Under consideration for publication in Math. Struct. in Comp. Science

Some logical and syntactical observations concerning the first order dependent type system λP

HERMAN GEUVERS[1] and ERIK BARENDSEN[2]

[1] *Computing Science Department, Eindhoven Univ. of Technology, NL*
[1],[2] *Computing Science Department, Nijmegen University, NL*

Received 31 March 1999

We look at two different ways of interpreting logic into the dependent type system λP. The first is by a *direct* formulas-as-types interpretation à la Howard where the logical derivation rules are mapped to derivation rules in the type system. The second is by viewing λP as a *Logical Framework*, following (Harper et al. 1987) and (Harper et al. 1993). Then the type system is used as the meta-language in which various logics can be coded.
We give a (brief) overview of known (syntactical) results about λP. Then we discuss two issues in some more detail. The first is the completeness of the formulas-as-types embedding of minimal first-order predicate logic into λP. This is a remarkably complicated issue, a first proof of which appeared in (Geuvers 1993), following ideas in (Barendsen and Geuvers 1989) and (Swaen 1989). The second issue is the *minimality* of λP as a logical framework. We will show that some of the rules are actually superfluous (even though they contribute nicely to the generality of presentation of λP).
At the same time we will attempt to provide a gentle inroduction to λP and its various aspects and we will try to use little inside knowledge.

1. Introduction and motivation

The typed λ-calculus λP is the extension of simple typed λ-calculus with (first order) dependent types. It occurs in different variants in the work of (Martin-Löf 1975) (intuitionistic type theory), (van Daalen 1973) (Automath), (Harper et al. 1987) (Logical Framework) and also in (Hindley and Seldin 1986) (Generalized Type Assignment). The precise definition that we will be using is the one in (Barendregt 1992), which is – of the forementioned – closest to (Harper et al. 1987). In the literature we find several ways of motivating the definition of λP and explaining its use. These can be devided in two: λP can be seen as a system for interpreting minimal first order predicate logic and λP can be seen as a logical framework. In both views one uses a kind of formulas-as-types interpretation to interpret (in the case of a logical framework one would say 'encode') the logic, the difference being that in the first case there is *one logic*, the internal logic of λP, whereas in the second case almost any formal system can be encoded. We will not give a complete overview of the different possible interpretations, but instead motivate the definition of λP by explaining the two interpretations by examples.

Using λP to represent logic, either by a direct interpretation or an encoding, raises a major question, namely: is the interpretation complete? Or, if one takes the view of λP as a logical framework: is the encoding adequate? If we take the point of view of a direct encoding of minimal first order predicate logic, L, into λP, then the question would be whether the implication

$$\Gamma_\Sigma \vdash_{\lambda\mathrm{P}} M : \varphi \;\Rightarrow\; \vdash_L \varphi$$

holds for all formulas φ of minimal first order predicate logic. (Here Γ_Σ represents the context that declares the constants of the first order signature Σ; φ is a formula over this signature Σ.) So, the question is whether, if φ, considered as a type in λP, is inhabited, then φ is derivable in L. The same question obviously applies to λP seen as a logical framework, with the difference that there is not just one logic, but that for every logic L we have to define a cotext Γ_L that codes the logic. How this works in detail will be discussed later by treating some examples.

One may wonder whether the *soundness* of the interpretation is not an issue. Well, it is an issue: we have to prove that

$$\vdash_L \varphi \;\Rightarrow\; \exists M[\Gamma_\Sigma \vdash_{\lambda\mathrm{P}} M : \varphi]$$

holds for all formulas φ of minimal first order predicate logic L. However, soundness is not a *major* issue, because it is easily satisfied. Also for the logical framework view, soundness is usually relatively easy: it boils down to choosing the 'correct' Γ_L as an encoding of the logic L.

The question of adequacy of the encoding of a logic L into λP, as a logical framework, was first dealt with by (Harper et al. 1987). (A full version of this paper has appeared as (Harper et al. 1993).) As a matter of fact, they were the first to actually state the problem. To prove adequacy of an encoding (Harper et al. 1987) devise a general technique that applies to many different logics L. The idea is to construct, out of a proof term $M : \varphi$, a canonical proof term M' (technically: the so called *long-$\beta\eta$-normal form of M*). From such a canonical proof term a proof of φ in the logic L is immediately constructed. We will illustrate this technique briefly in Section 4 by an example. (For *proving* adequacy of the encoding it is convenient to extend λP with η-conversion. This yields the actual type system of LF as defined in (Harper et al. 1987). However, for the adequacy result this is not needed – as we will also argue in 4 – because λP is a subsystem of the LF type system.) In Section 4 we will also show how we can define a minimal version of λP that can serve as a logical framework.

The question of completeness of the interpretation of minimal first order predicate logic into λP will be treated in Section 3. This issue was already raised by Martin-Löf in the seventies. A proof of completeness was first sketched in (Barendsen and Geuvers 1989). A precise proof – based on this proof – occurs in (Geuvers 1993). We give it in Section 3 with some more explanation and examples. Independently, (Berardi 1990) proved the same completeness result. It's maybe most remarkable that the completeness is such an intricate problem. To grasp this we have to understand how exactly minimal first order predicate logic is interpreted in λP. This is done by interpreting *both sets and formulas* as types. A predicate P on a set A is then interpreted as a function from (the type) A to the collection of all types, **type**, so $P : A{\to}\mathbf{type}$. From this one constructs e.g. the type $\Pi x{:}A.Px{\to}Px$, representing the formula $\forall x{:}a.Px{\to}Px$, but one can also construct types like $\Pi x{:}A.Px{\to}A$, which does not represent any set or formula of the logic. In the completeness proof one has to take care of all these 'meaningless' types and it is not at all clear whether these meaningless types can somehow spoil the completeness.

2. The system λP

We begin by defining the system λP. Then we give some examples of well-typed terms and list some of the general issues (and properties) of type systems. Finally we give a brief list of some meta-theoretic properties.

Definition 2.1. λP (Harper et al. 1987) is a system for deriving judgements of the following two forms

$$\begin{array}{rl} \Gamma \vdash M : B & M \text{ is of type } B \text{ in context } \Gamma, \\ \Gamma \vdash & \Gamma \text{ is a correct context.} \end{array}$$

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

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

The derivation rules for deriving the judgements $\Gamma \vdash M : B$ and $\Gamma \vdash$ are the following. ($\mathbf{s}$ ranges over $\{\mathbf{type}, \mathbf{kind}\}$.)

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

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

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

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

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

As usual, we omit brackets in sequences of applications by associating them to the left and we omit brackets in sequences of abstraction terms by associating them to the right. So MNP denotes $((MN)P)$, $\lambda x{:}A.\lambda y{:}B.M$ denotes $(\lambda x{:}A.(\lambda y{:}B.M))$ and $\Pi x{:}A.\Pi y{:}B.C$ denotes $(\Pi x{:}A.(\Pi y{:}B.C))$.
We use the convention of writing $A{\to}B$ for $\Pi x{:}A.B$ if $x \notin \mathrm{FV}(B)$.

In (Harper et al. 1987), a system called LF is defined ('Logical Framework'), specifically geared towards the coding of logics (and formal systems in general) in this typed λ-calculus. The system λP can be seen as the 'basic underlying type system' of LF. In the definition of λP we have ignored some features of LF, the main one being 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. Another difference is that we do not consider η-conversion. In LF, η-conversion is used for proving the adequacy of the encoding. However, we don't need η-conversion for adequacy, as will be argued in Section 4.

The system λP is an extension of the simply typed λ-calculus $\lambda{\to}$. This is not entirely obvious, as the types (and contexts) in $\lambda{\to}$ are usually defined separately from the terms, whereas in λP these definitions are interwoven. We therefore treat some examples of well-typed terms in $\lambda{\to}$.

Example 2.2.

1 $\alpha{:}\mathbf{type}, \beta{:}\mathbf{type}, \gamma{:}\mathbf{type} \vdash \lambda x{:}\alpha{\to}\beta{\to}\gamma.\lambda y{:}\alpha{\to}\beta.\lambda z{:}\alpha.xz(yz) :$
$(\alpha{\to}\beta{\to}\gamma){\to}(\alpha{\to}\beta){\to}\alpha{\to}\gamma.$

2 $\alpha{:}\mathbf{type}, \beta{:}\mathbf{type}, y{:}\beta \vdash \lambda x{:}(\alpha{\to}\beta){\to}\alpha.x(\lambda z{:}\alpha.y) : ((\alpha{\to}\beta){\to}\alpha){\to}\alpha.$

It is well-known that $\lambda{\to}$ is isomorphic with minimal propositional logic (logic with just implication) via the formulas-as-types embedding. In the example above, the first λ-term represents a proof (natural deduction derivation) of $(\alpha{\to}\beta{\to}\gamma){\to}(\alpha{\to}\beta){\to}\alpha{\to}\gamma$, whereas the second represents a proof of $((\alpha{\to}\beta){\to}\alpha){\to}\alpha$ from the assumption β.

In a similar fashion one can interpret in λP minimal first order predicate logic (logic with just implication and universal quantification). To be able to do this we have to follow one basic principle:

A formula is associated with the type of its proofs, hence
a *formula is provable* if and only if the associated *type is not empty* ('inhabited').

As a consequence, we associate with a predicate over the set (type) A a term of type $A{\to}\mathbf{type}$, the idea being that for $a : A$,

Pa holds if and only if the type *Pa is inhabited.*

This amounts to an interpretation of minimal predicate logic in λP where both sets and formulas are interpreted as types. We will define this interpretation precisely later and restrict to some motivating examples now.

Example 2.3.

