Gamma \vdash PN : s$ with $\Gamma \vdash P : \Pi y{:}B.C : s_3$ and $\Gamma \vdash N : B$. Then $C[N/y] =_\beta s$ and hence $s{:}s_3 \in \mathcal{A}$. Now we can apply the first part of the Lemma to the term PN to find that either $\mathsf{h}(PN) = x \in \mathsf{Var}^{s_3}$ or $\mathsf{h}(PN) = s''$ with $s''{:}s'{:}s_3$ for some s'. By the restrictions on the rules and the fact that $s : s_3$, we find that either $\mathsf{h}(PN) = x \in \mathsf{Var}^{s_3}$ with $s{:}s_3 \in \mathcal{A}$ or $\mathsf{h}(PN) = s''$ with $s''{:}s \in \mathcal{A}$ and we are done. ⊠

Chapter 5

The Church-Rosser property for $\beta\eta$-reduction

5.1. Introduction

In this chapter we want to treat the proof of the Church-Rosser property for $\beta\eta$-reduction in functional normalizing Pure Type Systems. By the restriction to functional normalizing systems we don't mean that the general property is false: At this moment this is still an open question, but we strongly believe that $\mathrm{CR}_{\beta\eta}$ holds in general for all Pure Type Systems. At the end of this section we shall make some comments on this and also on the proof, which we believe has some deficits.

In giving the proof we roughly follow [Geuvers 1992]. In fact the proof we give here is an expanded and updated version of the one that was given in there. We have changed the order of the lemmas a bit to stress which properties are general and which ones are specific properties of functional normalizing PTSs.

5.2. The proof of $\mathrm{CR}_{\beta\eta}$ for normalizing systems

Before giving the proof we want to fix some terminology and highlight some properties that come in handy for the proofs.

NOTATION. Suppose $\Gamma \vdash M : A$ is a derivable judgement in a functional PTS. If P is a subterm of M, we can speak of *the type of* P *in the derivation of* $\Gamma \vdash M : A$. In fact this type is unique up to $\beta\eta$-equality, due to the uniqueness of type property (Lemma 4.4.29). We therefore introduce the notation $\mathsf{ty}(P)$, which depends on Γ, M and A (but this dependency will usually not be mentioned explicitly) and is unique up to $=_{\beta\eta}$. We also want to fix the notion of a variable x being free in $\mathsf{ty}(P)$ or not. As $\mathsf{ty}(P)$ is unique up to $=_{\beta\eta}$ we shall usually be interested to know whether we can find a type for P in which x is not free. We therefore introduce the notation $x \notin_! \mathsf{ty}(P)$ to denote that there is a type B of P such that $x \notin \mathrm{FV}(B)$. (Note that all this is still relative to Γ, M and A.)

5.2.1. REMARK. 1. For terms that have a sort as type, the Key Lemma 4.4.18, gives in practice more specific information: If $\Pi x{:}A_1.A_2 =_{\beta\eta} C$ and $C : s$ for $s \in \mathcal{S}$, then $C \twoheadrightarrow_\beta \Pi x{:}C_1.C_2$ with $C_i =_{\beta\eta} A_i$. Similarly if $xP_1 \cdots P_n =_{\beta\eta} C$ and $C : s$ for $s \in \mathcal{S}$, then $C \twoheadrightarrow_\beta xQ_1 \cdots Q_n$ with $P_i =_{\beta\eta} Q_i$. This is true because C can not be a λ-abstraction.

2. For well-typed terms (in an arbitrary PTS) that are $\beta\eta$-equal to a sort, the Key Lemma 4.4.18, also gives some extra information: If $A \in \mathsf{Term}(\zeta)$ and $A \twoheadrightarrow_{\beta\eta} s(\in \mathcal{S})$, then $A \twoheadrightarrow_\beta s$. (This is easily verified by noticing that, if $A =_{\beta\eta} s$, then $A \twoheadrightarrow_\beta \lambda\vec{y}{:}A.s\vec{R} \twoheadrightarrow_\eta s$ (see Proposition 4.4.14) and that $\lambda\vec{y}{:}A.s\vec{R}$ can only be well-typed if $\vec{y}$ is empty.)

We first list some lemmas that are valid in all PTS (not just the functional normalizing ones.) We have not listed them under the general meta theory for Pure Type Systems because all the properties are about terms being (equal to a term) in normal form, so for systems that are not normalizing these properties loose their interest.

5.2.2. LEMMA.

$$\left.\begin{array}{r} \Gamma \vdash A{:}s \\ A \ in\ \beta\eta\text{-}nf \\ A =_{\beta\eta} B \\ x \notin FV(B, types\ of \Gamma) \end{array}\right\} \Rightarrow x \notin FV(A).$$

Here the types of Γ *are the terms C such that $y : C \in \Gamma$ for some variable y.*

PROOF. The proof is by induction on the structure of A. For A a sort or a variable it's trivial. For $A \equiv \Pi x{:}A_0.A_1$, we are done by induction hypothesis. Suppose now that A is an application term. Then x can only be free in domains of A. (Note that $|B| =_{\beta\eta} |A| \twoheadrightarrow_\eta \mathrm{nf}(|A|)$, and in untyped lambda calculus η-reductions do not remove any free variables, so $x \notin \mathrm{FV}(|A|)$.) Say C is the leftmost domain of A in which x occurs free, say in the subterm $zR_1 \cdots R_q(\lambda y_1{:}E_1.\cdots\lambda y_p{:}E_p.\lambda y{:}C.P)$. Then $x \notin_! \mathsf{ty}(z)$, because z is declared in the context or z is abstracted inside A to the left of C. This implies that also $x \notin_! \mathsf{ty}(zR_1 \cdots R_q)$. Now $\mathsf{ty}(zR_1 \cdots R_q) = \Pi q{:}(\Pi y_1{:}E_1.\cdots\Pi y_p{:}E_p.\Pi y{:}C.D).F$ and hence $C =_{\beta\eta} E$, for some E with $x \notin \mathrm{FV}(E)$. Now we can apply IH because C in β-nf and $x \notin \mathrm{FV}(E)$. So, $x \notin FV(C)$ and there is no leftmost domain in A in which x occurs free. ⊠

5.2.3. PROPOSITION. *For $M \in \mathsf{Term}$, if M in $\beta\eta$-nf, then $|M|$ in $\beta\eta$-nf.*

PROOF. Suppose M in $\beta\eta$-nf and $|M|$ not in $\beta\eta$-nf. Then $|M|$ is in β-nf by Proposition 4.4.14. So there is an η-redex in $|M|$, which is not an η-redex in M, say $\lambda x.|N|x$ is the left most such. Then $x \in \mathrm{FV}(N)$ while $x \notin \mathrm{FV}(|N|)$, so x occurs only free in domains of N. We now follow roughly the same method as in the proof of Lemma 5.2.2: Say C is the leftmost domain in which x is free, say C occurs in the subterm $zR_1 \cdots R_q(\lambda y_1{:}E_1.\cdots\lambda y_p{:}E_p.\lambda y{:}C.P)$. Again $x \notin_! \mathsf{ty}(z)$. (If z is declared in the context or abstracted left from the abstraction over x, then $x \notin_! \mathsf{ty}(z)$ by the convention that all bound variables are different and different from the free ones. If z is abstracted right from x, then $x \notin_! \mathsf{ty}(z)$ by the assumption that C is the leftmost domain containing x.) This implies that $x \notin_! \mathsf{ty}(zR_1 \cdots R_q)$ and by the fact that $\mathsf{ty}(zR_1 \cdots R_q) = \Pi q{:}(\Pi y_1{:}E_1.\cdots\Pi y_p{:}E_p.\Pi y{:}C.D).F$ we find that $C =_{\beta\eta} E$, for some E with $x \notin \mathrm{FV}(E)$. Now, by Lemma 5.2.2, $x \notin FV(C)$, so there is no leftmost domain in M in which x occurs free. Hence $|M|$ is in $\beta\eta$-nf. $\boxtimes$

5.2.4. LEMMA.

$$\left.\begin{array}{r}\Gamma \vdash M{:}A \\ \Gamma \vdash M'{:}A' \\ A =_{\beta\eta} A' \\ |M| \equiv |M'| \\ M, M' \textit{ in } \beta\textit{-nf}\end{array}\right\} \Rightarrow M \equiv_d M'$$

(The equality $\equiv_d$, was defined in Definition 4.4.13: $M \equiv_d M'$ if $M \equiv_d M'$ and all corresponding domains are $\beta\eta$-equal.)

PROOF. M and M' have the same structure (apart from the domains) and we have to show that all respective domains in M and M' are pairwise $\beta\eta$-equal. Say $M = \lambda x_1{:}A_1.\ldots\lambda x_n{:}A_n.N$ and $M' = \lambda x_1{:}A'_1.\ldots\lambda x_n{:}A'_n.N'$, with N and N' not abstractions. Then $A =_{\beta\eta} \Pi x_1{:}A_1.\ldots\Pi x_n{:}A_n.B =_{\beta\eta} \Pi x_1{:}A'_1.\ldots\Pi x_n{:}A'_n B' =_{\beta\eta} A'$, for some B and B' by Stripping, so $A_i =_{\beta\eta} A'_i$. Now compare from left to right all domains in N and N'.
Say C occurs as $zR_1 \cdots R_q(\lambda y_1{:}E_1.\ldots\lambda y_p{:}E_p.\lambda x{:}C.P)$ in N and C' occurs as $zR'_1 \cdots R'_q(\lambda y_1{:}E'_1.\ldots\lambda y_p{:}E'_p.\lambda x{:}C'P')$ in N' and for all domains to the left of C (respectively C') we are already done by induction. So $R_i =_{\beta\eta} R'_i$ for all i and $E_i =_{\beta\eta} E'_i$ for all i and hence $\mathsf{ty}(zR_1 \cdots R_q) = \mathsf{ty}(zR'_1 \cdots R'_q)$. This implies that

$$\mathsf{ty}(\lambda y_1{:}E_1.\ldots\lambda y_p{:}E_p.\lambda x{:}C.P) = \mathsf{ty}(\lambda y_1{:}E'_1.\ldots\lambda y_p{:}E'_p.\lambda x{:}C'P')$$

and so $B =_{\beta\eta} B'$. $\boxtimes$

The following Lemma collects the results of the previous Lemmas, establishing the confluence of $\beta\eta$-equality *for types* in normalizing $\mathsf{PTS}^s_{\beta\eta}$.

5.2.5. LEMMA. *Let $s, s' \in \mathcal{S}$.*

$$\left.\begin{array}{r} \Gamma \vdash A{:}s \\ \Gamma \vdash B{:}s' \\ A =_{\beta\eta} B \\ A, B \textit{ in } \beta\eta\text{-}\textit{nf} \end{array}\right\} \Rightarrow A \equiv B.$$

PROOF. By induction on the structure of A, using the Key Lemma, 5.2.4 and 5.2.3.
If $A \equiv \Pi x{:}A_1.A_2$, then $B =_{\beta\eta} \Pi x{:}B_1.B_2$ with $A_1 =_{\beta\eta} B_1$ and $A_2 =_{\beta\eta} B_2$. By induction hypothesis $A_1 \equiv B_1$ and $A_2 \equiv B_2$.
If $A \equiv xP_1 \cdots P_n$, then $B \equiv xQ_1 \cdots Q_n$ with $P_i =_{\beta\eta} Q_i$ (by Key Lemma.) Now, $\mathsf{ty}(xP_1 \cdots P_n) = \mathsf{ty}(xQ_1 \cdots Q_n)$, and so $s \equiv s'$. Further, $xP_1 \cdots P_n$ and $xQ_1 \cdots Q_n$ are in $\beta\eta$-nf, so, by 5.2.3, $|xP_1 \cdots P_n|$ and $|xQ_1 \cdots Q_n|$ are, so $|xP_1 \cdots P_n| \equiv |xQ_1 \cdots Q_n|$. We can apply 5.2.4 and conclude that all respective domains in $xP_1 \cdots P_n$ and $xQ_1 \cdots Q_n$ are $\beta\eta$-equal. By induction hypothesis (comparing the domains in $xP_1 \cdots P_n$ and $xQ_1 \cdots Q_n$ from left to right) we conclude that all respective domains in $xP_1 \cdots P_n$ and $xQ_1 \cdots Q_n$ are syntactically equal,that is $xP_1 \cdots P_n \equiv xQ_1 \cdots Q_n$. ⊠

5.2.6. THEOREM ($\mathrm{CON}_{\beta\eta}$ for normalizing functional $\mathsf{PTS}^s_{\beta\eta}$).

$$\left.\begin{array}{r} \Gamma \vdash M{:}A \\ \Gamma \vdash M'{:}A \\ M =_{\beta\eta} M' \end{array}\right\} \Rightarrow M \downarrow_{\beta\eta} M'.$$

PROOF. Define $N := \mathrm{nf}(M), N' := \mathrm{nf}(M')$. We prove $N \equiv N'$ and we are done. By SR_β and SR_η we find $\Gamma \vdash N{:}A$ and $\Gamma \vdash N'{:}A$. By 5.2.3, $|N|$ and $|N'|$ are in normal form, so $|N| \equiv |N'|$. By 5.2.4, all respective domains in N and N' are $\beta\eta$-equal. We now compare all respective domains in N and N', from left to right. By Lemma 5.2.5 all respective domains in N and N' are syntactically equal ($\equiv$), so $N \equiv N'$. ⊠

Obviously, the normalization is essential for the proof. Note however that also the restriction to $\mathsf{PTS}^s_{\beta\eta}$ is essential, because in $\mathsf{PTS}_{\beta\eta}$ we don't know how to prove SR_η. Of course we are still interested in proving $\mathrm{CR}_{\beta\eta}$ for $\mathsf{PTS}_{\beta\eta}$ (functional and normalizing). Somewhat surprisingly maybe, that is easy now : Using the work on $\mathsf{PTS}^s_{\beta\eta}$ that has been done in this section, we can show that (streng) is a derived rule in a functional normalizing $\mathsf{PTS}_{\beta\eta}$ and hence that Theorem 5.2.6 holds for any functional normalizing $\mathsf{PTS}_{\beta\eta}$. In fact, everything that is required is a simple corollary of Lemma 5.2.5. Then the proof of the derivedness of (streng) in functional normalizing $\mathsf{PTS}_{\beta\eta}$ can be found by redoing the proof of derivedness of (streng) in PTS_β. (Sublemma 4.4.34 and Lemma 4.4.35.)

The property that proves strengthening and hence SR_η is interesting enough to give it a name and treat it as a specific feature on its own. This is because in practice it holds quite generally for functional systems, even if they are not normalizing (like $\lambda\star$), or if we do not yet have a proof of normalization (as is the case for $\mathrm{CC}_{\beta\eta}$ at this point in the text).

5.2.7. DEFINITION. We say that a $\mathsf{PTS}_{\beta\eta}$ or $\mathsf{PTS}^s_{\beta\eta}$ satisfies *$\beta\eta$-preservation of sorts*, if

$$\left.\begin{array}{r}\Gamma\vdash A:s\\ \Gamma\vdash B:s'\\ A=_{\beta\eta}B\end{array}\right\}\Rightarrow s\equiv s'.$$

Obviously, there are non-functional PTS that do not satisfy $\beta\eta$-preservation of sorts (because Uniqueness of Types doesn't hold.) It should also be clear that we strongly believe the property to hold for all functional PTS: It comes as an immediate consequence of Confluence, Subject Reduction and Uniqueness of Types. The Corollary of Lemma 5.2.5 that we are interested in in the present context is that all functional normalizing $\mathsf{PTS}_{\beta\eta}$ satisfy $\beta\eta$-preservation of sorts. The reason to highlight this property here as a special definition is twofold: First, this is the specific feature we need to make the proof of strengthening and hence SR_η, work. Second, the $\beta\eta$-preservation of sorts is quite easily proved for other systems like $\mathrm{CC}_{\beta\eta}$ and $\lambda\star$.

5.2.8. COROLLARY (of Lemma 5.2.5). *A functional, normalizing* $\mathsf{PTS}_{\beta\eta}$ *satisfies $\beta\eta$-preservation of sorts.*

PROOF. Suppose $\Gamma\vdash A:s$ and $\Gamma\vdash B:s'$ in a functional normalizing system without (streng). Then also $\Gamma\vdash A:s$ and $\Gamma\vdash B:s'$ in the extension of the system with the rule (streng). Now A and B both normalize, so, by SR_β and SR_η in the extended system, $\Gamma\vdash \mathrm{nf}(A):s$ and $\Gamma\vdash\mathrm{nf}(B):s'$ (still in the extended system). By Lemma 5.2.5, this implies $\mathrm{nf}(A)\equiv\mathrm{nf}(B)$, so by Uniqueness of Types, $s\equiv s'$.$\boxtimes$

Trivially, the Corollary also holds for functional normalizing $\mathsf{PTS}^s_{\beta\eta}$.

5.2.9. SUBLEMMA. *If a* $\mathsf{PTS}_{\beta\eta}$ *satisfies $\beta\eta$-preservation of sorts, then*

$$\left.\begin{array}{r}\Gamma_1,x{:}A,\Gamma_2\vdash M:B\\ x\notin FV(\Gamma_2,M)\end{array}\right\}\Rightarrow\exists B'[B=_{\beta\eta}B'\ \&\ \Gamma_1,\Gamma_2\vdash M:B'].$$

PROOF. The proof is by induction on the derivation of $\Gamma_1,x{:}A,\Gamma_2\vdash M:B$, distinguishing cases according to the last rule. The only interesting cases are when the last rule is (λ) or (app), so we treat those. (The other cases sometimes use the Remark 5.2.1.)

(λ) Say $M \equiv \lambda y{:}C.N$, $B \equiv \Pi y{:}C.D$ and

$$\frac{\Gamma_1, x{:}A, \Gamma_2, y{:}C \vdash N : D \quad \Gamma_1, x{:}A, \Gamma_2 \vdash \Pi y{:}C.D : s_3}{\Gamma_1, x{:}A, \Gamma_2 \vdash \lambda y{:}C.N : \Pi y{:}C.D}$$

Then by IH $\Gamma_1, \Gamma_2, y{:}C \vdash N : D'$ for some D' with $D =_{\beta\eta} D'$.
Also, $\Gamma_1, x{:}A, \Gamma_2 \vdash C : s_1$ and $\Gamma_1, x{:}A, \Gamma_2, y{:}C \vdash D : s_2$ are conclusions of subderivations (with $(s_1, s_2, s_3) \in \mathcal{R}$), so by IH $\Gamma_1, \Gamma_2 \vdash C : E$ with $E =_{\beta\eta} s_1$ and hence $\Gamma_1, \Gamma_2 \vdash C : s_1$ by 5.2.1 and SR_β.
By Correctness of Types we find that $\Gamma_1, \Gamma_2, y{:}C \vdash D' : s_2'$ or $D' \equiv s \in \mathcal{S}$.
In the second case we have $D \twoheadrightarrow_\beta s$ and $s{:}s_2 \in \mathcal{A}$. So $\Gamma_1, \Gamma_2 \vdash \Pi y{:}C.s : s_3$ and hence $\Gamma_1, \Gamma_2 \vdash \lambda y{:}C.N : \Pi y{:}C.s$ with $\Pi y{:}C.D =_{\beta\eta} \Pi y{:}C.s$.
In the first case we have by $\beta\eta$-preservation of sorts that $s_2 \equiv s_2'$. So $\Gamma_1, \Gamma_2 \vdash \Pi y{:}C.D' : s_3$ and hence $\Gamma_1, \Gamma_2 \vdash \lambda y{:}C.N : \Pi y{:}C.D'$ with $\Pi y{:}C.D =_{\beta\eta} \Pi y{:}C.D'$.

(app) Say $M \equiv NP$, $B \equiv D[P/y]$ and

$$\frac{\Gamma_1, x{:}A, \Gamma_2 \vdash N : \Pi y{:}C.D \quad \Gamma_1, x{:}A, \Gamma_2 \vdash P : C}{\Gamma_1, x{:}A, \Gamma_2 \vdash NP : D[P/y]}$$

Then by IH, $\Gamma_1, \Gamma_2 \vdash N : E$ and $\Gamma_1, \Gamma_2 \vdash N : F$ with $\Pi y{:}C.D =_{\beta\eta} E$ and $F =_{\beta\eta} C$. By the Key Lemma we find that $E \twoheadrightarrow_\beta \Pi y{:}C'.D'$ with $C' =_{\beta\eta} C$ and $D' =_{\beta\eta} D$. So, by Corollary 4.4.31, $\Gamma_1, \Gamma_2 \vdash N : \Pi y{:}C'.D'$. We can apply $(\mathrm{conv}_{\beta\eta})$ to $\Gamma_1, \Gamma_2 \vdash P : F$ and $F =_{\beta\eta} C'$ to conclude $\Gamma_1, \Gamma_2 \vdash P : C'$ and hence $\Gamma_1, \Gamma_2 \vdash NP : D'[N/y]$, where $D'[N/x] =_{\beta\eta} D[N/x]$. ⊠

5.2.10. Lemma. *If a $\mathsf{PTS}_{\beta\eta}$ satisfies $\beta\eta$-preservation of sorts, then it satisfies strengthening, that is*

$$\left.\begin{array}{r}\Gamma_1, x{:}A, \Gamma_2 \vdash M : B\\ x \notin FV(\Gamma_2, M, B)\end{array}\right\} \Rightarrow \Gamma_1, \Gamma_2 \vdash M : B.$$

Proof. By the Sublemma we find a B' such that

$$\Gamma_1, \Gamma_2 \vdash M : B' \text{ and } B =_{\beta\eta} B'.$$

By Correctness of Types there are two possibilities, $\Gamma_1, x{:}A, \Gamma_2 \vdash B : s$ or $B \equiv s \in \mathcal{S}$. In the second case we are immediately done by SR_β, because $B' \twoheadrightarrow_\beta s$. In the first case we have

$$\Gamma_1, x{:}A, \Gamma_2 \vdash B : s$$

and by once again applying the Sublemma we find that

$$\Gamma_1, \Gamma_2 \vdash B : E =_{\beta\eta} s.$$

Now we are done by the fact that $E \twoheadrightarrow_\beta s$, SR_β and one application of (conv). ⊠

5.2.11. COROLLARY ($\beta\eta$-preservation of sorts implies SR_η). *A* $\mathsf{PTS}_{\beta\eta}$ *that satisfies $\beta\eta$-preservation of sorts, satisfies SR_η.*

PROOF. The proof is exactly the same as for Lemma 4.4.30, so one proves simultaneously the following.

$$\begin{aligned} \Gamma \vdash P : D \;\&\; P \longrightarrow_\eta P' &\;\;\Rightarrow\;\; \Gamma \vdash P' : D, \\ \Gamma \vdash P : D \;\&\; \Gamma \longrightarrow_\eta \Gamma' &\;\;\Rightarrow\;\; \Gamma' \vdash P : D. \end{aligned}$$

The proof uses the fact that we have strengthening, which was stated in the Lemma. ⊠

5.2.12. REMARK. In fact we can do with less then $\beta\eta$-preservation of sorts to prove strengthening and hence SR_η. The specific property that we need in the proof of strengthening is the following.

$$\left.\begin{array}{r} \Gamma \vdash A : s_2 \\ \Gamma \vdash B : s_2' \\ A =_{\beta\eta} B \\ (s_1, s_2, s_3) \in \mathcal{R} \end{array}\right\} \Rightarrow \exists s_3'[(s_1, s_2', s_3') \in \mathcal{R}].$$

(This is used in the case of the (λ) rule.)

If the system satisfies $\beta\eta$-preservation of sorts, the above property is obviously satisfied. But there are more Pure Type Systems that satisfy the above property, for example the *semi-full* ones. Remember that a PTS is semi-full if

$$(s_1, s_2, s_3) \in \mathcal{R} \;\&\; s_2' \in \mathcal{S} \Rightarrow \exists s_3'[(s_1, s_2', s_3') \in \mathcal{R}].$$

It is easy to verify that the above mentioned property holds. Consequently, all semi-full $\mathsf{PTS}_{\beta\eta}$ satisfy strengthening and hence all semi-full $\mathsf{PTS}_{\beta\eta}$ satisfy SR_η.

5.2.13. THEOREM ($\mathrm{CON}_{\beta\eta}$ for normalizing functional $\mathsf{PTS}_{\beta\eta}$).

$$\left.\begin{array}{r} \Gamma \vdash M{:}A \\ \Gamma \vdash M'{:}A \\ M =_{\beta\eta} M' \end{array}\right\} \Rightarrow M \downarrow_{\beta\eta} M'.$$

PROOF. The Theorem follows immediately from Theorem 5.2.6 by the fact that in any functional normalizing $\mathsf{PTS}_{\beta\eta}$ the rule (streng) is satisfied, which again follows immediately from Corollary 5.2.8 and Lemma 5.2.10. ⊠

5.3. Discussion

We have proved $\mathrm{CON}_{\beta\eta}$ for terms in a fixed context of a fixed type, but only for functional normalizing $\mathsf{PTS}_{\beta\eta}$. This immediately implies $\mathrm{CR}_{\beta\eta}$ on Term, because we have SR_β and SR_η for these systems. Confluence for well-typed terms of different types doesn't hold: Just consider the well-known counterexample. The same can be said for well-typed terms in different contexts: Take $A \neq_{\beta\eta} B$ and Γ and Γ' such that

$$\Gamma \vdash x(\lambda y : A.y) : \star \text{ and } \Gamma' \vdash x(\lambda y : B.y) : \star.$$

Then $x(\lambda y : A.y) =_{\beta\eta} x(\lambda y : B.y)$, but not $x(\lambda y : A.y) \downarrow_{\beta\eta} x(\lambda y : B.y)$.

We think that, using the work of [van Benthem Jutting 199+], who gives an analysis of typing in PTSs, these results can be extended to arbitrary normalizing type systems. The most interesting extension, however, is the one to non-normalizing type systems like $\lambda\star$. First because the proof given here relies very heavily on the normalization. Second, and maybe even more important, because from $\mathrm{CON}_{\beta\eta}$ on $\mathsf{Term}(\Gamma, A)$ in $\lambda\star$ (with $(\mathrm{conv}_{\beta\eta})$) we hope to get $\mathrm{CON}_{\beta\eta}$ on $\mathsf{Term}(\Gamma, A)$ for an arbitrary $\mathsf{PTS}_{\beta\eta}$, by imitating the reduction steps in $\lambda\star$ in the other $\mathsf{PTS}_{\beta\eta}$, using the terminality of $\lambda\star$ in the category $\mathsf{PTS}_{\beta\eta}$.

Let's now prove a general statement along these lines, i.e. describe a $\mathsf{PTS}_{\beta\eta}$ ζ such that, if $\zeta \models \mathrm{CON}_{\beta\eta}$, then $\mathrm{CON}_{\beta\eta}$ holds for any $\mathsf{PTS}_{\beta\eta}$. Note Remark 5.2.12, saying that SR_η holds for any semi-full $\mathsf{PTS}_{\beta\eta}$.

5.3.1. Definition. The $\mathsf{PTS}_{\beta\eta}$ $\lambda\mathbb{N}$ is the system $\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$ with

$$\begin{array}{rcl} \mathcal{S} & = & \mathbb{N}, \\ \mathcal{A} & = & \mathbb{N} \times \mathbb{N}, \\ \mathcal{R} & = & \mathbb{N} \times \mathbb{N} \times \mathbb{N} \end{array}$$

So $\lambda\mathbb{N}$ is full (and hence semi-full), which implies that $\lambda\mathbb{N}$ satisfies SR_η. (See Remark 5.2.12.) We now have the following Proposition.

5.3.2. Proposition. *If $\lambda\mathbb{N}$ satisfies $CON_{\beta\eta}$, then all $\mathsf{PTS}_{\beta\eta}$ satisfy $CON_{\beta\eta}$.*

Proof. Suppose $\lambda\mathbb{N}$ satisfies $\mathrm{CON}_{\beta\eta}$ and let ζ be an arbitrary $\mathsf{PTS}_{\beta\eta}$ with $\Gamma \vdash_\zeta M, N : A$ and $M =_{\beta\eta} N$. We have to show that $M \downarrow_{\beta\eta} N$. Now let $\tilde{-}$ be a mapping from the sorts of ζ to $\mathbb{N}$ that is injective on the set of sorts of ζ that occur in Γ, M or A. $\lambda\mathbb{N}$ is full, so the map $\tilde{-}$ is a PTS-morphism, so

$$\tilde{\Gamma} \vdash_{\lambda N} \tilde{M}, \tilde{N} : \tilde{A}.$$

Now, $\tilde{M} \downarrow_{\beta\eta} \tilde{N}$ and due to the local injectivity of $\tilde{-}$, the reduction paths from $\tilde{M}$, resp. $\tilde{N}$ can be faithfully translated back to reduction paths from M, resp. N, and so $M \downarrow_{\beta\eta} N$. ⊠

Because of the restriction to normalizing systems, we need to prove normalization of $\beta\eta$-reduction without using the Church-Rosser property. This may look problematic but in practice it isn't. For example for the Calculus of Constructions, the strong normalization proof in [Geuvers and Nederhof 1991] for the system with (conv_β) can quite easily be adapted to a proof of strong normalization for the system with ($\mathrm{conv}_{\beta\eta}$). We conjecture here the general theorem that, if a PTS_β is (strongly) normalizing, then the $\mathsf{PTS}_{\beta\eta}$ is.

That the proof of $\mathrm{CR}_{\beta\eta}$ for non-normalizing systems need not be very complicated is shown by the example of λU. This is the system defined in Definition 4.3.12 for which normalization does not hold. If we extend the system by replacing the conversion rule with the ($\mathrm{conv}_{\beta\eta}$) rule, the separation of contexts (Proposition 4.3.14) still holds. Due to this property, the proof of $\mathrm{CR}_{\beta\eta}$ is easy. It works as follows:

1. Note that, if $\Gamma \vdash A : \mathsf{Type}'$, then A contains no redexes.
2. Hence, if $\Gamma \vdash M, N : A(: \mathsf{Type})$, then the domains in M and N contain no redexes.
3. Conclude that $\mathrm{CON}_{\beta\eta}$ holds for such M and N.
4. Note that, if $\Gamma \vdash M : A(: \mathsf{Prop})$, then the domains of M are terms B with $\Gamma \vdash B : \mathsf{Type}$ or $\Gamma \vdash B : \mathsf{Prop}(: \mathsf{Type})$.
5. Hence, for these domains $\mathrm{CON}_{\beta\eta}$ already holds.
6. Hence $\mathrm{CON}_{\beta\eta}$ holds for M and N with $\Gamma \vdash M, N : A(: \mathsf{Prop})$.

If we look at the Church-Rosser property from a point of view as to how to compute the common reduct, we see that the situation is really a bit more complicated then for untyped lambda calculus. In untyped lambda calculus, if $M \twoheadrightarrow_{\beta\eta} M_1$ and $M \twoheadrightarrow_{\beta\eta} M_2$, a common reduct of M_1 and M_2 can be found using complete developments. (See [Barendregt 1984].) Here one has to do something more, namely reduce the domains: Consider again $M := \lambda x{:}A.(\lambda y{:}B.y)x$, $M_1 := \lambda x{:}A.x$ and $M_2 := \lambda y{:}B.y$. There are no residuals of the β-redex in M_2, nor are there any residuals of the η-redex in M_1, so we have a complete development of the set of both redexes, but $M_1 \not\equiv M_2$. (They would have been in the untyped case.) We still have to unify A and B.

