Investigations into Proof-Search in a System of First-Order Dependent Function Types

by

David Pym and Lincoln Wallen

LFCS Report Series ECS-LFCS-90-111

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

March 1990

To appear: Proc. 10th International Conference on Automated Deduction, Kaiserslautern, FRG, July 1990. Springer-Verlag. Please reference this publication.

INVESTIGATIONS INTO PROOF-SEARCH IN A SYSTEM OF FIRST-ORDER DEPENDENT FUNCTION TYPES

David Pym
University of Edinburgh
Scotland, U.K.
dpym@uk.ac.ed.lfcs

Lincoln Wallen
University of Oxford
England, U.K.
lw@uk.ac.ox.prg

Abstract

We present a series of proof systems for $\lambda\Pi$-calculus: a theory of *first-order dependent function types.* The systems are complete for the judgement of interest but differ substantially as bases for algorithmic proof-search. Each calculus in the series induces a search space that is properly contained within that of its predecessor. The $\lambda\Pi$-calculus is a candidate *general logic* in that it provides a metalanguage suitable for the encoding of logical systems and mathematics. Proof procedures formulated for the metalanguage extend to suitably encoded object logics, thus removing the need to develop procedures for each logic independently. This work is also an exploration of a *systematic* approach to the design of proof procedures. It is our contention that the task of designing a computationally efficient proof procedure for a given logic can be approached by formulating a series of calculi that possess specific proof-theoretic properties. These properties indicate that standard computational techniques such as *unification* are applicable, sometimes in novel ways. The study below is an application of this design method to an intuitionistic type theory.

Our methods exploit certain forms of *subformula property* and *reduction ordering* — a notion introduced by Bibel for classical logic, and extended by Wallen to various non-classical logics — to obtain a search calculus for which we are able to define notions of *compatibility* and *intrinsic well-typing* between a derivation ψ and a substitution σ calculated by unification which closes ψ (a derivation is closed when all of its leaves are axioms). Compatibility is an acyclicity test, a generalization of the *occurs-check* which, subject to intrinsic well-typing, determines whether the derivation ψ and substitution σ together constitute a *proof.*

Our work yields the (operational) foundations for a study of *logic programming* in this general setting. This potential is not explored here.

1 Introduction

We present a series of proof systems for $\lambda\Pi$-calculus: a theory of *first-order dependent function types* [vDa80,MR86,HHP87]. The systems are complete for the judgement of interest but differ substantially as bases for algorithmic proof-search. Each calculus in the series induces a search space that is properly contained within that of its predecessor.

Our interest in proof-search in $\lambda\Pi$-calculus stems from two sources. Firstly, the theory is a candidate *general logic* in that it provides a metalanguage suitable for the encoding of logical systems [AHM87] and mathematics [vDa80]. Proof procedures formulated for the metalanguage extend to suitably encoded object logics, thus removing the need to develop procedures for each logic independently.[1] The second reason for our interest arises out from a desire (expressed in [Wal89]) for a *systematic* approach to the design of proof procedures. It is our contention that the task of designing a computationally efficient proof procedure for a given logic can be approached by formulating a series of calculi that possess specific proof-theoretic properties. These properties are the indicators that standard computational techniques such as *unification* are applicable, sometimes in novel ways. The study below is an application of this design method to an intuitionistic type theory.

[1]This should be compared with the use of proof procedures for classical logic to effect proof-search within a modal logic, say, by means of an encoding of the latter in the former; see [Ohl88] for example.

The first system of the series, called **N**, is a natural deduction formulation of $\lambda\Pi$-calculus.[2] The main judgement of **N** is the typing assertion: $\Gamma \vdash M{:}A$, meaning that the term M has type A, given the type assignments for free variables and constants recorded in Γ. This relation is decidable [HHP87]. A typical rule of **N** is the elimination rule for the dependent function type constructor (Π):[3]

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

where $B[N/x]$ denotes capture-avoiding substitution of N for free occurrences of x in B.

The second system, called **L**, is sound and complete (relative to **N**) for the semi-decidable relation of *inhabitation:* $\Gamma \Rightarrow A$, with the meaning $(\exists M)(\Gamma \vdash M{:}A)$. The judgements of **L** assert the *existence* of proofs of the judgements of **N**, as in the case of first-order logic [Pra65]. **L** is the starting point for our investigation into proof-search. A typical rule of **L** is the Π-left rule, the counterpart of the Π-elimination rule of **N**:

$$\Pi l \quad \frac{\Gamma, z{:}B[N/x] \Rightarrow C}{\Gamma \Rightarrow C} \qquad \begin{array}{l}\text{(a) } w : (\Pi x{:}A.B) \in \Gamma \\ \text{(b) } z \notin \mathrm{Dom}(\Gamma) \\ \text{(c) } \Gamma \vdash N{:}A.\end{array}$$

L is almost a *logicistic* system, in the sense of Gentzen, meaning that there is only one localized appeal to an external notion, that notion being an appeal to the system **N** in side condition (c) of the rule above. Indeed, with respect to the Π-type structure of terms in the language, **L** has a *subformula property* [Gen34]. As a consequence, if the inference rules are used as *reduction operators* from conclusion to premisses then **L** induces a search space of derivations of a given sequent.[4] Notice that if the Πl rule is used as a reduction, the choice of term N to use in the premiss is unconstrained by the conclusion of the rule. The sub*formula* property of the Π-types does not extend to a full sub*term* property (*cf.* the quantifier rules of the predicate calculus). The *axiom* sequent (or *closure* condition for the reduction system) is:

$$\Gamma, z{:}A, \Gamma' \Rightarrow A$$

i.e., the conclusion occurs as the type of a declaration in the context.

The third and fourth systems are also systems of sequents. The system **U** is formed from **L** by removing the appeal to **N** in the Πl rule. The Πl rule of **U** is thus:

$$\Pi l \quad \frac{\Gamma, z{:}B[\alpha/x] \Rightarrow C}{\Gamma \Rightarrow C} \qquad \begin{array}{l}\text{(a) } w : (\Pi x{:}A.B) \in \Gamma \\ \text{(b) } z \notin \mathrm{Dom}(\Gamma).\end{array}$$

This rule introduces a free, or *universal* variable into the proof, denoted here by α. Universal variables are distinct from the usual *eigenvariables* that are bound by the derivation (and explicitly declared in contexts). The axiom sequent of **U** is used to compensate for the omission of term information in the Πl rule as follows:

$$\Gamma, z{:}B, \Gamma' \Rightarrow A \qquad \text{(a) } (\exists\sigma)\, B\sigma \equiv A\sigma$$

where σ is an *instantiation* of the universal variables of the sequent, and $B\sigma$ denotes the term resulting from the application of σ (as a substitution) to B. The calculation of instantiations

[2]This system is also known as the type system of the (Edinburgh) Logical Framework or LF [HHP87].

[3]Readers that are unfamiliar with dependent types should read the construction $\Pi x{:}A.B$ as a (typed) universal quantifier such as $\forall x{:}A.B$; x is the bound variable, ranging over the type A, which may occur free in the term B.

[4]Kleene [Kle68] explains this in the case of the predicate calculus. Sequent systems used in this way are systems of *block tableaux* [Smu68].

can be performed by a *unification algorithm* for the language. A suitable algorithm has been developed by the first author [Pym90] based on a standard algorithm for simple type theory [Hu75].