1 $\alpha{:}\mathbf{type}, P{:}\alpha{\to}\mathbf{type} \vdash \lambda x{:}\alpha.\lambda p{:}Px.p : \Pi x{:}\alpha.Px{\to}Px.$

2 $\alpha{:}\mathbf{type}, f{:}\alpha{\to}\alpha, R{:}\alpha{\to}\alpha{\to}\mathbf{type},$
$h_1 : \Pi x{:}\alpha.Rx(fx), h_2 : \Pi x,y,z{:}\alpha.(Rxy){\to}(Ryz){\to}(Rxz) \vdash$
$\lambda x{:}\alpha.h_2x(fx)(f(fx))(h_1x)(h_1(fx)) : \Pi x{:}\alpha.Rx(f(fx)).$

In a predicate logical interpretation, the first term is a proof of $\forall x \in A.P(x){\to}P(x)$ and the second is a proof of $\forall x \in A.R(x, f(f(x))$ from the hypotheses $\forall x \in A.R(x, f(x))$ and $\forall x,y,z \in A.R(x,y){\to}R(y,z){\to}R(x,z)$.

As a third type of example we treat a *coding* of minimal propositional logic in λP. The idea is to declare a type **prop** in the context to represent the type of (names) of propositions and to add a $T : \mathbf{prop}{\to}\mathbf{type}$, taking a name of a proposition to the type of its proofs. (By adding suitable declarations for the derivations we establish that for $a : \mathbf{prop}$, the terms of type Ta indeed represent natural deduction proofs.)

Example 2.4. Define

$$\begin{array}{rcl}\Gamma & := & \mathbf{prop} : \mathbf{type}, T : \mathbf{prop}{\to}\mathbf{type},\\ & & \supset : \mathbf{prop}{\to}\mathbf{prop}{\to}\mathbf{prop},\\ & & \supset_I : \Pi x, y{:}\mathbf{prop}.(Tx{\to}Ty){\to}T(\supset xy),\\ & & \supset_E : \Pi x, y{:}\mathbf{prop}.T(\supset xy){\to}Tx{\to}Ty.\end{array}$$

Then we have the following typings.

1 $\Gamma, x{:}\mathbf{prop} \vdash \supset_I(\supset xx(\lambda p{:}Tx.p)) : T(\supset xx)$.
This term codes a proof of '$x{\to}x$'.

2 For the purpose of presentation we write $\supset xy$ as $x \supset y$ and we omit the first two arguments of $\supset_I$ and $\supset_E$. We then find
$\Gamma, x, y{:}\mathbf{prop}, h{:}Ty \vdash \supset_I(\lambda p{:}T((x \supset y) \supset x).\supset_E(\supset_I(\lambda q{:}Tx.h)))$:
$T(((x \supset y) \supset x) \supset x)$.
This term codes a derivation of $((x{\to}y){\to}x){\to}x$ from the hypothesis y.

2.1. *Properties of* λP

In the examples above we saw that the general use of λP is to code (or represent directly) either terms (programs) or derivations (proofs). The β-reduction relation then corresponds to evaluation (of programs) or cut-elimination (of proofs). Important and natural properties to have are then that the typing is preserved by evaluation, that evaluation is confluent and that typing is decidable.

We list the main properties of the typed λ-calculus λP. Proofs can be found in (Harper et al. 1987), (Barendregt 1992) or in (Geuvers and Nederhof 1991). (The proofs in (Harper et al. 1987) are for λP with η; the proofs here are roughly the same, sometimes a bit simpler due to the absence of η.)

Proposition 2.5. (Subject Reduction) If $\Gamma \vdash M : A$ and $M \longrightarrow_\beta N$, then $\Gamma \vdash N : A$.

The following Proposition follows from Subject Reduction and the fact that β-reduction is confluent on the pseudo-terms T.

Proposition 2.6. (Typed Confluence) If $\Gamma \vdash M : A$, $\Gamma \vdash N : A$ and $M =_\beta N$, then there is a term P with $M \twoheadrightarrow_\beta P$, $N \twoheadrightarrow_\beta P$ and $\Gamma \vdash P : A$.

Proposition 2.7. (Uniqueness of Types) If $\Gamma \vdash M : A$ and $\Gamma \vdash M : B$, then $A =_\beta B$.

Proposition 2.8. (Strong Normalization) All well-typed expressions of λP are Strongly Normalizing with respect to β-reduction.

This Proposition was first proved in (Harper et al. 1987), by defining a reduction preserving mapping to the simple typed λ-calculus.

In view of Example 2.3, strong normalization of β-reduction is very natural, as it corresponds to cut-elimination in the logic (evaluation of natural deductions), which we know to be terminating for first order predicate logic.

Proposition 2.9. (Decidability of Typing) Given a context Γ and a pseudo-term M, it is decidable whether M has a type in Γ and if M has a type it can be computed.

As a matter of fact, 'typechecking a term' (i.e. find out whether a given pseudoterm has a type in a given context) proceeds by just computing it (with a 'fail' option if the pseudoterm does not have a type). Similarly, checking whether a given pseudoterm M has a *given* type A proceeds by computing the type of M and cheking whether it is β-equal to A.

Decidability of Typing holds because the abstracted variables come with their types. So, if we want to typecheck $\lambda x{:}A.M$ in Γ, we can proceed by typechecking M in $\Gamma, x{:}A$. If we want to typecheck MN in Γ, we have to typecheck M in Γ and see whether its type reduces to something of the form $\Pi x{:}A.B$. Then we typecheck N in Γ and see whether its type is β-equal to A. In the typechecking algorithm, an algorithm for checking β-equality is used. As all terms in λP are Strongly Normalizing, this equality-checking algorithm terminates, and hence the type-checking algorithm does.

If we pursue the view of λP as a logic, one may wonder whether the *inhabitation problem* is decidable, that is, given a context Γ and a type A, is it decidable whether there exists a term of type A in Γ?

Proposition 2.10. (Undecidability of Inhabitation) (Bezem and Springintveld 1996) In general it is undecidable whether, given a context Γ and a type A in Γ, there exists an M such that $\Gamma \vdash M : A$.

The proof proceeds by translating register machine programs P into a pair of a context Γ_P and a type A_P in such a way that A_P is inhabited in Γ_P iff P terminates.

¿From a more computational perspective, viewing the terms as programs, Strong Normalization may not seem very natural to have. In the computational view, one may wonder whether types can be reconstructed automatically, i.e. if they are *not* given in the λ-abstraction. This is known as the *Type Assignment Problem*, which is known to be decidable for simple typed λ-calculus. To be more precise, we define the *erasure*, $|-|$, of a pseudoterm by $|\lambda x{:}A.M| := \lambda x.|M|$ and for the other cases by structural recursion. Then, if $\Gamma \vdash M : A(: \mathbf{type})$, the term $|M|$ is an ordinary untyped λ-term. Now, given an untyped λ-term N one may wish to find out whether N is *typable*, i.e. whether a well-typed term M exists such that $|M| \equiv N$.

Proposition 2.11. (Undecidability of Type Assignment) (Dowek 1993) It is in general undecidable whether, given a context Γ and an untyped λ term N, there is a term M such that M is well-typed in Γ and $|M| \equiv N$.

The proof proceeds by defining a special context Γ such that for every Post correspondence problem P there is a (untyped) term t_P such that t_P is typable in Γ iff the problem P has a solution.

2.2. *Minimal* λP

It turns out that for many practical uses, there is no need to be able to abstract over a type to form a constructor (i.e. form $\lambda x{:}A.M$ for $\Pi x{:}A.B : \mathbf{kind}$). We therefore define a minimal version of λP.

Definition 2.12. In λP 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 λP without the rule (λ_P) we call λP^-, and we write $\vdash^-$ for judgements in λP^-. On the terms of λP 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.

All the nice properties that we know from λP remain to hold for λP^-. In Section 4 we show that this minimal version of λP is adequate as a logical framework. It is also adequate for interpreting minimal first order predicate logic, as we will see in the next Section.

3. λP as minimal first order predicate logic

We want to make a precise study of the embedding of minimal first order predicate logic into λP , following the so called *formulas-as-types* embedding, originally due to Curry, Howard and De Bruijn. The embedded logic, minimal first order predicate logic, has as connectives only implication $\supset$ and first-order universal quantification. There is no negation and (hence) there is no double negation rules, so the logic is constructive. A peculiarity of the embedding is that both domains (of the logic) and formulas are interpreted as types, which makes the *completeness* of the formulas-as-types embedding not at all obvious. Moreover, there are two constructions possible in λP that are – although not alien to predicate logic – not in the realm of first order logic. These are: (1) The possibility to *define new predicates by abstraction.* E.g. if $R : A{\to}A{\to}\mathbf{type}$ is a binary predicate on A, then $\lambda x{:}A.Rxx$ is of type $A{\to}\mathbf{type}$, a unary predicate on A. (2) The possibility to declare and construct *higher order functions.* E.g. one can declare $f : (A{\to}A){\to}A$ in the context, a function from $A{\to}A$ (functions from A to A) to A. Also one can construct higher order functions, like $\lambda f : (A{\to}A){\to}A.f(\lambda x{:}A.x)$ which is of type $((A{\to}A){\to}A){\to}A$.

In the following we first give a precise definition of the logical systems: minimal first order predicate logic, PRED and its extension with definable predicates and higher order functions, PRED^{fr}. Then we define a typed λ-calculus, λPRED^{fr} that we show to be isomorphic to PRED^{fr} via the formulas-as-types embedding. Then we prove in two phases that the embedding of minimal first order predicate logic (PRED) into λP is indeed complete. In the first phase we show completeness of the embedding of λPRED^{fr} into λP. In the second phase we show conservativity of PRED^{fr} over PRED.