Chapter 6

The Calculus of Constructions and its fine structure

6.1. Introduction

In pragraph 4.3.1 we encountered the Calculus of Constructions (CC) as an example of a Pure Type System, where it was also called $\lambda P\omega$. In this chapter we want to study this system in more detail. This will be done in various ways. First we say something about the practical meaning of the system in terms of logic and data types. If we want to see the Calculus of Constructions as a logic we have to study the formulas-as-types embedding from higher order predicate logic into CC. We have already defined this embedding in Chapter 4.1 (paragraph 4.3.1) as an embedding from the system λPREDω to CC. As we have already convinced ourselves of the fact that λPREDω and PREDω are isomorphic systems via the formulas-as-types analogy we shall only be studying the embedding from λPREDω into CC. In paragraph 4.3.1 we also encountered the so called cube of typed lambda calculi, which gives a fine structure for CC. We shall also study the other systems of this cube, especially in relation to the formulas-as-types embedding. The central question for each of these systems will be whether the formulas-as-types embedding is complete. As we are mainly concerned with the cube from a point of view of logic, it is also interesting to see to which extent the systems of the cube are conservative over one another.

Two of the more complicated issues regarding CC are not treated in this Chapter, namely the strong normalization and the Church-Rosser property for $\beta\eta$-reduction on terms of CC. Strong normalization will be dealt with in Chapter 7.1. We discussed the Church-Rosser property in Chapter 5.1. From the normalization it follows by the techniques developed in Chapter 5.1 that the Church-Rosser property holds for $\beta\eta$-reduction in CC.

6.2. The cube of typed lambda calculi and the logic cube

We recall some definitions of previous chapters. First remember that the Barendregt's cube of typed lambda calculi (Definition 4.3.1) consists of eight PTS_βs. Each of them has

$$\begin{array}{rcl} \mathcal{S} & := & \{\star, \Box\}, \\ \mathcal{A} & := & \{\star : \Box\}. \end{array}$$

The rules for each system are as given in the following table.

$\lambda\!\to$	$(\star, \star,$			
$\lambda 2$	$(\star, \star)$	$(\Box, \star,$		
$\lambda\mathrm{P}$	$(\star, \star)$		$(\star, \Box)$	
$\lambda\overline{\omega}$	$(\star, \star)$			$(\Box, \Box)$
$\lambda\omega$	$(\star, \star)$	$(\Box, \star)$		$(\Box, \Box)$
$\lambda\mathrm{P}2$	$(\star, \star)$	$(\Box, \star)$	$(\star, \Box)$	
$\lambda\mathrm{P}\overline{\omega}$	$(\star, \star)$		$(\star, \Box)$	$(\Box, \Box)$
$\lambda\mathrm{P}\omega$	$(\star, \star)$	$(\Box, \star)$	$(\star, \Box)$	$(\Box, \Box)$.

The system $\lambda\mathrm{P}\omega$ is the Calculus of Constructions, sometimes called the *Pure* Calculus of Constructions to distinguish it from its variants and extensions. We shall refer to it as CC. The systems of the cube are usually presented as follows.

$$\begin{tikzcd}
 & \lambda\omega \arrow[rr] & & \lambda\mathrm{P}\omega\,(=\mathrm{CC}) \\
\lambda 2 \arrow[ur] \arrow[rr] & & \lambda\mathrm{P}2 \arrow[ur] & \\
 & \lambda\overline{\omega} \arrow[uu] \arrow[rr] & & \lambda\mathrm{P}\overline{\omega} \arrow[uu] \\
\lambda\!\to \arrow[uu] \arrow[ur] \arrow[rr] & & \lambda\mathrm{P} \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

where an arrow denotes inclusion of one system in another.

Remember that we also defined the logic cube (Definition 4.3.5), following [Berardi 1990] as follows. It consists of eight PTS_βs, each of them having

$$\begin{array}{rcl} \mathcal{S} & = & \mathsf{Prop}, \mathsf{Set}, \mathsf{Type}^p, \mathsf{Type}^s, \\ \mathcal{A} & = & \mathsf{Prop} : \mathsf{Type}^p \mathsf{Set} : \mathsf{Type}^s. \end{array}$$

and the rules of each of the systems as given by the following table

λPROP				
	$(\mathsf{Prop}, \mathsf{Prop})$			
λPROP2				
	$(\mathsf{Prop}, \mathsf{Prop})$		$(\mathsf{Type}^p, \mathsf{Prop})$	
$\lambda\mathrm{PROP}\overline{\omega}$				$(\mathsf{Type}^p, \mathsf{Type}^p)$,
	$(\mathsf{Prop}, \mathsf{Prop})$			
$\lambda\mathrm{PROP}\omega$				$(\mathsf{Type}^p, \mathsf{Type}^p)$
	$(\mathsf{Prop}, \mathsf{Prop})$		$(\mathsf{Type}^p, \mathsf{Prop})$	
λPRED	$(\mathsf{Set}, \mathsf{Set})$	$(\mathsf{Set}, \mathsf{Type}^p)$		
	$(\mathsf{Prop}, \mathsf{Prop})$	$(\mathsf{Set}, \mathsf{Prop})$		
λPRED2	$(\mathsf{Set}, \mathsf{Set})$	$(\mathsf{Set}, \mathsf{Type}^p)$		
	$(\mathsf{Prop}, \mathsf{Prop})$	$(\mathsf{Set}, \mathsf{Prop})$	$(\mathsf{Type}^p, \mathsf{Prop})$	
$\lambda\mathrm{PRED}\overline{\omega}$	$(\mathsf{Set}, \mathsf{Set})$	$(\mathsf{Set}, \mathsf{Type}^p)$	$(\mathsf{Type}^p, \mathsf{Set})$	$(\mathsf{Type}^p, \mathsf{Type}^p)$
	$(\mathsf{Prop}, \mathsf{Prop})$	$(\mathsf{Set}, \mathsf{Prop})$		
$\lambda\mathrm{PRED}\omega$	$(\mathsf{Set}, \mathsf{Set})$	$(\mathsf{Set}, \mathsf{Type}^p)$	$(\mathsf{Type}^p, \mathsf{Set})$	$(\mathsf{Type}^p, \mathsf{Type}^p)$
	$(\mathsf{Prop}, \mathsf{Prop})$	$(\mathsf{Set}, \mathsf{Prop})$	$(\mathsf{Type}^p, \mathsf{Prop})$	

The systems are presented in a picture as follows.

$$
\begin{tikzcd}
 & \lambda\mathrm{PROP}\omega \arrow[rr] & & \lambda\mathrm{PRED}\omega \\
\lambda\mathrm{PROP2} \arrow[rr] \arrow[ur] & & \lambda\mathrm{PRED2} \arrow[ur] & \\
 & \lambda\mathrm{PROP}\overline{\omega} \arrow[rr] \arrow[uu] & & \lambda\mathrm{PRED}\overline{\omega} \arrow[uu] \\
\lambda\mathrm{PROP} \arrow[rr] \arrow[uu] \arrow[ur] & & \lambda\mathrm{PRED} \arrow[uu] \arrow[ur] &
\end{tikzcd}
$$

where an arrow denotes inclusion of one system in another.

Because we have convinced ourselves of the fact that the formulas-as-types embedding of a logic into the corresponding system of the logic cube is in fact an isomorphism, we can restrict our study of the formulas-as-types embedding into the systems of the Barendregt's cube to the study of the *collapsing mapping* H. Remember that H is defined as the family of PTS-morphisms from logic cube to Barendregt's cube given by

$$\begin{array}{rcl} H(\mathsf{Prop}) & = & \star, \\ H(\mathsf{Set}) & = & \star, \\ H(\mathsf{Type}^p) & = & \Box, \\ H(\mathsf{Type}^s) & = & \Box. \end{array}$$

6.3. Some more meta-theory for CC

Before going into studying the systems, we want to make some further definitions. This will also be necessary for the proof of strong normalization that will be given in a later chapter. In the rest of this chapter we always assume that we are working in a system with *sorted variables*, so e.g. for the cube we have two sets of variables $\mathsf{Var}^\Box$ and $\mathsf{Var}^\star$. See Definition 4.2.9 for details about the sorted variables.

6.3.1. Definition. For ζ a system of the cube we define the sets of *kinds*, *types*, *constructors* and *objects* as follows.

$$\begin{array}{rcl} \mathsf{Kind}(\zeta) & := & \Box\text{-}\mathsf{Term}, \\ \mathsf{Type}(\zeta) & := & \star\text{-}\mathsf{Term}, \\ \mathsf{Constr}(\zeta) & := & \Box\text{-}\mathsf{Elt}, \\ \mathsf{Obj}(\zeta) & := & \star\text{-}\mathsf{Elt}. \end{array}$$

Usually ζ will be clear from the context, in which case we omit it. Note that $\mathsf{Type}(\zeta) \subset \mathsf{Constr}(\zeta)$.

Now we can apply Lemma 4.4.37 to conclude that

$$\begin{array}{rcl} \mathsf{Kind} \cap \mathsf{Type} & = & \emptyset, \\ \mathsf{Constr} \cap \mathsf{Obj} & = & \emptyset. \end{array}$$

This will be very useful when defining mappings on terms of a system of the cube. A related property that is useful for defining mappings is given by Lemma 4.4.39, which allows to distinguish cases according to the 'heart' of a term. (See Definition 4.4.38.) In the cube, the heart of a term A, $\mathsf{h}(A)$, is a variable, $\star$ or $\Box$. From Lemma 4.4.39 we derive the following.

6.3.2. LEMMA. *For A a well-typed term of the cube we have*

$$\begin{aligned} A \in \mathsf{Kind} &\Leftrightarrow \mathsf{h}(A) = \star, \\ A \in \mathsf{Type} &\Rightarrow \mathsf{h}(A) \in \mathsf{Var}^{\Box}, \\ A \in \mathsf{Constr} &\Leftrightarrow \mathsf{h}(A) \in \mathsf{Var}^{\Box}, \\ A \in \mathsf{Obj} &\Leftrightarrow \mathsf{h}(A) \in \mathsf{Var}^{\star}. \end{aligned}$$

In [Barendregt 1992], mappings on (subsets of the) well-typed terms of the cube are often defined on a specific subset of the pseudoterms T, and the case distinction in the definition is then made according to the *level* of terms. This notion of 'level' is very close to our notion of 'heart', and in fact all the mappings in [Barendregt 1992] can be defined similarly by using case distinctions according to the heart of subterms. We try to refrain from defining mappings on the pseudoterms, and instead define mappings only on the well-typed terms as much as possible, because we feel that this is more intuitive. For completeness we define the notion of level though, and give the main property that one would want for it.

6.3.3. DEFINITION. For M a pseudoterm of the cube, the *level of* M, $\sharp(M)$, is the natural number defined as follows.

$$\begin{aligned} \mathsf{h}(M) = x \in \mathsf{Var}^{\star} &\Rightarrow \sharp(M) = 0, \\ \mathsf{h}(M) = x \in \mathsf{Var}^{\Box} &\Rightarrow \sharp(M) = 1, \\ \mathsf{h}(M) = \star &\Rightarrow \sharp(M) = 2, \\ \mathsf{h}(M) = \Box &\Rightarrow \sharp(M) = 3. \end{aligned}$$

The notion of level is closely related to the Automath notion of 'degree'. (In Automath the numbering is reversed.) The main property for levels is the following.

6.3.4. LEMMA. *In a system of the cube,*

$$\Gamma \vdash M : A \Rightarrow \sharp(M) + 1 = \sharp(A).$$

PROOF. Immediate consequence of Lemma 6.3.2. ⊠

One important mapping from the well-typed terms to the untyped lambda terms we have already encountered: The map $|\text{-}|$ that erases all domains (i.e. types in a λ-abstraction.) This is a very syntactical mapping, which leaves in a lot of type information that is of no importance for the underlying algorithm that the λ-term represents. We therefore define a mapping $|\text{-}|^t$ that erases all type information.

6.3.5. DEFINITION. The mapping $|\text{-}|^t$ from the objects of a system of the cube to the untyped lambda calculus is defined as follows.

$$\begin{array}{rcl} |x| & := & x, \\ |\lambda x{:}A.M| & := & \lambda x.|M|, \text{ if } A \text{ is a type,} \\ |\lambda\alpha{:}A.M| & := & |M|, \text{ if } A \text{ is a kind,} \\ |MN| & := & |M||N|, \text{ if } N \text{ is an object,} \\ |MN| & := & |M|, \text{ if } N \text{ is a constructor.} \end{array}$$

6.3.6. DEFINITION. The λ-abstractions in a well-typed term of CC (but the definition immediately extends to pseudoterms of CC) are split into four classes, the 0-, 2-, P- and ω-abstractions, as follows.

1. $\lambda x{:}A.M$ is a *0-abstraction* if M is an object, A a type,
2. $\lambda\alpha{:}A.M$ is a *2-abstraction* if M is an object, A a kind,
3. $\lambda x{:}A.M$ is a *P-abstraction* if M is a constructor, A a type,
4. $\lambda\alpha{:}A.M$ is a *ω-abstraction* if M is a constructor, A a kind.

We can decorate the λs correspondingly, so we can speak of the λ_0s λ_ω of a term etc. We now also define the notions of $\beta(\eta)^0$-reduction, $\beta(\eta)^2$-reduction, $\beta(\eta)^P$-reduction and $\beta(\eta)^\omega$-reduction by just restricting reduction to the redexes with the appropriate subscript attached to the λ. We use an arrow with a superscript above it to denote these restricted reductions, so $\xrightarrow{\omega}_{\beta\eta}$ etcetera.

We want to state two of the most important properties of CC.

6.3.7. THEOREM. *CC is strongly normalizing. (All β-reduction sequences starting from an $M \in \mathsf{Term}(CC)$ are finite.)*

PROOF. A detailed proof is given in Chapter 7.1. ⊠

A first proof of normalization can be found in [Coquand 1985], but the proof contained a bug as was remarked by Jutting. Coquand repaired his own proof in [Coquand 1986] and together with Gallier he gave a (different) proof of strong normalization in [Coquand and Gallier 1990]. There are various other versions of (strong) normalization proofs for CC in the literature. All of them use a higher order variant of the 'candidat de réducibilité' method as developped by Girard for proving strong normalisation for his system F and Fω. (See [Girard et al. 1989] for the proof for system F.) The idea is to define a kind of realisability model in which propositions are interpreted as sets of lambda terms (the realisers). A detailed explanation of the method can be found in [Gallier 1990]. The proof of strong normalization in Chapter 7.1 is given by defining a reduction preserving mapping from CC to Fω. Then SN for CC follows from SN for Fω. This makes things slightly easier because we don't have to bother about type dependency. (Fω is easier to handle than CC.) A complicating matter of Chapter 7.1 is that the proof is given for CC with a ($\text{conv}_{\beta\eta}$) rule. (That is, the $\mathsf{PTS}_{\beta\eta}$ CC.) The Strong Normalization of this system was an open problem up to now.

Intuituively it is clear that the hard part (proof-theoretically speaking) of a proof of SN for CC should be the normalization of λ_0 redexes. For one thing, it can be observed that this is the case for Fω. In the proof of Chapter 7.1 this becomes also clear: The whole problem of SN for CC is reduced to the problem of SN for erased terms in Fω (in which case we have only the 0-redexes left.) In [Coquand and Huet 1988], a version of CC is discussed in which the conversion rule is restricted to performing β^P- and β^ω-reductions. There it is called the *restricted Calculus of Constructions*.

6.3.8. Definition. The *restricted Calculus of Constructions* is the system CC with the (conv) rule restricted to $\beta_{P\omega}$-equality.

Let us show that for that restricted case, SN is relatively easy (like in the simply typed lambda calculus.)

Recall the definitions of β^ω-redex and β^P-redex of Definition 6.3.6: A β-redex is a β^ω-redex if it is of the form $(\lambda\alpha{:}A.B)P$ with A a kind and B a constructor. A β-redex is a β^P-redex if it is of the form $(\lambda x{:}A.B)t$ with A a type and B a constructor. We write $\xrightarrow{\omega}_\beta$ and $\xrightarrow{P}_\beta$ for the corresponding reductions. In the following we show that $\beta^{P\omega}$-reduction is normalizing.

6.3.9. Proposition. *The combination of β-reduction of P-redexes and ω-redexes, $\beta^{P\omega}$-reduction, is normalizing in CC.*

Proof. The proof is in flavour and complexity quite close to the normalization proof for $\lambda\!\rightarrow$. We assign to every term M of CC a pair (d, n), where d is the maximum of the *depths* of all $\beta^{P\omega}$-redexes in M and n is the number of $\beta^{P\omega}$-redexes of maximal depth. Then we proceed by contracting an innermost redex of maximal depth. That this procedure yields the $\beta^{P\omega}$-normal form is then shown by induction on the lexicographical ordering on the pairs (d, n). Before giving the definition of depth, let us remind us of the fact that there are the following three ways in which new β-redexes can be created by a β-reduction.

$$\begin{array}{rcll}(\lambda x{:}A.x)(\lambda y{:}B.M)Q & \longrightarrow_\beta & (\lambda y{:}B.M)Q, & (1)\\ (\lambda x{:}A.C[xQ])(\lambda y{:}B.M) & \longrightarrow_\beta & C[(\lambda y{:}B.M)Q], & (2)\\ (\lambda x{:}A.\lambda y{:}B.M)PQ & \longrightarrow_\beta & (\lambda y{:}B[P/x].M[P/x])Q, & (3)\end{array}$$

where the last possibility can at the same time be an example of the second. Further there is one way in which existing redexes can be duplicated by a β-reduction:

$$(\lambda x{:}A.M)C[(\lambda y{:}B.P)Q] \longrightarrow_\beta M[C[(\lambda y{:}B.P)Q]/x],$$

with x having more then one free occurrence in M. Now we define the *depth* of a β^P- or β^ω-redex by

$$\mathrm{depth}((\lambda u{:}A.M)Q) := \mathrm{rank}(\text{type of } \lambda u{:}A.M),$$

where the *rank* of a kind (the type of $\lambda_P x{:}A.M$ or $\lambda_\omega \alpha{:}A.M$ is always a kind) is defined by

$$\begin{array}{rcl}\mathrm{rank}(\star) & = & 1,\\ \mathrm{rank}(\Pi x{:}A.B) & = & 1+\mathrm{rank}(B),\ \text{if } A \text{ is a type},\\ \mathrm{rank}(\Pi\alpha{:}A.B) & = & \mathrm{rank}(A)+\mathrm{rank}(B),\ \text{if } A \text{ is a kind.}\end{array}$$

All this is well-defined by the Uniqueness of Types property for CC (Lemma 4.4.29) and the fact that if two kinds are β-equal, their ranks are the same.
The normalization procedure is now by contracting each time an innermost $\beta^{P\omega}$-redex of maximal depth. If we define for any term M its complexity $c(M)$ as the pair (d,n) with d the maximal depth of all $\beta^{P\omega}$-redexes and n the number of $\beta^{P\omega}$-redexes of depth d in M, the normalization procedure as given above reduces the complexity of terms (in the lexicographical ordering.) We show this by distinguishing the three different possibilities for creating new redexes that are mentioned above. (The duplication of redexes can only happen with redexes of rank smaller then r, so duplication is no problem.)

- Note that, in the first case the contracted redex can not be a β^P-redex. Further, if in the second case the contracted redex is a β^P-redex, the created redex is not a $\beta^{P\omega}$-redex.

- If, in the first two cases, the contracted redex is a β^ω-redex of depth d (with as type of the λ-part $\Pi\alpha{:}A.B$ so $d=\mathrm{rank}(\Pi\alpha{:}A.B)$), the depth of the new redex is $\mathrm{rank}(A)$, so the number of redexes of depth d is reduced by one.

- If, in the third case, the contracted redex is of depth d (with as type of the λ-part $\Pi u{:}A.B$ so $d=\mathrm{rank}(\Pi\alpha{:}A.B)$), the depth of the new redex is $\mathrm{rank}(B)$, so the number of redexes of depth d is reduced by one. (This uses the fact that the rank of a kind is stable under substitution.) ⊠

The restricted Calculus of Constructions is of limited interest, because it is not possible to first $\beta^{P\omega}$ normalize and then perform only β^{02} steps to obtain the β-normal form. This is because e.g. a β^2 reduction can create a β^P-redex (and a β_0-reduction can again create β_2 redexes). An example is

$$(\lambda_2 Q{:}\alpha{\to}\star.Qy)(\lambda_P x{:}\alpha.\tau) \xrightarrow{2}_\beta (\lambda_P x{:}\alpha.\tau)y.$$

The importance of the (strong) normalisation property lies in the fact that it gives a handle on the number of proofs of a proposition. (One can for example show that every closed term of type Nat is β-equal to a numeral (i.e. a term of the form $S(\dots S(Z)\dots)$.) Further, by using normalization one can prove the decidability of typing.

6.3.10. THEOREM. *In CC, given a context Γ and a pseudoterm M, it is decidable whether there exists a term A with $\Gamma \vdash M : A$. If such a term A exists, it can be computed effectively.*

The proof is prooftheoretically hard beacuse it depends on normalization. Note therefore that type checking in the restricted calculus is much easier, due to the 'easy' normalization proof.

Some hints towards a proof can be found in [Coquand and Huet 1988] and more details in [Coquand 1985] and especially in [Martin-Löf 1971]. See also [Harper and Pollack 1991] for an exposition on the decidability of typing for an extended version of CC, which also describes an algorithm for computing a type.

6.4. Intuitions behind the Calculus of Constructions

Let's first make some remarks about the impredicative coding of data types in (higher order) polymorphic lambda calculus. We feel this is necessary for a good understanding of CC. For this purpose it doesn't matter if we consider the versions that we called F and Fω or the PTS_β-versions that we called $\lambda 2$ and $\lambda\omega$. Details of the encoding can be found in [Böhm and Berarducci 1985] and [Girard et al. 1989]. We just treat three examples.

6.4.1. EXAMPLES.
1. The natural numbers in $\lambda 2$ and $\lambda\omega$ are defined by the type
$$\text{Nat} := \Pi\alpha : \mathsf{Prop}.\alpha{\to}(\alpha{\to}\alpha){\to}\alpha$$
and we find zero and succesor by taking $Z := \lambda\alpha{:}\mathsf{Prop}.\lambda x{:}\alpha.\lambda f{:}\alpha{\to}\alpha.x$ and $S := \lambda n{:}\text{Nat}.\lambda\alpha{:}\mathsf{Prop}.\lambda x{:}\alpha.\lambda f{:}\alpha{\to}\alpha.f(n\alpha x f)$. Now it is easy to define functions by iteration on Nat, by taking for $c{:}\sigma$, $g{:}\sigma{\to}\sigma$, $\text{It}cg{:}\text{Nat}{\to}\sigma$ as $\text{It}cg := \lambda x{:}\text{Nat}.x\sigma cg$. It is also possible to define functions by primitive recursion, but this is a bit more involved and also inefficient.

2. For σ a type, the type of list over σ is defined by the type
$$\text{List}(\sigma) := \Pi\alpha{:}\mathsf{Prop}.\alpha{\to}(\sigma{\to}\alpha{\to}\alpha){\to}\alpha)$$
and we find the constructors $\text{Nil} := \lambda\alpha{:}\mathsf{Prop}.\lambda x{:}\alpha.\lambda f{:}\sigma{\to}\alpha{\to}\alpha.x$ and $\text{Cons} := \lambda t{:}\sigma l{:}\text{List}(\sigma).\lambda\alpha{:}\mathsf{Prop}.\lambda x{:}\alpha.\lambda f{:}\sigma{\to}\alpha{\to}\alpha.ft(l\alpha x f)$. Again function (like 'head' and 'tail') can be defined by iteration and primitive recursion over lists.

3. Also coinductive dat types can be defined in $\lambda 2$ and $\lambda\omega$, which can be understood as greatest fixed points in a domain (the inductive data types correspond to smallest fixed points.) As an example we treat the type of streams (infinite lists) of natural numbers.

$$\mathrm{Str}(\mathrm{Nat}) := \exists\alpha.(\alpha{\to}\mathrm{Nat})\&(\alpha{\to}\alpha)\&\alpha.$$

For convenience we write

$$\langle f, g, x\rangle : (\alpha{\to}\mathrm{Nat})\&(\alpha{\to}\alpha)\&\alpha$$

if $f{:}\alpha{\to}\mathrm{Nat}$, $g{:}\alpha{\to}\alpha$ and $x{:}\alpha$, with projections π_1, π_2 and π_3. Then we have destructors
Head:Str(Nat)→Nat and Tail:Str(Nat)→Str(Nat) defined by

$$\begin{array}{rcl} \mathrm{Head} & := & \lambda s{:}\mathrm{Str}(\mathrm{Nat}).s\mathrm{Nat}(\lambda\alpha z.(\pi_1 z)(\pi_3 z), \\ \mathrm{Tail} & := & \lambda s{:}\mathrm{Str}(\mathrm{Nat}).s\mathrm{Str}(\mathrm{Nat})(\lambda\alpha z.\lambda\beta k.k\alpha(\pi_1 z)(\pi_2 z)(\pi_2 z(\pi_3 z)). \end{array}$$

It is possible to define function to Str(Nat) by coiteration and corecursion.

The impredicative data types of $\lambda 2$ and $\lambda\omega$ have a lot of structure already. (Girard has shown that in $\lambda\omega$ one can define on the type Nat all recursive functions that are provably total in higher order arithmetic.) It seems a good idea to use them for the domains of the logic. So now we view $\lambda\omega$ not as higher order proposition logic, but as a term calculus in which one can construct functions (as λ terms.) Then, because we want to do *predicate* logic, we have to add to $\lambda\omega$ the possibility of defining predicates on these new domains by adding the rule $(\star, \Box)$ to $\mathcal{R}$. The kind $A{\to}\star$ then represents the type of predicates on A and we can declare variables of type $A{\to}\star$ in the context. This is the Calculus of Constructions, CC, the Pure Type System with

$$\begin{array}{rcl} \mathcal{S} & = & \star, \Box, \\ \mathcal{A} & = & \star : \Box, \\ \mathcal{R} & = & (\star, \star), (\star, \Box), (\Box, \Box), (\Box, \star). \end{array}$$

Using our understanding of higher order predicate logic, the sort $\star$ is the universe of both propositions and domains in which a whole range of (closed) data types is present. There is however another way to see things. This is to understand $\star$ just as the universe of propositions (refraining from understanding the propositions as domains), in which case a type like $\varphi{\to}\star$ ($\varphi : \star$) can be understood as the type of predicates on proofs of φ. For practical purposes this latter approach doesn't seem to be so fruitful. For example one can not distinguish between proofs that are cut-free and proofs that are not. This is because lambda terms that are β-equal (proofs that are equal via cut-elimination) are identified: If Pt is provable and $t =_\beta t'$, then also Pt' is provable. If one is looking for these kind of applications, it is much more promising to use the 'coding' of a logic in a relatively weak framework like Automath or LF. There is however also the possibility to restrict the conversion rule of CC, such that only some convertible propositions are identified. (A system like this is described in [Coquand and Huet 1988].)

It should be clear that in any of the two approaches the distinction between domains, objects and proofs is blurred: propositions may contain proofs and there is no a priori distinction between domains and propositions. On the other hand it does take the formulas-as-types approach very seriously in the sense that formulas are not only treated *in the same way* as the types (domains) but just as if they were types, putting them in the same universe. Because of this mixing of formulas and domains, the Curry-Howard embedding from higher order predicate logic into CC is not complete. The embedding from higher order propositional logic into CC (i.e. if one refrains from understanding the propositions as domains) is complete.

We want to treat some examples to get the flavour of the system. In these examples, the impredicative coding of data types will be used as described in 6.4.1. First we want to discuss induction over the terms of type Nat and see to which extent Nat represents the free algebra of natural numbers. Then we treat two formulas that represent specifications of programs. This touches upon one of the most interesting aspects of CC: To use it as a higher order constructive logic in which one can represent specifications as formulas (about data types.) From a proof of the formula the constructive content can then be extracted as a program (more precisely a lambda term typable in $\lambda\omega$.) A lot of work on this subject has been done in [Paulin 1989]; we shall say a little bit more about this in paragraph 6.7.

6.4.2. Example. We know from the normalization property that in CC each closed term of type Nat is β-equal to a term of the form

$$\lambda\alpha{:}\star.\lambda x{:}\alpha.\lambda f{:}\alpha\to\alpha.f(\ldots(fx)\ldots).$$

That is, modulo β-equality the closed terms of type Nat are precisely the ones formed by S out of Z. This induction property can be expressed in CC, but is not provable inside it. To be precise, if we define

$$\mathrm{Ind}_{\mathrm{Nat}} := \forall P{:}\mathrm{Nat}\to\star.PZ\to(\forall x{:}\mathrm{Nat}.Px\to P(Sx))\to(\forall x{:}\mathrm{Nat}.Px),$$

then $\mathrm{Ind}_{\mathrm{Nat}}$ is not provable. If we assume $\mathrm{Ind}_{\mathrm{Nat}}$, we still can't prove that the type Nat is the free structure generated by Z and S. To establish this we have to add the premises $Z \neq_{\mathrm{Nat}} SZ$ and $\forall x, y{:}\mathrm{Nat}.(Sx = Sy)\to(x = y)$. None of these two propositions is provable in CC. In higher order predicate logic (working in the natural numbers-signature $\langle N, Z, S\rangle$) these three assumptions are independent, so we would have to add all three of them to obtain the free algebra of natural numbers. In CC this is not so. Due to the specific structure of the type Nat, the assumptions $\mathrm{Ind}_{\mathrm{Nat}}$ and $Z \neq_{\mathrm{Nat}} SZ$ suffice to prove the freeness of Nat. (This is so because one can define $P{:}\mathrm{Nat}{\to}\mathrm{Nat}$ with $\mathrm{Ind}_{\mathrm{Nat}} \vdash \forall x{:}\mathrm{Nat}.P(Sx) =_{\mathrm{Nat}} x$ in CC.)

6.4.3. EXAMPLES. 1. Abbreviate List(Nat) to List. The proposition stating that for every finite list of numbers there is a number that majorizes all its elements can be expressed by

$$\forall l{:}\mathrm{List}.\exists n{:}\mathrm{Nat}.\forall m{:}\mathrm{Nat}.m \in l \to m \leq n,$$

where $m \in l$ stands for

$$\forall P{:}\mathrm{List}{\to}\star.(\forall k{:}\mathrm{List}.P(\mathrm{Cons}mk)) \to \forall k{:}\mathrm{List}\forall r{:}\mathrm{Nat}.(Pk{\to}P(\mathrm{Cons}rk)) \to Pl$$

and $m \leq n$ stands for

$$\forall R{:}\mathrm{Nat}{\to}\mathrm{Nat}{\to}\star.(\forall x{:}\mathrm{Nat}.Rxx){\to}(\forall x,y{:}\mathrm{Nat}.Rxy{\to}Rx(Sy)){\to}Rmn.$$

A proof of this proposition constructs for every list l a number n and a proof of the fact that n majorizes l. From it one can extract a program of type List→Nat that satisfies this specification.