A **U**-proof is a pair $\langle \psi, \sigma \rangle$ consisting of a **U**-derivation ψ and an instantiation σ such that $\psi\sigma$ (the application of σ as a substitution to ψ) is an **L**-proof. Not every instantiation that closes the leaves of a given **U**-derivation will yield an **L**-proof when applied. It is sufficient to check that the instantiation can be *well-typed* in the derivation to ensure that the result is an **L**-proof. If Γ is the context and A the type of the universal variable α when introduced into the **U**-derivation, the well-typing condition for α amounts to:

$$\Gamma\sigma \vdash \alpha\sigma : A\sigma$$

ensuring that side condition (c) of the Πl rule of **L** — the condition omitted from the Πl rule of **U** — is nevertheless satisfied in $\psi\sigma$. The unconstrained choice of term in the Πl rule of **L** is replaced by a highly constrained choice in **U**.[5] This wholesale reduction in the search space is analogous, of course, to that obtained by Robinson in the context of the predicate calculus [Rob65].

The well-typing of an instantiation depends on the structure of the derivation from which it is calculated. However, even if it fails to be well-typed in that derivation it may be well-typed in some permutation of the derivation, since rule applications (or reductions) can sometimes be permuted whilst leaving the endsequent (*i.e.,* root) of the derivation and its leaves unchanged. The degree to which this can be done is summarized in the form of a *Permutation Theorem,* in the sense of Kleene [Kle52] and Curry [Cur52], and underlies the fourth and final system of the paper called **R**.

The rules of **R** are just the rules of **U** (so the derivations are the same) but the condition for instantiations to yield a proof is weakened. An **R**-proof is again a pair consisting of a derivation ψ and an instantiation σ under which ψ is closed, but now we require only that there exist perhaps another (closed) derivation ψ^* in which the instantiation is well-typed; *i.e.,* $\psi^*\sigma$ is an **L**-proof. Of course the crucial computational question is whether the existence of at least one suitable ψ^*, given ψ and σ, can be determined as the search progresses. We show that this is indeed the case using a *reduction ordering:* a notion which was introduced by Bibel [Bib81] for classical connectives and extended by the second author in [Wal89] to various non-classical connectives.[6] An **R**-proof therefore corresponds to an equivalence class of **U**-proofs of the same endsequent consisting of all permutation variants of the original derivation in which the calculated instantiation is well-typed.

The reader may find it helpful to refer back to the overview given above to identify the motivation for various technicalities below.

[5]The inference system that corresponds to the calculation of instantiations by unification does indeed have a *subterm property.* The soundness and completeness result for **U** is a form of *Herbrand Theorem* for the theory. The unification algorithm searches amongst the terms of a "Herbrand universe" defined by each leaf sequent. This aspect is not explored in detail in this paper, see [Pym90].

[6]The condition is equivalent to an enhanced "occurs-check" in the unification algorithm if a suitable notion of *Skolem function* were introduced. The suitable notion is *not* the obvious one that the reader might suppose from experience of classical quantifiers, not least because the logic under investigation here is intuitionistic. In general, the theoretical diversion via Skolemization is unnecessary and can be difficult to justify semantically, even in simple type theory (*cf.* [Mil83]). Our approach follows Herbrand's Theorem (which is finitary) rather than the Skolem-Herbrand-Gödel Theorem (which is not).

2 $\lambda\Pi$-calculus: a theory of dependent function types

The syntax of $\lambda\Pi$-calculus is given by the following grammar:

$$\begin{array}{llcl} \textit{Signatures} & \Sigma & ::= & \langle\rangle \mid \Sigma, c{:}K \mid \Sigma, c{:}A \\ \textit{Contexts} & \Gamma & ::= & \langle\rangle \mid \Gamma, x{:}A \\ \textit{Kinds} & K & ::= & \mathrm{Type} \mid \Pi x{:}A.K \\ \textit{TypeFamilies} & A & ::= & c \mid \Pi x{:}A.A \mid \lambda x{:}A.A \mid AM \\ \textit{Objects} & M & ::= & c \mid x \mid \lambda x{:}A.M \mid MM \end{array}$$

where c ranges over type and object constants. The proof system defined in [HHP87] for deriving assertions of the following forms:

$$\begin{array}{ll} \vdash \Sigma\ \mathrm{sig} & \Sigma \text{ is a valid signature} \\ \vdash_\Sigma \Gamma\ \mathrm{context} & \Gamma \text{ is a valid context} \\ \Gamma \vdash_\Sigma K\ \mathrm{kind} & K \text{ is a valid kind} \\ \Gamma \vdash_\Sigma A{:}K & A \text{ has kind } K \\ \Gamma \vdash_\Sigma M{:}A & M \text{ has type } A \end{array}$$

may be found in Appendix 1. We shall refer to this system as **N** to emphasize that it is a system of natural deduction. We stress that **N** is a system of first-order types in the following sense: the Π-type formation rule has the form:

$$\frac{\Gamma \vdash_\Sigma A{:}\mathrm{Type} \quad \Gamma, x{:}A \vdash_\Sigma B{:}\mathrm{Type}}{\Gamma \vdash_\Sigma \Pi x{:}A\,.\,B : \mathrm{Type}};$$

both A and B must be of kind Type (see also Rule 11 of Appendix 1). There are no variables of kind Type and consequently there are no higher-order types (of kind Type). A summary of the major metatheorems pertaining to **N** and its reduction properties may be found in the Appendix. We note here only that all five relations are decidable.

3 A metacalculus for N.

Definition 3.1 (Sequent) A *sequent* is a triple $\langle\Sigma, \Gamma, A\rangle$, written $\Gamma \Rightarrow_\Sigma A$, where Σ is a signature, Γ a context and A a type (family). The intended interpretation of the sequent is the (meta-)assertion:

$$(\exists M) \quad \mathbf{N} \text{ proves } \Gamma \vdash_\Sigma M{:}A.$$

□

We define a semi-logicistic calculus, **L**, for deriving sequents. The system is comprised of two axiom schemata and two operational rules (one left and one right) for the Π-types.

Definition 3.2 (**L**)

$$\begin{array}{lll} Ax1 & \Gamma, x{:}A, \Gamma' \Rightarrow_\Sigma A & \\ & & \\ Ax2 & \Gamma \Rightarrow_{\Sigma, c:A, \Sigma'} A & \\ & & \\ \Pi r & \dfrac{\Gamma, x{:}A \Rightarrow_\Sigma B}{\Gamma \Rightarrow_\Sigma \Pi x{:}A.B} & \text{(a) } x \notin \mathrm{Dom}(\Gamma) \\ & & \\ \Pi l & \dfrac{\Gamma, z{:}B[M/x] \Rightarrow_\Sigma C}{\Gamma \Rightarrow_\Sigma C} & \begin{array}{l} \text{(a) } @{:}\,\Pi x{:}A.B \in \Sigma \cup \Gamma \\ \text{(b) } z \notin \mathrm{Dom}(\Gamma) \\ \text{(c) } \mathbf{N} \text{ proves } \Gamma \vdash_\Sigma M{:}A \end{array} \end{array}$$

Here $B[M/x]$ denotes capture avoiding substitution of M for x, and the conditions $x, z \notin \mathrm{Dom}(\Gamma)$ mean that x and z do not label any declaration in the context Γ. For simplicity and efficiency, we work exclusively with $\beta\eta$-normal forms, and for such terms syntactic identity ($\equiv$) is taken up to α-congruence (change of bound variable). As usual we refer to the variable x of the Πr rule as the *eigenvariable* of the inference. We can ensure that in any derivation eigenvariables occur only in sequents above the inference at which they are introduced. $\Pi x{:}A.B$ is said to be the *principal formula* of the operational rules. A and B are the *side formulae* of the Πr rule, and A and $B[M/x]$ are the side formulae of the Πl rule. **L**-*derivations* are trees of sequents regulated by the operational rules, and **L**-*proofs* are derivations whose leaves are axioms. □

When x is not a free variable of B in $\Pi x{:}A.B$ we have $B[M/x] \equiv B$ and we write $A \to B$ for $\Pi x{:}A.B$: the third side condition on the Πl rule may be weakened to the inhabitation condition:

(c$'$) $(\exists M)$ **N** proves $\Gamma \vdash_\Sigma M{:}A$.