In the following diagram the steps in the proof of completeness of the formulas-as-typed embedding from PRED into λP are depicted. An arrow denotes the inclusion embedding of one system in another.

$$\begin{tikzcd}
\mathrm{PRED} \arrow[r, "3.17"] & \mathrm{PRED}^{\mathrm{fr}} \arrow[d, "\simeq"', "3.16"] & \\
& \lambda\mathrm{PRED}^{fr} \arrow[r, "3.36"] & \lambda\mathrm{P}
\end{tikzcd}$$

3.1. *Minimal first order predicate logic*

Definition 3.1. The language of the system PRED is defined as follows.

1 The *domains* are given by

$$\mathcal{D} ::= \mathcal{B} \,|\, \mathcal{P} \,|\, \mathcal{F}$$

where $\mathcal{B}$ is a set of *basic domains*, $\mathcal{P}$ is the set of *predicate domains*, defined by

$$\mathcal{P} ::= \mathsf{Prop} \,|\, \mathcal{B} {\to} \mathcal{P},$$

and $\mathcal{F}$ is the set of *functional domains*, defined 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.) The intention is that $B_1{\to}B_2{\to}B_3$ should be read as $B_1{\to}(B_2{\to}B_3)$, the set of functions taking an argument in B_1, an argument in B_2 and returning a value in B_3. Similarly, $B_1{\to}B_2{\to}\mathsf{Prop}$ is intended to represent the set of relations on $B_1 \times B_2$.

2 The terms of the language of PRED are described as follows.

— There are countably many constants c_i^D for every domain $D \in \mathcal{D}$,

— There are countably many variables of each basic domain $B \in \mathcal{B}$,

— If c_i^D is a constant of domain $D \equiv B_1{\to}\cdots{\to}B_1{\to}C$, where $C \in \mathcal{B} \cup \mathsf{Prop}$ and for $1 \leq i \leq p$, t_i is a term of domain B_i, then $c_i^D t_1 \ldots t_p$ is a term of domain C,

— If $\varphi \,\epsilon\, \mathsf{Prop}$ and x is a variable of basic domain B, then $\forall x \epsilon B.\varphi$ is a term of domain Prop.

— If φ and ψ are terms of domain Prop, then $\varphi \supset \psi$ is a term of domain Prop.

If t is a term of domain D, we shall just write $t \,\epsilon\, D$.

The derivation rules of PRED are the following. (The usual rules for $\supset$ and $\forall$, where quantification is restricted to the basic domains.)

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

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

$(*)$: in the ∀-I rule we make the usual restriction that the variable x may not occur free in a non-discharged assumption of the derivation. In the ⊃-I rule, i is an index to label the formulas that are discharged with that rule.
For Γ a set of formulas of PRED and φ a formula of PRED, we say that φ *is derivable from* Γ *in* PRED, notation $\Gamma \vdash_{\text{PRED}} \varphi$, if there is a derivation with root φ and all non-discharged formulas in Γ.

This is the usual system of minimal first order predicate logic. We extend this definition to allow for definable predicates (by λ-abstraction) and higher order functions.

Definition 3.2. The system PRED^{fr} is PRED plus the following extra clauses. The functional domains $\mathcal{F}$ are defined by

$$\mathcal{F} ::= \mathcal{B}{\to}\mathcal{B} \mid \mathcal{F}{\to}\mathcal{B}.$$

In the rules for term-formation we add

— There are countably many variables of each functional domain $F \in \mathcal{F}$,
— If $t\ \epsilon\ D_2$, $D_2 \in \mathcal{D}$ (an arbitrary domain) and x a variable of domain $D_1 \in \mathcal{B}\cup\mathcal{F}$, then $\lambda x\epsilon D_1.t\ \epsilon\ D_1{\to}D_2$,
— If $t\ \epsilon\ D_1{\to}D_2$ and $q\ \epsilon\ D_1$, then $tq\ \epsilon\ D_2$.
— If $\varphi\ \epsilon\ \mathsf{Prop}$ and x a variable of domain $D_1 \in \mathcal{B}\cup\mathcal{F}$, then $\forall x\epsilon D_1.\varphi\ \epsilon\ \mathsf{Prop}$.

The last rule replaces the application rule in the terms of PRED. The λ-abstraction (and application) on terms comes together with the usual notion of β-equality: two terms are β-equal if they are equal via the transitive, symmetric, reflexive closure, compatible with application and abstraction, of the one step β-reduction $(\lambda x\epsilon D.t)q{\longrightarrow}_\beta t[q/x]$. In the derivation rules we add the rule

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

So in PRED^{fr} there is no real distinction – in treatment – between basic domains and functional domains. For clarity and to be able to compare the systems, we keep the distinction.

The need for a conversion rule, establishing that β-equal formulas are equivalent, is felt in the following examples.

Examples 3.3. We work in the system PRED^{fr}. Let A be a basic domain, $R\ \epsilon\ A{\to}A{\to}\mathsf{Prop}$, $f\ \epsilon\ A{\to}A{\to}A$, $a\ \epsilon\ A$.

— We can define a new predicate $\lambda x\epsilon A.Rxx \;\epsilon\; A{\to}\mathsf{Prop}$ by λ-abstraction. Let's abbreviate it to Q. Now $Raa \vdash Qa$ by the conversion rule.
— We can define a new function $\lambda x\epsilon A.fx(fxa) \;\epsilon\; A{\to}A$ by λ-abstraction. Let's abbreviate it to g. Now $\forall x\epsilon A.Rx(fx(fxa)) \vdash \forall x\epsilon A.Rx(gx)$.

3.2. *A typed λ-calculus for minimal first order predicate logic*

Definition 3.4. $\lambda\mathrm{PRED}^{fr}$ is the typed λ-calculus defined as follows. Just like in λP, there are two forms of judgement.

$$\begin{array}{rl} \Gamma \vdash M : B & M \text{ is of type } B \text{ in context } \Gamma, \\ \Gamma \vdash & \Gamma \text{ is a correct context.} \end{array}$$

The set of pseudoterms is now defined by

$$\mathsf{T} ::= \mathsf{Var} \mid \mathsf{Set} \mid \mathsf{Prop} \mid \mathsf{Type}^s \mid \mathsf{Type}^p \mid (\mathsf{T}\mathsf{T}) \mid (\lambda x{:}\mathsf{T}.\mathsf{T}) \mid (\Pi x{:}\mathsf{T}.\mathsf{T}).$$

The intended interpretations are that Set represents the universe of basic sets (including sets of higher type), Prop represents the universe of formulas, Type^p represents the universe of predicate sets and Type^s contains just Set. The derivation rules for deriving the judgements $\Gamma \vdash M : B$ and $\Gamma \vdash$ are the following. Here, $\mathbf{s}, \mathbf{s_1}$ and $\mathbf{s_2}$ range over $\{\mathsf{Set}, \mathsf{Prop}, \mathsf{Type}^p, \mathsf{Type}^s\}$ and $\mathcal{R} = \{(\mathsf{Set},\mathsf{Set}), (\mathsf{Prop},\mathsf{Prop}), (\mathsf{Set},\mathsf{Prop}), (\mathsf{Set},\mathsf{Type}^p)\}$.

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

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

$$\text{(ax)}\quad \frac{\Gamma \vdash}{\Gamma \vdash \mathsf{Set} : \mathsf{Type}^s} \qquad\qquad \text{(ax)}\quad \frac{\Gamma \vdash}{\Gamma \vdash \mathsf{Prop} : \mathsf{Type}^p}$$

$$(\Pi)\quad \frac{\Gamma \vdash A : \mathbf{s_1} \quad \Gamma, x{:}A \vdash B : \mathbf{s_2}}{\Gamma \vdash \Pi x{:}A.B : \mathbf{s_2}}\ \text{if } (\mathbf{s}_1, \mathbf{s}_2) \in \mathcal{R}$$

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

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

We use the same notational conventions as for λP.

Example 3.5. $A{:}\mathsf{Set}, R{:}A{\to}A{\to}\mathsf{Prop}, s{:}\Pi x,y{:}A.Rxy{\to}Ryx, t{:}\Pi x,y,z{:}A.Rxy{\to}Ryz{\to}Rxz \vdash \lambda x,y{:}A.\lambda h{:}Rxy.txyxh(sxyh) : \Pi x,y{:}A.Rxy{\to}Rxx$.
This is a proof of the fact that if R is symmetric and transitive, then R is reflexive on its domain (the x for which Rxy for some y).

We list the main properties of the $\lambda\mathrm{PRED}^{fr}$. They are a consequence of the fact that λPRED^{fr} is a Pure Type System. Unless stated otherwise, proofs can be found in (Barendregt 1992) or in (Geuvers and Nederhof 1991).

Proposition 3.6. (Subject Reduction) If $\Gamma \vdash M : A$ and $M \twoheadrightarrow_\beta N$, then $\Gamma \vdash N : A$.

Proposition 3.7. (Typed Confluence) If $\Gamma \vdash M : A$, $\Gamma \vdash N : A$ and $M =_\beta N$, then there is a term P with $M \twoheadrightarrow_\beta P$, $N \twoheadrightarrow_\beta P$ and $\Gamma \vdash P : A$.

Proposition 3.8. (Uniqueness of Types) If $\Gamma \vdash M : A$ and $\Gamma \vdash M : B$, then $A =_\beta B$.

Proposition 3.9. (Decidability of Typing) Given a context Γ and a pseudo-term M, it is decidable whether M has a type in Γ and if M has a type it can be computed.

Proposition 3.10. (Strong Normalization) All well-typed expressions of λPRED^{fr} are Strongly Normalizing with respect to β-reduction.

Proposition 3.11. (Undecidability of Inhabitation) In general it is undecidable whether, given a context Γ and a type A in Γ, there exists an M such that $\Gamma \vdash M : A$.