2. Abbreviate Str(Nat) to Str. The proposition that every (infinite) stream that is majorizable has a maximal element can be expressed by

$$\forall s{:}\mathrm{Str}.(\exists n{:}\mathrm{Nat}.\forall m{:}\mathrm{Nat}.m \in s \to m \leq n){\to}(\exists n{:}\mathrm{Nat}.\text{`}n\text{ is maximum of }s\text{'}),$$

where $m \in s$ now stands for

$$\exists p{:}\mathrm{Nat}.\mathrm{Head}(p\mathrm{StrTail}s) = m,$$

and 'n is maximum of s' stands for

$$(n \in s)\&(\forall m{:}\mathrm{Nat}.m \in s \to m \leq n).$$

From a proof of this formula one would like to be able to extract a term of type Str→Nat that computes the maximum of a stream, if it exists. This means that we want to extract a partial function (the maximum may not exist), which is not possible, because in CC all functions are total. (Due to the normalization.) In practice this is no problem, because the extracted function will produce an 'arbitrary' number in case there is no maximum. This corresponds to the fact that in the proof of the formula, if s has no maximum we can take any number n to satisfy the conclusion 'n is maximum of s'. It will be clear that the construction in the proof (and hence the algorithm) depends heavily on the proof of the premise that s is majorizable.

6.5. Formulas-as-types of logics into the cube

The Curry-Howard embedding from logics into the typed lambd calculi of the cube makes an essential distinction between on the one hand basic and functional domains (including the definable data types) and on the other hand predicate domains like $A{\to}\mathsf{Prop}$. The basic domains are interpreted as variables of type $\star$, the functional domains as implicational formulas and the definable data types via the embedding of data types in system F. The predicate domains are interpreted as kinds, e.g. $A{\to}\star : \Box$. Using the logic cube we have described the formulas-as-types embedding as a PTS-morphism. In fact this was the reason for introducing the logic cube in the first place. In this section we study the completeness of the formulas-as-types embedding into the different systems of the cube by studying the PTS-morphism H from the logic cube into the cube. Although the main concern of this Chapter is the Calculus of Constructions, we also look at the embedding into the other systems.

In fact there are other ways of interpreting PREDω in CC, but the one we describe here is what the inventor(s) of CC aim at (see [Coquand 1985] and [Coquand and Huet 1988]), and which is sometimes called the 'canonical embedding' of higher order predicate logic into CC. The same holds for the system λP2: From [Longo and Moggi 1988] it becomes clear that the intention of the system is the formulas-as-types embedding of PRED2 into it in the way we have described it by the mapping H. In our setting the canonicity is partly forced upon by the syntax and therefore it is worthwile to also understand the embedding from a more semantical point of view.

It is well-known by now that the embedding into CC is not complete, i.e. there are sentences that are not provable in PREDω that become provable when mapped into CC. We shall treat some examples of those sentences. This incompleteness result is sometimes referred to as the 'non-conservativity of CC over higher order predicate logic', but this terminology is a bit ambiguous because (non-)conservativity actually only applies if a system is a subsystem of the other. Therefore we shall use the more correct terminology of '(in)completeness of the embedding' here. For the embedding into λP2 the question is still open, although there are reasons to believe that the embedding is not complete. This was explained to us by [Berardi 1990a] and we shall discuss these reasons briefly later. The embedding of PRED into λP is complete, as was shown independently by [Berardi 1988] and [Barendsen and Geuvers 1989]. We shall give the proof of the latter, which uses a method developped by [Swaen 1989] to show completeness of the formulas-as-types embedding of full first order pedicate logic into Martin-Löf's intuitionistic theory of types. Although the completeness of the embedding into λP is quite non-trivial, the result is not very interesting from a practical point of view. The logic PRED is too minimal to be of practical mathematical interest: There is no notion of negation in it.

6.5.1. The formulas-as-types embedding into CC

Let's first remark that there are terms of type $\star$, typable in CC in a context that comes from λPREDω, that do not have an intuitive meaning in higher order predicate logic, like $\alpha{:}\mathsf{Prop}, P{:}\alpha{\to}\mathsf{Prop}, x{:}\alpha \vdash Px{\to}\alpha : \mathsf{Prop}$. (Is $Px{\to}\alpha$ a domain or a proposition in λPREDω?)

As has been pointed out already,one can refrain from predicate logic and view CC as a higher order propositional logic with propositions about (proofs of) propositions. The typed lambda calculus corresponding to higher order propositional logic is λPROPω, which is exactly the same systems as $\lambda\omega$. So to understand the embedding from PROPω into CC we just have to look at the inclusion of $\lambda\omega$ in CC. Then all kind of rather exotic types can be understood as meta-propositions about higher order propositional logic. For example

$$\alpha{:}\star, P{:}\alpha{\to}\star, x{:}\alpha \vdash Px{\to}\alpha : \star$$

states that for α a proposition and x a proof of α, if P holds for x, then α holds. We can go to arbitrary high levels of meta-reasoning, for example

$$\alpha{:}\star, P{:}\alpha{\to}\star, x{:}\alpha, Q{:}Px{\to}\star, y{:}Px \vdash Px{\to}Qy : \star$$

but also

$$P{:}\Pi\alpha{:}\star.\alpha{\to}\star, \varphi{:}\star, x{:}\varphi, y{:}P\varphi x \vdash P(P\varphi x)y{:}\star.$$

It is well-known that the inclusion of $\lambda\omega$ into CC is complete, i.e. CC is conservative over $\lambda\omega$. This was proved independently by [Paulin 1989] and [Berardi 1989]; we give the proof in paragraph 6.5.3. It is quite similar to the proof of conservativity of PREDn over PROPn that we gave in Chapter 2.1.

As already pointed out, the formulas-as-types embedding from higher order predicate logic in CC is not complete. We now want to discuss some examples of sentences that are not provable in the logic but become inhabited when mapped into CC. At the same time one obtains a better understanding of the logical merits of CC. First we show that if one allows empty domains in the logic, the incompleteness is quite easy.

6.5.1. Remark. In CC, the existential quantifier has a first projection, similar to Martin-Löf's understanding of the existential quantifier as a strong Σ-type. (See e.g. [Martin-Löf 1984].) Remember that

$$\exists x{:}A.\varphi \equiv \Pi\alpha{:}\star.(\Pi x{:}A.\varphi{\to}\alpha){\to}\alpha$$

in λPREDω. Now, in CC there is a projection function

$$p : (\exists x{:}A.\varphi){\to}A$$

for $A, \varphi{:}\star$. Take

$$p :\equiv \lambda z{:}(\exists x{:}A.\varphi).zA(\lambda x{:}A.\lambda y{:}\varphi.x.).$$

So, if $\exists x{:}A.\varphi$ is provable one immediately obtains a closed term of type A by applying p. In general there is no second projection, so the $\exists$ is not a strong Σ. (If, for example, $\exists x{:}A.\varphi$ is assumed in the context, say by $z{:}\exists x{:}A.\varphi$, then $\varphi[pz/x]$ is not provable.) Obviously, in λPREDω the existential quantifier has no first projection: The expression $(\exists x{:}A.\varphi)\rightarrow A$ can not even be formed if $A{:}\mathsf{Set}, \varphi{:}\mathsf{Prop}$.

6.5.2. Lemma. *In* λPREDω, *for* $x \notin FV(\varphi)$,

$$A{:}\mathsf{Set}, P{:}A\rightarrow\mathsf{Prop}, \varphi{:}\mathsf{Prop} \nvdash (\exists x{:}A.Px) \supset (\forall x{:}A.\varphi) \supset \varphi,$$

but in CC there is a term M *with*

$$A{:}\star, P{:}A\rightarrow\star, \varphi{:}\star \vdash M : (\exists x{:}A.Px)\rightarrow(A\rightarrow\varphi)\rightarrow\varphi.$$

Proof. Because the λPREDω-context doesn't contain a declaration of a variable to A, we can't construct a term of type A, so we have no proof. In CC, take $M \equiv \lambda z{:}(\exists x{:}A.Px).\lambda y{:}(A\rightarrow\varphi).y(px)$, with p as in Remark 6.5.1. ⊠

Even without using empty domains the embedding is not complete, as was first independently shown by [Berardi 1989] and [Geuvers 1989]. We treat both counterexamples, starting with the latter as it is very short (but syntactic.) Both proofs give a counterexample already for the completeness of the embedding of third order predicate logic in so called third order dependent typed lambda calculus. (In this terminology, CC is higher order dependent typed lambda calculus and the system λP2 is second order dependent typed lambda calculus.) The counterexample with empty domains above already works for second order dependent typed lambda calculus; it is not known whether one can find a counterexample without allowing empty domains.

6.5.3. Proposition. *The formulas-as-types embedding of higher order predicate logic into CC is not complete.*

Proof ([Geuvers 1989]). We use the fact that if $x \notin \mathrm{FV}(\varphi)$, then $\forall x{:}A.\varphi$ and $A \supset \varphi$ can not be distinguished in CC. (In λPREDω they are distinguished by $A{:}\mathsf{Set}$ or $A{:}\mathsf{Prop}$.) Take

$$\Gamma := A{:}\mathsf{Set}, a{:}A, \varphi{:}\mathsf{Prop}, P{:}\mathsf{Prop}\rightarrow\mathsf{Prop}, z{:}P(\Pi x{:}A.\varphi),$$

and we try to find a proof t of $\exists\beta{:}\mathsf{Prop}.P(\beta\rightarrow\varphi)$. As no extensionality has been assumed in the context, such t can't be found. (Supposing there is such t, one easily shows that it can't be in normal form.) However, in CC one can take the type A for β because sets and propositions are not distinguished. More precisely, in $\Gamma' = A{:}\star, a{:}A, \varphi{:}\star, P{:}\star\rightarrow\star, z{:}P(\Pi x{:}A.\varphi)$,

$$\Gamma' \vdash \lambda\gamma{:}\star.\lambda h{:}(\Pi\beta{:}\star.P(\gamma\rightarrow\varphi)\rightarrow\beta).hAz : \exists\beta{:}\star.P(\beta\rightarrow\varphi). \boxtimes$$

PROOF ([Berardi 1989]). Define

$$\mathrm{EXT} := \Pi\alpha, \beta{:}\mathsf{Prop}.(\alpha{\leftrightarrow}\beta) \to (\alpha = \beta),$$

where $\alpha{\leftrightarrow}\beta$ denotes $(\alpha{\to}\beta)\&(\beta{\to}\alpha)$ and $=$ denotes the Leibniz equality on Prop, $\alpha =_{\mathsf{Prop}} \beta :\equiv \forall P{:}\mathsf{Prop}{\to}\mathsf{Prop}.P\alpha{\to}P\beta$. This 'EXT' is the extensionality axiom for propositions. Let's denote the CC-version of EXT by EXT$'$, so

$$\mathrm{EXT}' := \Pi\alpha, \beta{:}\star.(\alpha{\leftrightarrow}\beta) \to (\alpha = \beta).$$

In CC this axiom has some unexpected consequences: If we take $A : \mathsf{Set}$ nonempty, then in CC

$$a{:}A \vdash M : A \leftrightarrow (A{\to}A)$$

for some M, so from EXT$'$ it follows that all generic properties that hold for A, hold for $A{\to}A$ and vice versa. This can be used to construct in CC a proof p with

$$A{:}\star, a{:}A, z{:}\mathrm{EXT}' \vdash p : A \text{ is a } \lambda\text{-model},$$

where

$$\begin{array}{rcl} A \text{ is a } \lambda\text{-model} & := & \exists\Lambda{:}(A{\to}A){\to}A.\exists\mathrm{App}{:}A{\to}A{\to}A. \\ & & \mathrm{App} \circ \Lambda = \mathrm{Id}_{A\to A}\& \\ & & \Lambda \circ \mathrm{App} = \mathrm{Id}_A. \end{array}$$

This implies (among other things) that every term of type $A{\to}A$ has a fixed point. Of course, in higher order predicate logic, from EXT it doesn't follow that every function on a non-empty domain has a fixed point.

If we look for example at a context for Heyting arithmetic,

$$\begin{array}{rcl} \Gamma_{HA} & := & N{:}\star, 0{:}N, S{:}N{\to}N, \\ & & z_1{:}\Pi x{:}N.(Sx =_N Sy){\to}(x =_N y), \\ & & z_2{:}S0 \neq_N 0, \\ & & z_3{:}\Pi P{:}N{\to}\star.P0{\to}(\Pi y{:}N.Py{\to}P(Sy)){\to}(\Pi y{:}N.Py), \end{array}$$

then there is a term t in CC with

$$\Gamma_{HA}, z{:}\mathrm{EXT}' \vdash t : \bot. \boxtimes$$

6.5.2. The formulas-as-types embedding into subsystems of CC

The formulas-as-types embedding into the systems in the left plane of the cube is certainly complete: We have shown in chapter 3.1 that the embedding is even an isomorphism. This leaves us with the other three systems of the right plane. We do not treat the case of the embedding of $\lambda\mathrm{PRED}\overline{\omega}$ into $\lambda\mathrm{P}\overline{\omega}$, because we believe that a conservativity proof can be given by simply adapting the proof for λPRED and λP. More importantly this case is not of real interest, because the systems themselves are not of practical interest: They have just come up as a derivative of the definition of the cube as a fine structure for CC. (λPRED$\overline{\omega}$ corresponds to PRED$^\tau$, as it was defined in Definition 2.2.11. The systems PREDn^τ were introduced there for reasons of the semantics that we wanted to treat.)

This leaves us with two cases, λP2 and λP. The first case is open and for the second case the formulas-as-types embedding is complete. Let us first say something about the embedding of second order predicate logic into λP2.

First remark that the proofs of incompleteness of the embedding for CC (Proposition 6.5.3) also work for λPn for any $n > 2$. So the formulas-as-types embedding from nth order predicate logic into nth order dependent typed lambda calculus is incomplete for $n > 2$. Further, if we allow empty domains in the logic, the incompleteness is easily shown: Lemma 6.5.2 also holds for λPRED2 and λP2. Although we have no proof, there are reasons to believe that the embedding H from λPRED2 into λP2 is also incomplete if we do not allow empty domains in the logic. These reasons were provided by [Berardi 1990a] who suggests a proof of incompleteness. To understand the idea, we think it is best to look at an extension of λPRED2 with polymorphic sets.

6.5.4. Definition. The system of *second order predicate logic on polymorphic domains*, λPRED2^p is defined by extending the system λPRED2 with the rule $(\mathsf{Type}^s, \mathsf{Set})$ (i.e. extending λPRED2 with polymorphic domains.) So λPRED2^p is the following PTS_β.

$$\begin{array}{rcl}
\mathcal{S} & = & \mathsf{Prop}, \mathsf{Set}, \mathsf{Type}^p, \mathsf{Type}^s,\\
\mathcal{A} & = & \mathsf{Prop} : \mathsf{Type}^p, \mathsf{Set} : \mathsf{Type}^s,\\
\mathcal{R} & = & (\mathsf{Set}, \mathsf{Set}), (\mathsf{Type}^s, \mathsf{Set}), (\mathsf{Set}, \mathsf{Type}^p),\\
 & = & (\mathsf{Prop}, \mathsf{Prop}), (\mathsf{Set}, \mathsf{Prop}), (\mathsf{Type}^p, \mathsf{Prop}).
\end{array}$$

So now for example

$$\mathrm{Nat} := \Pi\alpha{:}\mathsf{Set}.\alpha{\to}(\alpha{\to}\alpha){\to}\alpha$$

is a basic domain. Similarly all the definable data types of the polymorphic lambda calculus are definable as sets in the system λPRED2^p.

The system λPRED2^p is still a logic in the sense that there is a separation between domains, terms (among which are the propositions) and proofs. We can prove a proposition similar to Proposition 4.3.6 for λPRED2^p, which states this fact that the system is built up in stages.

6.5.5. Proposition. *In* λPRED2^p *we have the following. If* $\Gamma \vdash M : A$ *then* $\Gamma_D, \Gamma_T, \Gamma_P \vdash M : A$ *with*

- $\Gamma_D, \Gamma_T, \Gamma_P$ *is a permutation of* Γ,

- Γ_D *only contains declarations of the form* $x : \mathsf{Set}$,

- Γ_T *only contains declarations of the form* $x : A$ *with* $\Gamma_D \vdash A : \mathsf{Set}/\mathsf{Type}^p$,

- Γ_P *only contains declarations of the form* $x : \varphi$ *with* $\Gamma_D, \Gamma_T \vdash \varphi : \mathsf{Prop}$,

- *if* $A \equiv \mathsf{Set}/\mathsf{Type}^p$, *then* $\Gamma_D \vdash M : A$,

- *if* $\Gamma \vdash A : \mathsf{Set}/\mathsf{Type}^p$, *then* $\Gamma_D, \Gamma_T \vdash M : A$.

The system λPRED2 is a subsystem of λPRED2^p and the PTS-morphism H is still an embedding from λPRED2^p into λP2. (Hence λPRED2^p is consistent due to the consistency of λP2.) We have introduced λPRED2^p as a system in between λPRED2 and λP2, because our argument already holds for λPRED2^p , which is more readily understood as λP2.

A straightforward semantics for λPRED2^p is given by an arbitrary model for the polymorphic lambda calculus (to interpret the Set-part) with a second order predicate logic on top of it (giving the Prop-part for example the Tarskian semantics). An arbitrary model for the polymorphic lambda calculus has a lot of specific structure and this may raise the question whether λPRED2^p is conservative over λPRED2. We don't have a definite answer to this, but we do have reasons to believe that the extension is not conservative. The idea comes from [Berardi 1990a].

Look at the context

$$\Gamma := A{:}\mathsf{Set}, a, a'{:}A, z{:}a \neq_A a',$$

which describes a similarity type in the logic. In λPRED2 this similarity type has a finite model (without going into details about models, it will be clear that if we take for A the two element set, for $A{\rightarrow}A$ the set-theoretic function space, for $A{\rightarrow}\mathsf{Prop}$ the set of subsets of A and so forth, this yields a model.) If we now look at a model for the similarity type Γ in λPRED2^p, we see that there are a lot of new domains (types of type Set) which will have an interpretation in the model as well. For example the domain Nat $:= \Pi\alpha{:}\mathsf{Set}.\alpha{\rightarrow}(\alpha{\rightarrow}\alpha){\rightarrow}\alpha$. In case of an empty similarity type, Nat could consistently be interpreted by a one element set (because $Z \neq SZ$ is not provable in λPRED2^p in the empty context). In the similarity type Γ however, the interpretation of Nat has to be an infinite set, which makes it impossible for Γ to have a finite model in λPRED2^p. The point is that from $a \neq a'$ one can prove $Z \neq SZ$ and hence $S^n(Z) \neq S^{n+1}(Z)$ (for all n), viz. Suppose $Z = SZ$, then $ZAa(\lambda x{:}A.a') =_A SZAa(\lambda x{:}A.a')$ so $a =_A a'$, quod non.

6.5.6. FACT (Berardi). The similarity type (context)

$$\Gamma := A{:}\mathsf{Set}, a, a'{:}A, z{:}a \neq_A a'$$

has a finite model in λPRED2 but no finite model in λPRED2^p.

We want to stress here that we don't know how to use this fact (syntactically or semantically) to show the non-conservativity; it may still be possible that, although Γ has essentially only infinite models in λPRED2^p, it still doesn't prove more λPRED2-propositions then those alraedy provable in λPRED2 from Γ. It is easily seen though, that if λPRED2^p is not conservative over λPRED2, then also the formulas-as-types embedding from second order predicate logic into λP2 is incomplete.

Now we want to show the completeness of the formulas-as-types embedding from first order predicte logic (PRED) into λP. We do this by showing completeness of the PTS-morphism H from λPRED to λP. As remarked in Chapter 2.1, the system PRED is on the one hand minimal (we only have $\supset$ and $\forall$), but on the other hand it has some extra features like higher order functions and λ-definable predicates that do not belong to the realm of 'standard' first order predicate logic that we have called PRED^{-fr} in Definition 2.3.9. We are actually interested in the completeness of the embedding of PRED^{-fr} into λP. That it is sufficient to study the mapping H is shown by Proposition 2.3.8 and Corollary 2.3.11 that establish the conservativity of PRED over PRED^{-fr}.

As has been pointed out already, the system PRED is too minimal to be of real interest for practical mathematics, also because a system like λP is usually seen as a logical framework (like LF or AUT-68 that we discussed in Chapter 3.1) However, the completeness result can be extended a little bit to systems with a bottom type. We are then considering the formulas-as-types embedding from PRED$^\perp$ to λP$^\perp$, where PRED$^\perp$ is the system defined in 2.2.14 and λP$^\perp$ is λP extended with a constant type $\perp : \star$ and a constant term $\mathcal{E}_\perp$ with an extra rule

$$\frac{\Gamma \vdash M : \perp \quad \Gamma \vdash A : \star}{\Gamma \vdash \mathcal{E}_\perp M A : A}$$

The system PRED$^\perp$ is more interesting because the full classical first order predicate logic is a subsystem of it. More precisely, there is a faithful embedding of classical first order predicate logic into PRED$^\perp$ by a double negation translation. The embedding of classical first order predicate logic in to λP$^\perp$ via the system PRED$^\perp$ is now complete, due to the completeness of the embedding of PRED$^\perp$ into λP$^\perp$.

We now give the technical details of the proof of completeness of $H : \lambda\text{PRED} \rightarrow \lambda\text{P}$. In [Barendsen and Geuvers 1989] this proof appears in a slightly different form. The proof uses techniques developped in [Swaen 1989] to show completeness of the formulas-as-types embedding from first order predicate logic into

Martin-Löf's intuitionistic theory of types. A different proof of the same result can be found in [Berardi 1990].

Following Proposition 6.5.5 (which also holds for λPRED), we can write any context Γ of λPRED in the format

$$\Gamma_D, \Gamma_T, \Gamma_P \vdash M : A$$

where

- $\Gamma_D, \Gamma_T, \Gamma_P$ is a permutation of Γ,
- Γ_D only contains declarations of the form $x : \mathsf{Set}$,
- Γ_T only contains declarations of the form $x : A$ with $\Gamma_D \vdash A : \mathsf{Set}/\mathsf{Type}^p$,
- Γ_P only contains declarations of the form $x : \varphi$ with $\Gamma_D, \Gamma_T \vdash \varphi : \mathsf{Prop}$.

Then, if $\Gamma \vdash M : A$, we have

- if $A \equiv \mathsf{Set}/\mathsf{Type}^p$, then $\Gamma_D \vdash M : A$,
- if $\Gamma \vdash A : \mathsf{Set}/\mathsf{Type}^p$, then $\Gamma_D, \Gamma_T \vdash M : A$.

We shall refer to Γ_D a *set-context*, to Γ_T as an *object-context*, to Γ_P as a *proof-context* and to the concatenation Γ_D, Γ_T as a *language-context*.

The question of completeness is whether for any λPRED-context $\Gamma_D, \Gamma_T, \Gamma_P$ and proposition φ with $\Gamma_D, \Gamma_T \vdash \varphi : \mathsf{Prop}$, if

$$H(\Gamma_D, \Gamma_T, \Gamma_P) \vdash M : H(\varphi) \text{ in } \lambda\text{P},$$

then there exists a term N with

$$\Gamma_D, \Gamma_T, \Gamma_P \vdash N : \varphi \text{ in } \lambda\text{PRED}.$$

In the following we assume for any λPRED-context Γ that

1. $\Gamma \equiv \Gamma_D, \Gamma_T, \Gamma_P$
2. Γ_D is not empty,
3. all declared sets in Γ_D are nonempty
4. Γ_T begins with a declaration β:Prop and Γ_P begins with z:β.

The third and fourth clause are added for convenience, we shall refer to the β:Prop with z:β as True. In case there are empty domains in the logic, the completeness result would still hold with a slightly adapted argument. If the second were not satisfied we would in fact be working in propositional logic. The clause has as a consequence that we can always refer to 'the first declaration of a set variable in Γ'. For this set variable we choose a fixed name 0, so we may in the following always assume that $0 : \mathsf{Set}$ is the first declaration of the λPRED-context Γ.

6.5.7. DEFINITION. For Γ_D, Γ_T a language-context and Δ a context of λP, we say that Δ *is an elementary extension of* $H(\Gamma_D, \Gamma_T)$, notation $H(\Gamma_D, \Gamma_T) \Subset \Delta$ if $\Delta \supseteq H(\Gamma_D, \Gamma_T)$ and the extra declarations in Δ are all of the form $x{:}\sigma$ with $H(\Gamma_D, \Gamma_T) \vdash \sigma : \star$ in λP.

For example, $H(\Gamma_D, \Gamma_T, \Gamma_P)$ is always an elementary extension of $H(\Gamma_D, \Gamma_T)$. We now define a mapping $| - |^p$ from λP to the objevt language of λPRED

6.5.8. DEFINITION. The mapping $| - |^p$ from terms of λP to terms of λPRED is defined as follows.

$$
\begin{array}{rrcl}
(i) & |\star|^p & := & \mathsf{Set}, \\
(ii) & |\Box|^p & := & \mathsf{Type}^s, \\
(iii) & |x|^p & := & 0, \text{ if } x \text{ is a variable of type } \cdots \to \star, \\
(iv) & |x|^p & := & x, \text{ for } x \text{ another variable}, \\
(v) & |\Pi x{:}A.B|^p & := & |B|^p \text{ if } A{:}\star, B{:}\Box, \\
 & & := & \Pi x{:}|A|^p.|B|^p \text{ else}, \\
(vi) & |\lambda x{:}A.M|^p & = & |M|^p \text{ if } A{:}\star, M{:}B{:}\Box, \text{ (for some } B), \\
 & & = & \lambda x{:}|A|^p.|B|^p \text{ else}, \\
(vii) & |PM|^p & = & |P|^p \text{ if } M{:}A{:}\star, P{:}B{:}\Box, \text{ (for some } A, B), \\
 & & = & |P|^p|M|^p \text{ else}
\end{array}
$$

The definition extends immediately to contexts of λP, where a declaration of the form $x : \cdots \to \star$ is removed.

That the mapping $| - |^p$ is indeed from λP to λPRED is justified by the following Proposition.

6.5.9. PROPOSITION.

$$\Delta \vdash M : A \ (in\ \lambda P) \Rightarrow |\Delta|^p \vdash |M|^p : |A|^p.$$

PROOF. By induction on the derivation of $\Delta \vdash M : A$ in λP. ⊠

6.5.10. FACT. If $\Gamma_D, \Gamma_T \vdash M : A(: \mathsf{Set})$, then $|H(A)|^p \equiv A$ and $|H(M)|^p \equiv M$. (Note that H is the identity on these kind of terms.)

6.5.11. COROLLARY. *For* $\Delta \Supset H(\Gamma_D, \Gamma_T)$, *say* $\Delta \equiv H(\Gamma_D, \Gamma_T), \Delta'$ *we have*

$$\Delta \vdash M : A(: \star) \Rightarrow \Gamma_D, \Gamma_T, |\Delta'|^p \vdash |M|^p : |A|^p.$$

PROOF. Immediate by the fact that $|H(\Gamma_D)|^p \equiv \Gamma_D$ and for a declaration $x : A$ in Γ_T, if $A{:}\mathsf{Set}$, then $|x{:}A|^p \equiv x{:}A$ and if $A{:}\mathsf{Type}^p$, then $|x|^p \equiv 0$ (and in that case this declaration doesn't play a role anymore). ⊠

All this means that $|-|^p$ is a mapping back from terms of λP to the object-language of λPRED that does not change the terms that originated from the object-language.

Now we define a mapping back from λP to the proof-language of λPRED, so now types in λP will become propositions and objects will become proofs of λPRED.

6.5.12. DEFINITION. Let $\Delta \Supset H(\Gamma_D, \Gamma_T)$. The map Tr on constructors of λP in Δ is defined as follows.

$$\begin{array}{rrcl}
(i) & \mathsf{Tr}(\alpha) & := & \text{True, if } \alpha{:}\mathsf{Set} \in \Gamma_D,\\
(ii) & \mathsf{Tr}(\alpha) & := & \alpha, \text{ if } \alpha{:}\cdots\to\mathsf{Prop} \in \Gamma_T,\\
(iii) & \mathsf{Tr}(\lambda x{:}A.M) & := & \lambda x{:}|A|^p.\mathsf{Tr}(M),\\
(iv) & \mathsf{Tr}(Qt) & := & \mathsf{Tr}(Q)|t|^p,\\
(v) & \mathsf{Tr}(\Pi x{:}A.B) & := & \Pi x{:}|A|^p.\mathsf{Tr}(A)\to\mathsf{Tr}(B).
\end{array}$$

6.5.13. PROPOSITION. *For $\Delta \Supset H(\Gamma_D, \Gamma_T)$, say $\Delta \equiv H(\Gamma_D, \Gamma_T), \Delta'$ we have*

$$\begin{array}{rl}
& \Delta \vdash C : \Pi x_1{:}A_1.\ldots\Pi x_n{:}A_n.\star \ \textit{ in } \lambda P\\
\Rightarrow & \Gamma_D, \Gamma_T, |\Delta'|^p \vdash \mathsf{Tr}(C) : |A_1|^p\to\cdots\to|A_n|^p\to\mathsf{Prop} \textit{ in } \lambda\text{PRED}.
\end{array}$$

PROOF. By induction on the derivation. Note that if $A{:}\star$ in λP, then $|A|^p$ contains no object-variables. Furthermore, if $\Delta \vdash M : A(:\star)$, then $\Gamma_D, \Gamma_T, |\Delta'|^p \vdash |M|^p : |A|^p$ by Corollary 6.5.11. ⊠

6.5.14. COROLLARY. *For $\Delta \Supset H(\Gamma_D, \Gamma_T)$, say $\Delta \equiv H(\Gamma_D, \Gamma_T), \Delta'$ we have*

$$\Delta \vdash A : \star \textit{ in } \lambda P \Rightarrow \Gamma_D, \Gamma_T, |\Delta'|^p \vdash \mathsf{Tr}(A) : \mathsf{Prop} \textit{ in } \lambda\text{PRED}.$$

6.5.15. LEMMA. *If $\Gamma_D \vdash A : \mathsf{Set}$ in λPRED, then*

$$\exists M[\Gamma_D, \Gamma_T, \Gamma_P \vdash M : \mathit{True} \leftrightarrow \mathsf{Tr}(A)] \quad \textit{in } \lambda\text{PRED}.$$

(To be precise we would have to write $\mathsf{Tr}(H(A))$ in stead of $\mathsf{Tr}(A)$, but H is the identity on terms of type Set.)