This in turn may be expressed within the system by the sequent $\Gamma \Rightarrow_\Sigma A$ yielding a modified Πl rule which we call $\to l$:

$$\to l \;\; \frac{\Gamma \Rightarrow_\Sigma A \qquad \Gamma, z{:}B \Rightarrow_\Sigma C}{\Gamma \Rightarrow_\Sigma C} \qquad \begin{array}{l}\text{(a) } @{:}A \to B \in \Sigma \cup \Gamma \\ \text{(b) } z \notin \mathrm{Dom}(\Gamma).\end{array}$$

Consequently, we extend the syntax conservatively to include *non-dependent* function types, $A \to B$, corresponding to $\Pi x{:}A.B$ whenever x is not free in B, and include the $\to l$ rule above and the $\to r$ rule below as derived rules.

$$\to r \;\; \frac{\Gamma, x{:}A \Rightarrow_\Sigma B}{\Gamma \Rightarrow_\Sigma A \to B} \qquad \text{(a) } x \notin \mathrm{Dom}(\Gamma).$$

DEFINITION 3.3 (Well-formed sequent) A sequent $\Gamma \Rightarrow_\Sigma A$ is said to be *well-formed* just in case $\Gamma \vdash_\Sigma A$: Type. □

PROPOSITION 3.4 (HHP87) *The well-formedness problem for sequents is decidable.* □

In practice, derivations are constructed from the root, or *endsequent*, toward the leaves, in the spirit of Kleene [Kle68] and systems of tableaux [Smu68]. In support of this usage we have the following result:

PROPOSITION 3.5 (PYM90) *For well-formed sequents $\Gamma \Rightarrow_\Sigma A$,*

$$\mathbf{L} \text{ proves } \Gamma \Rightarrow_\Sigma A \quad \text{iff} \quad (\exists M) \;\; \mathbf{N} \text{ proves } \Gamma \vdash_\Sigma M{:}A.$$

□

We revert to the appropriate fragment of **N** to decide if the endsequent is well-formed. If so, **L** may be utilized to prove inhabitation of A with respect to the context Γ. Moreover an inhabiting term can be extracted from the **L**-proof. Details may be found in [Pym90]. **L** is not fully logicistic since an appeal is still made to **N** for each application of the Πl rule (third side condition).

With the introduction of **L** we have made two conceptual steps. Firstly, we have shifted our attention from a decidable judgement (type assignment) to a semi-decidable judgement (inhabitation) since the former is uninteresting from the point of view of general theorem proving. (Terms code the proofs of their types, hence the decidability of the judgements of **N**.)

The second step concerns proof-search. We moved directly to a sequent system with a limited subformula property. An alternative choice would have been to formulate a natural deduction system for inhabitation assertions which would have given us a Π-elimination rule of the form:

$$\Pi\mathcal{E} \;\; \frac{\Gamma \Rightarrow_\Sigma \Pi x{:}A\,.\,B}{\Gamma \Rightarrow_\Sigma B[M/x]} \qquad \text{(a) } \mathbf{N} \text{ proves } \Gamma \vdash_\Sigma M{:}A$$

similar to the usual natural deduction rule for quantifiers. The fact that the type A in the premiss is not a subformula of the conclusion means that a proof procedure based on such a calculus would have to invent the type. The limited subformula property of $\mathbf{L}$ restricts the non-determinism to the choice of term M in the Πl rule.

4 A metacalculus for L

We introduce a new syntactic class of *universal* variables denoted by lowercase Greek letters α, β, *etc.*, and extend the syntactic category of objects to include them thus:

$$Objects \quad M \quad ::= \quad c \mid \alpha \mid x \mid \lambda x{:}A.M \mid MM.$$

Notice that universal variables cannot appear λ-bound. By virtue of this extension, entities of all syntactic classes may now contain universal variables as subterms. When we wish to emphasize that a syntactic entity does not contain universal variables we shall refer to it as being *ground.*

We define a calculus for sequents by dropping the axiom schemata of $\mathbf{L}$ and modifying the Πl rule as follows:

DEFINITION 4.1 (U-derivation) The rules of $\mathbf{U}$ consist of the $\rightarrow r$, $\rightarrow l$ and Πr rules of $\mathbf{L}$, together with

$$\Pi l \quad \frac{\Gamma, z{:}B[\alpha/x] \Rightarrow_\Sigma C}{\Gamma \Rightarrow_\Sigma C} \qquad \begin{array}{l}\text{(a) } @ : \Pi x{:}A.B \in \Sigma \cup \Gamma \\ \text{(b) } z \notin \mathrm{Dom}(\Gamma).\end{array}$$

$\mathbf{U}$-*derivations* are trees regulated by the above rules such that the sequent at the root of the tree is well-formed. In applications of the Πl rule we call Γ the *typing context* of α and A the *type* of α. (Note that we have not yet defined $\mathbf{U}$-*proofs*; there are no axiom schemata.) □

$\mathbf{U}$ thus consists of four operational rules. Notice that the Πl rule no longer contains an external appeal to $\mathbf{N}$ or a choice of term, but is otherwise identical to the Πl rule of $\mathbf{L}$.

DEFINITION 4.2 (Instantiation) An *instantiation* is a mapping from universal variables to objects. The capture-avoiding application of instantiations to all of the constructs of the language is defined in the obvious way. □

The following notion compensates for the absence of axiom schemata in $\mathbf{U}$.

DEFINITION 4.3 (Closure) A sequent $\Gamma \Rightarrow_\Sigma A$ is said to be *closed under* an instantiation σ just in case $B\sigma \equiv A\sigma$ for some declaration $@{:}B \in \Sigma \cup \Gamma$. A $\mathbf{U}$-derivation is said to be *closed under* σ just in case all of its leaf sequents are closed under σ. (Again, we work exclusively with $\beta\eta$-normal forms.) □

We are interested in instantiations that are *well-typed* in the following sense.

DEFINITION 4.4 (Well-typing) An instantiation σ is said to be *well-typed* in a given $\mathbf{U}$-derivation just in case for every universal variable α of the derivation, with typing context Γ and type A, we have:

$$\mathbf{N} \text{ proves } \Gamma\sigma \vdash_\Sigma \alpha\sigma{:}A\sigma \qquad \Box$$

We are now in a position to define a notion of proof for $\mathbf{U}$.