The proof of Undecidability of Inhabitation for λP in (Bezem and Springintveld 1996) (see also Proposition 2.10) applies immediately to λPRED^{fr}. As a matter of fact, the proof in (Bezem and Springintveld 1996) already shows that minimal first order predicate logic PRED is undecidable.

Proposition 3.12. (Undecidability of Type Assignment) It is in general undecidable whether, given a context Γ and an untyped λ term N, there is a term M such that M is well-typed in Γ and $|M| \equiv N$.

Modulo some small changes, the proof of Undecidability of Type Assignment for λP (see 2.11) in (Dowek 1993) can be applied to λPRED^{fr} as well.

The following is a specific property of λPRED^{fr}, that does not hold for λP. In fact it states that λPRED^{fr} is really a *logical* system, where first the terms are built up, then the propositions and then the proofs.

Proposition 3.13. In λPRED^{fr}, if $\Gamma \vdash M : A$, then

$$\Gamma_D, \Gamma_T, \Gamma_P \vdash M : A$$

where

— $\Gamma_D, \Gamma_T, \Gamma_P$ is a permutation of Γ,
— Γ_D only contains declarations $\alpha : \mathsf{Set}$,
— Γ_T only contains declarations $x : A$ with $\Gamma_D \vdash A : \mathsf{Set}/\mathsf{Type}^p$,
— Γ_P only contains declarations $p : \varphi$ with $\Gamma_D, \Gamma_T \vdash \varphi : \mathsf{Prop}$.

The Γ_D, Γ_T and Γ_P are determined uniquely up to permutation. We refer to Γ_D as the *set-context* of Γ, to Γ_T as the *object-context* of Γ and to Γ_P as the *proof-context* of Γ and to the concatenation Γ_D, Γ_T as a *language-context*. Furthermore, if $\Gamma \vdash M : A$, then

— 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 will refer to the M such that Γ_D, $\Gamma_T \vdash M : A$ with $A : \mathsf{Set}$ as the *object-language*.

The proof of this Proposition is not difficult and proceeds by induction on the derivation of $\Gamma \vdash M : A$. A detailed proof can be found in (Geuvers 1993).

By restricting the rules we obtain a system we call λPRED which corresponds to PRED in the same way that $\lambda\mathrm{PRED}^{fr}$ corresponds to $\mathrm{PRED}^{\mathrm{fr}}$, via the formulas-as-types embedding.

Definition 3.14. The system λPRED is defined by

— Restricting in the rules of $\lambda\mathrm{PRED}^{fr}$ the (λ)-rule to the case where $\mathsf{s} = \mathsf{Prop}$,
— Adding a universe Fun,
— Removing $(\mathsf{Set}, \mathsf{Set})$ from $\mathcal{R}$,
— Allowing in triples $(\mathbf{s_1}, \mathbf{s_2}, \mathbf{s_3})$ of universes in $\mathcal{R}$, where $\mathbf{s_3}$ is the type in the conclusion of the (Π)-rule, and adding the triples $(\mathsf{Set}, \mathsf{Set}, \mathsf{Fun})$ and $(\mathsf{Set}, \mathsf{Fun}, \mathsf{Fun})$.

The intention of Fun is to represent the universe of function types. (The $\mathcal{F}$ in Definition 3.1.) We can now only form basic sets of type Set and $A{\to}A : \mathsf{Fun}$, $A{\to}A{\to}A : \mathsf{Fun}$, etcetera.

3.3. *The conservativity results*

The formulas-as-types embedding from $\mathrm{PRED}^{\mathrm{fr}}$ into $\lambda\mathrm{PRED}^{fr}$ is defined as follws.

— A basic domain B is mapped to a declaration $B : \mathsf{Set}$ in the context. Hence, the functional domains are mapped to terms of type Set and the predicate domains are mapped to terms of type Type^p. For convenience, we don't distinguish between the notation of the domain in the $\mathrm{PRED}^{\mathrm{fr}}$ and in $\lambda\mathrm{PRED}^{fr}$.
— A constant c^D is mapped to a declaration $c : D$ in the context. Hence, a term $f(t_1, \ldots, t_n)$ is translated to $f t_1 \cdots t_n$.
— A proposition is mapped to a term of type Prop by translating $\forall x{:}A.\varphi$ to $\Pi x{:}A.\varphi$ and $\varphi \supset \psi$ to $\varphi{\to}\psi$.
— An assumption φ is translated into a declaration $p : \varphi$ in the context.
— A derivation is mapped to a typed term translating an I-rule into a λ-abstraction and an E-rule into an application.

As the translation alters so little for domains, terms and propositions, we usually don't write it and identify e.g. $\forall x \epsilon A.Px \supset Qx$ with $\Pi x{:}A.Px{\to}Qx$. The translation from derivations to proof-terms will be denoted by $\llparenthesis - \rrparenthesis$. So if $\Delta \vdash^{\Theta}_{\mathrm{PRED}^{\mathrm{fr}}} \varphi$, denoting that Θ is a derivation of φ from Δ, then $\llparenthesis \Theta \rrparenthesis$ is the associated proof-term.

We do not define the translation of derivations into proof-terms precisely, as it is rather well-known and involves quite a lot of syntactical detail. Instead we give an example of a derivation and how it is (inductively) translated to a term in $\lambda\mathrm{PRED}^{fr}$. Morevover, this example should provide enough evidence for the fact that there is also a translation back from proof-terms (in $\lambda\mathrm{PRED}^{fr}$) to derivations in $\mathrm{PRED}^{\mathrm{fr}}$. The translation from terms to derivations will be denoted by $\Sigma(-)$: if t is a proof-term, then $\Sigma(t)$ is the associated derivation.

The isomorphism between $\mathrm{PRED}^{\mathrm{fr}}$ and $\mathrm{PRED}^{\mathrm{fr}}$ arises from the fact that these two translations $\llparenthesis - \rrparenthesis$ and $\Sigma(-)$ are eachothers inverses. Details can be found in (Geuvers 1993).

Example 3.15. Let a domain A, a predicate P on A and a binary relation Q on A be given. The following is a deriavtion of $(\forall x \epsilon A.Px) \supset (\forall x \epsilon A.Qxx)$ from the hypothesis

$\forall x,y\epsilon A.Px \supset Py \supset Qxy$. Let's call this derivation Θ.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall x,y\epsilon A.Px \supset Py \supset Qxy}{\forall y\epsilon A.Px \supset Py \supset Qxy}}{Px \supset Px \supset Qxx} \quad \dfrac{[\forall x\epsilon A.Px]^1}{Px}}{Px \supset Qxx} \quad \dfrac{[\forall x\epsilon A.Px]^1}{Px}}{Qxx}}{\forall x\epsilon A.Qxx}}{(\forall x\epsilon A.Px) \supset (\forall x\epsilon A.Qxx)}\ 1$$

We decorate Θ by λ-terms, thus inductively defining the λ-term $⦃\Theta⦄$ from the derivation Θ.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{g : \forall x,y\epsilon A.Px \supset Py \supset Qxy}{gx : \forall y\epsilon A.Px \supset Py \supset Qxy}}{gxx : Px \supset Px \supset Qxx} \quad \dfrac{h : \forall x\epsilon A.Px}{hx : Px}}{gxx(hx) : Px \supset Qxx} \quad \dfrac{h : \forall x\epsilon A.Px}{hx : Px}}{gxx(hx)(hx) : Qxx}}{\lambda x{:}A.gxx(hx)(hx) : \forall x\epsilon A.Qxx}}{\lambda h{:}(\forall x\epsilon A.Px).\lambda x{:}A.gxx(hx)(hx) : (\forall x\epsilon A.Px) \supset (\forall x\epsilon A.Qxx)}$$

So $⦃\Theta⦄ := \lambda h{:}(\Pi x{:}A.Px)\lambda x{:}A.gxx(hx)(hx)$. In λPRED^{fr}, we can derive

$$A{:}\mathsf{Set}, P{:}A{\to}\mathsf{Prop}, Q{:}A{\to}A{\to}\mathsf{Prop}, g{:}\Pi x,y{:}A.Px{\to}Py{\to}Qxy \quad \vdash$$
$$\lambda h{:}(\Pi x{:}A.Px)\lambda x{:}A.gxx(hx)(hx) : (\Pi x{:}A.Px){\to}(\Pi x{:}A.Qxx).$$

Proposition 3.16. Given a PRED^{fr}-signature Σ (containing a finite number of domains and relations and constants over these domains), a finite set of formulas (over Σ) Δ and a formula φ,

$$\Delta \vdash^{\Theta}_{\text{PRED}^{\text{fr}}} \varphi \Rightarrow \Gamma_\Sigma, \Gamma_{\Delta,\varphi,\Theta}, \vec{p} : \Delta \vdash_{\lambda\text{PRED}^{fr}} ⦃\Theta⦄ : \varphi,$$

where $\Gamma_\Sigma, \Gamma_{\Delta,\varphi,\Theta}$ is the canonically defined language context containing declarations for all the constants and free variables in $\Sigma, \Delta, \varphi, \Theta$.
Furthermore, if $\Gamma_D, \Gamma_T \vdash \psi : \mathsf{Prop}$ for all $\psi \in \Delta \cup \{\varphi\}$, then

$$\Gamma_D, \Gamma_T, \vec{p} : \Delta \vdash_{\lambda\text{PRED}^{fr}} M : \varphi \Rightarrow \Delta \vdash^{\Sigma(M)}_{\text{PRED}^{\text{fr}}} \varphi.$$

The mappings $⦃-⦄$ and $\Sigma(-)$ constitute a bijection between derivations in PRED^{fr} and proof-terms in λPRED^{fr}.