PROOF. Immediate from the definition of Tr. ⊠

6.5.16. LEMMA. *For $\Delta \Supset H(\Gamma_D, \Gamma_T)$, say $\Delta \equiv H(\Gamma_D, \Gamma_T), \Delta'$, with $\Delta \vdash A, B : \star$ and $\Delta \vdash t : B$ we have*

$$\mathsf{Tr}(A)[|t|^p/x] \equiv \mathsf{Tr}(A[t/x])$$

and if $A =_\beta A'$,then

$$\exists M[\Gamma_D, \Gamma_T, |\Delta'|^p \vdash M : \mathsf{Tr}(A) \leftrightarrow \mathsf{Tr}(A')] \textit{ in } \lambda\text{PRED}.$$

PROOF. The first is easily proved by induction on the structure of A. The second follows from the fact that $\mathsf{Tr}(A) =_\beta \mathsf{Tr}(A')$, which is justified by the first and the Church-Rosser property. ⊠

6.5.17. PROPOSITION. *For each language-context* Γ_D, Γ_T *and* φ *with* $\Gamma_D, \Gamma_T \vdash \varphi : \mathsf{Prop}$ *we have*

$$\exists M[\Gamma_D, \Gamma_T \vdash M : \varphi \leftrightarrow \mathsf{Tr}(H(\varphi)).$$

(Note that H *is the identity on expressioons of type* Prop*, so we can skip it.)*

PROOF. By induction on the structure of φ. By Lemma 6.5.16 we may assume that φ is in normal form.

(base) If $\varphi \equiv \alpha t_1 \cdots t_n$ with α a variable, then $\mathsf{Tr}(\varphi) \equiv \varphi$ by the fact that $|t_i|^p \equiv t_i$. (Fact 6.5.10.)

($\supset$) Say $\varphi \equiv \psi \to \chi$ with $\psi, \chi{:}\mathsf{Prop}$. Then $\mathsf{Tr}(\varphi \to \psi) \equiv \forall x{:}|\varphi|^p.\mathsf{Tr}(\varphi) \to \mathsf{Tr}(\psi)$. Now we are done by IH: The variable x will not occur free in $\varphi \to \psi$ and one easily constructs the required derivation trees.

($\forall$) Say $\varphi \equiv \Pi x{:}A.\psi$ with $A : \mathsf{Set}$. Then $\mathsf{Tr}(\Pi x{:}A.\psi) \equiv \Pi x{:}|A|^p.\mathsf{Tr}(A) \to \mathsf{Tr}(\psi)$. Now by Fact 6.5.10 and Lemma 6.5.15, $\Pi x{:}|A|^p.\mathsf{Tr}(A) \to \mathsf{Tr}(\psi)$ is equivalent to $\Pi x{:}A.\mathsf{Tr}(\psi)$, so we are done by IH. ⊠

6.5.18. DEFINITION. For $\Delta \Supset H(\Gamma_D, \Gamma_T)$, say $\Delta \equiv H(\Gamma_D, \Gamma_T), \Delta'$, we define the context $\mathsf{TR}(\Delta)$ as

$$\mathsf{TR}(\Delta) := \Gamma_D, \Gamma_T, |\Delta|^p, \mathsf{Tr}(\Delta),$$

where $\mathsf{Tr}(\Delta)$ is defined by replacing every declaration $z{:}A$ in Δ' by $z' : \mathsf{Tr}(A)$. (We have to make sure that the declared variables in $\mathsf{Tr}(\Delta)$ are different from the ones in $|\Delta|^p$.)

6.5.19. PROPOSITION. *Let* $\Delta \Supset \Gamma_D, \Gamma_T$*, then*

$$\Delta \vdash M : A(:\star) \text{ in } \lambda P \Rightarrow \exists N[\mathsf{TR}(\Delta) \vdash N : \mathsf{Tr}(A)] \text{ in } \lambda\text{PRED}.$$

PROOF. By induction on the derivation of $\Delta \vdash M : A$ in λP.

(var) $M \equiv x$ then either $x{:}A$ in Γ_T or in Δ'. In the first case $\mathsf{Tr}(A) \leftrightarrow$ True and in the second case $x{:}\mathsf{Tr}(A) \in \mathsf{TR}(\Delta)$.

(app) Say

$$\frac{\Delta \vdash M : \Pi x{:}A.B \quad \Delta \vdash t : A}{\Delta \vdash Mt : B[t/x]}$$

By IH, $\mathsf{TR}(\Delta) \vdash N : \mathsf{Tr}(\Pi x{:}A.B) \equiv \Pi x{:}|A|^p.\mathsf{Tr}(A) \to \mathsf{Tr}(B)$ and $\mathsf{TR}(\Delta) \vdash Q : \mathsf{Tr}(A)$. We also have $\mathsf{TR}(\Delta) \vdash |t|^p : |A|^p$, by Corollary 6.5.11. So we may conclude $\mathsf{TR}(\Delta) \vdash N|t|^p Q : \mathsf{Tr}(B)[|t|^p/x] \equiv \mathsf{Tr}(B[t/x])$.

(λ) Say

$$\frac{\Delta, x{:}B \vdash M : C \quad \Delta \vdash \Pi x{:}B.C : \star}{\Delta \vdash \lambda x{:}B.M : \Pi x{:}B.C}$$

By IH, $\mathsf{TR}(\Delta, x{:}B) \vdash N : \mathsf{Tr}(C)$. $\mathsf{TR}(\Delta, x{:}B) \equiv \mathsf{TR}(\Delta), x{:}|B|^p, x'{:}\mathsf{Tr}(B)$, so we have

$$\mathsf{TR}(\Delta) \vdash \lambda x{:}|B|^p.\lambda x'{:}\mathsf{Tr}(B).N : \Pi x{:}|B|^p.\mathsf{Tr}(B) \rightarrow \mathsf{Tr}(C) \equiv \mathsf{Tr}(\Pi x{:}B.C).$$

(conv) We are immediately done by Lemma 6.5.16. $\boxtimes$

6.5.20. Corollary. *The embedding H from λPRED into λP is complete, i.e. if Γ_D, Γ_T is a language-context with $\Gamma_D, \Gamma_T \vdash \varphi : \mathsf{Prop}$ and Γ_P a proof-context, then*

$$H(\Gamma_D, \Gamma_T, \Gamma_P) \vdash M : H(\varphi) \textit{ in } \lambda P \Rightarrow \exists N[\Gamma_D, \Gamma_T, \Gamma_P \vdash N : \varphi \textit{ in } \lambda\text{PRED}.$$

Proof. $H(\Gamma_D, \Gamma_T, \Gamma_P)$ is an elementary extension of Γ_D, Γ_T, so by the Proposition we have

$$\Gamma_D, \Gamma_T, |\Gamma_P|^p, \mathsf{Tr}(\Gamma_P) \vdash N : \mathsf{Tr}(\varphi)$$

for some term N. Now all declarations in $|\Gamma_P|^p$ are of the form $y : B$ where $B : \mathsf{Set}$, so we can substitute other terms for each of these variables. Furthermore, for every B for which $y'{:}B \in \mathsf{Tr}(\Gamma_P)$ we have $\exists M.\Gamma_D, \Gamma_T \vdash M : B \leftrightarrow \mathsf{Tr}(B)$ by Proposition 6.5.17. So we can replace each $y'{:}\mathsf{Tr}(B)$ by $y''{:}B$, at the same time substituting My'' for y' inside N. (These variables do not occur in $\mathsf{Tr}(\varphi)$.) We obtain a term N' with

$$\Gamma_D, \Gamma_T, \Gamma_P \vdash N' : \varphi.$$

By again applying Proposition6.5.17,we can transform this N' into a N'' with

$$\Gamma_D, \Gamma_T, \Gamma_P \vdash N'' : \varphi. \boxtimes$$

6.5.3. Conservativity relations inside the cube

We now want to address the question of conservativity inside the cube of typed lambda calculi and the logic cube. We first look at the cube of typed lambda calculi, because the situation for the logic cube is very similar. There are four results that do the whole job, resulting in the following picture.

$$
\begin{tikzcd}
 & \lambda\omega \arrow[rr] & & \lambda \mathrm{P}\omega \\
\lambda 2 \arrow[ur] \arrow[rr] & & \lambda \mathrm{P}2 \arrow[ur, dotted] & \\
 & \lambda\underline{\omega} \arrow[uu] \arrow[rr] & & \lambda \mathrm{P}\underline{\omega} \arrow[uu] \\
\lambda{\rightarrow} \arrow[uu] \arrow[ur] \arrow[rr] & & \lambda \mathrm{P} \arrow[uu] \arrow[ur] &
\end{tikzcd}
$$

where an arrow denotes a conservative inclusion and a dotted arrow denotes a non-conservative inclusion. By transitivity of conservativity (if system 3 is conservative over system 2 and system 2 is conservative over system 1, then system 3 is conservative over system 1), it is no problem to fill in the picture further. (Draw the arrows between two non-adjacent systems) We can collect all this in the following Proposition.

6.5.21. PROPOSITION. *For S_1 and S_2 two systems in the cube of typed lambda calculi such that $S_1 \subseteq S_2$:*

$$S_2 \textit{ is conservative over } S_1 \Leftrightarrow S_2 \neq \lambda P\omega \ \& \ S_1 \neq \lambda P2.$$

PROOF. It suffices to prove the following four results.

1. If $S_2 \supseteq S_1$, S_1 a system of the lower plane in the cube, then S_2 is conservative over S_1.(Proposition 6.5.22.)

2. If S_2 a system in the right plane of the cube, S_1 the adjacent system in the left plane, then S_2 is conservative over S_1.(Proposition 6.5.25.)

3. $\lambda\mathrm{P}\omega$ is not conservative over $\lambda\mathrm{P}2$,

4. $\lambda\omega$ is conservative over $\lambda 2$. (Corollary 2.4.27.)

The fourth is a consequence of Corollary 2.4.27, saying that PROPω is conservative over PROP2 and of the fact that PROPω and PROP2 are isomorphic to, respectively, $\lambda\omega$ and $\lambda 2$ via the formulas-as-types embedding. (See paragraph 4.3.1 and especially Proposition 4.3.4.) The third was verfied in detail by

[Ruys 1991], following an idea from Berardi. The idea is to look at a context Γ in λP2 that represents Arithmetic. Then Γ with λP2 is as strong as second order Arithmetic and Γ with $\lambda P\omega$ is as strong as higher order Arithmetic. Hence we can use Gödel's Second Incompleteness Theorem to show that in λP2 one can not derive from Γ that Γ is consistent in λP2. On the other hand in $\lambda P\omega$ one can derive from Γ that Γ is consistent in λP2. Hence the non-conservativity. ⊠

We first prove the Proposition about conservativity of systems over systems in the lower plane. The Proposition was also proved in [Verschuren 1990] in a slightly different way.

6.5.22. PROPOSITION. *Let S_1 be a system of the lower plane and S_2 be any system of the cube such that $S_1 \subseteq S_2$. Then*

$$\left.\begin{array}{r}\Gamma \vdash_{S_1} B : \star \\ \Gamma \vdash_{S_2} M : B \\ \Gamma \text{ and } M \text{ in normal form}\end{array}\right\} \Rightarrow \Gamma \vdash_{S_1} M : B.$$

PROOF. By induction on the structure of M.

applic. Say $M \equiv xP_1 \cdots P_n$. Then $x{:}\Pi y_1{:}C_1.D_1 \in \Gamma$, so

$$\begin{array}{lll}\Gamma & \vdash_{S_1} & C_1 : \star, \\ \Gamma & \vdash_{S_2} & P_1 : C_1.\end{array}$$

Now by IH, $\Gamma \vdash_{S_1} P_1 : C_1$, so $\Gamma \vdash_{S_1} xP_1 : D_1[P_1/y_1]$. We can now go further with P_2: We know that $D_1[P_1/y_1] \twoheadrightarrow_\beta \Pi y_2{:}C_2.D_2$ with

$$\Gamma \vdash_{S_1} C_2 : \star.$$

Also

$$\Gamma \vdash_{S_2} P_2 : C_2,$$

so again by IH $\Gamma \vdash_{S_1} P_2 : C_2$ and hence $\Gamma \vdash_{S_1} xP_1P_2 : D_2[P_2/y_2]$. Continuing in this way upto n we find that $\Gamma \vdash_{S_1} xP_1 \cdots P_n : D_n[P_n/y_n]$ with $D_n[P_n/y_n] = B$. Now by one application of the conversion rule (using $\Gamma \vdash_{S_1} B : \star$) we conclude $\Gamma \vdash xP_1 \cdots P_n : B$.

abstr. Say $M \equiv \lambda x{:}A.N$. Then $B \longrightarrow_\beta \Pi x{:}A.C$ (because A in normal form). So $\Gamma \vdash_{S_1} \Pi x{:}A.C : \star$ and $\Gamma, x{:}A \vdash_{S_2} M : C$ (by Stripping and the conversion rule). We can apply IH to conclude $\Gamma, x{:}A \vdash_{S_1} M : C$. Now we are done: By one λ-abstraction and one conversion we conclude $\Gamma \vdash_{S_1} \lambda x{:}AM : B$. ⊠

The side condition Γ in normal form has just been added for convenience (in giving the proof.) It is not essential and it may be dropped.

We now prove the conservativity of the right plane over the left plane. The idea is to define a mapping that removes all type dependencies. This mapping will go from a system in the right plane to the adjacent system in the left plane and is the identity on terms that are already well-typed in the left plane. Hence the conservativity. The proof is originally independently due to [Paulin 1989] and [Berardi 1990]. The first described the mapping from $\lambda\mathrm{P}\omega$ to $\lambda\omega$ in the first place to use it for program extraction; the second described the collection of four mappings (which is a straightforward generalisation of the mapping from $\lambda\mathrm{P}\omega$ to $\lambda\omega$) to give a conservativity proof. The mappings are very much related to similar mappings one can define from predicate logic to proposition logic to prove conservativity of the first over the second.

6.5.23. Definition ([Paulin 1989] and [Berardi 1990]). Let S_2 be a system of the right plane and S_1 the adjacent system in the left plane. The mapping $[-] : \mathsf{Term}(S_2) \to \mathsf{Term}(S_1)$ is defined as follows.

$$\begin{array}{rcl}
[\Box] & = & \Box, \\
[\star] & = & \star, \\
[x] & = & x, \text{ for } x \text{ a variable}, \\
[\Pi x{:}A.B] & = & [B] \text{ if } A{:}\star, B{:}\Box, \\
 & = & \Pi x{:}[A].[B] \text{ else}, \\
[\lambda x{:}A.M] & = & [M] \text{ if } A{:}\star, M{:}B{:}\Box, \text{ (for some } B), \\
 & = & \lambda x{:}[A].[M] \text{ else}, \\
[PM] & = & [P] \text{ if } M{:}A{:}\star, P{:}B{:}\Box, \text{ (for some } A, B), \\
 & = & [P][M] \text{ else},
\end{array}$$

6.5.24. Remark. The side conditions in the defintion are justified by the Classification Lemma (4.4.37). We could also have distinguished cases according to the heart or the level of subterms. (See Lemma 6.3.2 and Lemma 6.3.4.)

The mapping $[-]$ extends straightforwardly to contexts. The following justifies the statement in the definition that the mapping $[-]$ goes from the right plane to the left plane.

6.5.25. Proposition. *([Paulin 1989],[Berardi 1990]) Let S_2 be a system in the right plane and S_1 the adjacent system in the left plane of the cube.*

$$\Gamma \vdash_{S_2} M : A \Rightarrow [\Gamma] \vdash_{S_1} [M] : [A]$$

Proof. By a straightforward induction on the derivation of $\Gamma \vdash_{S_2} M : A$. $\boxtimes$

6.5.26. Corollary. *([Paulin 1989],[Berardi 1990]) For S_2 a system in the right plane and S_1 the adjacent system in the left plane of the cube we have*

$$S_2 \text{ is conservative over } S_1.$$

Proof. The only thing to check is that for $M \in \mathsf{Term}(S_1)$, $[M] \equiv M$. This is done by an easy induction on the structure of M. ⊠

The conservativity relations in the logic cube (Definition 4.3.5) are as follows. (An arrow denotes a conservative extension, a dotted arrow a non-conservative extension.)

$$\begin{tikzcd}
 & \lambda\mathrm{PROP}\omega \arrow[rr] & & \lambda\mathrm{PRED}\omega \\
\lambda\mathrm{PROP2} \arrow[ur] \arrow[rr] & & \lambda\mathrm{PRED2} \arrow[ur, dotted] & \\
 & \lambda\mathrm{PROP}\overline{\omega} \arrow[uu] \arrow[rr] & & \lambda\mathrm{PRED}\overline{\omega} \arrow[uu] \\
\lambda\mathrm{PROP} \arrow[uu] \arrow[ur] \arrow[rr] & & \lambda\mathrm{PRED} \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

6.5.27. Proposition. *For S_1 and S_2 two systems in the logic cube such that $S_1 \subseteq S_2$:*

$$S_2 \text{ is conservative over } S_1 \Leftrightarrow S_2 \neq \lambda\mathrm{PRED}\omega \ \&\ S_1 \neq \lambda\mathrm{PRED2}.$$

Proof. Completely analoguous to the proof for the cube of typed lambda calculi, of Proposition 6.5.21. ⊠

In Chapter 2.1 we also discussed first order predicate logic with (PRED) and without (PRED^{-f}) functional domains. We stated a conservativity result of PRED over PRED^{-f} in Proposition 2.3.8. In Chapter 4.1 we saw that λPRED corresponds to PRED and we also defined the system $\lambda\mathrm{PRED}^{-f}$ that corresponds to PRED^{-f} (Definition 4.3.7). The conservativity of PRED over PRED^{-f} can now easily be stated and proved in terms of typed lambda calculi. Let therefore

$H' : \lambda\text{PRED}^{-f} \to \lambda\text{PRED}$ be the PTS-morphism defined by

$$\begin{array}{rcl} H'(\mathsf{Set}) & = & \mathsf{Set}, \\ H'(\mathsf{Fun}) & = & \mathsf{Set}, \\ H'(\mathsf{Prop}) & = & \mathsf{Prop}, \\ H'(\mathsf{Type}^p) & = & \mathsf{Type}^p, \\ H'(\mathsf{Type}^s) & = & \mathsf{Type}^s. \end{array}$$

It is easy to verify that H' is almost the identity: for M a term of λPRED^{-f}, if $M \not\equiv \mathsf{Fun}$, then $H'(M) \equiv M$. We have the following. Compare with Proposition 2.3.8.

6.5.28. Proposition. *For* Γ *a context and* $\sigma; \mathsf{Prop}$ *in* λPRED^{-f},

$$\Gamma \vdash_{\text{PRED}} M : \varphi \;\Rightarrow\; nf(\Gamma) \vdash_{\text{PRED}^{-f}} nf(M) : nf(\varphi).$$

So the embedding H' *is complete with respect to provability and* PRED *is conservative over* PRED^{-f}.

Proof. By induction on the derivation. ⊠

6.6. Consistency of (contexts of) CC

As the embedding H from $\lambda\text{PRED}\omega$ into CC is not complete (CC proves more propositions than $\lambda\text{PRED}\omega$), one may wonder whether there are propositions that CC can not prove, or to pose the question differently, is CC consistent? That this is the case can be shown quite easily by giving a two-point model for CC. (See [Coquand 1990].) The type $\star$ is interpreted as $\{\emptyset, \{\emptyset\}\}$ (or $\{0, 1\}$ in the language of ZF) and if $\vdash M : A$, the interpretation of M is in the set A. This model is also called the 'proof-irrelevance' model (e.g. in [Coquand 1990]) because in the model all proofs of a proposition are mapped to the same element 0. So the model also implies that

$$\neg\exists M[\vdash M : a \neq_A a' \text{ for } \vdash a, a' : A].$$

The interpretation will be such that the proposition $\bot (:\equiv \Pi\alpha{:}\mathsf{Prop}.\alpha)$ is interpreted by 0, so

$$\neg\exists M[\vdash M : \bot],$$

that is, CC doesn't prove $\bot$. We shall make the model construction precise here. It is in fact a model construction for $\lambda\omega$. Using the mapping $[-]$ of Definition 6.5.23, we find that it is also a model for CC. So the consistency of CC follows from the consistency of higher order propositional logic and the conservativity of CC over $\lambda\omega$. (Proposition 6.5.21.) It is not so easy to construct the model immediately for CC, a problem that is solved in [Coquand 1990] by describing the model for a variant of CC. Here we use the mapping $[-]$ from CC to $\lambda\omega$ for this purpose.

Before constructing the model we want to recall some properties of $\lambda\omega$ that will be used. They have already been stated in Proposition 4.3.4. First, the set of kinds of $\lambda\omega$ (those terms A for which $\Gamma \vdash A : \Box$ for some Γ) can be described by K, where

$$K ::= \star, \mid K \to K.$$

Second, no proposition-variables are subterms of propositions or constructors, i.e.

$$\Gamma \vdash M : A : \mathsf{Kind} \Rightarrow \Gamma' \vdash M : A : \mathsf{Kind},$$

where Γ' consists just of those declarations $x{:}B$ in Γ for which $\Gamma \vdash B : \mathsf{Kind}$.

These two properties imply that we can build the interpretation in three stages by first giving a meaning to the kinds, then to the types and constructors and then to the objects. Also recall that the variables are seperated into two sets, $\mathsf{Var}^\star$ for object-variables and $\mathsf{Var}^\Box$ for constructor-variables. The first will be denoted by Latin characters, the latter by Greek characters.

In general, an interpretation of terms of $\lambda\omega$ uses a valuation ξ of constructor-variables and a valuation ρ of proof-variables. In our simple model all free object-variables have the value 0, so we only need ξ. For convenience we think of contexts of $\lambda\omega$ as being split up in a $\Gamma^\Box$, containing the declarations of constructor variables, and a $\Gamma^\star$, containing the declarations of object-variables.

6.6.1. Definition. (i) The valuation ξ *satisfies* $\Gamma^\Box$ (notation $\xi \models \Gamma^\Box$) if for all $\alpha : A \in \Gamma^\Box$, $\xi(\alpha)$ is in the interpretation of A. ($A : \Box$, so A doesn't contain any free variables.)

(ii) The valuation ξ *satisfies* Γ (notation $\xi \models \Gamma$) if ξ satisfies $\Gamma^\Box$ and for all $x{:}A \in \Gamma^\star$, the interpretation of A under ξ is not empty. ($A : \star$, so A can only contain free constructor-variables.)

6.6.2. Definition. For $\Gamma \vdash M{:}A$ we define the interpretation function $[\![-]\!] : \mathsf{Term}(\lambda\omega) \to \text{Sets}$ as follows.

1. For types, $[\![\star]\!] = 2$ and $[\![k_1 \to k_2]\!] = [\![k_1]\!] \to [\![k_2]\!]$ (for $k_1,\ k_2 \in K$), where the latter arrow denotes set-theoretic function space.

2. For constructors, let ξ be a valuation of constructor-variables such that $\xi \models \Gamma_1$,

$$\begin{array}{rcl}
[\![\alpha]\!]_\xi &=& \xi(\alpha),\\
[\![\Pi x{:}A.B]\!]_\xi &=& 1 \text{ if } \forall a \in [\![A]\!][[\![B]\!]_{\xi(x:=a)} = 1],\\
&=& 0 \text{ else, (for } A : \Box, B : \star),\\
[\![A \to B]\!]_\xi &=& [\![A]\!]_\xi \to [\![B]\!]_\xi, \text{ (for } A, B : \star),\\
[\![PQ]\!]_\xi &=& [\![P]\!]_\xi [\![Q]\!]_\xi,\\
[\![\lambda\alpha{:}A.P]\!]_\xi &=& \boldsymbol{\lambda} a \in [\![A]\!]_\xi . [\![P]\!]_{\xi(x:=a)}.
\end{array}$$

3. All objects are interpreted as 0.

Here, $\boldsymbol{\lambda} a \in U.V(a)$ denotes a set-theoretic function. Further we identify all singleton sets (like e.g. $[\![A]\!]_\xi \to [\![A]\!]_\xi$) with 1 and we use the fact that no proof-variables occur in propositions.

By induction on derivations one can prove the following property.

6.6.3. PROPOSITION. *If* $\Gamma \vdash M : A$, *then for all valuations* ξ *with* $\xi \models \Gamma$, $[\![M]\!]_\xi \in [\![A]\!]_\xi$.

It is good to realise here that for example for $\Gamma = x{:}\bot (\equiv \Pi\alpha{:}\star.\alpha)$, there is no ξ with $\xi \models \Gamma$, so in this case the conclusion of the proposition is vacuously satisfied.

6.6.4. COROLLARY. $\lambda\omega$, *and hence CC, is consistent.*

PROOF. For all valuations ξ, $[\![\bot]\!]_\xi = 0$. All valuations satisfy the empty context, so if $\vdash M : \bot$, then $0 \in 0$, quod non. ⊠

One may wonder whether $\mathrm{EXT}' := \Pi\alpha, \beta{:}\star.(\alpha{\leftrightarrow}\beta) \to (\alpha =_\star \beta)$, is *consistent* in CC. That this is the case can be seen by using the proof-irrelevance model of Definition 6.6.2. The interpretation of EXT′ in the model is 1, so if CC would prove $\mathrm{EXT}'{\to}\bot$, CC itself would be inconsistent, quod non. The same argument applies to show that CC with classical logic is consistent. Define

$$\mathrm{CL} := \Pi\alpha : \star.\alpha \vee \neg\alpha.$$

Then

$$[\![\mathrm{CL}]\!] = 1,$$

so z : CL is a consistent context. A more interesting example is the Axiom of Choice. Let

$$\mathrm{AC} := \Pi P{:}A{\to}B{\to}\star.(\Pi x{:}A.\exists y{:}B.Pxy) \to (\exists f{:}A{\to}B.\Pi x{:}A.Px(fx)).$$

Applying the mapping of Definition 6.5.23 we obtain

$$[\mathrm{AC}] = \forall P{:}\star.(A{\to}B\&P) \to (A{\to}B)\&(A{\to}P).$$

Now [AC] is inhabited by a closed term in $\lambda\omega$, so AC is not inconsistent in CC (by the consistency of $\lambda\omega$.) Notice that in all these cases the proof of consistency of an assumption is done by giving a model in which the assumption is satisfied; for EXT and CL the proof-irrelevance model and for AC the system $\lambda\omega$.

In some (quite trivial) cases it is even possible to use CC itself as model: If the context Γ consists only of declarations $x : A$ with $A : \square$ or $A =_\beta zt_1 \ldots t_p$ with z a variable, then Γ is consistent. Contexts of this kind are called *strongly consistent* in [Seldin 1990].

6.6.5. Proposition ([Seldin 1990]). *Strongly consistent contexts of CC are consistent.*

Proof. Let $\Gamma = x_1{:}A_1, \ldots, x_n{:}A_n$ be a strongly consistent context and suppose that $\Gamma \vdash M : \bot$ for some M. Now we consecutively substitute closed terms for all free variables that are declared in Γ, such that all the assumed propositions become $\top(=\Pi\alpha.\alpha{\to}\alpha)$. It works as follows: if $x_i : A_i \in \Gamma$ with $\Gamma \vdash A_i : \Box$, then $A_i =_\beta \Pi\vec{y}{:}\vec{B}.\star$, (with $\mathrm{FV}(\vec{B}) \subseteq \{x_1, \ldots, x_{i-1}\}$) and we substitute $\lambda\vec{y}{:}\vec{B}^*.\top$ for x_i, where the B^* are the terms in which the substitution for $x_1, \ldots, x_{i-1}$ has already been done. If $x : zt_1 \ldots t_p(:\star)$ with z a variable, we substitute x by $\lambda\alpha{:}\star.\lambda x{:}\alpha.x$, which is of type $\top$. If we denote this substitution by *, we can conclude from $\Gamma \vdash M : \bot$ and the Substitution Lemma that $\vdash M^* : \bot$. So Γ is consistent by the consistency of CC. ⊠

The techniques described above to show that a context is consistent are not sufficient to handle the more interesting examples. For mere proof theoretic reasons it will for example not be possible to show the consistency of Γ_{HA} (defined in the second proof of Proposition 6.5.3) with these techniques: This would give us a first order consistency proof of higher order arithmetic. These kind of contexts have to be handled by a normalization argument: Assuming the inconsistency of Γ_{HA}, show that a proof of $\bot$ in Γ_{HA} can not be in normal form, and so there is no such proof. In [Seldin 1990] one can find a detailed proof of the consistency of a context that represents Peano Arithmetic in a system that is a slight extension of CC. Coquand shows in [Coquand 1990] by a normalization argument that the context

$$\begin{array}{rcl}\mathrm{INF} & = & A{:}\star, a{:}A, f{:}A{\to}A, R{:}A{\to}A{\to}\star \\ & & z_1{:}\forall x{:}A.(Rxx){\to}\bot, z_2{:}\forall x,y,z{:}A.Rxy{\to}Ryz{\to}Rxz, z_3{:}\forall x{:}A.Rx(fx)\end{array}$$

is consistent. When contexts become larger, a consistency proof by the normalization argument can of course get very involved. Semantics is then a very helpful tool for showing consistency and in general to show the non-derivability of a formula from a specific set of assumptions. Of course one has to use more interesting models then the one of 6.6.2 to establish this. In [Streicher 1991] there are some examples of this technique using realisability semantics.

Knowing that a certain context is consistent is of course not enough to use it safely for doing proofs. Due to the incompleteness of the formulas-as-types embedding, a well-understood context that is beyond suspicion in higher order predicate logic, may have unexpected side-effects when embedded in CC. Furthermore, CC has a greater expressibility then higher order predicate logic so we may also put in the context axioms that do have a meaning but can not be expressed in the logic. An example is given by the axiom of definite descriptions that makes a generic statement about all domains. It is described in [Pottinger 1989] as follows

$$\mathrm{DD} := \forall\alpha{:}\star.\forall P{:}\alpha{\to}\star.\forall z{:}(\exists! x{:}\alpha.Px).P(\iota\alpha Pz),$$

where

$$\exists! x{:}\alpha.Px := (\exists x{:}\alpha.Px)\&(\forall x,y{:}\alpha.Px \rightarrow Py \rightarrow (x =_\alpha y))$$

and ι is a term of type $\forall\alpha{:}\star.\forall P{:}\alpha\rightarrow\star.(\exists! x{:}\alpha.Px)\rightarrow\alpha$. (One can take some fixed closed term for ι but also declare it as variable in the context.) We assume the intended meaning of DD in PREDω to be clear. Together with classical logic, the axiom of definite descriptions has an unexpected side-effect in CC.

6.6.6. Proposition. *[[Pottinger 1989]] 'Classical logic' and 'definite descriptions' yield proof irrelevance in CC*

We have already encountered the semantical notion of proof irrelevance in the discussion of the model in 6.6.2. It can also be expressed in purely syntactical terms as the phenomenon that for all propositions φ, all proofs of φ are Leibniz-equal. It is then formalised in CC by the proposition

$$\mathrm{PI} := \forall\alpha{:}\star.\forall x,y{:}\alpha.(x =_\alpha y).$$

Of course, PI holds in the proof-irrelevance model of 6.6.2 (the interpretation of PI is 1), so PI doesn't imply inconsistency. However, if we intend to use CC for predicate logic it is clearly undesirable: if Γ proves PI, then any assumption $a \neq a'$ makes Γ inconsistent. We see that PI, which is a very useful principle for proofs, is a very odd principle when applied to domain-objects. Because of the treatment of domains and propositions at the same level, principles about (proofs of) propositions have unwanted applications to the domains.