DEFINITION 4.5 (U-proof) A $\mathbf{U}$-*proof* is a pair $\langle\psi, \sigma\rangle$, where ψ is a $\mathbf{U}$-derivation and σ an instantiation, such that (1) ψ is closed under σ, and (2) σ is well-typed in ψ. □

THEOREM 4.6 *If $\langle\psi, \sigma\rangle$ is a $\mathbf{U}$-proof, $\psi\sigma$ is an $\mathbf{L}$-proof.*

PROOF. By induction on the structure of **U**-derivations. The closure condition (1) ensures that the leaves of $\psi\sigma$ are **L**-axioms. The well-typing condition (2) ensures that the image under σ of each instance of a Πl rule (of **U**) in ψ satisfies the side conditions on the Πl rule of **L**. The remaining operational rules are common to the two systems. $\square$

We remark that it is immediate that any **L**-proof arises as a **U**-proof: this is the converse of Theorem 4.6.

We shall postpone discussion of proof-search in **U** until we have introduced our fourth and final refinement. One could stop here, however, using the unification algorithm developed in [Pym90] (based on that of [Hu75]) to calculate instantiations when a putative leaf has been reached, then checking the well-typing condition using **N** (recall that **N** is a decidable system for well-typing). The search space induced by **U** is a proper subspace of that induced by **L** since the choice of term at Πl reductions in the former is constrained by the syntactic content of the leaf sequents (*cf.* [Rob65]). The main reason for considering a further refinement is that **U** distinguishes between derivations that are *intrinsically* identical in a sense made precise below. Consequently, the search space induced by **U** still contains a major source of redundancy.

5 A metacalculus for U

The content of the typing contexts of universal variables in a **U**-derivation depends on the structure of the derivation. For example, the two **U**-derivations below differ in the order in which the Πl and Πr rule have been applied (the figures should be read from endsequent to premisses and we assume that there is some $@ : (\Pi y{:}A.B(y)) \in \Sigma \cup \Gamma$):

$$\dfrac{\dfrac{\Gamma, z{:}B(\alpha), x{:}A \Rightarrow_\Sigma B(x)}{\Gamma, z{:}B(\alpha) \Rightarrow_\Sigma \Pi x{:}A.B(x)}}{\Gamma \Rightarrow_\Sigma \Pi x{:}A.B(x)} \qquad \dfrac{\dfrac{\Gamma, x{:}A, z{:}B(\alpha) \Rightarrow_\Sigma B(x)}{\Gamma, x{:}A \Rightarrow_\Sigma B(x)}}{\Gamma \Rightarrow_\Sigma \Pi x{:}A.B(x)}$$

The principal formula of the Πl reduction in each derivation is the declaration $@ : (\Pi y{:}A.B(y))$ — assumed to be in $\Sigma \cup \Gamma$. We also assume that $x \notin \mathrm{Dom}(\Gamma)$. The instantiation σ that maps α to x closes both derivations. The typing context of α in the first derivation is Γ, while in the second it is $\Gamma, x{:}A$. Since $\alpha\sigma = x$, in order to check the well-typing condition for α we must show $\Gamma \vdash_\Sigma x{:}A$ for the first derivation and $\Gamma, x{:}A \vdash_\Sigma x{:}A$ for the second. Consequently σ is well-typed in the second derivation but not in the first (since $x \notin \mathrm{Dom}(\Gamma)$).

Our final refinement is to introduce a calculus, **R**, in which the existence of the **U**-derivation on the left, together with the closing instantiation, is sufficient to infer the existence of the **U**-proof on the right. That is, we investigate conditions under which rule instances may be *permuted* whilst leaving the endsequent and leaves of the derivation essentially unchanged.

DEFINITION 5.1 (**R**-derivation) The rules of **R** are exactly the rules of **U**; consequently the derivations of **R** are exactly those of **U**. (The notion of **R**-proof however differs from that of **U**-proof; see below.) $\square$

Let ψ be an **R**-derivation of a given endsequent and let $\mathcal{F}_\psi$ denote the collection of inferences that comprise ψ. We use $\mathcal{F}_\psi(\Xi) \subseteq \mathcal{F}_\psi$ to denote the inferences of a given type Ξ, for Ξ one of the following: $\rightarrow l$, $\rightarrow r$, Πl or Πr. Let σ be an instantiation for ψ.

DEFINITION 5.2 The following binary relations are defined on $\mathcal{F}_\psi$:

(i) $R <_\psi R'$ iff a side formula of R is the principal formula of R'[7];

[7]More accurately: iff a side formula of R is a "descendent" of the principal formula of R; we distinguish the "occurrences" of a formula in a derivation [Kle68].

(ii) $R \ll_{\psi} R'$ iff R occurs below R' in ψ;

(iii) $R \sqsubset_{\sigma} R'$ iff the universal variable or eigenvariable introduced by R is a free variable of $\alpha\sigma$, where α is the universal variable introduced by R'.

□

Notice that $\ll_{\psi}$ decomposes into sixteen subrelations: $\ll_{\psi}^{\Xi,\Omega} \subseteq \mathcal{F}_{\psi}(\Xi) \times \mathcal{F}_{\psi}(\Omega)$ for Ξ, Ω amongst $\rightarrow l$, $\rightarrow r$, Πl and Πr. $\ll_{\psi}$ and its subrelations are called the *skeletal orderings* of the derivation ψ. Notice also that $<_{\psi}$ is a subrelation of $\ll_{\psi}$.

DEFINITION 5.3 (Reduction ordering) The *reduction ordering* $\lhd_{\psi,\sigma}$ induced by an **R**-derivation ψ and a instantiation σ is defined by:

$$\lhd_{\psi,\sigma} \quad =_{\text{def}} \quad (<_{\psi} \cup \prec_{\psi} \cup \sqsubset_{\sigma})^{+}$$

where + indicates transitive closure and the relation $\prec_{\psi}$ is defined by:

$$\prec_{\psi} \quad =_{\text{def}} \quad \ll_{\psi} \setminus \bigcup_{\Xi} (\ll_{\psi}^{\Pi l,\Xi} \cup \ll_{\psi}^{\Xi,\Pi l});$$

Ξ ranges over the operational rules of **R**. □

The presence of a relation in $\lhd_{\psi,\sigma}$ indicates that relationship between specific inferences may not be altered by permutation. Consequently, the definition of $\prec_{\psi}$ fixes the relative positions of all rule applications in ψ *except* Πl rule applications (since they are removed from $\ll_{\psi}$ to form $\prec_{\psi}$).

DEFINITION 5.4 (i) (Compatibility) A derivation is said to be *compatible* with an instantiation just in case the reduction ordering induced is irreflexive.

(ii) (Degree) The *degree* of a compatible derivation is the number of pairs of inferences in the derivation whose skeletal order is inconsistent with the reduction ordering. That is, $\langle R, R' \rangle$ for which both $R \ll_{\psi} R'$ (R is below R' in ψ) and $R' \lhd_{\psi,\sigma} R$. If a derivation is compatible with an instantiation with degree n, we say it is *n-compatible.* □

THEOREM 5.5 (PERMUTATION THEOREM) *If ψ is compatible with σ, then there is a 0-compatible* **R***-derivation ψ^* of the same endsequent. Moreover, if ψ is closed under σ, so is ψ^*. We say that ψ^* is a* permutation *of ψ.*

PROOF. By induction on the degree of ψ. We interchange Πl inferences with other inferences to reduce the degree. One such case was given as an example at the start of this section (Πl over Πr). The others are left to the reader. One must check that the leaves of the permuted derivation contain the same declarations (though in a different order). □

We have as a corollary to the construction performed in the proof of the Permutation Theorem:

COROLLARY 5.6 $\lhd_{\psi^*,\sigma} = \lhd_{\psi,\sigma}$.