For a detailed proof see (Geuvers 1993).

Proposition 3.17. PRED^{fr} is conservative over PRED, that is, for Δ a set of formulas and φ a formula of PRED,

$$\Delta \vdash_{\text{PRED}^{\text{fr}}} \varphi \ \Rightarrow \ \Delta \vdash_{\text{PRED}} \varphi.$$

Proof. The proof is by eliminating cuts (in derivations) and normalizing the terms as follows. In PRED^{fr} all terms are Strongly Normalizing and all cuts in derivations can be eliminated. Moreover if Δ and φ are taken from PRED and $\Delta \vdash_{\text{PRED}^{\text{fr}}} \varphi$ by a cut-free derivation in which only normal terms occur, then this is already a derivation in PRED.

The easiest way to treat the normalization and cut-elimination argument is by looking at λPRED^{fr} and λPRED instead. The argument then runs as follows.

1 The system λPRED^{fr} is Strongly Normalizing (e.g. there is a reduction-preserving embedding into the Calculus of Constructions, which is known to be SN).
2 Suppose that the term-context Γ_T is actually a λPRED-context (i.e. there occur no Set-types of higher type in it). Suppose furthermore that $\Delta \cup \{\varphi\}$ contains only λPRED-formula.
3 If $\Gamma_D, \Gamma_T \vdash_{\lambda\text{PRED}^{fr}} t : A$ with t in normal form and $A : \mathsf{Set} \in \Gamma_D$ (i.e. A is a basic domain), then $\Gamma_D, \Gamma_T \vdash_{\lambda\text{PRED}} t : A$
4 If $\Gamma_D, \Gamma_T, \Gamma_P \vdash_{\lambda\text{PRED}^{fr}} q : \varphi$ with q in normal form, then $\Gamma_D, \Gamma_T, \Gamma_P \vdash_{\lambda\text{PRED}} q : \varphi$.

The third and fourth step are proved by induction on the structure of terms (t and q). Together, these four steps prove the proposition. □

Now to show the completeness of the formulas-as-types embedding from first order predicte logic (PRED) into λP, we only have to show the completeness of the embedding of λPRED^{fr} into λP. Both λPRED^{fr} and λP are Pure Type Systems (PTS) and the embedding of λPRED^{fr} into λP that we are looking at is a *PTS-morphism H* given by

$$H := \begin{cases} \mathsf{Set} & \mapsto \ \textbf{type} \\ \mathsf{Prop} & \mapsto \ \textbf{type} \\ \mathsf{Type}^s & \mapsto \ \textbf{kind} \\ \mathsf{Type}^p & \mapsto \ \textbf{kind} \end{cases}$$

A PTS-embedding extends immediately to all terms and contexts and it preserves typing, i.e.

$$\Gamma \vdash_{\lambda\text{PRED}^{fr}} M : A \ \Rightarrow \ H(\Gamma) \vdash_{\lambda\text{P}} H(M) : H(A).$$

We now give the technical details of the proof of completeness of $H : \lambda\text{PRED}^{fr} \to \lambda\text{P}$. The proof uses techniques developped in (Swaen 1989) who shows completeness of the formulas-as-types embedding from first order predicate logic into Martin-Löf's intuitionistic theory of types. A different proof of our result can be found in (Berardi 1990).

The question of completeness is whether for any λPRED^{fr}-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

$$\Gamma_D, \Gamma_T, \Gamma_P \vdash N : \varphi \text{ in } \lambda\text{PRED}^{fr}.$$

Convention 3.18. In the following we assume for any λPRED^{fr}-context Γ that

1 $\Gamma \equiv \Gamma_D, \Gamma_T, \Gamma_P$,
2 Γ_D contains $O : \mathsf{Set}$ (so there is at least one basic domain),
3 all basic domains in Γ_D are nonempty,
4 Γ_T begins with a declaration $\beta{:}\mathsf{Prop}$ and Γ_P begins with $z{:}\beta$. This β will be referred to as True, the z will be referred to as t.

The first clause is validated by Proposition 3.13. If the second clause were not satisfied we would in fact be working in propositional logic. The third and fourth clause are added for convenience. In case there are empty domains in the logic, the completeness result would still hold with a slightly adapted argument.

We fix a language-context of $\lambda\mathrm{PRED}^{fr}$ Γ_D, Γ_T. We first want to define a map $|-|^p$ from λP-terms, typable in $H(\Gamma_D, \Gamma_T)$, to the object language of $\lambda\mathrm{PRED}^{fr}$. This map should be the inverse of H on the object language. That is, if $\Gamma_D, \Gamma_T \vdash_{\lambda\mathrm{PRED}^{fr}} M : A$ with $A : \mathsf{Set}$, then $|H(M)|^p \equiv M$. We need to define $|-|^p$ not just on terms typable in $H(\Gamma_D, \Gamma_T)$, but in a *elementary extension* of $H(\Gamma_D, \Gamma_T)$.

Definition 3.19. For Γ_D, Γ_T a language-context of $\lambda\mathrm{PRED}^{fr}$ and Δ a context of λP, we say that Δ *is an elementary extension of* $H(\Gamma_D, \Gamma_T)$, notation $\Delta \succ H(\Gamma_D, \Gamma_T)$, if $\Delta \supseteq H(\Gamma_D, \Gamma_T)$ and the extra declarations in Δ are all of the form $x{:}\sigma$ with $H(\Gamma_D, \Gamma_T) \vdash_{\lambda\mathrm{P}} \sigma : \mathbf{type}$.

For example, $H(\Gamma_D, \Gamma_T, \Gamma_P)$ is always an elementary extension of $H(\Gamma_D, \Gamma_T)$.

Definition 3.20. Let $\Delta \succ H(\Gamma_D, \Gamma_T)$. The mapping $|-|^p$ from λP-terms in the context Δ to terms of $\lambda\mathrm{PRED}^{fr}$ is defined as follows.

(i)	$\lvert\mathbf{type}\rvert^p$	$:=$	Set,
(ii)	$\lvert\mathbf{kind}\rvert^p$	$:=$	Type^s,
(iii)	$\lvert x\rvert^p$	$:=$	x, if $x \in \Gamma_D$(so $x : \mathsf{Set}$),
(iv)	$\lvert x\rvert^p$	$:=$	O, if $x \in \Gamma_T$ and $x : \cdots \to \mathsf{Prop}$,
(v)	$\lvert x\rvert^p$	$:=$	x, for x another variable,
(vi)	$\lvert\Pi x{:}A.B\rvert^p$	$:=$	$\lvert B\rvert^p$ if $A{:}\mathbf{type}, B{:}\mathbf{kind}$,
		$:=$	$\Pi x{:}\lvert A\rvert^p.\lvert B\rvert^p$ else,
(vii)	$\lvert\lambda x{:}A.M\rvert^p$	$:=$	$\lvert M\rvert^p$ if $A{:}\mathbf{type}, M{:}B{:}\mathbf{kind}$, (for some B),
		$:=$	$\lambda x{:}\lvert A\rvert^p.\lvert B\rvert^p$ else,
(ix)	$\lvert PM\rvert^p$	$:=$	$\lvert P\rvert^p$ if $M{:}A{:}\mathbf{type}, P{:}B{:}\mathbf{kind}$, (for some A, B),
		$:=$	$\lvert P\rvert^p\lvert M\rvert^p$ else

The definition extends immediately to the context Δ itself, where a declaration coming from $x : \cdots \to \mathsf{Prop} \in \Gamma_T$ (case (iv)) is removed.

Example 3.21. As a running example throughout this Section, we will use the following $\lambda\mathrm{PRED}^{fr}$-context.

$$\begin{aligned}
&O : \mathsf{Set}, A : \mathsf{Set},\\
&\mathrm{True} : \mathsf{Prop}, c_O : O, c_A : A, f : O \to A, g : (O \to A) \to A, R : O \to A \to \mathsf{Prop},\\
&\mathsf{t} : \mathrm{True}, p_1 : \Pi x{:}O.Rx(fx), p_2 : \Pi x{:}A.(\Pi y{:}O.Ry(g(\lambda z{:}O.x))) \to Rc_O x.
\end{aligned}$$

So here

$$\begin{aligned}
\Gamma_D &= O : \mathsf{Set}, A : \mathsf{Set},\\
\Gamma_T &= \mathrm{True} : \mathsf{Prop}, c_O : O, c_A : A, f : O \to A, g : (O \to A) \to A, R : O \to A \to \mathsf{Prop},\\
\Gamma_P &= \mathsf{t} : \mathrm{True}, p_1 : \Pi x{:}O.Rx(fx), p_2 : \Pi x{:}A.(\Pi y{:}O.Ry(g(\lambda z{:}O.x))) \to Rc_O x.
\end{aligned}$$

(In the examples of this Section we will use these abbreviations. Note that in the Definitions, Lemmas and Propositions, Γ_D, Γ_T is *some* fixed language context.) Now consider the following elementary extension Δ of $H(\Gamma_D,\Gamma_T)$:

$$\Delta := H(\Gamma_D,\Gamma_T,\Gamma_P), z_1 : O{\to}A, z_2 : \Pi x{:}O.Rx(gz_1){\to}A, z_3 : \Pi x{:}A.Rc_O x.$$

Note that the type of z_2 does not correspond to a logical formula (it is not in the image of H). We compute the $|-|^p$-image of this context. It is

$$\begin{aligned}&O : \mathsf{Set}, A : \mathsf{Set}, c_O : O, c_A : A, f : O{\to}A, g : (O{\to}A){\to}A,\\ &\mathsf{t} : O, p_1 : O{\to}O, p_2 : A{\to}(O{\to}O){\to}O, z_1 : O{\to}A, z_2 : O{\to}O{\to}A, z_3 : A{\to}O.\end{aligned}$$

That the mapping $|-|^p$ is indeed from λP-terms in the context Δ to $\lambda\mathrm{PRED}^{fr}$ is justified by the following Proposition.