The proof of Proposition 6.6.6 in [Pottinger 1989] uses an adapted form of a proof by Coquand ([Coquand 1990]), showing that CC with classical logic and a derivation rule for a strong version of disjoint sum yields proof irrelevance. Let's also state this result, but not by adding a derivation rule but by adding an axiom, which really amounts to the same as the rule used in [Coquand 1990]. (Using the result by Reynolds that polymorphism is not set-theoretic, Berardi has proved that in CC, classical logic with a stronger form of definite descriptions (replacing the $\exists!$ by $\exists$) implies PI. See [LEGO-examples] for details.)

6.6.7. Proposition ([Coquand 1990]). *'Classical logic' with 'disjunction property for classical proofs' implies proof irrelevance in CC.*

Here we mean by 'disjunction property for classical proofs', that for $c : \mathrm{CL}$ in the context and $\varphi : \star$, $c\varphi$ is in the smallest set of proofs of $\varphi \vee \neg\varphi$ that contains all proofs that are obtained by $\vee$-introduction from a proof of φ or a proof of $\neg\varphi$. Put in syntactical terms this says that, for i and j the injections from A to $A \vee B$, respectively from B to $A \vee B$, the proposition

$$\forall P{:}(A\vee B)\rightarrow\star.(\forall x{:}A.P(ix))\rightarrow(\forall x{:}B.P(jx))\rightarrow P(c\varphi)$$

holds. So proof irrelevance follows from the context

$$cl{:}\mathrm{CL}, z{:}\forall\alpha{:}\star.(\alpha+\neg\alpha)(cl\alpha),$$

where for $A, B{:}\star$,

$$A+B := \lambda y{:}A\vee B.\forall P{:}(A\vee B)\to\star.(\forall x{:}A.P(ix))\to(\forall x{:}B.P(jx))\to Py.$$

In presence of CL also the reverse can be proved, so we can construct a proof p with

$$cl{:}\mathrm{CL}\vdash p : PI \leftrightarrow (\forall\alpha{:}\star.(\alpha+\neg\alpha)(cl\alpha)).$$

The implication from right to left is the most interesting. In [Coquand 1990] it is proved by using the fact that if in Γ one can construct $A:\star$, $E : A\to\star$, $\epsilon:\star\to A$ and a proof of $\forall\alpha{:}\star.\alpha\leftrightarrow E(\epsilon\alpha)$, then Γ proves $\perp$.

6.7. Formulas about data-types in CC

Having seen the incompleteness of the formulas-as-types embedding of higher order predicate logic in CC, we shall now see that the distance between CC and PREDω is not so large when it comes to propositions about inductive data types. This follows from a recent result by Berardi, which we shall discuss here only for what concerns the implications for the formulas-as-types embedding. For details and proofs we refer to [Berardi 199+]. The point is that for purposes of deriving programs from proofs, it doesn't seem to make sense to declare a theory in the context. Instead one uses the definable impredicative data types and inductive predicates on them, as is done in the examples of 6.4.3. This is not the place to discuss in detail the topic of extracting programs from proofs in CC, for which we refer to [Paulin 1989], but to get some flavor we treat the first example of 6.4.3. Roughly, the program extracted from the proof is the $\lambda\omega$-term obtained by the mapping $[-]$, as defined in Definition 6.5.23.

Suppose t is a proof of

$$\Pi l{:}\mathrm{List}.\exists n{:}\mathrm{Nat}.\Pi m{:}\mathrm{Nat}.\text{`}m\in l\to m\leq n\text{'}$$

in the context $a{:}\mathrm{Ind}_{\mathrm{Nat}}$. Then in $\lambda\omega$ we have

$$a{:}\Pi P{:}\star.P\to(\mathrm{Nat}\to P\to P)\to(\mathrm{Nat}\to P)\vdash[t] : \mathrm{List}\to(\mathrm{Nat}\times\mathrm{Nat}\to\mathrm{True}_1\to\mathrm{True}_2),$$

where True_1 and True_2 are some trivially provable propositions. Now $[t]$ still contains computationally irrelevant information; the real program to be extracted should be something like $\lambda x{:}\mathrm{Nat}.\pi_1([t]^*x) : \mathrm{List}\to\mathrm{Nat}$, where * substitutes some closed term for a in $[t]$. Of course it is not irrelevant what we substitute for a, but the general picture should be clear: From the proof of the specification one can obtain the program that satisfies the specification. In [Paulin 1989] it is also shown how to extract from the proof the logical content which is a proof that the extracted program satisfies the specification. Some parts of the proof have computational content while others don't. Therefore, to mechanize the extraction proces, in [Paulin 1989] the type $\star$ is divided in Prop, Data and Spec, the first consisting of the propositions with purely logical content, the second consisting of the propositions with purely computaional content and the third consisting of propositions containing both logical and computaional content.

In view of the discussion of the example above it is an interesting question whether CC proves more propositions *about inductive data types* then higher order predicate logic does. It is clear that we have to be more precise if we want to have a negative answer, because in general the answer will be positive. (E.g. in CC we can still prove $\mathrm{EXT} \rightarrow \exists x{:}\mathrm{Nat}.Sx =_{\mathrm{Nat}} x$ (see the second proof of Proposition 6.5.3) and $\mathrm{Ind}_{\mathrm{Nat}}\&(Z \neq_{\mathrm{Nat}} SZ) \rightarrow \Pi x, y{:}\mathrm{Nat}.(Sx =_{\mathrm{Nat}} Sy) \rightarrow (x =_{\mathrm{Nat}} y)$ (see Example 6.4.2.)) First we have to consider only the strongest version of inductive data types, called *parametric data types* in [Berardi 199+]. A parametric data type is in set-theoretic terms the smallest set X closed under some fixed operators (functions of type $A_1 \rightarrow A_2 \rightarrow \ldots \rightarrow A_n \rightarrow X$, where $n \geq 0$ and each A_i is X or an already defined parametric data type.) If D is a parametric data type this implies that the induction and uniqueness properties for D are satisfied. In algebraic terms, a parametric data type is just a free (or initial) algebra. Further we have to restrict ourselves to a specific class of propositions, what Berardi calls the *propositions on functional types*. The functional types are the ones obtained by putting arrows between the parametric data types; further there are the so called *logical types*, which is the class of (higher order) predicate types on functional types. The *propositions on functional types* are the propositions obtained from the basic propositions by the usual logical connectives $\supset, \vee, \&, \neg, \forall_L$ and $\exists_L$, where L is a logical type. The *basic propositions* are those propositions obtained by applying an inductive predicate to the right number of terms (of the right type), so this class is already quite big. (Inductive predicates are minimal subsets among those closed under some fixed monotone constructors; they can be defined in higher order predicate logic by the higher order quantification over all such predicates. For example $\leq \subseteq \mathrm{Nat} \times \mathrm{Nat}$ and $\in \subseteq \mathrm{Nat} \times \mathrm{List}$ of the Examples in 6.4.3 are inductive predicates.) In [Berardi 199+] all this is defined in set-theoretic terms and then translated into CC. Following [Berardi 199+], we do not denote this translation explicitly (but there are no ambiguities about this.)

The main result of [Berardi 199+] is now saying that for φ a proposition in the set Pos, if $\Gamma \vdash M{:}\varphi$ in CC for some term M, and Γ is satisfied in the model PER, then φ is provable in Set theory. Here PER is some model based on the interpretation of propositions of CC as partial equivalence relations on Λ (the set of untyped lambda terms.) The model-construction is in [Berardi 199+]; we will not go into it here but state the important facts that for all parametric data type D, the interpretation of Ind_D in PER is not empty, which means that $z{:}\mathrm{Ind}_D$ is satisfied. The set of propositions Pos consists of those propositions on functional types that are built up from the basic propositions using $\supset, \vee, \&, \neg$ and $\forall x{:}D, \exists x{:}D$ (for D a parametric data type) with the restriction that a $\forall x{:}D$ that is not bound may only occur in a positive place. (The $\forall x{:}\mathrm{Nat}$ for example, is bound if it appears as $\forall x{:}\mathrm{Nat}.(\leq (x,n) \to \ldots)$.)

One of the obvious examples where the result applies is the first of 6.4.3. Berardi shows that also the statement of Girard's normalization theorem, saying that all typable terms in system F are strongly normalizable, is in Pos. It is of the form

$$\Pi t{:}Te.\Pi A{:}Ty.Oft(t,A) \supset \exists n{:}\mathrm{Nat}.\Pi t'{:}Te.\Pi m{:}\mathrm{Nat}.Redd(t,t',m) \supset m \leq n,$$

where the type of pseudoterms Te and the type of types Ty are parametric data types and $Oft \subseteq Te \times Ty$ and $Redd \subseteq Te \times Te \times \mathrm{Nat}$ are inductive predicates with $Oft(t,A)$ if t is of type A in F, $Redd(t,t',m)$ if t reduces to t' in m steps. We see that the restrictions on the form of the propositions is not very serious; a specification will usually be of the form $\Pi x{:}D.\exists y{:}D'.P(x,y)$ with $P(x,y) \in \mathsf{Pos}$. Further the result is very general, as there are no restrictions at all on the shape of Γ or M. So Γ may even contain assumptions that can not be expressed in set-theoretical terms: As long as the assumptions are satisfied in PER, the conclusion is valid.

It would be interesting to see whether the result discussed above can be rephrased syntactically by extending λPREDω with inductive data types and describing a formulas-as-types embedding from the extended higher order predicate logic to CC. This extension of λPREDω can be defined by adding a scheme for inductive types (by allowing a kind of least fixed point construction for positive type constructors), but also by extending λPREDω with polymorphic domains. As we know how to define inductive data types in polymorphic lambda calculus and the formulas-as-types embedding from λPREDω to CC immediately extends to λPREDω with polymorphic domains, we want to say a bit more about the latter possibility. Let λPREDω^p be the following Pure Type System.

$$\begin{array}{rcl}
\mathcal{S} & = & \mathsf{Prop}, \mathsf{Set}, \mathsf{Type}^p, \mathsf{Type}^s, \\
\mathcal{A} & = & \mathsf{Prop} : \mathsf{Type}^p \mathsf{Set} : \mathsf{Type}^s, \\
\mathcal{R} & = & (\mathsf{Set}, \mathsf{Set}), (\mathsf{Type}^s, \mathsf{Set}), (\mathsf{Type}^p, \mathsf{Set}) \\
 & = & (\mathsf{Set}, \mathsf{Type}^p), (\mathsf{Type}^p, \mathsf{Type}^p), \\
 & = & (\mathsf{Prop}, \mathsf{Prop}), (\mathsf{Set}, \mathsf{Prop}), (\mathsf{Type}^p, \mathsf{Prop}).
\end{array}$$

So this is λPREDω with $(\mathsf{Type}^s, \mathsf{Set})$: a higher order predicate logic built on the polymorphic lambda calculus in stead of the simple theory of types. Note the similarity with Definition 6.5.4. In view of the description of parametric data types in the beginning of this section it is natural to leave the rule $(\mathsf{Type}^p, \mathsf{Set})$ out of the system to eliminate things like $\Pi\alpha{:}\mathsf{Set}.(\alpha \to \star) \to \alpha : \mathsf{Set}$. This is an option that we want to leave open.

The formulas-as-types embedding from $\lambda\text{PRED}\omega^p$ into CC is now induced by the formulas-as-types embedding from $\lambda\text{PRED}\omega$ into CC of Definition 3.2.6, so it is the PTS-morphism H with

$$\begin{array}{rcl} H(\star) & = & \star, \\ H(\mathsf{Set}) & = & \star, \\ H(\mathsf{Type}^p) & = & \Box, \\ H(\mathsf{Type}^s) & = & \Box. \end{array}$$

This immediately shows that $\lambda\text{PRED}\omega^p$ is consistent. (In fact the mapping H shows that all extensions of $\lambda\text{PRED}\omega$ with rules of the form (s, s'), $s, s' \in \{\mathsf{Prop}, \mathsf{Set}, \mathsf{Type}^p, \mathsf{Type}^s\}$, are consistent.) The embedding H is not complete; the same counterexamples as for $\lambda\text{PRED}\omega$ do the job. (See the proof of Proposition 6.5.3.) However, if we restrict ourselves to propositions in the set Pos, we may still be able to prove that if

$$z_1{:}\text{Ind}_{D_1}, \ldots, z_n{:}\text{Ind}_{D_n}, a{:}\text{Ind}_{\text{Nat}}, b{:}Z \neq_{\text{Nat}} SZ \vdash M : \varphi \text{ in CC},$$

then there is a proof P in $\lambda\text{PRED}\omega^p$ with

$$z_1{:}\text{Ind}_{D_1}, \ldots, z_n{:}\text{Ind}_{D_n}, a{:}\text{Ind}_{\text{Nat}}, b{:}Z \neq_{\text{Nat}} SZ \vdash P : \varphi,$$

where $D_1, \ldots, D_n$ are the parametric data types that occur in φ. (We omit the mapping H for reasons of readability.) In view of the proof of the original result in [Berardi 199+], we have a strong feeling that this adapted completeness of the formulas-as-types embedding from $\lambda\text{PRED}\omega^p$ into CC holds. However, it is not as general as the original result; one would like to allow more assumptions then just those stating the parametricity of the data types. Still the matter could be interesting for further investigations, because it may give a more syntactical handle as to which propositions about data types are provable in CC.

Let's end this section with the remark that, just like for the system $\lambda\text{PRED}2^p$, it is an open question whether $\lambda\text{PRED}\omega^p$ is conservative over $\lambda\text{PRED}\omega$. The same reasons for believing that $\lambda\text{PRED}2^p$ is not conservative over $\lambda\text{PRED}2$, apply to $\lambda\text{PRED}\omega^p$. A possible non-conservativity result does, however, not affect the use of the system $\lambda\text{PRED}\omega^p$ when the use is restricted to proving the kind of propositions about parametric data types that we discussed above.

Chapter 7

Strong Normalization for $\beta\eta$ in the Calculus of Constructions

7.1. Introduction

In this Chapter we prove the strong Normalization for CC with $\beta\eta$ conversion rule. We shall denote this system by $\mathrm{CC}_{\beta\eta}$, to distinguish it from CC_{β}, which is the original Calculus of Constructions, with only β conversion. Similarly we have $\mathrm{F}\omega_{\beta\eta}$ and $\mathrm{F}\omega_{\beta\eta}$.

One of the main problems with proving $\mathrm{SN}_{\beta\eta}$ for $\mathrm{CC}_{\beta\eta}$ is that we do not know whether $\mathsf{Term}(\mathrm{CC}_{\beta\eta})$ is closed under η-reduction. We know that SR_{η} holds for $\mathrm{CC}^{s}_{\beta\eta}$ (Lemma 4.4.32), but that doesn't immediately imply SR_{η} for $\mathrm{CC}_{\beta\eta}$. One thing to do is to prove $\mathrm{SN}_{\beta\eta}$ for $\mathrm{CC}^{s}_{\beta\eta}$, which immediately implies $\mathrm{SN}_{\beta\eta}$ for $\mathrm{CC}_{\beta\eta}$ (because the set of terms of the latter is a subset of the set of terms of the first). We choose to prove first SR_{η} for $\mathrm{CC}_{\beta\eta}$ and then $\mathrm{SN}_{\beta\eta}$ for $\mathrm{CC}_{\beta\eta}$ directly. On the one hand this is more natural and on the other hand we have found in Chapter 5.1 a simple criterion for SR_{η} to hold, which also applies to $\mathrm{CC}_{\beta\eta}$.

7.2. Meta-theory for CC with $\beta\eta$-conversion

In the section where we studied the meta theory for general Pure Type Systems we have seen some properties that we could only prove for PTS_{β}, whereas we would like to have them also for the other notions $\mathsf{PTS}_{\beta\eta}$ and $\mathsf{PTS}^{s}_{\beta\eta}$. In fact this was one of the reasons for introducing $\mathsf{PTS}^{s}_{\beta\eta}$ in the first place: We couldn't prove SR_{η} for $\mathsf{PTS}_{\beta\eta}$, so we introduced $\mathsf{PTS}^{s}_{\beta\eta}$. One of the properties that we were unable to prove for both $\mathsf{PTS}_{\beta\eta}$ and $\mathsf{PTS}^{s}_{\beta\eta}$ is the Classification Lemma, 4.4.37. As this Lemma is very important for defining mappings on the set of typable terms in an easy way, we shall show that the Lemma does hold for $\mathrm{CC}_{\beta\eta}$. So, in the following we use the syntax with *sorted variables*, as it was described in Definition 4.2.9.

7.2.1. SUBLEMMA. *The system $CC_{\beta\eta}$ has the following (expected) properties.*

1. *If $M \in \mathsf{Term}(CC_{\beta\eta})$, $M =_{\beta\eta} \Box$, then $M \equiv \Box$.*

2. *There are no terms of the form $\Pi u{:}A.\Box$ in $CC_{\beta\eta}$.*

PROOF. 1. If $M =_{\beta\eta} \Box$, then $M \twoheadrightarrow_\beta \Box$ by the Key Lemma 4.4.18. We can not have the situation that $\Gamma \vdash M : A$, because this implies (using the Stripping Lemma 4.4.27 and the Key Lemma 4.4.18) that there must be an axiom $(\Box : s)$ among the axioms of $\mathrm{CC}_{\beta\eta}$. So there is an $s \in \{\star, \Box\}$ with $M \equiv s$. It is easily seen that the s can only be $\Box$.

2. Suppose $\Pi u{:}A.\Box \in \mathsf{Term}(\mathrm{CC}_{\beta\eta})$. Then $\Gamma \vdash \Pi u{:}A.\Box : B$ for some Γ and B. So $\Gamma, u{:}A \vdash \Box : s$ for some sort s, which is not the case. $\boxtimes$

7.2.2. LEMMA. *$CC_{\beta\eta}$ satisfies $\beta\eta$-*preservation of sorts *(Definition 5.2.7). That is, for A and A' terms of $CC_{\beta\eta}$, Γ and Γ' contexts of $CC_{\beta\eta}$ and $s, s' \in \{\star, \Box\}$,*

$$\left.\begin{array}{r}\Gamma \vdash A : s\\ \Gamma' \vdash A' : s'\\ A =_{\beta\eta} A'\end{array}\right\} \Rightarrow s \equiv s'$$

PROOF. By induction on the structure of A we show

$$\left.\begin{array}{r}A' \in \mathsf{Term}(\Gamma')\\ \Gamma \vdash A : \Box\\ A =_{\beta\eta} A'\end{array}\right\} \Rightarrow \Gamma' \vdash A' : \Box.$$

Then we are done because, by Uniqueness of Types (Lemma 4.4.29), a term can not at the same time be a type and a kind. We distinguish cases according to the possible structure of A.

- $A \equiv A_1 A_2$. Then $\Gamma \vdash A_1{:}\Pi u{:}C.\Box$, which is not possible. (Sublemma 7.2.1.)

- $A \equiv \lambda u{:}A_1.A_2$. Then A can not be of type $\Box$ by the Stripping Lemma 4.4.27.

- $A \equiv \Pi u{:}A_1.A_2$. Then $A' \twoheadrightarrow_\beta \Pi u{:}A_1'.A_2'$ with (among other things) $A_2 =_{\beta\eta} A_2'$ and $\Gamma, x{:}A_1 \vdash A_2 : \Box$. We are now done by induction hypothesis.

- $A \equiv \star$. Then $A' \twoheadrightarrow_{\beta\eta} \star$, hence $\Gamma' \vdash A' : \Box$ and we are done. $\boxtimes$

7.2.3. COROLLARY (Classification in $\mathrm{CC}_{\beta\eta}$). *In $CC_{\beta\eta}$ we have*

$$\begin{array}{rcl}\mathsf{Kind} \cap \mathsf{Type} &=& \emptyset,\\ \mathsf{Constr} \cap \mathsf{Obj} &=& \emptyset.\end{array}$$

PROOF. Note that, just as in the proof of the Classification Lemma 4.4.37, it suffices to prove the following two statements (let $s, s' \in \{\star, \Box\}$.)

$$\Gamma \vdash A : s,\ \Gamma \vdash A : s' \;\Rightarrow\; s \equiv s',$$
$$\Gamma \vdash M : A : s,\ \Gamma \vdash M : A' : s' \;\Rightarrow\; s \equiv s'.$$

These follow immediately from Lemma 7.2.2, using Uniqueness of Types. ⊠

7.2.4. COROLLARY. *$CC_{\beta\eta}$ satisfies strengthening and SR_η.*

PROOF. In Chapter 5.1 we have shown that a $\mathsf{PTS}_{\beta\eta}$ that satisfies $\beta\eta$-preservation of sorts satisfies strengthening (Lemma 5.2.10) and SR_η (Corollary 5.2.11.) ⊠

7.3. The proof of SN for $\beta\eta$ in CC

We now turn to the proof of strong normalization for $\beta\eta$-reduction in the Calculus of Constructions with $\beta\eta$-conversion.This is the most general property about normalization in versions of CC that one would want: It implies SN for $\beta(\eta)$-reduction for CC with $\beta(\eta)$-conversion. The proof we give here is a generalisation of the proof of SN_β for CC_β, given in [Geuvers and Nederhof 1991].

Before giving the proof we want to see why $\mathrm{SN}_{\beta\eta}$ for $\mathrm{CC}_{\beta\eta}$ does not follow immediately from SN_β for CC_β by a 'postponement of η-reduction' argument. (That is, we strongly believe that there should be some 'easy' combinatorial argument deriving one from the other, but we haven't been able to find it.) The postponement of η still works, as was shown in paragraph 4.4.2. From it we get that SN_β for CC_β implies $\mathrm{SN}_{\beta\eta}$ for CC_β. Now the problem is that the set of typable terms of $\mathrm{CC}_{\beta\eta}$ is larger then the set of typable terms of CC_β. An example is given by the term

$$\lambda x{:}P(\lambda y{:}A.My)\!\to\!\star.\lambda z{:}PM.xz$$

which can be typed in $\mathrm{CC}_{\beta\eta}$, but not in CC_β if $y \notin \mathrm{FV}(M)$.

We do have the following, which says that it is enough to prove strong normalization for β-reduction on $\mathrm{CC}_{\beta\eta}$.

7.3.1. PROPOSITION.

$$CC_{\beta\eta} \models SN_\beta \Rightarrow CC_{\beta\eta} \models SN_{\beta\eta}.$$

PROOF. The proof follows immediately from Theorem 4.4.9, which says that $X \models \mathrm{SN}_\beta \Rightarrow \downarrow_\eta X \models \mathrm{SN}_{\beta\eta}$ if X is a set of pseudoterms closed under β-reduction. Note that $\mathsf{Term}(\mathrm{CC}_{\beta\eta})$ is closed under β and η. (The first by SR_β for arbitrary PTSs; the second by Corollary 7.2.4.) So $\mathsf{Term}(\mathrm{CC}_{\beta\eta}) = \downarrow_\eta \mathsf{Term}(\mathrm{CC}_{\beta\eta})$ and we are done. ⊠

Although the Proposition says that it is sufficient to study β-reduction, we prove $\mathrm{SN}_{\beta\eta}$ for $\mathrm{CC}_{\beta\eta}$, because the proof of SN_β for $\mathrm{CC}_{\beta\eta}$ would be exactly the same.

7.3.1. Obtaining $\mathbf{SN}_{\beta\eta}$ for CC from $\mathbf{SN}_{\beta\eta}$ for $\mathbf{F}\omega$

We define a reduction preserving mapping from $\mathrm{CC}_{\beta\eta}$ to $\mathrm{F}\omega_{\beta\eta}$. The mapping is the same as the one in [Geuvers and Nederhof 1991], where it was defined as a mapping from CC_{β} to $\mathrm{F}\omega_{\beta}$ to prove the strong normalization property for CC_{β}. The problem with the extension to $\mathrm{CC}_{\beta\eta}$ is that we don't have all the meta theory for $\mathrm{CC}_{\beta\eta}$ that was used in [Geuvers and Nederhof 1991] for the CC_{β} case. In the following we verify that the whole argument can still go through.

The original intuition of the mapping is due to [Harper et al. 1987] who define a $\beta\eta$-reduction preserving mapping from LF to $\lambda\!\rightarrow$ to prove the strong normalization of LF. The map $[\![-]\!]$ that will be used can be seen as a higher order version of the map defined by [Harper et al. 1987], although things get quite a bit more complicated here. It's also possible to restrict the map $[\![-]\!]$ to $\mathsf{Term}(\lambda\mathrm{P}2)$, to derive the result $\lambda 2 \models \mathrm{SN}_{\beta\eta} \Rightarrow \lambda\mathrm{P}2 \models \mathrm{SN}_{\beta\eta}$.

The map $[\![-]\!]$ doesn't work uniformly on the terms of $\mathrm{CC}_{\beta\eta}$. That is, we can't define $[\![-]\!]$ such that for all Γ, M and A,

$$\Gamma \vdash_{CC_{\beta\eta}} M : A \Rightarrow [\![\Gamma]\!] \vdash_{F\omega_{\beta\eta}} [\![M]\!] : [\![A]\!].$$

To show that $[\![-]\!]$ really maps terms of $\mathrm{CC}_{\beta\eta}$ on terms of $\mathrm{F}\omega_{\beta\eta}$, one has to define another map τ from types and kinds and sorts of CC to types and kinds and sorts of $\mathrm{F}\omega_{\beta\eta}$ such that

$$\Gamma \vdash_{CC_{\beta\eta}} M : A \Rightarrow \tau(\Gamma) \vdash_{F\omega_{\beta\eta}} [\![M]\!] : \tau(A).$$

In order to get a feeling for the mappings $[\![-]\!]$ and τ we give some heuristics (following [Geuvers and Nederhof 1991].)

The idea of the mappings in [Harper et al. 1987] is to replace redexes that use type dependency by $\lambda\!\rightarrow$-redexes. We follow this idea, so let for example A be a type such that

$$\frac{\Gamma \vdash_{CC_{\beta\eta}} F : A \rightarrow \star \quad \Gamma \vdash_{CC_{\beta\eta}} t : A}{\Gamma \vdash_{CC_{\beta\eta}} Ft : \star}$$

then $[\![-]\!]$ and τ must erase all type dependencies such that

$$\frac{\tau(\Gamma) \vdash_{F\omega_{\beta\eta}} [\![F]\!] : \tau(A) \rightarrow \star \quad \tau(\Gamma) \vdash_{F\omega_{\beta\eta}} [\![t]\!] : \tau(A)}{\Gamma \vdash_{F\omega_{\beta\eta}} [\![Ft]\!] : \tau(\star)}$$

is sound. This is solved for LF by taking $[\![Ft]\!] = [\![F]\!][\![t]\!]$, $\tau(A \rightarrow \star) = \tau(A) \rightarrow 0$ and $\tau(\star) = 0$, where 0 is a fixed type variable. A redex that is obtained by type dependency, say $(\lambda x{:}A.M)t$, with A a type, M a constructor and t an object, is replaced by $(\lambda z{:}0.\lambda x{:}\tau(A).[\![M]\!])[\![A]\!][\![t]\!]$, where z is a fresh variable. This term is then typable in the system without type dependency and also the possible redexes in A are preserved by the abstraction over $z{:}0$ and the application to $[\![A]\!]$.

If we add polymorphism the situation gets more complicated. Let for example

$$\frac{\Gamma \vdash_{CC_{\beta\eta}} F : \Pi\alpha{:}\star.\alpha\to\alpha \quad \Gamma \vdash_{CC_{\beta\eta}} \sigma : \star}{\Gamma \vdash_{CC_{\beta\eta}} F\sigma : \sigma\to\sigma}$$

then

$$\frac{\tau(\Gamma) \vdash_{F\omega_{\beta\eta}} [\![F]\!] : \tau(\Pi\alpha{:}\star.\alpha\to\alpha) \quad \tau(\Gamma) \vdash_{F\omega_{\beta\eta}} [\![\sigma]\!] : 0}{\Gamma \vdash_{F\omega_{\beta\eta}} [\![F\sigma]\!] : \tau(\sigma\to\sigma)}$$

must be sound. This means that taking $\tau(\Pi\alpha{:}\star.\alpha\to\alpha) = 0\to\tau(\alpha\to\alpha)$, $[\![F\sigma]\!] = [\![F]\!][\![\sigma]\!]$ doesn't work. (The application is not sound.) But also the option taking $\tau(\Pi\alpha{:}\star.\alpha\to\alpha) = \Pi\alpha{:}\star.\alpha\to\alpha$, $[\![F\sigma]\!] = [\![F]\!]\tau(\sigma)$ doesn't seem right, because the possible reductions in σ are not preserved. The solution is to do both and take

$$\begin{aligned} \tau(\Pi\alpha{:}\star.\alpha\to\alpha) &= \Pi\alpha{:}\star.0\to\alpha\to\alpha, \\ [\![F\sigma]\!] &= [\![F]\!]\tau(\sigma)[\![\sigma]\!]. \end{aligned}$$

This implies that a higher order abstraction should have a different interpretation too. For example the interpretation of $F : \Pi\alpha{:}\star.\alpha\to\alpha$ now has to be applied to two arguments. The solution for the case $F \equiv \lambda\alpha{:}\star.\lambda x{:}\alpha.x$ is to take something like $\lambda\alpha{:}\star.\lambda z{:}0.\lambda x{:}\alpha.x$, but the general picture is of course quite a bit more complicated because kinds can have much more structure (and have objects as subexpresions) then in $F\omega_{\beta\eta}$. Therefore we define a mapping ρ which provides a type for the image of τ (so we have $\Gamma \vdash_{CC_{\beta\eta}} A : B \Rightarrow \tau(\Gamma) \vdash\vdash_{F\omega_{\beta\eta}} \tau(A) : \rho(B)$ for A a type constructor or a kind).

The mapping ρ in fact just takes what is usually called the 'order' of a kind, in terms of the underlying $F\omega_{\beta\eta}$ kind. The definition is as follows.

7.3.2. Definition. The map $\rho : \{\Box\} \cup \mathsf{Kind}(CC_{\beta\eta}) \to \mathsf{Kind}(F\omega_{\beta\eta})$ is defined by

1. $\rho(\star) = \rho(\Box) = \star$,
2. $\rho(\Pi\alpha{:}A.B) = \rho(A)\to\rho(B)$ if A is a kind,
3. $\rho(\Pi x{:}A.B) = \rho(B)$ if A is a type.

Note that the case distinction in the Definition is allright in $CC_{\beta\eta}$. As the mapping ρ removes all type dependencies and all variables we have the following easy properties. (Also use the fact that for A and B typable terms, if $A =_{\beta\eta} B$, then A is a kind if and only B is. This was proved in Lemma 7.2.2.)

7.3.3. Fact. For A, B kinds of $CC_{\beta\eta}$, u a variable and M a term,

1. $\rho(A[M/u] \equiv \rho(A) \equiv \rho(A)[M/u]$,
2. $A =_{\beta\eta} B \Rightarrow \rho(A) \equiv \rho(B)$.