PROOF. The only relationships changed in the permutation are those excluded from $\lhd_{\psi,\sigma}$. □

The following lemma shows that the 0-compatibility of a derivation and instantiation is a necessary condition for the well-typing of the latter in the former.

LEMMA 5.7 *If σ is well-typed in ψ, ψ is 0-compatible with σ.*

PROOF. An inconsistency between the skeletal ordering of ψ and the reduction ordering arises from an inconsistency between $\sqsubset_\sigma$ and the skeletal ordering. Since $\sqsubset_\sigma \subseteq \mathcal{F}_\psi(\Xi) \times \mathcal{F}_\psi(\Pi l)$, it must arise from a Πl inference, introducing α say, being nearer the root of the derivation than a Πl or Πr inference that gives rise to a variable, v say, free in $\alpha\sigma$. But then v cannot be declared in the typing context of α since it is introduced above α, and therefore σ is not well-typed, contradicting our hypothesis. Therefore there can be no inconsistencies and ψ is 0-compatible with σ. $\square$

We give a simple example of an **R**-derivation and a closure instantiation which fails to be well-typed in the given derivation, but which is well-typed in a reordering of that derivation.

Let $\Sigma \equiv_{\mathrm{def}} A : \mathrm{Type}, B : A \to \mathrm{Type}, C : \mathrm{Type}, p : A \to \mathrm{Type}, a : A, f : B(a) \to B(a)$, and let $\Gamma \equiv_{\mathrm{def}} x_1 : \Pi x_2{:}A\,.\Pi x_3{:}B(x_2)\,.\,px_3$.

We search for a proof of the (well-formed) endsequent $\Gamma \Rightarrow_\Sigma \Pi x_4 : B(a)\,.\,C \to p(fx_4)$. Consider the following **R**-derivation, ψ:

$$\begin{array}{cl}
\Gamma, x_6 : \Pi x_3{:}B(\alpha)\,.\,px_3, x_7 : p\beta, x_4{:}B(a), x_5{:}C \Rightarrow_\Sigma p(fx_4) & \\ \hline
\Gamma, x_6 : \Pi x_3{:}B(\alpha)\,.\,px_3, x_7 : p\beta, x_4{:}B(a) \Rightarrow_\Sigma C \to p(fx_4) & (\to r) \\ \hline
\Gamma, x_6 : \Pi x_3{:}B(\alpha)\,.\,px_3, x_7 : p\beta \Rightarrow_\Sigma \Pi x_4{:}B(a)\,.\,C \to p(fx_4) & (\Pi\, r) \\ \hline
\Gamma, x_6 : \Pi x_3{:}B(\alpha)\,.\,px_3 \Rightarrow_\Sigma \Pi x_4{:}B(a)\,.\,C \to p(fx_4) & (\Pi\, l) \\ \hline
\Gamma \Rightarrow_\Sigma \Pi x_4{:}B(a)\,.\,C \to p(fx_4) & (\Pi\, l).
\end{array}$$

The leaf sequent is closed by the instantiation $\sigma \equiv \langle\langle a, \alpha\rangle, \langle fx_4, \beta\rangle\rangle$, but fx_4 is not well-typed in $\Gamma, x_6 : \Pi x_3{:}B(a)\,.\,Px_3$. However, if we reorder the rules in the derivation so that the $\Pi\, l$s are used after the $\to r$ and $\Pi\, r$ we obtain the derivation:

$$\begin{array}{cl}
\Gamma, x_4{:}B(a), x_5{:}C, x_6 : \Pi x_3{:}B(\alpha)\,.\,px_3, x_7 : p\beta \Rightarrow_\Sigma p(fx_4) & \\ \hline
\Gamma, x_4{:}B(a), x_5{:}C, x_6 : \Pi x_3{:}B(\alpha)\,.\,px_3 \Rightarrow_\Sigma p(fx_4) & (\Pi\, l) \\ \hline
\Gamma, x_4{:}B(a), x_5{:}C \Rightarrow_\Sigma p(fx_4) & (\Pi\, l) \\ \hline
\Gamma, x_4{:}B(a) \Rightarrow_\Sigma C \to p(fx_4) & (\to r) \\ \hline
\Gamma \Rightarrow_\Sigma \Pi x_4{:}B(a)\,.\,C \to p(fx_4) & (\Pi\, r).
\end{array}$$

This sequent also is closed by the instantiation σ and fx_4 is well-typed in the context $\Gamma, x_4 : B(a), x_5{:}C, x_6 : \Pi x_3{:}B(a)\,.\,px_3$.

From a computational point of view testing for compatibility is a simple matter given a derivation and an instantiation: it is an acyclicity check in a directed graph. Compatibility is not, however, a *sufficient* test for well-typing. The Permutation Theorem gives us the existence of 0-compatible derivations in which we might test for well-typing of the instantiation incrementally (*i.e.*, as it is found) but this involves repeatedly constructing permutations using the constructive proof of the theorem. This is inelegant and computationally expensive.

Another alternative would be to ignore well-typing until a closed, compatible derivation and instantiation have been found, and then utilize the Permutation Theorem once and check well-typing. We reject this option on the grounds that typing constraints reduce the search space of the unification algorithm drastically.

We develop instead a computationally tractable test on a derivation and instantiation that, if passed, guarantees the well-typing of the instantiation in all 0-compatible permutations of the derivation. Our ability to define such a notion is a corollary of the normalization (cut-elimination) result for $\lambda\Pi$-calculus [HHP87] with its attendant subformula property, just as the results obtained in [Bib81] and [Wal89] for other logics rely on metatheorems of this sort.

Henceforth we treat contexts as ordered structures or DAGs rather than sequences since the dependencies between declarations form such an order. Consequently the implicit union, denoted above by a comma, such as in "$\Gamma, x{:}A$", should be understood as an order preserving union of the order (DAG) Γ and the singleton order $x{:}A$. The latter will be higher in the resulting order than the declarations of the free variables in A, and incomparable with the other maximal elements of Γ. This assumption simplifies our discussion.

The following notions are introduced for an $\mathbf{R}$-derivation ψ of a well-formed endsequent. We use u, v, w possibly subscripted to denote universal and eigenvariables of ψ. Let $T(v)$ denote the typing expression for the variable v in ψ.

Definition 5.8 (Intrinsic typing context) The *intrinsic typing context* $I(v)$ for each (eigen- or universal) variable v of ψ is defined inductively on the structure of the endsequent as follows:

$$I(v) \quad =_{\text{def}} \quad \biguplus_{w \in \text{FV}(T(v))} \quad (I(w), w{:}T(w)).$$

$\biguplus$ denotes order-preserving union of orders; $\text{FV}(M)$ denotes the set of free variables of the term M. $\square$

$I(v)$ is well-defined since the endsequent is well-formed. Indeed, we have:

Lemma 5.9 $I(v) \vdash_\Sigma T(v)$: Type.

Proof. By construction and the well-formedness of the endsequent (see Appendix 1 for the notion of a well-formed context). $\square$

Let ψ be compatible with the instantiation σ. We give an inductive definition of the intrinsic typing context and type of a variable of ψ under a compatible instantiation σ. The induction is on the (well-founded) reduction ordering $(\lhd_{\psi,\sigma})$ over the domain of σ.