Proposition 3.22. Let $\Delta \succ H(\Gamma_D,\Gamma_T)$.

$$\Delta \vdash_{\lambda\mathrm{P}} M : A \;\Rightarrow |\Delta|^p \vdash_{\lambda\mathrm{PRED}^{fr}} |M|^p : |A|^p.$$

Proof. By induction on the derivation of $\Delta \vdash M : A$ in λP. There are no difficult cases. In the conversion rule, it is used that, if $M =_\beta N$, then $|M|^p \equiv |N|^p$. □

Example 3.23. (Definitions are as in Example 3.21.) In λP we have $\Delta \vdash_{\lambda\mathrm{P}} z_3 c_A : Rc_O c_A$, which is mapped to $|\Delta|^p \vdash_{\lambda\mathrm{PRED}^{fr}} z_3 c_A : O$. Similarly, $\Delta \vdash_{\lambda\mathrm{P}} z_2 c_O(z_3(gz_1)) : A$, is mapped to $|\Delta|^p \vdash_{\lambda\mathrm{PRED}^{fr}} z_2 c_O(z_3(gz_1)) : A$.

Fact 3.24. 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.)

Corollary 3.25. For $\Delta \succ H(\Gamma_D,\Gamma_T)$, say $\Delta \equiv H(\Gamma_D,\Gamma_T),\Delta'$ we have

$$\Delta \vdash_{\lambda\mathrm{P}} M : A(: \mathbf{type}) \Rightarrow \Gamma_D,\Gamma_T,|\Delta'|^p \vdash_{\lambda\mathrm{PRED}^{fr}} |M|^p : |A|^p.$$

Proof. We know by Proposition 3.22 that $|\Delta|^p \vdash_{\lambda\mathrm{PRED}^{fr}} |M|^p : |A|^p$. Now note that $|H(\Gamma_D)|^p \equiv \Gamma_D$. Furthermore, 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 $|x|^p \equiv O$ and the declaration of x is removed from the context by $|-|^p$. So $\Gamma_D,\Gamma_T,|\Delta'|^p \supseteq |\Delta|^p$ and so $\Gamma_D,\Gamma_T,|\Delta'|^p \vdash |M|^p : |A|^p$ by Thinning.

□

All this means that $|-|^p$ is a mapping back from λP-terms (typable in $\Delta \succ H(\Gamma_D,\Gamma_T)$) to the object-language of $\lambda\mathrm{PRED}^{fr}$ that does not change the terms that originated from the object-language.

Now we define a mapping Tr back from λP to the proof-language of $\lambda\mathrm{PRED}^{fr}$. So types in λP will become propositions and objects will become proofs of $\lambda\mathrm{PRED}^{fr}$: If $\Delta \succ H(\Gamma_D,\Gamma_T)$ and $\Delta \vdash_{\lambda\mathrm{P}} A : \mathbf{type}$, then $|\Delta|^p \vdash_{\lambda\mathrm{PRED}^{fr}} \mathsf{Tr}(A) : \mathsf{Prop}$. Moreover, if $\Gamma_D,\Gamma_T \vdash_{\lambda\mathrm{PRED}^{fr}} \varphi : \mathsf{Prop}$, then $\mathsf{Tr}(\varphi)$ should be equivalent to φ, i.e. $\mathsf{Tr}(\varphi)$ should be inhabited iff φ is. We prove that for such φ we can find terms M_1, M_2 such that $\Gamma_D,\Gamma_T \vdash_{\lambda\mathrm{PRED}^{fr}} M_1 : \varphi{\to}\mathsf{Tr}(\varphi)$ and $\Gamma_D,\Gamma_T \vdash_{\lambda\mathrm{PRED}^{fr}} M_2 : \mathsf{Tr}(\varphi){\to}\varphi$. We introduce some notation.

Notation 3.26. For Γ a context in $\lambda\mathrm{PRED}^{fr}$ and $\Gamma \vdash_{\lambda\mathrm{PRED}^{fr}} \varphi,\psi : \mathsf{Prop}$, we write

$$\Gamma \vdash_{\lambda\mathrm{PRED}^{fr}} \langle M_1, M_2\rangle : \varphi \leftrightarrow \psi$$

if $\Gamma \vdash_{\lambda\mathrm{PRED}^{fr}} M_1 : \varphi{\to}\psi$ and $\Gamma \vdash_{\lambda\mathrm{PRED}^{fr}} M_2 : \psi{\to}\varphi$.

The main trick in the definition of Tr is to decompose a type $\Pi x{:}A.B$ (A, B : **type**) into a quantification over $|A|^p$ and an implication $\mathsf{Tr}(A) \to \mathsf{Tr}(B)$. This idea occurs in different places in the literature. Our main inspiration is (Swaen 1989), who uses it for first-order predicate logic and Martin Löf's type theory, but one can also find it in (Mohring 1986), who uses it for defining a realisability interpretation for the Calculus of Constructions, and in (Geuvers 1995), where it is used to prove Strong Normalization for the Calculus of Constructions.

Definition 3.27. Let $\Delta \succ H(\Gamma_D, \Gamma_T)$. The map Tr on constructors of λP in Δ is defined as follows.

$$\begin{array}{lrcl}
(i) & \mathsf{Tr}(\alpha) & := & \text{True, if } \alpha{:}\mathsf{Set} \in \Gamma_D, \\
(ii) & \mathsf{Tr}(\alpha) & := & \alpha, \text{ if } \alpha{:}\cdots{\to}\mathsf{Prop} \in \Gamma_T, \\
(iii) & \mathsf{Tr}(\lambda x{:}A.M) & := & \lambda x{:}|A|^p.\mathsf{Tr}(M), \\
(iv) & \mathsf{Tr}(Qt) & := & \mathsf{Tr}(Q)|t|^p, \\
(v) & \mathsf{Tr}(\Pi x{:}A.B) & := & \Pi x{:}|A|^p.\mathsf{Tr}(A) \to \mathsf{Tr}(B).
\end{array}$$

Example 3.28. With the λP-context Δ defined as in Example 3.21, we find the following Tr translations for types in this context.

$$\begin{array}{rcl}
\mathsf{Tr}(\Pi x{:}O.Rx(fx)) & = & \Pi x{:}O.\text{True} \to Rx(fx), \\
\mathsf{Tr}(\Pi x{:}O.Rx(gz_1) \to A) & = & \Pi x{:}O.\text{True} \to Rx(gz_1) \to \text{True}.
\end{array}$$

Proposition 3.29. For $\Delta \succ H(\Gamma_D, \Gamma_T)$, say $\Delta \equiv H(\Gamma_D, \Gamma_T), \Delta'$ we have

$$\begin{array}{rl}
& \Delta \vdash_{\lambda\mathrm{P}} C : \Pi x_1{:}A_1.\ldots\Pi x_n{:}A_n.\mathbf{type} \\
\Rightarrow & \Gamma_D, \Gamma_T, |\Delta'|^p \vdash_{\lambda\mathrm{PRED}^{fr}} \mathsf{Tr}(C) : |A_1|^p \to \cdots \to |A_n|^p \to \mathsf{Prop}.
\end{array}$$

Proof. By induction on the derivation. Note that if A:**type** in λP, then $|A|^p$ contains no object-variables. Furthermore, if $\Delta \vdash_{\lambda\mathrm{P}} M : A(: \mathbf{type})$, then $\Gamma_D, \Gamma_T, |\Delta'|^p \vdash_{\lambda\mathrm{PRED}^{fr}} |M|^p : |A|^p$ by Corollary 3.25. □

All σ : **type** in λP are mapped to a $\mathsf{Tr}(\sigma)$: Prop in $\lambda\mathrm{PRED}^{fr}$. If $\sigma \equiv H(A)$ with A : Set in $\lambda\mathrm{PRED}^{fr}$, then $\mathsf{Tr}(\sigma)$ should be inhabited ('true').

Lemma 3.30. If $\Gamma_D \vdash_{\lambda\mathrm{PRED}^{fr}} A : \mathsf{Set}$, then

$$\exists M_1, M_2[\Gamma_D, \Gamma_T \vdash_{\lambda\mathrm{PRED}^{fr}} \langle M_1, M_2\rangle : \text{True} \leftrightarrow \mathsf{Tr}(A)].$$

(To be precise we would have to write $\mathsf{Tr}(H(A))$ instead of $\mathsf{Tr}(A)$, but H is the identity on terms of type Set.)

Proof. Immediate from the definition of Tr: if $\Gamma_D \vdash A : \mathsf{Set}$, then $A \equiv \cdots \to \alpha$ with $\alpha{:}\mathsf{Set} \in \Gamma_D$. Hence $\mathsf{Tr}(A) \equiv \cdots \to \text{True}$, which is logically equivalent to True. □

The mapping Tr preserves $=_\beta$. This is proved using a substitution Lemma for Tr.