We now want to devote some attention to the interpretation of types and kinds under $[\![-]\!]$, before giving the definition of τ. For example, if $\Gamma \vdash_{CC_{\beta\eta}} A : \star$ and $\Gamma, \alpha{:}A \vdash_{CC_{\beta\eta}} B : \Box$, then we want $\tau(\Gamma) \vdash_{F\omega_{\beta\eta}} [\![\Pi x{:}A.B]\!] : \tau(\Box)$. The intended interpretation of $\star$ under τ was 0 (some fixed type variable.) This leaves us with the possibility to also take $\tau(\Box) = 0$ and to take $[\![\Pi x{:}A.B]\!] = c[\![A]\!][\![B]\!][c'/x]$, with c some term of type $0{\to}0{\to}0$ and c' some term of type $\tau(A)$.

So it will be required that we have fixed terms of every type and every kind in $\mathrm{F}\omega_{\beta\eta}$. However, not every type in $\mathrm{F}\omega_{\beta\eta}$ is inhabited by a closed term and therefore it seems necessary to extend the syntax with a possibility of having (closed) constants of all types. However, this becomes a very complicated system (what if we substitute a term in a constant of a not-closed type?) and it turns out that we can stay away from these kind of atrocities. The solution is to work in a fixed context $0 : \star, d{:}\bot$ ($\bot \equiv \Pi\alpha{:}\star.\alpha$) in $\mathrm{F}\omega_{\beta\eta}$ and define a fixed term $c^A : A$ for every type or kind A.

We give the definition of τ, reflecting the intuitions about preservation of reductions etc.

7.3.4. Definition. The map $\tau : \{\Box\} \cup \mathsf{Kind}(\mathrm{CC}_{\beta\eta}) \cup \mathsf{Constr}(\mathrm{CC}_{\beta\eta}) \to \mathsf{Term}(\mathrm{F}\omega_{\beta\eta})$ is inductively defined by

$$
\begin{array}{rcll}
\tau(\star) & = & \tau(\Box) = 0, & \\
\tau(\alpha) & = & \alpha, & \\
\tau(\Pi\alpha{:}A.B) & = & \Pi\alpha{:}\rho(A).\tau(A){\to}\tau(B) & \text{if } A \text{ is a kind,} \\
\tau(\Pi x{:}A.B) & = & \Pi x{:}\tau(A).\tau(B) & \text{if } A \text{ is a type,} \\
\tau(\lambda\alpha{:}A.M) & = & \lambda\alpha{:}\rho(A).\tau(M) & \text{if } A \text{ is a kind,} \\
\tau(\lambda x{:}A.M) & = & \tau(M) & \text{if } A \text{ is a type,} \\
\tau(MN) & = & \tau(M)\tau(N) & \text{if } N \text{ is a constructor,} \\
\tau(MN) & = & \tau(M) & \text{if } N \text{ is an object.}
\end{array}
$$

The definition by cases is correct by Classification for $\mathrm{CC}_{\beta\eta}$, Corollary 7.2.3. That the range of τ is indeed a subset of $\mathsf{Term}(\mathrm{F}\omega_{\beta\eta})$ will be stated in Lemma 7.3.9. The mapping τ deletes object variables and therefore type dependency, and is compatible with substitution and reduction, as is stated by the following fact. (Proofs are by induction on the structure of the terms, using the Stripping Lemma 4.4.27 and Fact 7.3.3.)

7.3.5. Fact. For A, B kinds of $\mathrm{CC}_{\beta\eta}$, x an object variable, α a constructor variable, Q a constructor and M an object of $\mathrm{CC}_{\beta\eta}$,

1. $\tau(A)$ does not contain free object variables and $\tau(A[M/x]) \equiv \tau(A)$,
2. $\tau(A[Q/\alpha]) \equiv \tau(A)[\tau(Q)/\alpha]$,
3. $A \longrightarrow_\beta B \Rightarrow \tau(A) \longrightarrow_\beta \tau(B)$ or $\tau(A) \equiv \tau(B)$,
4. $A \longrightarrow_\eta B \Rightarrow \tau(A) \longrightarrow_\eta \tau(B)$.

Using the mapping τ we now define the mapping of contexts of $\mathrm{CC}_{\beta\eta}$ onto contexts of $\mathrm{F}\omega_{\beta\eta}$. This mapping will be called τ too, although it is not defined straightforwardly by applying τ to all types and kinds in the context. The reason for this is that constructor variables have to be 'split' in a constructor variable and an object variable, replacing $\alpha : A$ in the context by $\alpha : \rho(A), x_\alpha : \tau(A)$, where x_α is some fresh object variable connected with α. This splitting has to be done because a Π- or λ-abstraction over a constructor variable is replaced by two abstractions.

To make this splitting precise we assume an injection of $i : \mathsf{Var}^\square \hookrightarrow \mathsf{Var}^\star$ such that $\mathsf{Var}^\star \setminus i(\mathsf{Var}^\square)$ is countable, consisting of those object variables that are used in the derivations in $\mathrm{CC}_{\beta\eta}$ (so an object variable $i(\alpha)$ is always 'fresh'.) Notationally we don't work with the injection i but write x_α for $i(\alpha)$. So for every variable $\alpha \in \mathsf{Var}^\square$ we have a fresh variable x_α.

7.3.6. Definition. The mapping τ on declarations and contexts is defined as follows.

1. For A a type in $\mathrm{CC}_{\beta\eta}$, x an object variable,
$$\tau(x : A) := x : \tau(A),$$
2. For A a kind in $\mathrm{CC}_{\beta\eta}$, α a constructor variable,
$$\tau(\alpha : A) := \alpha : \rho(A), x_\alpha : \tau(A),$$
3. For $\Gamma = u_1{:}A_1, u_2{:}A_2, \ldots, u_n{:}A_n$ a context in $\mathrm{CC}_{\beta\eta}$,
$$\tau(\Gamma) := 0 : \star, d : \bot, \tau(u_1{:}A_1), \tau(u_2{:}A_2), \ldots, \tau(u_n{:}A_n).$$

The $0 : \star$ in the context serves as the image of $\star$ and $\square$ under τ. Further it is used as the canonical inhabitant of $\star$ and canonical inhabitans for the other kinds of $\mathrm{F}\omega_{\beta\eta}$ are built from it. In fact we could have left it out and used any closed $\mathrm{F}\omega_{\beta\eta}$-type for it. The $d : \bot$ in the context is necessary to have a canonical inhabitant for every type. It is essential for the construction of the reduction preserving mapping $[\![-]\!]$.

7.3.7. Definition. *Canonical inhabitants of types and kinds in* $\tau(\Gamma)$, denoted by c^A for A a type or kind, are defined as follows.

$$\begin{array}{rrcl}
(i) & c^\star & := & 0, \\
(ii) & c^{A\to B} & := & \lambda\alpha{:}A.c^B, \text{ for } A\to B \text{ a kind}, \\
(iii) & c^A & := & dA, \text{ for } A \text{ a type.}
\end{array}$$

Note that $c^B[N/u] \equiv c^{B[N/u]}$ for all kinds and types B, variables u and terms N. Further note that the inhabitant c^A of A is independent of the context in which A is typed (it only depends on the specific choice of the variables 0 and d, which are constants relative to our exposition.) Before showing the soundness of τ:

$$\Gamma \vdash_{CC_{\beta\eta}} M : A \Rightarrow \tau(\Gamma) \vdash_{F\omega_{\beta\eta}} \tau(M) : \rho(A), \text{ for } M \text{ not an object,}$$

we treat some examples of the application of the mapping τ to a $\mathrm{CC}_{\beta\eta}$-term.

7.3.8. Examples. These examples are also meant to show the connection (at least computationally) between τ and the Mohring-Berardi mapping from CC_β to Fb when it comes to constructors.

1. $\tau(\Pi\alpha{:}\star.\alpha{\to}\alpha{\to}\alpha) = \Pi\alpha{:}\star.0{\to}\alpha{\to}\alpha,$
2. $\tau(\Pi\alpha{:}\star.\alpha{\to}\alpha{\to}\alpha{\to}\star) = \Pi\alpha{:}\star.0{\to}\alpha{\to}\alpha{\to}0,$
3. $\tau(\lambda\alpha{:}\star.\lambda x{:}\alpha.\lambda P{:}\alpha{\to}\star.Px) = \lambda\alpha{:}\star.\lambda P{:}\star.P$

7.3.9. Lemma. *For $M \in \mathsf{Term}(CC_{\beta\eta})$, M not an object,*

$$\Gamma \vdash_{CC_{\beta\eta}} M : A \Rightarrow \tau(\Gamma) \vdash_{F\omega_{\beta\eta}} \tau(M) : \rho(A).$$

Proof. The proof is the same as in [Geuvers and Nederhof 1991] for CC_β, so by induction on the derivation. We treat the case that the last rule was (app) and the case that the last rule was (λ). (In the proof we omit the subscript under the turnstile as it will always be clear from the context whether we are working in $\mathrm{CC}_{\beta\eta}$ or in $\mathrm{F}\omega_{\beta\eta}$.)

(app) Say $M \equiv PQ$ and $\Gamma \vdash M : \Pi u{:}B.C$, $\Gamma \vdash N : B$, $A \equiv C[P/u]$. Now PQ is a constructor, and hence P is. We distinguish subcases between Q being a constructor or an object.
If Q is a constructor, we find by induction hypothesis that $\tau(\Gamma) \vdash \tau(P) : \rho(\Pi u{:}B.C)(\equiv \rho(B){\to}\rho(C))$ and $\tau(\Gamma) \vdash \tau(Q) : \rho(B)$. By one (app) we find $\tau(\Gamma) \vdash \tau(P)\tau(Q) : \rho(C)$ and we are done because $\tau(P)\tau(Q) \equiv \tau(PQ)$ and $\rho(C) \equiv \rho(C[Q/u])$.
If Q is an object, we find by induction hypothesis that $\tau(\Gamma) \vdash \tau(P) : \rho(\Pi u{:}B.C)(\equiv \rho(C))$. We are done because $\tau(P)\tau(Q) \equiv \tau(P)$ and $\rho(C) \equiv \rho(C[Q/u])$.

(λ) Say $M \equiv \lambda u{:}B.N$ and $\Gamma, u{:}B \vdash N : C$, $\Gamma \vdash \Pi u{:}B.C : \star/\Box$. We distinguish subcases between B being a type or a kind.
If B is a type, we have $\tau(\lambda u{:}B.N) \equiv \tau(N)$, $\rho(\Pi u{:}B.C) \equiv \rho(C)$ and further by induction hypothesis $\tau(\Gamma), u{:}\tau(B) \vdash \tau(N) : \rho(C)$. By substituting $c^{\tau(B)}$ for u we find $\tau(\Gamma) \vdash \tau(N) : \rho(C)$.
If B is a kind, then $\tau(\lambda u{:}B.N) \equiv \lambda u{:}\rho(B).\tau(N)$, $\rho(\Pi u{:}B.C) \equiv \rho(B){\to}\rho(C)$

and further by induction hypothesis $\tau(\Gamma), u{:}\rho(B), x_u{:}\tau(B) \vdash \tau(N) : \rho(C)$. By substituting $c^{\tau(B)}$ for x_u we find $\tau(\Gamma), u{:}\rho(B) \vdash \tau(N) : \rho(C)$. Now also $\tau(\Gamma) \vdash \rho(B) \rightarrow \rho(C){:}\Box$, and hence $\tau(\Gamma) \vdash \lambda u{:}\rho(B).\tau(N) : \rho(B) \rightarrow \rho(C)$. ⊠

7.3.10. DEFINITION. The map $[\![-]\!]$ from $\mathsf{Term}(\mathrm{CC}_{\beta\eta}) \setminus \{\Box\}$ to $\mathsf{Term}(\mathrm{F}\omega_{\beta\eta})$ is defined inductively by

$$
\begin{array}{rcll}
[\![\star]\!] & = & c^0, & \\
[\![x]\!] & = & x & \text{if } x \in \mathsf{Var}^\star, \\
[\![\alpha]\!] & = & x_\alpha & \text{if } \alpha \in \mathsf{Var}^\Box, \\
[\![\Pi x{:}A.B]\!] & = & c^{0\rightarrow 0\rightarrow 0}[\![A]\!][\![B]\!][c^{\tau(A)}/x] & \text{if } A \text{ a type}, \\
[\![\Pi \alpha{:}A.B]\!] & = & c^{0\rightarrow 0\rightarrow 0}[\![A]\!][\![B]\!][c^{\rho(A)}/\alpha, c^{\tau(A)}/x_\alpha] & \text{if } A \text{ a kind}, \\
[\![\lambda x{:}A.M]\!] & = & (\lambda z{:}0.\lambda x{:}\tau(A).[\![M]\!])[\![A]\!], & \text{if } A \text{ a type}, \\
[\![\lambda \alpha{:}A.M]\!] & = & (\lambda z{:}0.\lambda \alpha{:}\rho(A).\lambda x_\alpha{:}\tau(A).[\![M]\!])[\![A]\!], & \text{if } A \text{ a kind}, \\
[\![MN]\!] & = & [\![M]\!][\![N]\!], & \text{if } N \text{ an object}, \\
[\![MN]\!] & = & [\![M]\!]\tau(N)[\![N]\!], & \text{if } N \text{ a constructor}.
\end{array}
$$

Here z is always assumed to be a fresh object variable.

The definition by cases is allright by the Classification for $\mathrm{CC}_{\beta\eta}$, Corollary 7.2.3. It is not very difficult to verify that the mapping preserves β- and η-reductions, which will be stated in 7.3.16. That the image of the mapping $[\![-]\!]$ is indeed a subset of $\mathsf{Term}(\mathrm{F}\omega_{\beta\eta})$ is stated by the following lemma. It is only in the proof of this lemma that we really have to add something to the proof of strong normalization for β in CC_β (apart from the quite non-trivial verification of a lot of meta-theoretical facts for $\mathrm{CC}_{\beta\eta}$ of course, but this has already been done in Chapter 4.1.) What we have to do extra here is to verify that for A and B types in $\mathrm{CC}_{\beta\eta}$, if $A =_{\beta\eta} B$ then $\tau(A) =_{\beta\eta} \tau(B)$. For CC_β this problem was easily settled by the Church-Rosser property, which we lack here. This turns out to be not so easy: We can not just redo the reduction-expansion path from A to B to get $\tau(A) =_{\beta\eta} \tau(B)$, because τ removes abstractions (and hence redexes.) Also constructors can be $\beta\eta$-equal to objects, like in $\lambda\alpha{:}\star.\alpha =_{\beta\eta} \lambda x{:}\bot.x$, and although objects are not in the domain of τ, this may have an effect on the $\beta\eta$-conversion. An example where the equality between A and $B$is really established in a different manner then the equality between $\tau(A)$ and $\tau(B)$ is the following.

$$\lambda\alpha{:}\bot\rightarrow\star.\lambda x{:}\bot.\alpha x =_{\beta\eta} \lambda\alpha{:}\star\rightarrow\star.\lambda\beta{:}\star.\alpha\beta,$$

and

$$\tau(\lambda\alpha{:}\bot\rightarrow\star.\lambda x{:}\bot.\alpha x) \equiv \lambda\alpha{:}\star.\alpha =_{\beta\eta} \lambda\alpha{:}\star\rightarrow\star.\lambda\beta{:}\star.\alpha\beta \equiv \tau(\lambda\alpha{:}\star\rightarrow\star.\lambda\beta{:}\star.\alpha\beta).$$

In this case the two images are still $\beta\eta$-equal, but one could imagine that there are dirtier tricks. That there are however no dirtier tricks is shown in Lemma 7.3.13. For the proof of that Lemma it is convenient to modify the mapping τ a bit to a mapping τ' from the erased terms to erased terms. (Here we mean the erasure $| - |$ that removes only the domains; it was defined in Definition 4.4.11.) We then define τ' by induction on the structure of terms, distinguishing cases according to the *heart* of specific subterms. (The notion of 'heart' of a term A, $\mathsf{h}(A)$, is defined in Definition 4.4.38.)

7.3.11. DEFINITION. Consider the set E which is obtained from the set $\{\Box\} \cup \mathsf{Kind}(\mathrm{CC}_{\beta\eta}) \cup \mathsf{Constr}(\mathrm{CC}_{\beta\eta})$ by first applying the erasure mapping $| - |$ and then closing down under $\twoheadrightarrow_{\beta\eta}$. On this set E we define the mapping τ' by induction on the structure of terms as follows.

$$\begin{array}{rcll}
\tau'(\star) & = & \tau'(\Box) = 0, & \\
\tau'(\alpha) & = & \alpha, & \\
\tau'(\Pi\alpha{:}A.B) & = & \Pi\alpha{:}\rho(A).\tau'(A)\to\tau'(B) & \text{if } \alpha \in \mathsf{Var}^\Box, \\
\tau'(\Pi x{:}A.B) & = & \Pi x{:}\tau'(A).\tau'(B) & \text{if } x \in \mathsf{Var}^\star, \\
\tau'(\lambda\alpha.M) & = & \lambda\alpha.\tau'(M) & \text{if } \alpha \in \mathsf{Var}^\Box, \\
\tau'(\lambda x.M) & = & \tau'(M) & \text{if } x \in \mathsf{Var}^\star, \\
\tau'(MN) & = & \tau'(M)\tau'(N) & \text{if } \mathsf{h}(N) \in \mathsf{Var}^\Box, \\
\tau'(MN) & = & \tau'(M) & \text{if } \mathsf{h}(N) \in \mathsf{Var}^\star
\end{array}$$

The definition is justified by Lemma 4.4.39.

7.3.12. FACT. If $A \in \{\Box\} \cup \mathsf{Kind}(\mathrm{CC}_{\beta\eta}) \cup \mathsf{Constr}(\mathrm{CC}_{\beta\eta})$, then

$$|\tau(A)| \equiv \tau'(|A|).$$

7.3.13. LEMMA. *For A, B terms of $CC_{\beta\eta}$, not objects,*

$$A =_{\beta\eta} B \Rightarrow \tau(A) =_{\beta\eta} \tau(B).$$

PROOF. Immediately from the following.

$$\begin{array}{rcl}
A =_{\beta\eta} B & \Rightarrow & |A| =_{\beta\eta} |B| \\
& \Rightarrow & \tau'(|A|) =_{\beta\eta} \tau'(|B|) \\
& \Rightarrow & |\tau(A)| =_{\beta\eta} |\tau(B)| \quad \Rightarrow \quad \tau(A) =_{\beta\eta} \tau(B).
\end{array}$$

The first is a standard property of $|-|$, the third is justified by the fact that we just stated and the last step is also a standard property of $|-|$. (See Corollary 4.4.17.) This leaves us with the second step. Suppose $|A| =_{\beta\eta} |B|$, say $|A| \twoheadrightarrow_{\beta\eta} Q \;{}_{\beta\eta}\!\twoheadleftarrow |B|$. Then we can copy all the reduction steps from $|A|$ to Q and from $|B|$ to Q in the τ'-image. A precise proof of this fact can be given by verifying that the properties of 7.3.5 also hold for τ', i.e. for x an object variable ($x \in \mathsf{Var}^\star$) and α a constructor variable ($\alpha \in \mathsf{Var}^\Box$)

1. $\tau'(A)$ does not contain free object variables and $\tau'(A[M/x]) \equiv \tau'(A)$,
2. $\tau'(A[Q/\alpha]) \equiv \tau'(A)[\tau'(Q)/\alpha]$,

3. $A \longrightarrow_\beta B \Rightarrow \tau'(A) \longrightarrow_\beta \tau'(B)$ or $\tau'(A) \equiv \tau'(B)$,
4. $A \longrightarrow_\eta B \Rightarrow \tau'(A) \longrightarrow_\eta \tau'(B)$. ⊠

7.3.14. LEMMA.

$$\Gamma \vdash_{CC_{\beta\eta}} M : A \Rightarrow \tau(\Gamma) \vdash_{F\omega_{\beta\eta}} [\![M]\!] : \tau(A)$$

PROOF. By induction on the structure of terms as in [Geuvers and Nederhof 1991], using Lemma 7.3.13 and the Stripping Lemma 4.4.27. We treat the cases for M being a Π-abstraction, a λ-abstraction or an applicatioin.

Π-abstr. Say $M \equiv \Pi u{:}B.C$ and note that A can only be $\star$ or $\Box$. By induction hypothesis we obtain that $\tau(\Gamma) \vdash [\![B]\!] : 0$ and $\tau(\Gamma, u{:}B) \vdash C : 0$. Now we distinguish cases according to whether B is a type or a kind.
If B is a type, $\tau(\Gamma, u{:}B) = \tau(\Gamma), u{:}\tau(B)$, so by substituting $c^{\tau(B)}$ for u and applying $c^{0\to 0\to 0}$ to $[\![B]\!]$ and $[\![C]\!][c^{\tau(B)}/u]$ we conclude that $\tau(\Gamma) \vdash c^{0\to 0\to 0}[\![B]\!][\![C]\!][c^{\tau(B)}/u] : 0$ and we are done.
If B is a kind, $\tau(\Gamma, u{:}B) = \tau(\Gamma), u{:}\rho(B), x_u{:}\tau(B)$, so by substituting $c^{\rho(B)}$ for u, $c^{\tau(B)}$ for x_u and applying $c^{0\to 0\to 0}$ to $[\![B]\!]$ and $[\![C]\!][c^{\rho(B)}/u, c^{\tau(B)}/x_u]$ we conclude that $\tau(\Gamma) \vdash c^{0\to 0\to 0}[\![B]\!][\![C]\!][c^{\rho(B)}/u, c^{\tau(B)}/x_u] : 0$ and we are done.

λ-abstr. Say $M \equiv \lambda u{:}B.P$ and note that (by the Stripping Lemma 4.4.27) $A =_{\beta\eta} \Pi u{:}B.C$ with $\Gamma, u{:}B \vdash P : C$. By induction hypothesis we obtain that $\tau(\Gamma, u{:}B) \vdash [\![P]\!] : \tau(C)$ and $\tau(\Gamma) \vdash [\![B]\!] : 0$. Now we distinguish cases according to whether B is a type or a kind.
If B is a type, $\tau(\Gamma, u{:}B) = \tau(\Gamma), u{:}\tau(B)$. Now $\tau(B)$ and $\tau(C)$ are both types, so we can do a λ-abstraction and we obtain $\tau(\Gamma) \vdash \lambda u{:}\tau(B).[\![P]\!] : \Pi u{:}\tau(B).\tau(C)$. From this we easily conclude that
$\tau(\Gamma) \vdash (\lambda z{:}0\lambda u{:}\tau(B).[\![P]\!])[\![B]\!] : \Pi u{:}\tau(B).\tau(C)$. Now we are done because from $\Pi u{:}B.C =_{\beta\eta} A$ it follows by Lemma 7.3.13 that $\Pi u{:}\tau(B).\tau(C) =_{\beta\eta} \tau(A)$ and we can apply the conversion rule to obtain what was to be proved.
If B is a kind, $\tau(\Gamma, u{:}B) = \tau(\Gamma), u{:}\rho(B), x_u{:}\tau(B)$. Now $\tau(B)$ is a type and $\rho(B)$ is a kind, so we can do two λ-abstractions to obtain $\tau(\Gamma) \vdash \lambda u{:}\rho(B).\lambda x_u{:}\tau(B).[\![P]\!] : \Pi u{:}\rho(B).\tau(B){\to}\tau(C)$. From this we easily conclude that
$\tau(\Gamma) \vdash (\lambda z{:}0\lambda u{:}\rho(B).\lambda x_u{:}\tau(B).[\![P]\!])[\![B]\!] : \Pi u{:}\rho(B).\tau(B){\to}\tau(C)$. Now again we are done because from $\Pi u{:}B.C =_{\beta\eta} A$ it follows by Lemma 7.3.13 that $\Pi u{:}\tau(B).\tau(C) =_{\beta\eta} \tau(A)$.

applic. Say $M \equiv PQ$ with $\Gamma \vdash P : \Pi u{:}B.C$, $\Gamma \vdash Q : B$ such that $A =_{\beta\eta} C[Q/u]$. By induction hypothesis we find that $\tau(\Gamma) \vdash [\![P]\!] : \tau(\Pi u{:}B.C)$ and

$\tau(\Gamma) \vdash [\![Q]\!] : \tau(B)$. We distinguish subcases according to whether P is an object or a constructor.
If Q is an object then B is a type, so $[\![PQ]\!] \equiv [\![P]\!][\![Q]\!]$ and $\tau(\Pi u{:}B.C) \equiv \Pi u{:}\tau(B).\tau(C)$. We can conclude that $\tau(\Gamma) \vdash [\![PQ]\!] : \tau(C)[[\![Q]\!]/u]$ and we are done by the fact that $\tau(C)[[\![Q]\!]/u] \equiv \tau(C) =_{\beta\eta} \tau(A)$ (by Lemma 7.3.13.)
If Q is a constructor then B is a kind, so $[\![PQ]\!] \equiv [\![P]\!]\tau(Q)inteQ$ and $\tau(\Pi u{:}B.C) \equiv \Pi u{:}\rho(B).\tau(B)\to\tau(C)$. We can conclude that $\tau(\Gamma) \vdash [\![PQ]\!] : \tau(C)[\tau(Q)/u]$ and we are done by the fact that $\tau(C)[\tau(Q)/u] \equiv \tau(C[Q/u]) =_{\beta\eta} \tau(A)$ (by Lemma 7.3.13.) ⊠

7.3.15. Lemma. *For $M \in \mathsf{Term}(CC_{\beta\eta})$, $x \in \mathsf{Var}^{\star}$, $\alpha \in \mathsf{Var}^{\square}$, N an object and Q a constructor,*

1. $[\![M[N/x]]\!] \equiv [\![M[[\![N]\!]/x]]\!]$,
2. $[\![M[Q/x]]\!] \equiv [\![M]\!][\tau(Q)/\alpha, [\![Q]\!]/x_\alpha]$.

Proof. Both by induction on the structure of M, using the fact that a term $\rho(A)$ does not contain any free variables and that a term $\tau(A)$ does not contain any free object variables. Further one needs some (easy) substitution properties for the canonical inhabitants of types and kinds like

$$\begin{array}{rcl} c^{\tau(A)}[[\![N]\!]/x] & \equiv & c^{\tau(A[N/x])}, \\ c^{\rho(A)}[[\![N]\!]/x] & \equiv & c^{\rho(A[N/x])}, \\ c^{\tau(A)}[\tau(B)/\alpha, [\![B]\!]/x_\alpha] & \equiv & c^{\tau(A[B/\alpha])}, \\ c^{\rho(A)}[\tau(B)/\alpha, [\![B]\!]/x_\alpha] & \equiv & c^{\rho(A[B/\alpha])}. \boxtimes \end{array}$$

7.3.16. Theorem. *For $M, M' \in \mathsf{Term}(CC_{\beta\eta})$,*

$$M \longrightarrow_{\beta\eta} M' \Rightarrow [\![M]\!] \twoheadrightarrow^{\neq 0}_{\beta\eta} [\![M']\!].$$

Proof. By induction on the structure of M. The only interesting cases are when the reduced β- or η-redex is M itself, which are handled by distinguishing subcases according to the domain of the lambda abstraction. We only treat the cases for which the domain is a kind. (The cases for which the domain is a type are similar but easier.)

- $M \equiv (\lambda\alpha{:}A.N)Q$ with A a kind. Then

$$\begin{array}{rcl} [\![M]\!] & \equiv & (\lambda z{:}0.\lambda\alpha{:}\rho(A).\lambda x_\alpha{:}\tau(A).[\![N]\!])[\![A]\!]\tau(Q)[\![Q]\!] \\ & \twoheadrightarrow^{\neq 0}_{\beta} & [\![N]\!][\tau(Q)/\alpha, [\![Q]\!]/x_\alpha] \\ & \equiv & [\![N[Q/\alpha]]\!] \equiv [\![M']\!]. \end{array}$$

- $M \equiv \lambda\alpha{:}A.N\alpha$ with A a kind. Then

$$\begin{array}{rcl} [\![M]\!] & \equiv & (\lambda z{:}0.\lambda\alpha{:}\rho(A).\lambda x_\alpha{:}\tau(A).[\![N]\!]\alpha x_\alpha)[\![A]\!] \\ & \twoheadrightarrow^{\neq 0}_{\beta\eta} & [\![N]\!] \equiv [\![M']\!] \boxtimes \end{array}$$

7.3.17. THEOREM.

$$F\omega_{\beta\eta} \models SN_{\beta\eta} \Rightarrow CC_{\beta\eta} \models SN_{\beta\eta}.$$

PROOF. An infinite $\beta\eta$-reduction sequence in $\mathrm{CC}_{\beta\eta}$ yields an infinite $\beta\eta$-reduction sequence in $\mathrm{F}\omega_{\beta\eta}$ by the mapping $[\![-]\!]$. $\boxtimes$

One can be a bit more careful in the last proof and use the positive formulation of Strong Normalization: for every term M there is an upperbound to the length of all reduction sequences starting from M. Then one can show that, from an upperbound to the length of $\beta\eta$-reductions starting from $[\![M]\!]$, one can compute an upperbound to the length of $\beta\eta$-reductions starting from M.

7.3.2. Strong Normalization for $\beta\eta$-reduction in Fω

The proof of $\mathrm{F}\omega_{\beta\eta} \models \mathrm{SN}_{\beta\eta}$ will be done by first proving that β^ω-reduction is strongly normalizing and that the combination $\beta^{2\omega}$-reduction is strongly normalizing. Using this, we then show that, if β^0-reduction is strongly normalizing on the *erased* terms (the erasure here is the 'typed' erasure defined in 6.3.5, different from the one defined in 4.4.11, which is totally syntactical), then β-reduction is strongly normalizing. In this way we avoid the need to define the so called 'candidats de réducibilité' as typed sets, as is done for example in [Girard et al. 1989]. This makes the exposition more perspicious and clearly points out where the proof is essentially complex (in proof-theoretical terms.) This idea of proving strong normalization (reducing the problem to the set of underlying type-free terms) is applied to the polymorphic lambda calculus in [Mitchell 1986] (see also [Scedrov 1990]).