Definition 5.10 ($I_\sigma(v)$ and $T_\sigma(v)$) Base. For all $v \in \text{FV}(\psi)$, define $I_\epsilon(v) = I(v)$ and $T_\epsilon(v) = T(v)$. (ϵ is the empty instantiation.)

Step. Given $v \in \text{Dom}(\sigma)$, we assume that we have defined $I_\sigma(w)$ and $T_\sigma(w)$ for all $w \in \text{Dom}(\sigma)$ such that $w \lhd_{\psi,\sigma} v$ (Inductive Hypothesis). Let $w_1, w_2, \ldots, w_n$ be an enumeration of those variables declared in $I_\epsilon(v)$. By definition of $\lhd_{\psi,\sigma}$ and $I(v)$, we have $w_i \lhd_{\psi,\sigma} v$ for $0 < i < n+1$. Define

$$\begin{array}{rcl} D_0(v) & = & I_\epsilon(v) \vdash_\Sigma T_\epsilon(v){:}\text{Type} \\ D_{k+1}(v) & = & \text{CUT}(\quad I_\sigma(w_k) \vdash_\Sigma w_k\sigma{:}T_\sigma(w_k) \quad , \quad D_k(v) \quad), \qquad 0 \le k < n. \end{array}$$

If $D_n(v)$ is the assertion: $\Delta \vdash_\Sigma C{:}\text{Type}$, then define

$$\begin{array}{rcl} I_\sigma(v) & =_{\text{def}} & \Delta \\ T_\sigma(v) & =_{\text{def}} & C. \end{array} \qquad \square$$

The "CUT" operation in the above definition is the admissible rule of transitivity (see Appendix 1). That is, $D_{k+1}(v)$ is defined in terms of $D_k(v)$ by the following inference figure:

$$\frac{I_\sigma(w_k) \vdash_\Sigma w_k\sigma{:}T_\sigma(w_k) \qquad D_k(v)}{D_{k+1}(v)} \;\text{CUT}.$$

The cut rule is being used to effect substitution of the values (under σ) of universal variables throughout the judgement starting from the "uninstantiated" intrinsic typing context and type.

The definition is well-formed since the context of the left premiss of each cut is a subcontext of the right premiss. (This follows from the construction of $I(v)$.) The cut above then serves to eliminate the declaration $w_k{:}T_\sigma(w_k)$ from the context of $D_k(v)$, replacing w_k by $w_k\sigma$ throughout the rest of the assertion.

The enumeration taken is irrelevant since independent cuts commute. Consider

$$\frac{I_\sigma(u_2) \vdash_\Sigma u_2\sigma{:}T_\sigma(u_2) \qquad \dfrac{I_\sigma(u_1) \vdash_\Sigma u_1\sigma{:}T_\sigma(u_1) \qquad D_k(v)}{D_{k+1}(v)}}{D_{k+2}(v)}$$

and

$$\frac{I_\sigma(u_1) \vdash_\Sigma u_1\sigma{:}T_\sigma(u_1) \qquad \dfrac{I_\sigma(u_2) \vdash_\Sigma u_2\sigma{:}T_\sigma(u_2) \qquad D'_k(v)}{D'_{k+1}(v)}}{D'_{k+2}(v)}.$$

In the first derivation $w_k = u_1$ and $w_{k+1} = u_2$. In the second, $w_k = u_2$ and $w_{k+1} = u_1$. If u_1 and u_2 are assumed independent (*i.e.*, unrelated via $\lhd_{\psi,\sigma}$), we have $u_i \notin \mathrm{Dom}(I_\sigma(u_j))$, $i \neq j$. Hence substitution of the value $u_1\sigma$ for u_1 does not interfere with substitution of the value $u_2\sigma$ for u_2, and $D_{k+2} = D'_{k+2}$.

We can now state the desired well-typing condition for σ in ψ.

DEFINITION 5.11 (Intrinsic well-typing) σ is said to be ***intrinsically well-typed*** in ψ just in case for all universal variables α of ψ, we have: $\quad I_\sigma(\alpha) \vdash_\Sigma \alpha\sigma{:}T_\sigma(\alpha)$. □

The importance of the definition is summarized by:

PROPOSITION 5.12 *If σ is intrinsically well-typed in ψ, then it is intrinsically well-typed in all compatible permutations of ψ. In particular it is well-typed in 0-compatible permutations.*

PROOF. Reference to the definition will show that the intrinsic well-typing of σ in ψ does not depend on the Πl structure of ψ. (In fact we deliberately forbade such dependence by our definition of $\lhd_{\psi,\sigma}$.) Hence the conditions are unaffected by permutations allowed by the reduction ordering $\lhd_{\psi,\sigma}$, which is itself unaltered by permutation (Corollary 5.6). For a 0-compatible permutation ψ^* of ψ, the intrinsic typing context for a variable is a subcontext of the typing context in $\psi^*\sigma$. Since "Thinning" is admissible (Appendix 1), σ is well-typed in ψ^*. □

In a similar vein, we state without proof the following:

PROPOSITION 5.13 *If σ is well-typed in a 0-compatible derivation ψ, it is intrinsically well-typed in ψ.* □

We can now define a computationally acceptable notion of **R**-proof.

DEFINITION 5.14 (**R**-proof) An **R**-*proof* is a pair $\langle\psi, \sigma\rangle$ such that (1) ψ is closed under σ; (2) ψ is compatible with σ, and (3) σ is intrinsically well-typed in ψ. □

THEOREM 5.15 *For well-formed sequents $\Gamma \Rightarrow_\Sigma A$,*

$$\mathbf{R} \text{ proves } \Gamma \Rightarrow_\Sigma A \qquad \text{iff} \qquad \mathbf{U} \text{ proves } \Gamma \Rightarrow_\Sigma A.$$

PROOF. (Only if.) Suppose $\langle\psi, \sigma\rangle$ is an **R**-proof of $\Gamma \Rightarrow_\Sigma A$. The Permutability Theorem gives us a permutation ψ^* of ψ, closed under (hypothesis 1), and compatible with (hypothesis 2), the instantiation σ. Hypothesis (3), via Proposition 5.12, ensures that σ is well-typed in ψ^*. Hence $\langle\psi^*, \sigma\rangle$ is an **U**-proof.

(If.) Let $\langle\psi, \sigma\rangle$ be an **U**-proof of $\Gamma \Rightarrow_\Sigma A$. By definition ψ is closed under σ and σ is well-typed in ψ. Compatibility follows from Lemma 5.7, and intrinsic well-typing from Proposition 5.13. □

6 Some remarks

Instantiations are generated by a unification algorithm acting on putative axiom sequents. They are first checked for compatibility (occurs-check) and then for intrinsic well-typing. The incremental nature of intrinsic typing means that the unification algorithm can use the typing information to constrain its search. New values for previously uninstantiated variables may be used to eliminate those variables from the typing contexts of the remaining ones. No permutations need be calculated.

We have been somewhat cautious in this development and allowed only Πl rules to migrate. As a consequence the basic structure of a derivation is largely fixed. The next step is to remove the ordering constraints induced by the propositional structure of the logic, perhaps using unification here as was done in [Wal89] for first-order intuitionistic logic. The final result would be a *matrix method* in the style of Bibel [Bib81] or Andrews [And81].