Lemma 3.31. For $\Delta \succ H(\Gamma_D, \Gamma_T)$, say $\Delta \equiv H(\Gamma_D, \Gamma_T), \Delta'$, with $\Delta \vdash_{\lambda\mathrm{P}} A, B$: **type** and $\Delta \vdash_{\lambda\mathrm{P}} t : B$ we have

$$\mathsf{Tr}(A)[|t|^p/x] \equiv \mathsf{Tr}(A[t/x]).$$

If $\Delta \vdash_{\lambda\mathrm{P}} A, A'$: **type** and $A =_\beta A'$,then

$$\mathsf{Tr}(A) =_\beta \mathsf{Tr}(A').$$

Proof. The first is easily proved by induction on the structure of A. The second follows from the fact that, if $A \longrightarrow_\beta A'$, then $\mathsf{Tr}(A) =_\beta \mathsf{Tr}(A')$. □

Proposition 3.32. For each language-context Γ_D, Γ_T and φ with $\Gamma_D, \Gamma_T \vdash_{\lambda\mathrm{PRED}^{fr}} \varphi : \mathsf{Prop}$ we have

$$\exists M_1, M_2[\Gamma_D, \Gamma_T \vdash_{\lambda\mathrm{PRED}^{fr}} \langle M_1, M_2\rangle : \varphi \leftrightarrow \mathsf{Tr}(H(\varphi))].$$

(Note that H is the identity on expressioons of type Prop, so we can skip it.)

Proof. By induction on the structure of φ, assuming that φ is in normal form. (By Lemma 3.31, $\mathsf{Tr}(\varphi) =_\beta \mathsf{Tr}(\mathrm{nf}(\varphi))$.)

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 3.24.)

$\supset$ Say $\varphi \equiv \psi{\to}\chi$ with $\psi, \chi{:}\mathsf{Prop}$. Then $\mathsf{Tr}(\varphi{\to}\psi) \equiv \Pi x{:}|\varphi|^p.\mathsf{Tr}(\varphi){\to}\mathsf{Tr}(\psi)$. Now we are done by IH: The variable x will not occur free in $\varphi{\to}\psi$ and one easily constructs the required proof-terms.

$\forall$ 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 3.24 and Lemma 3.30, $\Pi x{:}|A|^p.\mathsf{Tr}(A){\to}\mathsf{Tr}(\psi)$ is equivalent to $\Pi x{:}A.\mathsf{Tr}(\psi)$, so we are done by IH.

□

Definition 3.33. For $\Delta \succ H(\Gamma_D, \Gamma_T)$, 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 $z' : \mathsf{Tr}(A)$. (Of course we make sure that the declared variables in $\mathsf{Tr}(\Delta)$ are different from the ones in $|\Delta|^p$.)

Example 3.34. We look at the translation of Δ as in Example 3.21.

$$\begin{array}{rcl}\mathsf{TR}(\Delta) & = & O : \mathsf{Set}, A : \mathsf{Set},\\ & & \mathrm{True} : \mathsf{Prop}, c_O : O, c_A : A, f : A{\to}O, g : (O{\to}A){\to}A, R : O{\to}A{\to}\mathsf{Prop},\\ & & \mathsf{t} : O, p_1 : O{\to}O, p_2 : A{\to}(O{\to}O){\to}O, z_1 : O{\to}A, z_2 : O{\to}O{\to}A, z_3 : A{\to}O,\\ & & \mathsf{t}' : \mathrm{True}, p_1' : \Pi x{:}O.\mathrm{True}{\to}Rx(fx),\\ & & p_2' : \Pi x{:}A.\mathrm{True}{\to}(\Pi y{:}O.\mathrm{True}{\to}Ry(g(\lambda z{:}O.x))){\to}Rc_Ox.\end{array}$$

Proposition 3.35. Let $\Delta \succ \Gamma_D, \Gamma_T$, then

$$\Delta \vdash_{\lambda\mathrm{P}} M : A(:\mathbf{type}) \text{ in } \lambda\mathrm{P} \Rightarrow \exists N[\mathsf{TR}(\Delta) \vdash_{\lambda\mathrm{PRED}^{fr}} N : \mathsf{Tr}(A)] \text{ in } \lambda\mathrm{PRED}^{fr}.$$

Proof. By induction on the derivation of $\Delta \vdash_{\lambda\mathrm{P}} M : A$ in λP.

(var) $M \equiv x$ then either $x{:}A$ in Γ_T or in Δ'. In the first case $\mathsf{Tr}(A) \leftrightarrow \mathrm{True}$ and in the second case $x{:}\mathsf{Tr}(A) \in \mathsf{TR}(\Delta)$.

(app) Say

$$\frac{\Delta \vdash_{\lambda\mathrm{P}} M : \Pi x{:}A.B \quad \Delta \vdash_{\lambda\mathrm{P}} t : A}{\Delta \vdash_{\lambda\mathrm{P}} Mt : B[t/x]}$$

By IH, $\mathsf{TR}(\Delta) \vdash_{\lambda\mathrm{PRED}^{fr}} N : \mathsf{Tr}(\Pi x{:}A.B) \equiv \Pi x{:}|A|^p.\mathsf{Tr}(A){\to}\mathsf{Tr}(B)$ and

$\mathsf{TR}(\Delta) \vdash_{\lambda\mathrm{PRED}^{fr}} Q : \mathsf{Tr}(A)$. We also have $\mathsf{TR}(\Delta) \vdash_{\lambda\mathrm{PRED}^{fr}} |t|^p : |A|^p$, by Corollary 3.25. So we may conclude $\mathsf{TR}(\Delta) \vdash_{\lambda\mathrm{PRED}^{fr}} N|t|^p Q : \mathsf{Tr}(B)[|t|^p/x] \equiv \mathsf{Tr}(B[t/x])$.

(λ) Say

$$\frac{\Delta, x{:}B \vdash_{\lambda\mathrm{P}} M : C \quad \Delta \vdash_{\lambda\mathrm{P}} \Pi x{:}B.C : \mathbf{type}}{\Delta \vdash_{\lambda\mathrm{P}} \lambda x{:}B.M : \Pi x{:}B.C}$$

By IH, $\mathsf{TR}(\Delta, x{:}B) \vdash_{\lambda\mathrm{PRED}^{fr}} N : \mathsf{Tr}(C)$. $\mathsf{TR}(\Delta, x{:}B) \equiv \mathsf{TR}(\Delta), x{:}|B|^p, x'{:}\mathsf{Tr}(B)$, so we have

$$\mathsf{TR}(\Delta) \vdash_{\lambda\mathrm{PRED}^{fr}} \lambda x{:}|B|^p.\lambda x'{:}\mathsf{Tr}(B).N : \Pi x{:}|B|^p.\mathsf{Tr}(B) \to \mathsf{Tr}(C) \equiv \mathsf{Tr}(\Pi x{:}B.C).$$

(conv) We are immediately done by Lemma 3.31.

□

Corollary 3.36. The embedding H from $\lambda\mathrm{PRED}^{fr}$ into λP is complete, i.e. if Γ_D, Γ_T is a language-context with $\Gamma_D, \Gamma_T \vdash_{\lambda\mathrm{PRED}^{fr}} \varphi : \mathsf{Prop}$ and Γ_P a proof-context, then

$$H(\Gamma_D, \Gamma_T, \Gamma_P) \vdash_{\lambda\mathrm{P}} M : H(\varphi) \Rightarrow \exists N[\Gamma_D, \Gamma_T, \Gamma_P \vdash_{\lambda\mathrm{PRED}^{fr}} N : \varphi].$$

Proof. $H(\Gamma_D, \Gamma_T, \Gamma_P)$ is an elementary extension of Γ_D, Γ_T, so by the Proposition we have

$$\Gamma_D, \Gamma_T, |\Gamma_P|^p, \mathsf{Tr}(\Gamma_P) \vdash_{\lambda\mathrm{PRED}^{fr}} N : \mathsf{Tr}(\varphi)$$

for some term N. Now every declaration in $|\Gamma_P|^p$ is of the form $y : B$ where $B : \mathsf{Set}$, so we can substitute for such a y a term of type B in the context Γ_D, Γ_T. Furthermore, if $z{:}B \in \mathsf{Tr}(\Gamma_P)$, then $\exists M_1, M_2.[\Gamma_D, \Gamma_T \vdash_{\lambda\mathrm{PRED}^{fr}} \langle M_1, M_2\rangle : B \leftrightarrow \mathsf{Tr}(B)]$ by Proposition 3.32. So we can replace each $z{:}\mathsf{Tr}(B)$ by $\hat{z}{:}B$, at the same time substituting $M_1\hat{z}$ for z inside N. (Such a variable z does not occur in $\mathsf{Tr}(\varphi)$.) We obtain a term N' such that

$$\Gamma_D, \Gamma_T, \Gamma_P \vdash_{\lambda\mathrm{PRED}^{fr}} N' : \mathsf{Tr}(\varphi).$$

By again applying Proposition 3.32,we can transform this N' into a N'' with

$$\Gamma_D, \Gamma_T, \Gamma_P \vdash_{\lambda\mathrm{PRED}^{fr}} N'' : \varphi.$$

□

As a Corollary to the proof of completeness of $H : \lambda\mathrm{PRED}^{fr} \to \lambda\mathrm{P}$, we get completeness of $H : \lambda\mathrm{PRED} \to \lambda\mathrm{P}^-$: the same Definitions apply and the same results can be proven.