7.3.18. PROPOSITION.

$$F\omega_{\beta\eta} \models SN_{\beta\eta^\omega}.$$

PROOF. We only have to consider the constructors, because an infinite $\beta\eta^\omega$-reduction in a term of $\mathrm{F}\omega_{\beta\eta}$ will always be due to an infinite $\beta\eta^\omega$-reduction in a subterm that is a constructor.
The proof is now by defining a $\beta\eta$-reduction preserving mapping $[-]$ from the constructors of $\mathrm{F}\omega_{\beta\eta}$ to the objects of $\lambda\!\to$ such that a constructor $M{:}k$ becomes an object $[M]{:}[k]$, where $[k]$ is defined inductively as follows.

$$\begin{array}{rcl} [\star] & = & 0, \\ [k_1 \to k_2] & = & [k_1] \to [k_2], \end{array}$$

where 0 is some fixed type variable to be declared in the context. The reduction preserving mapping $[-]$ on constructors is

$$\begin{array}{rcl} [\alpha] & = & \alpha, \\ [\sigma\to\tau] & = & c^{0\to 0\to 0}[\sigma]\to[\tau], \\ [\Pi\alpha{:}k.\sigma] & = & [\sigma][c^{[k]}/\alpha], \\ [\lambda\alpha{:}k.P] & = & \lambda\alpha{:}[k].[P], \\ [PQ] & = & [P][Q], \end{array}$$

where for k a kind of $\mathrm{F}\omega_{\beta\eta}$, the fixed object $c^{[k]}$ of type $[k]$ is defined inductively by taking c^0 as a fixed variable of type 0 in the context and defining $c^{k_1\to k_2} = \lambda x{:}[k_1].c^{[k_2]}$. We then have for Γ a context containing only declarations of constructor variables,

$$\Gamma \vdash_{F\omega_{\beta\eta}} P : k \Rightarrow 0{:}\star, c^0{:}0, [\Gamma] \vdash_{\lambda\to} [P] : [k],$$

where the extension of $[-]$ to contexts is the straightforward one. ⊠

7.3.19. Lemma. *For $M, M' \in \mathsf{Term}(F\omega_{\beta\eta})$, objects,*

$$\begin{array}{lrcl} (i) & M \xrightarrow{2}_{\beta\eta} M' & \Rightarrow & \#(\lambda_2 s \textit{ in } M) = \#(\lambda_2 s \textit{ in } M') - 1, \\ (ii) & M \xrightarrow{\omega}_{\beta\eta} M' & \Rightarrow & \#(\lambda_2 s \textit{ in } M) = \#(\lambda_2 s \textit{ in } M'), \\ (iii) & M \xrightarrow{2}_{\beta\eta} M' \textit{ or } M \xrightarrow{\omega}_{\beta\eta} M' & \Rightarrow & |M|^t \equiv |M'|^t. \end{array}$$

Proof. The only way in which the number of λs of a certain form can increase by a reduction step is when the λ of this particular form occurs in Q and

$$(\lambda x{:}A.N)Q \longrightarrow N[Q/x],$$

with x free in N more then once. So we look for each case of the lemma at a β-redex of the above form in the premise and check the conclusion.

1. $(\lambda_2\alpha{:}K.N)Q \longrightarrow_2 N[Q/\alpha]$. Then Q is a constructor, so it does not contain any objects as subexpressions, so it certainly contains no λ_2s. So the number of λ_2s is reduced with one.

2. $(\lambda_\omega\alpha{:}K.N)Q \xrightarrow{\omega}_\beta N[Q/\alpha]$. Then Q is a constructor again and so it contains no λ_2s. The number of λ_2s in the term remains the same.

3. By the definition of the erasure $|-|^t$, which removes all type information. A β^ω-reduction step will always be inside a type of the object M, so $|M|^t \equiv |M'|^t$. A β^2-reduction step inside M is of the form $(\lambda_2\alpha{:}K.N)Q \longrightarrow_2 N[Q/\alpha]$. After applying $|-|^t$ the first becomes $|N|^t$ and so does the second. ⊠

7.3.20. Lemma.

$$F\omega_{\beta\eta} \models SN_{\beta\eta^{2\omega}}$$

Proof. Suppose we have an infinite reduction sequence

$$M_1 \xrightarrow{2\omega}_{\beta\eta} M_2 \xrightarrow{2\omega}_{\beta\eta} M_3 \xrightarrow{2\omega}_{\beta\eta} \ldots,$$

in $\mathrm{F}\omega_{\beta\eta}$. By Proposition 7.3.18 we know that all the M_i must be objects and that this infinite reduction can not have a tail

$$M_n \xrightarrow{\omega}_{\beta\eta} M_{n+1} \xrightarrow{\omega}_{\beta\eta} M_{n+2} \xrightarrow{\omega}_{\beta\eta} \ldots.$$

So the infinite $\beta\eta^{2\omega}$-reduction sequence contains infinitely many $\beta\eta^2$-contractions. By Lemma 7.3.19 this is not possible: a $\beta\eta^2$-contraction reduces the number of λ_2s by one and a $\beta\eta^\omega$-contraction does not change the number of λ_2s. So there can be no infinite $\beta\eta^{2\omega}$-reduction in $\mathrm{F}\omega_{\beta\eta}$. ⊠

7.3.21. Proposition.

$$\forall M \in \mathsf{Obj}(F\omega_{\beta\eta})[SN(|M|^t) \Rightarrow SN(M)]$$

Proof. Let M be an object such that $\mathrm{SN}(|M|^t)$ holds. Suppose we have an infinite reduction sequence

$$M \longrightarrow_{\beta\eta} M_1 \longrightarrow_{\beta\eta} M_2 \longrightarrow_{\beta\eta} \ldots,$$

in $\mathrm{F}\omega_{\beta\eta}$. Then all M_i are objects of $\mathrm{F}\omega_{\beta\eta}$. By Lemma 7.3.20, only finitely many $\beta\eta^{2\omega}$-contractions are performed after one another, so the sequence contains infinitely many $\beta\eta^0$-contractions. Now we can apply $|{-}|^t$ to obtain an infinite $\beta\eta$-reduction sequence starting from $|M|^t$ (using Lemma 7.3.19.) This contradicts $\mathrm{SN}(|M|^t)$, so there is no infinite $\beta\eta$-reduction sequence starting from M. ⊠

The Proposition is telling us that we only have to check that the set of erasures of objects of $\mathrm{F}\omega_{\beta\eta}$ satisfies $\mathrm{SN}_{\beta\eta}$ in order to prove

$$\mathrm{F}\omega_{\beta\eta} \models \mathrm{SN}_{\beta\eta}.$$

This will be done by extending the well-known method of computability predicates to the higher order case. This method can be seen as the building of a model of $\mathrm{F}\omega_{\beta\eta}$ inside the untyped lambda calculus, where types become sets of strongly normalizing terms and the interpretation (modulo a valuation ρ that assigns untyped terms to the free variables) of a term M of type σ is an untyped term in the set that is represented by σ. The Strong Normalization property then follows from the fact that one can take the identity for the valuation ρ, in which case the interpretation of M becomes $|M|^t$, which is then Strongly Normalizing by the construction of the model.

Let in the following $\mathsf{SN} \subset \Lambda$ be the set of untyped lambda terms that is Strongly Normalizing under $\beta\eta$-reduction. (By posponement of η-reduction and the fact that η-reduction itself is Strongly Normalizing on Λ, this is the same as the set of terms that is Strongly Normalizing under β-reduction.)

7.3.22. DEFINITION. A set of untyped lambda terms X is *saturated* if

1. $X \subset \mathsf{SN}$,
2. $\forall \vec{Q} \in \mathsf{SN} \forall x \in \mathsf{Var}[x\vec{Q} \in X]$,
3. $\forall \vec{Q}, M, P \in \mathsf{SN}[M[P/x]\vec{Q} \in X \Rightarrow (\lambda x.M)P\vec{Q} \in X]$.

Note that SN is itself saturated and that all saturated sets are nonempty.

The types of $\mathrm{F}\omega_{\beta\eta}$ will be interpreted as saturated sets. This requires some closure properties for the set of saturated sets which will be proved in Lemma 7.3.24. The kinds of $\mathrm{F}\omega_{\beta\eta}$ will be interpreted as the set-theoretic function spaces except for the kind $\star$ which will be interpreted as the set of all saturated sets. Recall that

$$\mathsf{Kind}(\mathrm{F}\omega_{\beta\eta}) = K ::= \star \,|\, K \to K.$$

7.3.23. DEFINITION. For $k \in \mathsf{Kind}(\mathrm{F}\omega_{\beta\eta})$, *the set of computability predicates for* k, $\mathsf{CP}(k)$, is defined inductively as follows.

$$\begin{array}{rcl} \mathsf{CP}(\star) & = & \{X \,|\, X \subset \Lambda \text{ is saturated}\}, \\ \mathsf{CP}(k_1 \to k_2) & = & \{f \,|\, f : \mathsf{CP}(k_1) \to \mathsf{CP}(k_2)\}. \end{array}$$

The interpretation of a kind k in the intended model will now be by taking $\mathsf{CP}(k)$.

7.3.24. LEMMA. *The set of saturated sets is closed under arbitary intersections and taking function spaces. That is,*

1. *for I a set and X_i saturated for all $i \in I$,*

$$\cap_{i \in I} X_i \text{ is saturated}$$

2. *for X and Y saturated,*

$$X \to Y := \{M \in \Lambda \,|\, \forall N \in X[MN \in Y]\} \text{ is saturated.}$$

PROOF. The closure under arbitrary intersections is easy to prove. For the closure under function spaces, let X and Y be saturated sets and take $X \to Y$ as in the lemma. It is easy to see that all $M \in X \to Y$ are SN. Further, for x a variable and $\vec{Q} \in \mathsf{SN}$, we have that for all $N \in X$, $x\vec{Q}N \in Y$, because N is SN and Y is a saturated set. Finally, for $\vec{Q}, M, P \in \mathsf{SN}$ with $M[P/x]\vec{Q} \in X \to Y$, we know that $\forall N \in X[M[P/x]\vec{Q}N \in Y]$. So $\forall N \in X[(\lambda x.M)P\vec{Q}N \in Y]$ by the saturatedness of Y, so $(\lambda x.M)P\vec{Q} \in X \to Y$. ⊠

One may wonder why we need the saturated sets (a specific class of subsets of SN) and can not just interpret all the types by the set SN itself. However, this breaks down on the fact that $\mathsf{SN} \to \mathsf{SN} \neq \mathsf{SN}$. (For example, $\lambda x.xx \notin \mathsf{SN} \to \mathsf{SN}$.)

7.3.25. DEFINITION. For Γ a context of $\mathrm{F}\omega_{\beta\eta}$, a *constructor valuation of* Γ (notation $\xi \models_{\Box} \Gamma$) is a map $\xi : \mathsf{Var}^{\Box} \to \cup_{k\in K} CP(k)$ such that

$$\alpha : k \in \Gamma \Rightarrow \xi(\alpha) \in \mathsf{CP}(k).$$

7.3.26. DEFINITION. For Γ a context of $\mathrm{F}\omega_{\beta\eta}$ and ξ a constructor valuation of Γ, the interpretation function

$$[\![-]\!]^{\Gamma}_{\xi} : \Gamma\text{-}\mathsf{Constr}(\mathrm{F}\omega_{\beta\eta}) \to \cup_{k\in K} \mathsf{CP}(k)$$

is defined inductively as follows.

$$\begin{array}{rcl}
[\![\alpha]\!]^{\Gamma}_{\xi} & = & \xi(\alpha), \\
[\![PQ]\!]^{\Gamma}_{\xi} & = & [\![P]\!]^{\Gamma}_{\xi}[\![Q]\!]^{\Gamma}_{\xi}, \\
[\![\lambda\alpha{:}k.Q]\!]^{\Gamma}_{\xi} & = & \boldsymbol{\lambda} f \in \mathsf{CP}(k).[\![Q]\!]^{\Gamma}_{\xi(\alpha:=f)}, \\
[\![\sigma\to\tau]\!]^{\Gamma}_{\xi} & = & [\![\sigma]\!]^{\Gamma}_{\xi}\to[\![\tau]\!]^{\Gamma}_{\xi}, \\
[\![\Pi\alpha{:}k.\sigma]\!]^{\Gamma}_{\xi} & = & \cap_{f\in\mathsf{CP}(k)}[\![\sigma]\!]^{\Gamma}_{\xi(\alpha:=f)}.
\end{array}$$

In most situations the Γ will be clear from the context, and will therefore not be mentioned explicitly.

The definition is justified by the Stripping Lemma 4.4.27 and the following Lemma, which states that the interpretations of the constructors are elements of the right computability predicate.

7.3.27. LEMMA. *For Γ a context of $F\omega_{\beta\eta}$, $Q, k \in \mathsf{Term}(F\omega_{\beta\eta})$ and $\xi \models_{\Box} \Gamma$,*

$$\Gamma \vdash Q : k(:\Box) \Rightarrow [\![Q]\!]_{\xi} \in \mathsf{CP}(k).$$

PROOF. Easy induction over the structure of Q. ⊠

7.3.28. LEMMA. *For Γ a context of $F\omega_{\beta\eta}$, $Q, P \in \Gamma\text{-}\mathsf{Constr}(F\omega_{\beta\eta})$, $\alpha \in \mathsf{Var}^{\Box}$ and $\xi \models_{\Box} \Gamma$,*

(i) $[\![Q[P/\alpha]]\!]_{\xi} \equiv [\![Q]\!]_{\xi(\alpha:=[\![P]\!]_{\xi})}$,

(ii) $Q =_{\beta\eta} P \Rightarrow [\![Q]\!]_{\xi} = [\![P]\!]_{\xi}$.

PROOF. The first by an easy induction over the structure of Q. For the second it is sufficient to prove

$$Q \longrightarrow_{\beta\eta} P \Rightarrow [\![Q]\!]_{\xi} = [\![P]\!]_{\xi},$$

which is easily done, by induction over the structure of Q. That this is sufficient follows from the fact that the Church-Rosser property for $\beta\eta$-reduction and Subject Reduction for $\beta\eta$-reduction hold for $\mathrm{F}\omega_{\beta\eta}$. The first is easy by the separation of contexts in $\mathrm{F}\omega$. (See Proposition 4.3.4. In the discussion that ends Chapter 5.1 we have pointed out how to prove $\mathrm{CR}_{\beta\eta}$ for such a system.) SR_{η} for $\mathrm{F}\omega_{\beta\eta}$ is a consequence of Corollary 7.2.4 (but there are easier ways to obtain this result). ⊠

7.3.29. DEFINITION. For Γ a context of $F\omega_{\beta\eta}$ and $\xi \models_{\Box} \Gamma$, an *object valuation of* Γ *with respect to* ξ (notation $\rho, \xi \models \Gamma$) is a map $\rho : \mathsf{Var}^\star \to \Lambda$ such that

$$x : \sigma \in \Gamma \Rightarrow \rho(x) \in [\![\sigma]\!]_\xi.$$

7.3.30. DEFINITION. For Γ a context of $F\omega_{\beta\eta}$ and ρ and ξ valuations such that $\rho, \xi \models \Gamma$, the interpretation function

$$[\![-]\!]_\rho^\Gamma : \Gamma\text{-}\mathsf{Obj}(F\omega_{\beta\eta}) \to \Lambda$$

is defined inductively as follows.

$$\begin{array}{rcl} [\![x]\!]_\rho^\Gamma & = & \rho(x), \\ [\![PQ]\!]_\rho^\Gamma & = & [\![P]\!]_\rho^\Gamma [\![Q]\!]_\rho^\Gamma, \text{ if } Q \text{ is an object}, \\ [\![PQ]\!]_\rho^\Gamma & = & [\![P]\!]_\rho^\Gamma, \text{ if } Q \text{ is a constructor}, \\ [\![\lambda x{:}\sigma.Q]\!]_\rho^\Gamma & = & \lambda x.[\![Q]\!]_{\rho(x:=x)}^\Gamma, \text{ if } \sigma \text{ is a type}, \\ [\![\lambda\alpha{:}k.Q]\!]_\rho^\Gamma & = & [\![Q]\!]_\rho^\Gamma, \text{ if } k \text{ is a kind}. \end{array}$$

In most situations the Γ will be clear from the context, and will therefore not be mentioned explicitly.

The interpretation of objects of $F\omega_{\beta\eta}$ does not use the valuation for the constructor variables at all. We could therefore have given the previous definition without mentioning the ξ, letting ρ be an arbitrary mapping from $\mathsf{Var}^\star$ to Λ. We put the restriction on the ρ because on the one hand it is the natural restriction to make for an interpretation function and on the other hand it will be needed for the theorem we are to be proving.

The fact that the interpretation of objects does not depend on the interpretation of the types is also expressed by the following fact.

7.3.31. FACT. For M an object, ρ a valuation as in the definition and $\vec{x}$ the vector of free variables in M,

$$[\![M]\!]_\rho \equiv |M|^t[\rho(\vec{x})/\vec{x}],$$

where $\rho(\vec{x})$ is the vector obtained by consecutively applying ρ to $\vec{x}$.

7.3.32. DEFINITION. For Γ a context, M an object and σ a type of $F\omega_{\beta\eta}$, Γ *models* M *of type* σ, notation $\Gamma \models M : \sigma$ is defioned by

$$\Gamma \models M : \sigma := \forall \rho, \xi[\rho, \xi \models \Gamma \Rightarrow [\![M]\!]_\rho \in [\![\sigma]\!]_\xi].$$

7.3.33. THEOREM. *For Γ a context, M an object and σ a type of $F\omega_{\beta\eta}$,*

$$\Gamma \vdash M : \sigma \Rightarrow \Gamma \models M : \sigma.$$

PROOF. By induction on the structure of M we prove that if $\rho, \xi \models \Gamma$, then $[\![M]\!]_\rho \in [\![\sigma]\!]_\xi$. So let ρ and ξ be valuations such that $\rho, \xi \models \Gamma$.

- $M \equiv x \in \mathsf{Var}^\star$. Then $x{:}\tau \in \Gamma$ with $\tau =_{\beta\eta} \sigma$. So $[\![M]\!]_\rho = \rho(x) \in [\![\tau]\!]_\xi$ and $[\![\tau]\!]_\xi = [\![\sigma]\!]_\xi$ and we are done.
- $M \equiv \lambda x{:}\tau.Q$ with τ a type. Then $\Gamma, x{:}\tau \vdash Q : \mu$ for some μ with $\sigma =_{\beta\eta} \tau \rightarrow \mu$. By IH $[\![Q]\!]_{\rho(x:=p)} \in [\![\mu]\!]_\xi$ for all $p \in [\![\tau]\!]_\xi$, so $[\![\lambda x{:}\tau.Q]\!]_\rho p \in [\![\mu]\!]_\xi$ for all $p \in [\![\tau]\!]_\xi$, so $[\![\lambda x{:}\tau.Q]\!]_\rho \in [\![\tau]\!]_\xi \rightarrow [\![\rho]\!]_\xi = [\![\sigma]\!]_\xi$.
- $M \equiv \lambda\alpha{:}k.Q$, with k a kind. Then $\Gamma, \alpha{:}k \vdash Q : \tau$ for some τ with $\sigma =_{\beta\eta} \Pi\alpha{:}k.\tau$. By IH $[\![Q]\!]_\rho \in [\![\tau]\!]_{\xi(\alpha:=f)}$ for all $f \in \mathsf{CP}(k)$, and so $[\![\lambda\alpha{:}k.Q]\!]_\rho = [\![Q]\!]_\rho \in \cap_{f \in \mathsf{CP}(k)} [\![\tau]\!]_{\xi(\alpha:=f)} = [\![\sigma]\!]_\xi$.
- $M \equiv PQ$, with Q an object. Then $\Gamma \vdash P : \tau \rightarrow \mu$ and $\Gamma \vdash Q : \tau$ for some τ and μ with $\mu =_{\beta\eta} \sigma$. By IH $[\![P]\!]_\rho \in [\![\tau]\!]_\xi \rightarrow [\![\mu]\!]_\xi$ and $[\![Q]\!]_\rho \in [\![\tau]\!]_\xi$, so $[\![PQ]\!]_\rho = [\![P]\!]_\rho [\![Q]\!]_\rho \in [\![\mu]\!]_\xi = [\![\sigma]\!]_\xi$.
- $M \equiv PQ$, with Q a constructor. Then $\Gamma \vdash P : \Pi\alpha{:}k.\tau$ and $\Gamma \vdash Q : k$ for some τ with $\tau[Q/\alpha] =_{\beta\eta} \sigma$. By IH $[\![P]\!]_\rho \in [\![\tau]\!]_\xi \rightarrow [\![\mu]\!]_\xi$ and $[\![Q]\!]_\rho \in [\![\tau]\!]_\xi$, so $[\![PQ]\!]_\rho = [\![P]\!]_\rho [\![Q]\!]_\rho \in [\![\mu]\!]_\xi = [\![\sigma]\!]_\xi$. By Induction Hypothesis $[\![P]\!]_\rho \in \cap_{f \in \mathsf{CP}(k)} [\![\tau]\!]_{\xi(\alpha:=f)}$. Further we know that $[\![Q]\!]_\xi \in \mathsf{CP}(k)$, so in any case $[\![PQ]\!]_\rho = [\![P]\!]_\rho \in [\![\tau]\!]_{\xi(\alpha:=Q)}$. ⊠

7.3.34. THEOREM.

$$\forall M \in \mathsf{Obj}(F\omega_{\beta\eta})[SN(|M|^t)].$$

PROOF. Let M be an object of $\mathrm{F}\omega_{\beta\eta}$, say that Γ and σ are a context and a type such that $\Gamma \vdash M : \sigma$. Then $\Gamma \models M : \sigma$ by the previous theorem.
Now we define canonical elements c^k in the sets $\mathsf{CP}(k)$ as follows.

$$\begin{array}{rcl} c^\star & := & \mathsf{SN}, \\ c^{k_1 \rightarrow k_2} & := & \boldsymbol{\lambda} f \in \mathsf{CP}(k_1).c^{k_2}. \end{array}$$

For the constructor valuation for Γ we take ξ with $\xi(\alpha) = c^k$ if $\alpha{:}k \in \Gamma$ (and $\xi(\alpha)$ arbitrary otherwise), and for the object valuation for Γ with respect to this ξ we take ρ with $\rho(x) = x$.
Now $\rho, \xi \models \Gamma$ and so $[\![M]\!]_\rho \in [\![\sigma]\!]_\xi$. This implies that $|M|^t$ is Strongly Normalising, because $[\![M]\!]_\rho \equiv |M|^t$ and $[\![\sigma]\!]_\xi \subseteq \mathsf{SN}$. ⊠

Chapter 8

Discussion

At the end of this thesis we want to make some remarks about points that deserve some extra attention. We first try to make the situation around the proof of $\mathrm{SN}_{\beta\eta}$ and $\mathrm{CON}_{\beta\eta}$ for $\mathrm{CC}_{\beta\eta}$ clear. In the middle of all the general Lemmas and Propositions it may have become a bit obscure what exactly is required for these proofs. Then we compare the PTS- syntax with a different formalization of Pure Type Systems which has a more 'semantical' nature.

8.1. Confluence and Normalization

8.1.1. Remark. If one wants to study the confluence of $\beta\eta$-reduction in a Pure Type System, one should be looking at the property $\mathrm{CON}_{\beta\eta}$, i.e.

$$\Gamma \vdash M, N : A \text{ with } M =_{\beta\eta} N \overset{?}{\Rightarrow} M \downarrow_{\beta\eta} N,$$

because $\mathrm{CON}_{\beta\eta}$ is not a consequence of $\mathrm{CR}_{\beta\eta}$ on the well-typed terms. This because a $\beta\eta$-reduction-expansion path from M to N can contain terms that are not typable. ($M =_{\beta\eta} N$ means that they are equal as pseudoterms.) For these non-typable terms, $\mathrm{CR}_{\beta\eta}$ on the well-typed terms does not apply.

The proof of $\mathrm{CON}_{\beta\eta}$ for $\mathrm{CC}_{\beta\eta}$ in this thesis is done in the following steps.

1. Prove the Key Lemma 4.4.18.
2. Prove SR_{β} (Lemma 4.4.30). This is relatively easy, by induction on derivations, using the Key Lemma.
3. Prove SR_{η}. This follows quite easily from the fact that $\mathrm{CC}_{\beta\eta}$ satisfies $\beta\eta$-preservation of sorts. (See Definition 5.2.7, Lemma 7.2.2 and Corollary 7.2.4.)
4. Prove $\mathrm{F}\omega_{\beta\eta} \models \mathrm{CON}_{\beta\eta}$. This is easy, by the fact that contexts in $\mathrm{F}\omega_{\beta\eta}$ can be separated. (See paragraph 5.3 for a proof of $\mathrm{CON}_{\beta\eta}$ of a calculus containing $\mathrm{F}\omega_{\beta\eta}$.)

5. Prove $F\omega_{\beta\eta} \models SN_{\beta\eta}$. This is hard; the proof in paragraph 7.3.2 is done by first showing that it is sufficient to prove $SN_{\beta\eta}$ for erased terms. The proof uses $F\omega_{\beta\eta} \models CON_{\beta\eta}$.

6. Prove $CC_{\beta\eta} \models SN_{\beta\eta}$. This is hard. It is done by defining a reduction preserving mapping from $CC_{\beta\eta}$ to $F\omega_{\beta\eta}$, so the proof uses $F\omega_{\beta\eta} \models SN_{\beta\eta}$.

7. Prove $CC_{\beta\eta} \models CON_{\beta\eta}$. This is hard; it requires $CC_{\beta\eta} \models WN_{\beta\eta}$, so it uses $CC_{\beta\eta} \models SN_{\beta\eta}$. The proof in Chapter 5.1 is for a more general case. For $CC_{\beta\eta}$ it suffices to prove Lemmas 5.2.2, 5.2.4, 5.2.5, Proposition 5.2.3 and Theorem 5.2.6.

Obviously, the fourth, fifth and sixth item can be compressed to one, namely to prove $CC_{\beta\eta} \models SN_{\beta\eta}$. Up to now there is however no other proof of this fact then the one given in this thesis along the lines sketched above.

Some issues immediately come up here. First that we prove Strong Normalization whereas we only need Weak Normalization (usually this property is just called Normalization) for the proof of $CR_{\beta\eta}$. Also in other situations, Weak Normalization often suffices. (For example to prove consistency of a context.) This raises the following conjecture.

8.1.2. Conjecture. *For all Pure Type Systems ζ,*

$$\zeta \models WN_{\beta(\eta)} \Rightarrow \zeta \models SN_{\beta(\eta)}.$$

Another thing that we do not know is if Strong Normalization for a system with (conv_β) implies Strong Normalization for the system with $(\mathrm{conv}_{\beta\eta})$. The problem is that if we extend the conversion rule with η, there are more well-typed terms. (See the discussion in the beginning of section 7.3.) Our intuition says that this extension can not affect the normalization, so we have the following conjecture.

8.1.3. Conjecture. *For all Pure Type Systems ζ,*

$$\zeta \textit{ with } (\mathit{conv}_\beta) \models SN_{\beta(\eta)} \;\Rightarrow\; \zeta \textit{ with } (\mathit{conv}_{\beta\eta}) \models SN_{\beta(\eta)}.$$

Finally we still have the open problem whether $CON_{\beta\eta}$ holds for all Pure Type Systems. We strongly believe that this is so and raise the following conjecture. (Motivated by Proposition 5.3.2.)

8.1.4. Conjecture. *In all Pure Type Systems,*

$$\left.\begin{array}{r} \Gamma \vdash M{:}A \\ \Gamma \vdash M'{:}A \\ M =_{\beta\eta} M' \end{array}\right\} \Rightarrow\; M \downarrow_{\beta\eta} M'.$$

For each of these questions, a counter-example showing that the property does not hold would probably be much more interesting then a proof. (Which makes it unlikely that they will soon be proved, unless there are 'easy' proofs.)

There are reasons to believe that Conjecture 8.1.4 is false. It was shown to us by Werner that Confluence of $\beta\eta$-reduction conflicts with a fixed point combinator. Let us state this precisely for the system $\lambda\star$ with ($\mathrm{conv}_{\beta\eta}$) rule. A fixed point combinator in $\lambda\star$ is a term

$$Y : \Pi\alpha{:}\star.(\alpha\to\alpha)\to\alpha$$

such that

$$Y\sigma F =_{\beta\eta} F(Y\sigma F)$$

for $\sigma : \star$ and $F : \sigma\to\sigma$.

8.1.5. FACT. [Werner 1993] If $\lambda\star$ has a fixed point combinator then $\lambda\star \not\models \mathrm{CON}_{\beta\eta}$ and $\lambda\star \not\models \mathrm{CR}_{\beta\eta}$.

The proof is more general and applies to all PTSs that have a sort $\star$ for which $(\star,\star)$ is a rule and for which there is a sort s such that $(s,\star)$ is a rule and $\star : s$ is an axiom. Hence we have the following Corollary by the fact that $\lambda U \models \mathrm{CON}_{\beta\eta}$. (A proof of this fact was sketched in section 5.3.)

8.1.6. COROLLARY. *In the system λU there is no fixed point combinator.*

Up to now it is not known whether there exists a fixed point combinator in $\lambda\star$. Our conviction that $\mathrm{CON}_{\beta\eta}$ holds has led us to believe that there is no fixed point combinator. (There is a so called 'looping' combinator, which is a family of combinators $Y_0, Y_1, Y_2, \ldots$, all of type $\Pi\alpha{:}\star.(\alpha\to\alpha)\to\alpha$, such that $Y_n\sigma F =_\beta F(Y_{n+1}\sigma F)$. See for example [Coquand and Herbelin 1992].)

8.2. Semantical version of the systems

In fact the Confluence property (Conjecture 8.1.4) is the one that justifies the use of Pure Type Systems with ($\mathrm{conv}_{\beta\eta}$) in the first place.

If one wants to give a semantics to a Pure Type System, one only wants to assign a meaning to the well-typed terms. The pseudoterms are just introduced because they make meta-theory easier, being so closely related to the untyped lambda calculus. Even those who are just interested in syntax will agree with the point of view that only the well-typed terms have a meaning. This point of view implies that if two well-typed terms are equal, but only via a path that passes through the non-typable terms, then these terms should not really be considered as being equal.

Because pseudoterms do not have a semantics, a 'semantical' presentation of Pure Type Systems would not contain a conversion rule of the form that we have. The side-condition in the conversion rule would be stated by an equality judgement of the form $\Gamma \vdash M = N : A$ in stead of an equality condition on the set of pseudoterms. This equality judgement would then be axiomatised in such a way that $\Gamma \vdash M = N : A$ holds only if there is a reduction-expansion path from M to N that passes through the set of well-typed terms of type A in Γ. Obviously, this is also the intended meaning of the equality in the conversion rule of a Pure Type System: If $\Gamma \vdash A, B : \mathsf{Type}$ and $A =_{\beta\eta} B$, then it should be the case that the equality of A and B can be established via a path that passes through the set of Γ-types only. However, when we consider $\beta\eta$-equality it is not clear that this intended meaning is also the actual meaning. (If one only considers β-equality this is obviously the case, due to CR_β on the pseudoterms.)