Closure instantiations are calculated by unification. In general, the substitutions calculated by the unification algorithm introduce new variables (*i.e.*, variables that are not present in the original context)[8]. However, we require only those substitutions which are well-typed instantiations (under some reordering), and so for a a given **R**-derivation ψ we accept (for further analysis) just those substitutions σ which do not introduce new variables. The unification algorithm of [Pym90] is both *sound* and *complete* for the calculation of such instantiations.

7 Non-ground endsequents and logic programming

We have considered ground endsequents and ground instantiations explicitly. The extension of **R** to non-ground endsequents is straightforward. A non-ground sequent together with a *typing constraint* $\mathcal{T}$ for its universal variables is considered to stand for the set of its well-formed ground instances. (The typing constraint consists of intrinsic typing contexts and types for each universal variable occuring in the endsequent, ensuring that the mutual dependencies do not render any extension of the initial reduction ordering cyclic.) This determines a set $\mathcal{S}(\mathcal{T})$ of ground *answer instantiations.* Any non-ground instantiation calculated from an **R**-proof $\langle\psi, \sigma\rangle$ of the sequent determines a set of ground well-typed *extensions* $\mathcal{S}_{\psi}(\mathcal{T})$.

We consider non-ground endsequents because they have an interesting logic programming interpretation. A sequent $\Gamma \Rightarrow_{\Sigma} A$ may be interpreted as a *logic program* in the following sense: Σ determines a language, Γ a list of *program clauses* and A a *query* written in the language Σ. universal variables correspond to *program variables* or *logic variables*; these correspond to the logical variables of the programming language PROLOG [CM84]. The whole sequent represents a request to compute a instantiation σ for the universal variables of the sequent such that any ground extension σ' of σ renders the sequent $\Gamma\sigma' \Rightarrow_{\Sigma} A\sigma'$ **L**-provable. We are exploiting the fact that the underlying lambda calculus of the $\lambda\Pi$-calculus encodes a computation of type A; *i.e.,* a term M for which $\Gamma \vdash_{\Sigma} M{:}A$. A full discussion of this notion of logic programming, including both operational (as presented here) and model-theoretic semantics, may be found in [Pym90] and in a forthcoming paper by the authors, where we also discuss the application of our techniques to a form of *resolution rule* which generalizes Paulson's higher-order resolution [Pau86] to the $\lambda\Pi$-calculus.

Acknowledgements. The authors are grateful to Anne Salvesen, James Harland, Robert Harper, Furio Honsell, Gordon Plotkin, Randy Pollack and two anonymous referees for helpful suggestions and comments.

[8]Indeed, this is true of the basic algorithm for the simply-typed λ-calculus of [Hu75]

References

[And81] Andrews, P.B. Theorem-proving via general matings. *J. Assoc. Comp. Mach.* 28(2):193–214, 1981.

[AHM87] Avron, A., Honsell, F., Mason, I. Using Typed Lambda Calculus to Implement Formal Systems on a Machine. University of Edinburgh, 1987, ECS-LFCS-87-31.

[Bib81] Bibel, W. Computationally Improved Versions of Herbrand's Theorem. In J. Stern, editor, Proc. of the Herbrand Symposium, *Logic Colloquium* '81, pp. 11-28, North-Holland, 1982.

[CM84] Clocksin, W.F., Mellish, C.S. Programming in Prolog, Springer-Verlag, 1984.

[Cur52] Curry, H.B. The permutability of rules in the classical inferential calculus. *J. Symbolic Logic* **17**, pp. 245–248, 1952.

[vDa80] van Daalen, D.T., The language theory of AUTOMATH. PhD thesis, Technical University of Eindhoven, The Netherlands, 1980.

[Gen34] Gentzen, G. Untersuchungen über das logische Schliessen, *Mathematische Zeitschrift* 39 (1934) 176–210, 405-431.

[HHP87] Harper, R., Honsell, F., Plotkin, G. A Framework for Defining Logics. Proc. LICS '87.

[HHP89] Harper, R., Honsell, F., Plotkin, G. A Framework for Defining Logics. Submitted to the J. Assoc. Comp. Mach., 1989.

[Hu75] Huet, G. A Unification Algorithm for Typed λ-calculus. *Theor. Comp. Sci.*, 1975.

[Kle52] Kleene, S.C. Permutability of inferences in Gentzen's calculi *LK* and *LJ*. *Memoirs of the American Mathematical Society* **10**, pp. 1–26, 1952.

[Kle68] Kleene, S.C. Mathematical logic. Wiley and Sons, 1968.

[MR86] Meyer, A. and Reinhold, M. 'Type' is not a type: preliminary report. in Proc. 13th ACM Symp. on the Principles of Programming Languages, 1986.

[Mil83] Miller, D. Proofs in higher-order logic. PhD thesis, Carnegie-Mellon University, Pittsburgh, USA, 1983.

[Ohl88] Ohlbach, H-J. A resolution calculus for modal logics. Proc. 9th Conf. on Automated Deduction, LNCS 310, 1988.

[Pau86] Paulson, L. Natural Deduction Proof as Higher-order Resolution. *J. Logic Programming* **3**, pp. 237–258, 1986.

[Pra65] Prawitz, D. Natural Deduction: A Proof-theoretical Study. Almqvist & Wiksell, Stockholm, 1965.

[Pym90] Pym, D.J. Proofs, Search and Computation in General Logic. PhD thesis. University of Edinburgh, forthcoming.

[Rob65] Robinson, J. A machine-oriented logic based on the resolution principle. *J. Assoc. Comp. Mach.* **12**, pp. 23–41, 1965.

[Sa89] Salvesen, A. In preparation. University of Edinburgh, 1989.

[Smu68] Smullyan, R.M. First-order logic, *Ergebnisse der Mathematik*, Volume **43**, Springer Verlag, 1968.

[Wal89] Wallen, L.A. Automated deduction in non-classical logics, MIT Press, 1989.

Appendix 1

The $\lambda\Pi$-calculus is closely related to the Π-fragment of AUT-PI, a language belonging to the so-called AUTOMATH family. The $\lambda\Pi$-calculus is a language with entities of three levels: *objects*, *types and families of types*, and *kinds*. Objects are classified by types, types and families of types by kinds. The kind Type classifies the types; the other kinds classify functions f which yield a type $f(x_1)\dots(x_n)$ when applied to objects $x_1, \dots, x_n$ of certain types determined by the kind of f. Any function definable in the system has a type as domain, while its range can either be a type, if it is an object, or a kind, if it is a family of types. The $\lambda\Pi$-calculus is therefore predicative.

The theory we shall deal with is a formal system for deriving assertions of one of the following shapes:

$\vdash \Sigma$ sig	Σ is a signature
$\vdash_\Sigma \Gamma$ context	Γ is a context
$\Gamma \vdash_\Sigma K$ kind	K is a kind
$\Gamma \vdash_\Sigma A{:}K$	A has kind K
$\Gamma \vdash_\Sigma M{:}A$	M has type A

where the syntax is specified by the following grammar:

$$\begin{array}{lcl}
\Sigma & ::= & \langle\rangle \mid \Sigma, c : K \mid \Sigma, c : A \\
\Gamma & ::= & \langle\rangle \mid \Gamma, x : A \\
K & ::= & \mathrm{Type} \mid \Pi x : A.K \\
A & ::= & c \mid \Pi x : A.B \mid \lambda x : A.B \mid AM \\
M & ::= & c \mid x \mid \lambda x : A.M \mid MN
\end{array}$$

We let M and N range over expressions for objects, A and B for types and families of types, K for kinds, x and y over variables, and c over constants. We write $A \to B$ for $\Pi x : A.B$ when x does not occur free in B. We refer to the collection of variables declared in a context Γ as $\mathrm{Dom}(\Gamma)$. We assume α-conversion throughout. The inference rules of the $\lambda\Pi$-calculus appear in Table 1.