8.2.1. Definition. The semantical version of a Pure Type System $\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R}))$ has the following rules. The typing rules are (sort), (weak), (var), (Π), (λ), and (app) as for ordinary PTSs. (To denote that we are in a semantical version we write $\vdash_=$ in the rules.) The conversion rule is

$$(\mathrm{conv}'_{\beta\eta}) \quad \frac{\Gamma \vdash_= M : A \quad \Gamma \vdash_= A = B : s}{\Gamma \vdash_= M : B}$$

The judgement $\Gamma \vdash_= A = B : s$ is generated by

$$(\beta) \quad \frac{\Gamma \vdash_= \lambda x{:}A.M : \Pi x{:}C.D \quad \Gamma \vdash_= N : C}{\Gamma \vdash_= (\lambda x{:}A.M)N = M[N/x] : D[N/x]}$$

$$(\eta) \quad \frac{\Gamma \vdash_= M : \Pi x{:}A.B}{\Gamma \vdash_= \lambda y{:}A.My = M : \Pi x{:}A.B}$$

$$(\mathrm{axiom}) \quad \frac{\Gamma \vdash_= M : A}{\Gamma \vdash_= M = M : A}$$

$$(\mathrm{sym}) \quad \frac{\Gamma \vdash_= M = N : A}{\Gamma \vdash_= N = M : A}$$

$$(\mathrm{trans}) \quad \frac{\Gamma \vdash_= M = N : A \quad \Gamma \vdash_= N = Q : A}{\Gamma \vdash_= M = Q : A}$$

$$(\Pi\text{-eq}) \quad \frac{\Gamma \vdash_= A = A' : s_1 \quad \Gamma, x{:}A \vdash_= B = B' : s_2}{\Gamma \vdash_= \Pi x{:}A.B = \Pi x{:}A'.B' : s_3} \quad \text{if } (s_1, s_2, s_3) \in \mathcal{R},$$

$$(\lambda\text{-eq}) \quad \frac{\Gamma \vdash_= A = A' : s \;\; \Gamma, x{:}A \vdash_= M = M' : B \;\; \Gamma \vdash_= \Pi x{:}A.B : s}{\Gamma \vdash_= \lambda x{:}A.M = \lambda x{:}A'.M' : \Pi x{:}A.B}$$

$$(\text{app-eq}) \quad \frac{\Gamma \vdash_= M = M' : \Pi x{:}A.B \quad \Gamma \vdash_= N = N' : A}{\Gamma \vdash_= MN = M'N' : B[N/x]}$$

$$(\text{conv-eq}) \quad \frac{\Gamma \vdash_= M = M' : A \quad \Gamma \vdash_= A = B : s}{\Gamma \vdash_= M = M' : B}$$

We would like to be able to show the equivalence of our version of the syntax of Pure Type Systems and the semantical version in the sense that, if ζ is a $\mathsf{PTS}_{\beta\eta}$ and $\zeta_=$ the semantical version of ζ, then the following holds.

$$\left.\begin{array}{r} \Gamma \vdash_\zeta M : A \\ \Gamma \vdash_\zeta N : A \\ M =_{\beta\eta} N \end{array}\right\} \Leftrightarrow \Gamma \vdash_{\zeta_=} M = N : A.$$

Now, *the* method for proving this is by showing that $\mathrm{CON}_{\beta\eta}$ holds for ζ as it was expressed in Conjecture 8.1.4:

$$\left.\begin{array}{r} \Gamma \vdash_\zeta M{:}A \\ \Gamma \vdash_\zeta M'{:}A \\ M =_{\beta\eta} M' \end{array}\right\} \Rightarrow M \downarrow_{\beta\eta} M'.$$

Let us focus on a possible proof of the equivalence of ζ and $\zeta_=$ to see why $\mathrm{CON}_{\beta\eta}$ is so essential. The implication from right to left should be relatively straightforward by showing that, if $\Gamma \vdash_{\zeta_=} M = N : A$, then $M =_{\beta\eta} N$ as pseudoterms and $\Gamma \vdash_{\zeta_=} M : A$. It is obvious from the rules of $\zeta_=$ that the first holds. The second is by induction on the derivation of $\Gamma \vdash_{\zeta_=} M = N : A$.

The implication from left to right is more interesting. It implies the following statement

(1)If M and N are two terms that are typable with the same type in a context, then they are equal via a $\beta\eta$-reduction-expansion path
through the well-typed terms.

It is even impossible to imagine that one could prove the implication ($\Rightarrow$) without having first proved (1). Obviously, the way to prove (1) is by proving $\mathrm{CON}_{\beta\eta}$.

This stresses the importance of the final open problem (8.1.4) that we raised.

Bibliography

[Avron et al. 1987] A. Avron, F. Honsell and I. Mason, Using typed lambda calculus to implement formal systems on a machine, Report 87-31, LFCS Edingurgh, UK.

[Barendregt 1984] H.P. Barendregt, *The lambda calculus: its syntax and semantics*, revised edition. Studies in Logic and the Foundations of Mathematics, North Holland.

[Barendregt 1992] H.P. Barendregt, Typed lambda calculi. In *Handbook of Logic in Computer Science*, eds. Abramski et al., Oxford Univ. Press.

[Barendsen 1989] E. Barendsen, Representation of logic, data types and recursive functions in typed lambda calculi, Master's thesis, University of Nijmegen, Netherlands, March 1989.

[Barendsen and Geuvers 1989] E. Barendsen and H. Geuvers, λP is conservative over first order predicate logic, Manuscript, Faculty of Mathematics and Computer Science, University of Nijmegen, Netherlands,

[van Benthem Jutting 1977] L.S. van Benthem Jutting, Checking Landau's "Grundlagen" in the Automath system, Ph.D. thesis, Eindhoven University of Technology, Netherlands, 1977.

[van Benthem Jutting 199+] L.S. van Benthem Jutting, Typing in Pure Type Systems. To appear in *Information and Computation*.

[van Benthem Jutting et. al. 1992] L.S. van Benthem Jutting, J. McKinna and R.Pollack, Checking Algorithms for Pure Type Systems, Manuscript.

[Berardi 1988] S. Berardi, Towards a mathematical analysis of the Coquand-Huet calculus of constructions and the other systems in Barendregt's cube. Dept. Computer Science, Carnegie-Mellon University and Dipartimento Matematica, Universita di Torino, Italy.

[Berardi 1989] S. Berardi, Talk given at the 'Jumelage meeting on typed lambda calculus', Edinburgh, September 1989.

[Berardi 1990] S. Berardi, Type dependence and constructive mathematics, Ph.D. thesis, Universita di Torino, Italy.

[Berardi 1990a] S. Berardi, Private Communication.

[Berardi 199+] S. Berardi, Encoding of data types in Pure Construction Calculus: a semantic justification. To appear in the Proceedings of the second BRA meeting on Logical Frameworks, Edinburgh, May 1991.

[Böhm and Berarducci 1985] C. Böhm and A. Berarducci, Automatic synthesis of typed Λ-programs on term algebras *Theor. Comput. Science*, 39, pp 135-154.

[Boyer and Moore 1988] R.S. Boyer and J.S. Moore, *A Computational Logic Handbook*. Academic Press, Boston.

[de Bruijn 1974] N.G. de Bruijn, Some extensions of AUTOMATH: The AUT-4 family, Internal Report, University of Technology, Eindhoven, Netherlands.

[de Bruijn 1980] 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.P. Seldin, J.R. Hindley, Academic Press, New York, pp 580-606.

[CC-documentation] The Calculus of Constructions, documentation and users guide, version 4.10, Technical report, INRIA, August 1989.

[Church 1940] A. Church, A formulation of the simple theory of types *J. Symbolic Logic*, 5, pp 56-68.

[Constable et.al. 1986] R.L. Constable et.al., *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall.

[Coquand 1985] Th. Coquand, Une théorie des constructions, Thèse de troisième cycle, Université Paris VII, France, January 1985.

[Coquand 1986] Th. Coquand, An analysis of Girard's paradox, *Proceedings of the first symposium on Logic in Computer Science, Cambridge Mass.*, IEEE, pp 227-236.

[Coquand 1986a] Th. Coquand, A Calculus of Constructions, Manuscript, INRIA, France.

[Coquand 1990] Th. Coquand, Metamathematical investigations of a calculus of constructions. In *Logic and Computer Science*, ed. P.G. Odifreddi, APIC series, vol. 31, Academic Press, pp 91-122.

[Coquand 1991] Th. Coquand, An algorithm for testing conversion in type theory. In Huet and Plotkin (eds.), *Logical Frameworks*, Cambridge Univ. Press.

[Coquand 199+] Th. Coquand, A new paradox in type theory, to appear in *Proceedings of the 9th International Congress of Logic, Methodology and Philosophy of Science, Uppsala, Sweden 1991.*

[Coquand and Gallier 1990] Th. Coquand and J. Gallier, A proof of strong normalization for the Theory of Constructions using a Kripke-like interpretation, Informal Proceedings of the BRA-Logical Frameworks meeting, Antibes 1990, pp 479–497

[Coquand and Herbelin 1992] Th. Coquand and H. Herbelin, An Application of A-translation to the existence of families of looping combinators in inconsistent Type Systems, to appear in *Journal of Functional Programming.*

[Coquand and Huet 1988] Th. Coquand and G. Huet, The calculus of constructions, *Information and Computation*, 76, pp 95-120.

[Coquand and Huet 1985] Th. Coquand and G. Huet, Constructions: a higher order proof system for mechanizing mathematics. *Proceedings of EUROCAL '85, Linz*, LNCS 203.

[Coquand and Mohring 1990] Th. Coquand and Ch. Paulin-Mohring Inductively defined types, In P. Martin-Löf and G. Mints editors. *COLOG-88 : International conference on computer logic, LNCS 417.*

[Curry and Feys 1958] H.B. Curry and R. Feys, *Combinatory Logic, Vol. 1*. North-Holland.

[van Daalen 1973] D. van Daalen, A description of AUTOMATH and some aspects of its language theory, In P. Braffort, ed. *Proceedings of the symposium on APL, Paris.*

[van Daalen 1980] D. van Daalen, The language theory of AUTOMATH, Ph.D. thesis, Eindhoven Technological University, The Netherlands, Februari 1980.

[van Dalen 1983] D. van Dalen, *Logic and Structure*, second edition. Springer Verlag.

[Dowek et al. 1991] G. Dowek, A. Felty, H. Herbelin, G. Huet, Ch. Paulin-Mohring, B. Werner, The Coq proof assistant version 5.6, user's guide. INRIA Rocquencourt - CNRS ENS Lyon.

[Gallier 1990] J. Gallier, On Girard's "Candidats de Reductibilité". In *Logic and Computer Science*, ed. P.G. Odifreddi, APIC series, vol. 31, Academic Press, pp 123-204.

[Gardner 1992] P. Gardner, Representing logics in type theory, Ph.D. thesis, University of Edinburgh, UK, January 1992.

[Geuvers 1988] H. Geuvers, The interpretation of logics in type systems, Master's thesis, University of Nijmegen, Netherlands, August 1988.

[Geuvers 1989] J.H. Geuvers, Talk given at the 'Jumelage meeting on typed lambda calculus', Edinburgh, September 1989.

[Geuvers 1990] J.H. Geuvers, Type systems for higher order predicate logic, Manuscript, University of Nijmegen, Netherlands, May 1990.

[Geuvers and Nederhof 1991] J.H. Geuvers and M.J. Nederhof, A modular proof of strong normalisation for the calculus of constructions. *Journal of Functional Programming*, vol 1 (2), pp 155-189.

[Geuvers 1992] J.H. Geuvers, The Church-Rosser property for $\beta\eta$-reduction in typed lambda calculi. In *Proceedings of the seventh annual symposium on Logic in Computer Science, Santa Cruz, Cal.*, IEEE, pp 453-460.

[Geuvers 199+] J.H. Geuvers, The Calculus of Constructions and higher order logic, to appear in *The Curry-Howard isomorphism, 8ème volume des cahiers du Centre de Logique de l'Université Catholique de Louvain*, eds. M. Crabbe and Ph. de Groote.

[Girard 1971] J.-Y. Girard, Une extension de l'interprétation fonctionnelle de Gödel à l'analyse et son application à l'élimination des coupures dans l'analyse et la théorie des types. *Proceedings of the Second Scandinavian Logic Symposium*, ed. J.E. Fenstad, North-Holland.

[Girard 1972] J.-Y. Girard, Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur. Ph.D. thesis, Université Paris VII, France.

[Girard 1986] J.-Y. Girard, The system F of variable types, fifteen years later. *TCS* 45, pp 159-192.

[Girard et al. 1989] J.-Y. Girard, Y. Lafont and P. Taylor, *Proofs and types*, Camb. Tracts in Theoretical Computer Science 7, Cambridge University Press.

[Gordon et al. 1976] M.J. Gordon, A.J. Milner, A.P. Wadsworth, *Edinburgh LCF*, LNCS 89.

[Gordon 1988] M.J. Gordon, HOL: a proof generating system for higher-order logic. *VLSI specification, Verification and Synthesis*, Eds. G. Birtwistle and P.A. Subrahmanyam, Kluwer, Dordrecht, pp 73-128.

[Harper et al. 1987] R. Harper, F. Honsell and G. Plotkin, A framework for defining logics. *Proceedings Second Symposium on Logic in Computer Science*, (Ithaca, N.Y.), IEEE, Washington DC, pp 194-204.

[Harper and Pollack 1991] R. Harper and R. Pollack, Type checking with universes, *TCS* 89, pp 107-136.

[Heyting 1934] A. Heyting, *Mathematische Grundlagenforschung. Intuitionismus. Beweistheorie*, Springer, Berlin. Reprinted 1974.

[Howard 1980] W.A. Howard, The formulas-as-types notion of construction. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, eds. J.P. Seldin, J.R. Hindley, Academic Press, New York, pp 479-490.

[Hyland and Pitts 1988] M. Hyland and A. Pitts, The theory of constructions: categorical semantics and topos-theoretic models. In *Categories in computer science and logic, Proc. of the AMS Research Conference, Boulder, Col.*, eds. J.W. Gray and A.S. Scedrov, Contemporary Math., vol 92, AMS, pp 137-199.

[Kolmogorov 1932] A.N. Kolmogorov, Zur Deutung der Intuitionistischen Logik, *Math. Z.* 35, pp 58–65.

[Krivine and Parigot 1990] J.-L. Krivine and M. Parigot, Programming with proofs, *J. Inf. Process. Cybern.* EIK 26 3, pp 149–167.

[Lambek and Scott 1986] J. Lambek and P.J. Scott, *Introduction to higher order Categorical Logic*, Cambridge studies in advanced mathematics 7, Camb. Univ.Press.

[Läuchli 1970] H. Lauchli, An abstract notion of realizability for which intuitionistic predicate calculus is complete. In *Intuitionism and Proof Theory, Proceedings of the Summer School Conference at Buffalo, New York*, eds. G. Myhill, A. Kino and R. Vesley, North-Holland, pp 227–234.

[Longo and Moggi 1988] G. Longo and E. Moggi, Constructive Natural Deduction and its “Modest” Interpretation. Report CMU-CS-88-131.

[LEGO-examples] R. Pollack et al., Examples of proofs formalised in LEGO, Edinburgh.

[Löb 1976] M. Löb, Embedding first order predicate logic in fragments of intuitionistic logic, *J. Symbolic Logic* vol 41, 4 pp. 705–719.

[Luo 1989] Z. Luo, ECC: An extended Calculus of Constructions. *Proc. of the fourth ann. symp. on Logic in Comp. Science, Asilomar, Cal.* IEEE, pp 386-395.

[Luo and Pollack 1992] Z. Luo, R. Pollack, Lego proof development system: User's Manual, Dept. of Computer Science, University of Edinburgh, April 1992.

[Martin-Löf 1971] P. Martin-Löf, A theory of types, manuscript, October 1971.

[Martin-Löf 1975] P. Martin-Löf, An intuitionistic theory of types: predicative part. *Logic Colloquium '73*, North-Holland 1975, pp 73-118.

[Martin-Löf 1982] P. Martin-Löf, Constructive mathematics and computer programming. *Sixth International Congress for Logic, Methodology, and Philosophy of Science VI, 1979*, North-Holland 1982, pp 153-175.

[Martin-Löf 1984] P. Martin-Löf, *Intuitionistic Type Theory*, Studies in Proof theory, Bibliopolis, Napoli.

[Mendler 1987] N.P. Mendler, Inductive types and type constraints in second-order lambda calculus. *Proceedings of the Second Symposium of Logic in Computer Science*. Ithaca, N.Y., IEEE, pp 30-36.

[Mitchell 1986] J. Mitchell, A type-inference approach to reduction properties and semantics of polymorphic expressions. In *Proceedings of 1986 ACM Symposium on LISP and Functional Programming*, ACM New York, pp 308–319,

[Mohring 1986] Ch. Mohring, Algorithm development in the calculus of constructions. In *Proceedings of the first symposium on Logic in Computer Science, Cambridge, Mass.* IEEE, pp 84-91.

[Nederpelt 1973] R.P. Nederpelt, Strong normalization in a typed lambda calculus with lambda structured types. Ph.D. thesis, Eindhoven Technological University, The Netherlands, June 1973.

[Nordström et al. 1990] B. Nordström, K. Petersson, J.M. Smith, *Programming in Martin-Löf's Type Theory*. Oxford University Press.

[Paulin 1989] Ch. Paulin-Mohring, Extraction des programmes dans le calcul des constructions, Thèse, Université Paris VII, France.

[Paulson 1987] L.C. Paulson, *Logic and Computation.* Cambridge Tracts in Theoretical Computer Science 2, Cambridge University Press.

[Parigot 1992] M. Parigot, Recursive programming with proofs. *Theor. Comp. Science* 94, pp 335-356.

[Pollack 1989] R. Pollack, Talk given at the 'Jumelage meeting on typed lambda calculus', Edinburgh, September 1989.

[Pottinger 1989] G. Pottinger, Definite descriptions and excluded middle in the theory of constructions, TYPES network, November 1989.

[Prawitz 1965] D. Prawitz, *Natural Deduction*, Almqvist and Wiksell, Stockholm.

[Reynolds 1974] J.C. Reynolds, Towards a theory of type structure. *Proceedings, Colloque sur la Programmation*, LNCS 19, pp 408-425.

[Ruys 1991] M. Ruys, $\lambda P\omega$ is not conservative over $\lambda P2$, Master's thesis, University of Nijmegen, Netherlands, November 1991.

[Salvesen 1989] A. Salvesen, The Church-Rosser Theorem for LF with η reduction. Notes of a talk presented at the BRA-Logical Frameworks meeting, Antibes 1990.

[Salvesen1991] A. Salvesen, The Church-Rosser property for $\beta\eta$-reduction, manuscript.

[Scedrov 1990] A. Scedrov, A guide to polymorphic types. In *Logic and Computer Science*, ed. P.G. Odifreddi, APIC series, vol. 31, Academic Press, pp 387-420.

[Seldin 1990] J. Seldin, Excluded middle without definite descriptions in the theory of constructions, TYPES network, September 1990.

[Schütte 1977] K. Schütte, *Proof Theory*, Grundlehren der mathematischen Wissenschaften 225, Springer-Verlag.

[Smorynski 1973] C. Smorynski, Applications of Kripke models, in *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, ed. A. Troelstra, LNM 344, pp 324–391.

[Streicher 1988] T. Streicher, Correctness and completeness of a categorical semantics of the calculus of constructions, Ph.D. Thesis, Passau University, Germany.

[Streicher 1991] T. Streicher, Independence of the induction principle and the axiom of choice in the pure calculus of constructions, *TCS* 103(2), pp 395 - 409.

[Swaen 1989] Weak and strong sum-elimination in intuitionistic type theory, Ph.D. thesis, Faculty of Mathematics and Computer Science, University of Amsterdam, Netherlands, September 1989.

[Tait 1965] W.W. Tait, Infinitely long terms of transfinite type. In *Formal Systems and Recursive Functions*, eds. J.N. Crossley and M.A.E. Dummett, North-Holland.

[Tait 1975] W.W. Tait, A realizability interpretation of the theory of species. In *Proceedings of Logic Colloquium*, ed. R. Parikh, LNM 453, pp 240–251.

[Takeuti 1975] G. Takeuti, *Proof Theory*, Studies in Logic, vol. 81, North-Holland.

[Terlouw 1989a] J. Terlouw, Een nadere bewijstheoretische analyse van GSTT's (incl. appendix), Manuscript, Faculty of Mathematics and Computer Science, University of Nijmegen, Netherlands, March, April 1989. (In Dutch)

[Terlouw 1989b] J. Terlouw, Sterke normaliszatie in C á la Tait, Notes of atalk held at the Intercity Seminar on Typed Lambda Calculus, Nijmegen, Netherlands, April 1989. (In Dutch)

[Tonino anf Fujita 1992] H. Tonino and K.-E. Fujita, On the adequacy of representing higher order intuitionistic logic as a pure type system, *Annals of Pure and Applied Logic* 57, pp 251–276.

[Troelstra and Van Dalen 1988] A. Troelstra and D. van Dalen, *Constructivism in mathematics, an introduction, Volume I/II*, Studies in logic and the foundations of mathematics, vol 121 and volume 123, North-Holland.

[Verschuren 1990] E. Verschuren, Conservativity in Barendregt's cube, Master's thesis, University of Nijmegen, Netherlands, December 1990.

[Werner 1993] B. Werner, Private Communication.

Index

Samenvatting

Dit proefschrift behandelt het verband tussen logica's en getypeerde lambd-calculi, in het bijzonder door bestudering van de zogenaamde 'formules-als-types' inbedding. Deze inbedding geeft een betekenis aan logische bewijzen (het ware misschien beter te spreken van 'afleidingen') in termen van getypeerde lambda termen. Een belangrijk gevolg hiervan is dat de bewijzen een getrouwe lineaire representatie krijgen in een formeel systeem. Dit heeft aanleiding gegeven tot belangrijke toepassingen, allen gebaseerd op de mogelijkheid tot het manipuleren van bewijzen binnen een formeel systeem. Men denke hierbij aan de computer-verificatie van bewijzen en aan de mogelijkheid om uit een bewijs van een uitspraak van de vorm $\forall x.\exists y.\varphi(x, y)$ een algoritme te extraheren dat voor iedere x een y berekent waarvoor $\varphi(x, y)$ geldt. In dit proefschrift wordt met name gekeken naar de formules-als-types inbedding zelf en tevens worden de bijbehorende systemen van getypeerde lambda calculus uitgebreid bestudeerd. Slechts zijdelings wordt in de hoofdstukken 3.1 en 6.1 enige aandacht besteed aan de toepassingen.

De formules-als-types inbedding werd voor het eerst formeel beschreven in [Howard 1980], die ook de eerste was die de terminology 'formulas-as-types' gebruikte. Het manuscript van dit artikel dateert al uit 1968 en veel van de ideeën uit dit werk zijn nog ouder en gaan terug tot Curry (zie bijvoorbeeld [Curry and Feys 1958]). Howard stelt zich met name tot doel een interpretatie te geven van de intuitionistische logische voegtekens volgens de zogenaamde Brouwer-Heyting-Kolmogorov (BHK) interpretatie. Volgens deze BHK interpretatie (zie bijvoorbeeld [Troelstra and Van Dalen 1988]) wordt een voegteken verklaard door te zeggen wanneer iets een bewijs is van een uitspraak die opgebouwd is met behulp van dat voegteken. Howard geeft een formele semantiek van intuitionistische bewijzen in termen van een getypeerde lambda calculus door een interpretatie te geven van de introductie en eliminatie regels van de voegtekens. De introductie regels voor $\supset$ en $\forall$ corresponderen bijvoorbeeld met λ-abstractie en de eliminatie regels voor $\supset$ en $\forall$ met applicatie. Het werk van Howard is later verfijnd en uitgebreid door onder andere Martin-Löf en Girard.

Een andere benadering werd gekozen door de Bruijn in het Automath project [de Bruijn 1980], die onafhankelijk van Howard een soort van formules-als-types inbedding definiëerde. Deze inbedding heeft een andere vorm, met name vanwege het feit dat de Bruijn niet gericht was op bewijstheoretische bespiegelingen maar op een veel praktischer doel: het formaliseren van wiskundig redeneren op een computer. Het verschil in vorm zit hem erin dat men niet zoekt naar getypeerde lambda-calculi die getrouw met logische systemen corresponderen, maar dat men één systeem probeert te vinden dat kan dienen als raamwerk (logical framework) voor wiskundig redeneren. Dit raamwerk zal dus vrij 'kaal' zijn en alleen die onderliggende principes bevatten waar alle wiskundigen het over eens zijn. In de eerste plaats is de Bruijns werk dus een poging om deze onderliggende principes boven tafel te krijgen, met als mogelijk gevolg dat, zodra deze principes geformaliseerd zijn, deze geïmplementeerd kunnen worden als een programma voor verificatie van wiskundige redeneringen.

Uiteraard kunnen ook de lambda-calculi *à la Howard* geïmplementeerd worden. Met name voor de toepassing van het extraheren van algoritmen uit bewijzen blijken deze systemen het meest geschikt te zijn. Het is uiteraard ook mogelijk om beide benaderingen te gebruiken binnen één systeem.

Het voornaamste deel van dit proefschrift is gewijd aan de formules-als-types inbedding *à la Howard*. Interessante vragen hierbij zijn of de inbedding volledig is en in hoeverre zij een isomorfisme is. Volledigheid van de inbedding betekent hier dat voor alle formules φ uit de logica, als er een term is van type φ in de getypeerde lambda calculus, dan is φ bewijsbaar in de logica. Isomorphie wil zeggen dat de inbedding een structuur behoudende bijectie op het niveau van bewijzen is. Het is ook van belang eigenschappen van de getypeerde lambda-calculi zelf af te leiden. In de eerste plaats om met behulp van die eigenschappen iets over de formules-als-types interpretatie te zeggen, maar verder zijn deze eigenschappen ook van belang voor de implementatie van de calculus. Tot slot hebben zij vaak ook belangrijke corollaria in de logica's.

De twee belangrijkste van deze eigenschappen zijn confluentie en normalisatie. Zowel in de logische taal (zeker als die hogere orde is) als op de bewijzen is er een notie van reductie en een daaruit voortvloeiende notie van gelijkheid. In de logische taal worden deze meestal gerepresenteerd door de β- (of $\beta\eta$-)reductie en gelijkheid. Deze wordt vaak de definitionele gelijkheid van de taal genoemd. De gelijkheid op afleidingen komt voort uit de reductie-relatie die bestaat uit het elimineren van sneden. Nu is het zo dat in de bijbehorende getypeerde lambda-calculi zowel de definitionele gelijkheid als de gelijkheid op afleidingen gerepresenteerd worden door β- of $\beta\eta$-gelijkheid (afhankelijk van wat men precies als definitionele gelijkheid in de taal neemt en hoe men precies de notie van snede definieert.) De confluentie eigenschap (die zegt dat twee termen die gelijk zijn ook een gemeenschappelijk reduct hebben) is van vitaal belang om te laten zien dat niet alle termen aan elkaar gelijk zijn. De normalisatie-eigenschap (die zegt dat iedere term reduceert naar een term in normaal vorm, i. e. een term die niet verder gereduceerd kan worden) is van vitaal belang om de consistentie van een (logische) theorie te laten zien.

Dit proefschrift is opgebouwd uit de volgende componenten.

Hoofdstuk 2 geeft een overzicht van logische systemen, van eerste orde propositielogica tot en met hogere orde predicatenlogica, in de klassieke en intuitionistische varianten. We beschrijven en bewijzen eigenschappen en verbanden zoals beslisbaarheid en conservativiteit.

Hoofdstuk 3 geeft een gedetailleerde bescrijving van de formules-als-types inbedding, zowel die à la Howard als die à la de Bruijn. We geven een gedetailleerd bewijs van de isomorfie van eerste orde predicatenlogica en een corresponderende getypeerde lambda-calculus.

In Hoofdstuk 4 bestuderen we een algemeen raamwerk voor de beschrijving van getypeerde lambda-calculi, de zogenaamde 'Pure Type Systems'. We bewijzen een reeks eigenschappen voor deze systemen en geven voorbeelden van Pure Type Systems die corresponderen met logica's uit Hoofdstuk 1.

In Hoofdstuk 5 bestuderen we de confluentie van $\beta\eta$-reductie in getypeerde lambda calculi. Confluentie van β-reductie is relatief eenvoudig, maar voor $\beta\eta$ is het algemene probleem verrassend moeilijk. Het algemene resultaat dat we hier bewijzen is dat confluentie geldt voor $\beta\eta$ als het Pure Type System normalizerend is.

Hoofdstuk 6 gaat over de Calculus of Constructions (CC), een getypeerde lambda-calculus gedefiniëerd door Coquand en Huet waarin de hogere orde logica ingebed kan worden door middel van de formules-als-types inbedding. We bestuderen CC en zijn fijnstructuur en de inbedding van logica in (subsystemen van) CC.

Hoofdstuk 7 geeft een gedetailleerd bewijs van sterke normalisatie van $\beta\eta$-reductie in CC. (Sterke normalisatie betekent dat er geen oneindige reductie paden zijn.)

Tenslotte bespreken we in Hoofdstuk 8 nog een aantal vermoedens die naar voren komen naar aanleiding van de bewijzen van confluentie en normalisatie.

Curriculum Vitae

Ik ben geboren op 19 mei 1964 te Deventer, alwaar ik van zomer 1976 tot zomer 1978 de brugklas en de tweede klas van het Gymnasium volgde op de Alexander Hegius Scholengemeenschap. Van zomer 1978 tot zomer 1982 volgde ik de overige vier klassen van het Gymnasium (later OVWO) op het Stedelijk Lyceum te Zutphen en bekroonde dit met een diploma.

Van september 1982 tot en met augustus 1988 heb ik Wiskunde gestudeerd aan de KUN. Mijn afstudeerrichting was Grondslagen van de Wiskunde en ik heb mijn afstudeerwerk verricht bij Professor H.P. Barendregt. Hoewel ik in diezelfde periode nog zitting heb gehad in de onderwijscommissie, de sectieraad en het sectiebestuur van de vakgroep Wiskunde en in de faculteitsraad van de faculteit Wiskunde en Natuurwetenschappen, werden zowel het propedeutisch als het doctoraal examen met lof behaald. Verder zijn uit deze periode nog vermeldenswaard de studentassistentschappen bij Wiskunde voor Chemici, Wiskunde voor Biologen en Statistiek en Kansrekening, Inleiding in de Wiskunde en Logica, alle drie voor Informatici.

Na mijn afstuderen ben ik van september 1988 tot en met december 1988 uitgezonden geweest naar het Dipartimento di Informatica van de Universiteit van Pisa om te werken onder Professor G. Longo. Deze uitzending vond plaats in het kader van het 'Jumelage project getypeerde lambda calculus'. Daarna ben ik van 1 maart 89 tot 1 maart 1993 Assistent in Opleiding geweest bij de vakgroep Informatica van de KUN. Onder supervisie van Professor H.P. Barendregt en gefinancierd door het TLI netwerk deed ik daar onderzoek op het gebied van de getypeerde lambda calculus. In deze periode heb ik het college Grondslagen van de Informatica 2 gegeven en de helft van het college Theorie 1. Tevens heb ik geassisteerd bij het oriëntatiecollege Grondslagen van de Informatica en bij Grondslagen van de Informatica 3. Van 1 maart 93 tot 1 augustus 93 ben ik Toegevoegd Onderzoeker bij dezelfde vakgroep Informatica geweest in het kader van het ESPRIT BRA project 'Types for Proofs and Programs'. Vanaf 1 augustus 93 ben ik Universitair Docent bij de vakgroep Theoretische Informatica van de TUE. Tevens blijf ik voor één dag in de week Toegevoegd Onderzoeker bij de vakgroep Informatica van de KUN bij het voornoemde ESPRIT BRA project.