A term is said to be *well-typed in a signature and context* if it can be shown to either be a kind, have a kind, or have a type in that signature and context. A term is *well-typed* if it is well-typed in some signature and context. The notion of $\beta\eta$-reduction, written $\to_{\beta\eta}$, can be defined both at the level of objects and at the level of types and families of types in the obvious way, for details [HHP89]. $M =_{\beta\eta} N$ iff $M \to^*_{\beta\eta} P$ and $N \to^*_{\beta\eta} P$ for some term P, where * denotes transitive closure. For simplicity we shall write $\to_{\beta\eta}$ for $\to^*_{\beta\eta}$.

Since $\beta\eta$-conversion over $K \cup A \cup M$ is not Church-Rosser, so the order of technical priority in which the basic metatheoretical results are proved is crucial. The theorem below summarizes these results in a convenient order (here α ranges over the basic assertions of the type theory). The reader is referred to [HHP87], [HHP89] and [Sa89] for details of its proof.

Theorem (The Basic Metatheory of the $\lambda\Pi$-calculus)

1. *Thinning is an admissible rule: if* $\Gamma \vdash_\Sigma \alpha$ *and* $\vdash_{\Sigma,\Sigma'} \Gamma, \Gamma'$ context, *then* $\Gamma, \Gamma' \vdash_{\Sigma,\Sigma'} \alpha$.

2. *Transitivity is an admissible rule: if* $\Gamma \vdash_\Sigma M{:}A$ *and* $\Gamma, x : A, \Delta \vdash_\Sigma \alpha$, *then* $\Gamma, \Delta[M/x] \vdash_\Sigma \alpha[M/x]$.

3. *Uniqueness of types and kinds: if* $\Gamma \vdash_\Sigma M{:}A$ *and* $\Gamma \vdash_\Sigma M{:}A'$, *then* $A =_{\beta\eta} A'$, *and similarly for kinds.*

4. *Subject reduction: if* $\Gamma \vdash_\Sigma M{:}A$ *and* $M \to^*_{\beta\eta} M'$, *then* $\Gamma \vdash_\Sigma M'{:}A$, *and similarly for types.*

5. *All well-typed terms are strongly normalizing.*

6. *All well-typed terms are Church-Rosser.*

7. *Each of the five relations defined by the inference system of table 1 is decidable, as is the property of being well-typed.*

8. *Predicativity: if* $\Gamma \vdash_\Sigma M : A$ *then the type-free* λ*-term obtained by erasing all type information from* M *can be typed in the Curry type assignment system.*

9. *Strengthening is an admissible rule: if* $\Gamma, x : A, \Gamma' \vdash_\Sigma \alpha$ *and if* $x \notin \mathrm{FV}(\Gamma') \cup \mathrm{FV}(\alpha)$ *then* $\Gamma, \Gamma' \vdash_\Sigma \alpha$. $\square$

Valid Signature

$$\frac{}{\vdash \langle\rangle \ \text{sig}} \tag{1}$$

$$\frac{\vdash \Sigma \ \text{sig} \quad \vdash_\Sigma K \ \text{kind} \quad c \notin \mathrm{Dom}(\Sigma)}{\vdash \Sigma, c : K \ \text{sig}} \tag{2}$$

$$\frac{\vdash \Sigma \ \text{sig} \quad \vdash_\Sigma A\colon \text{Type} \quad c \notin \mathrm{Dom}(\Sigma)}{\vdash \Sigma, c : A \ \text{sig}} \tag{3}$$

Valid Context

$$\frac{\vdash \Sigma \ \text{sig}}{\vdash_\Sigma \langle\rangle \ \text{context}} \tag{4}$$

$$\frac{\vdash_\Sigma \Gamma \ \text{context} \quad \Gamma \vdash_\Sigma A\colon \text{Type} \quad x \notin \mathrm{Dom}(\Gamma)}{\vdash_\Sigma \Gamma, x : A \ \text{context}} \tag{5}$$

Valid Kinds

$$\frac{\vdash_\Sigma \Gamma \ \text{context}}{\Gamma \vdash_\Sigma \text{Type} \ \text{kind}} \tag{6}$$

$$\frac{\Gamma \vdash_\Sigma A\colon \text{Type} \quad \Gamma, x : A \vdash_\Sigma K \ \text{kind}}{\Gamma \vdash_\Sigma \Pi x : A.K \ \text{kind}} \tag{7}$$

Valid Elements of a Kind

$$\frac{\vdash_\Sigma \Gamma \ \text{context} \quad c : K \in \Sigma}{\Gamma \vdash_\Sigma c\colon K} \tag{8}$$

$$\frac{\Gamma \vdash_\Sigma A\colon \text{Type} \quad \Gamma, x : A \vdash_\Sigma B\colon \text{Type}}{\Gamma \vdash_\Sigma \Pi x : A.B\colon \text{Type}} \tag{9}$$

$$\frac{\Gamma \vdash_\Sigma A\colon \text{Type} \quad \Gamma, x : A \vdash_\Sigma B\colon K}{\Gamma \vdash_\Sigma \lambda x : A.B\colon \Pi x : A.K} \tag{10}$$

$$\frac{\Gamma \vdash_\Sigma B\colon \Pi x : A.K \quad \Gamma \vdash_\Sigma N\colon A}{\Gamma \vdash_\Sigma BN\colon K[N/x]} \tag{11}$$

$$\frac{\Gamma \vdash_\Sigma A\colon K \quad \Gamma \vdash_\Sigma K' \ \text{kind} \quad K =_{\beta\eta} K'}{\Gamma \vdash_\Sigma A\colon K'} \tag{12}$$

Valid Elements of a Type

$$\frac{\vdash_\Sigma \Gamma \ \text{context} \quad c : A \in \Sigma}{\Gamma \vdash_\Sigma c\colon A} \tag{13}$$

$$\frac{\vdash_\Sigma \Gamma \ \text{context} \quad x : A \in \Gamma}{\Gamma \vdash_\Sigma x\colon A} \tag{14}$$

$$\frac{\Gamma \vdash_\Sigma A\colon \text{Type} \quad \Gamma, x : A \vdash_\Sigma M\colon B}{\Gamma \vdash_\Sigma \lambda x : A.M\colon \Pi x : A.B} \tag{15}$$

$$\frac{\Gamma \vdash_\Sigma M\colon \Pi x : A.B \quad \Gamma \vdash_\Sigma N\colon A}{\Gamma \vdash_\Sigma MN\colon B[N/x]} \tag{16}$$

$$\frac{\Gamma \vdash_\Sigma M\colon A \quad \Gamma \vdash_\Sigma A'\colon \text{Type} \quad A =_{\beta\eta} A'}{\Gamma \vdash_\Sigma M\colon A'} \tag{17}$$

Table 1.