Corollary 3.37. The embedding H from λPRED into $\lambda\mathrm{P}^-$ is complete, i.e. if Γ_D, Γ_T is a language-context with $\Gamma_D, \Gamma_T \vdash_{\lambda\mathrm{PRED}} \varphi : \mathsf{Prop}$ and Γ_P a proof-context, then

$$H(\Gamma_D, \Gamma_T, \Gamma_P) \vdash_{\lambda\mathrm{P}^-} M : H(\varphi) \Rightarrow \exists N[\Gamma_D, \Gamma_T, \Gamma_P \vdash_{\lambda\mathrm{PRED}} N : \varphi].$$

3.4. *Some comments on the completeness result*

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 λP that will be discussed in Section 4.) 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 $\mathrm{PRED}^\perp$ to $\lambda\mathrm{P}^\perp$, where $\mathrm{PRED}^\perp$ is PRED, defined in 3.1, extended with a constant

$\bot \,\epsilon\,$ Prop and the 'ex falso sequitur quodlibet' rule, saying that $\bot{\to}\varphi$ is always inhabited. The system $\lambda\mathrm{P}^{\bot}$ is λP extended with a constant type $\bot$: **type** and a constant term $\mathcal{E}_{\bot}$ with an extra rule

$$\frac{\Gamma \vdash M : \bot \quad \Gamma \vdash A : \mathbf{type}}{\Gamma \vdash \mathcal{E}_{\bot} M A : A}$$

The system $\mathrm{PRED}^{\bot}$ 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 $\mathrm{PRED}^{\bot}$ by a double negation translation. The embedding of classical first order predicate logic in to $\lambda\mathrm{P}^{\bot}$ via the system $\mathrm{PRED}^{\bot}$ is now complete, due to the completeness of the embedding of $\mathrm{PRED}^{\bot}$ into $\lambda\mathrm{P}^{\bot}$.

4. λP as a Logical Framework

The idea of using a first order dependent type theory as a Logical Framework originates from De Bruijn and the Automath project. (See (de Bruijn 1980), (van Daalen 1973) or (Nederpelt et al. 1994).) However, the definition of λP, roughly as it is given here, and its use as a Logical Framework, originate from (Harper et al. 1987). It is also in (Harper et al. 1987) that the issue of *adequacy of the encoding* of a logic is first raised. Given a logic L and its encoding as a λP-context Γ_L, (see 2.4 for an example) we may wonder whether this encoding is *adequate*, i.e. whether for $\Delta = \{\psi_1, \ldots, \psi_n, \varphi\}$ a set of formulas of L,

$$\text{if } \Gamma_L, p_1{:}T\psi_1, \ldots, p_n{:}T\psi_n \vdash_{\lambda\mathrm{P}} M : T\varphi, \text{ then } \Delta \vdash_L \varphi\ ?$$

Of course, *soundness* of the encoding is also an important issue, but usually that is a rather straightforward induction on the derivation in L. As a matter of fact, the soundness lies in constructing the right context Γ_L (that represents the logic L appropriately). So we assume that we have chosen the right Γ_L and that the encoding of L in Γ_L is sound:

$$\text{If } \Delta \vdash_L \varphi \text{ then } \Gamma_L, p_1{:}T\psi_1, \ldots, p_n{:}T\psi_n \vdash_{\lambda\mathrm{P}} M : T\varphi, \text{ for some } M.$$

(This M will usually be a direct encoding of the derivation of $\Delta \vdash_L \varphi$.)

4.1. *Adequacy of the LF encoding*

The way to prove adequacy of the interpretation is by using so called 'long-$\beta\eta$-normal forms'. A long-$\beta\eta$-normal form is obtained by first taking the β-normal form and then doing *η-expansion*: a term $C[M]$ in β-normal form η-expands to $C[\lambda x{:}A.Mx]$ 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 $l\beta\eta(M)$ for the long-$\beta\eta$-normal form of the term M.

So, for proving adequacy of the encoding, it is most convenient to replace the β-conversion rule in λP with a $\beta\eta$-conversion rule:

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

This is also what is done in LF: The type system of LF is λP extended with η in the conversion rule. We will not go into this extension very deeply, but just remark that adding this rule complicates the meta-theory quite a bit. Normalization is relatively easy (shown in (Harper et al. 1987)), but confluence (of $\beta\eta$-reduction) is surprisingly complicated and was first proved by (Salvesen 1989) and (Geuvers 1992).

We note that for proving adequacy of an encoding, the extension with η is not strictly required, but it does make the proof easier. (Maybe it is virtually the only way to prove it.) The argument runs as follows. Suppose we have a $\beta\eta$-conversion rule, so we work in LF.

$$\begin{array}{rll} \text{If } \Gamma_L, \vec{p} : T\vec{\Delta} & \vdash_{LF} & M : T\varphi, \\ \text{then } \Gamma_L, \vec{p} : T(\overrightarrow{l\beta\eta(\Delta)}) & \vdash_{LF} & l\beta\eta(M) : T(l\beta\eta(\varphi)) \text{ using } =_{\beta\eta}, \\ \text{then } \Delta' & \vdash_L & \varphi' \end{array}$$

by a canonical translation from long-$\beta\eta$-nfs in the context Γ_L to L. Now, by the fact that the encoding of a formula of L as a a term of type **prop** yields a long-$\beta\eta$-nf, we are done: Δ, resp. φ are exactly the LF-encoding of Δ', resp. φ'.

Having proved adequacy of the encoding in LF, we immediately concluse that the encoding in λP is adequate as well, because $\lambda\text{P} \subset \text{LF}$. More precisely: If $\Gamma_L, \vec{p} : T\vec{\Delta} \vdash_{\lambda\text{P}} M : T\varphi$, then also $\Gamma_L, \vec{p} : T\vec{\Delta} \vdash_{LF} M : T\varphi$, and the rest of the argument runs as above.

We treat the adequacy for the encoding of minimal proposition logic (Example 2.4) in some more detail. Here we don't need the $\beta\eta$-conversion rule at all. Technically: because there are no functions of higher type in the context Γ of 2.4; in terms of the argument above: if $\Gamma \vdash_{\lambda\text{P}} A : \textbf{prop}$, then $A \equiv l\beta\eta(A)$. So, we only need that $\Gamma \vdash_{\lambda\text{P}} M : A \Rightarrow \Gamma \vdash_{\lambda\text{P}} l\beta\eta(M) : A$, which is easily proved in λP.

Now, consider the context Γ of 2.4. The proof of adequacy of the encoding proceeds in three steps.

(1) If $\Gamma, \vec{x} : \textbf{prop} \vdash_{\lambda\text{P}} A : \textbf{prop}$, then either one of the following two is the case.

$$\begin{array}{rll} A =_\beta \supset BC & \text{with} & \Gamma, \vec{x} : \textbf{prop} \vdash_{\lambda\text{P}} B, C : \textbf{prop}, \\ A =_\beta x & \text{with} & x{:}\textbf{prop} \text{ in the context..} \end{array}$$

(2) If $\Gamma, \vec{x} : \textbf{prop}, \vec{p} : T(\vec{A}) \vdash_{\lambda\text{P}} t : TB$ (with $\vec{A} : \textbf{prop}$ and $B : \textbf{prop}$), then either one of the following three is the case.

$$\begin{array}{rll} t =_\beta p & \text{with} & B =_\beta A, p{:}A \text{ in the context, for some } A, \\ t =_\beta \supset_E CDqr & \text{with} & \Gamma, \vec{x} : \textbf{prop}, \vec{p} : T(\vec{A}) \vdash_{\lambda\text{P}} q : T(\supset CD), r : TC \\ & & \text{and } B =_\beta D, \\ t =_{\beta\eta} \supset_I CD(\lambda z{:}TC.q) & \text{with} & \Gamma, \vec{x} : \textbf{prop}, \vec{p} : T(\vec{A}), z{:}TC \vdash_{\lambda\text{P}} q : TD \\ & & \text{and } B =_\beta \supset CD. \end{array}$$

(3) From the long-$\beta\eta$-nf of a term t with $\Gamma, \vec{x} : \textbf{prop}, \vec{p} : T(\vec{A}) \vdash_{\lambda\text{P}} t : TB$ (as in (2)) one can inductively define a derivation of the associated proposition in minimal first order proposition logic.

So there is an isomorphism between $\beta\eta$-equivalence classes of terms of type TA in Γ and derivations of A in the logic. The isomorphism is defined on the long-$\beta\eta$-normal forms, which form a complete set of representants for the $\beta\eta$-equivalence classes.

4.2. *A minimal version of* λP

Although the number of rules is limited, λP (or LF) is very powerful in interpreting a wide variety of formal systems. (See (Harper et al. 1987) or (Avron et al. 1987) for examples.) It is however not minimal yet: we can do without part of the (λ)-rule without weakening the power of the system. This is partly due to the way in which the system is being used. Once the context Γ_L that represents the formal system has been established, one is only interested in judgements of the form

$$\Gamma_L \vdash_{\lambda\mathrm{P}} M : A, \text{ with } A : \mathbf{type}$$

On the other hand there is no reason to let the context Γ_L 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 $\Pi x{:}A.B : \mathbf{kind}$. That is: *everything that can be encoded in* λP *can already be encoded in* $\lambda\mathrm{P}^-$. See Definition 2.12.

We 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).

Lemma 4.1. If $\Gamma \vdash_{\lambda\mathrm{P}} M : \mathbf{type}$ or $\Gamma \vdash_{\lambda\mathrm{P}} M : A(: \mathbf{type})$, then β_P-nf(M) contains no λ_P.

Proof. Suppose that M is in β_P-nf and that $\Gamma \vdash_{\lambda\mathrm{P}} M : \mathbf{type}$ or $\Gamma \vdash_{\lambda\mathrm{P}} M : A(: \mathbf{type})$. We prove by induction on the structure of M that it contains no λ_P. Note that $M \equiv \lambda_P x{:}B.N$ is not possible, because then $M : A : \mathbf{kind}$.

$M \equiv x\vec{t}$ Then for all i, $\Gamma \vdash t_i : B_i(: \mathbf{type})$ and t_i in β_P-nf, so by induction hypothesis, t_i contains no λ_P and we are done.

$M \equiv \lambda_0 x{:}B.N$ Then $\Gamma, x{:}B \vdash N : C(: \mathbf{type})$ and $\Gamma \vdash B : \mathbf{type}$. Both are in β_P-nf, so we are done by using the induction hypothesis.

$M \equiv \Pi x{:}B.C$ Then $\Gamma, x{:}B \vdash C : \mathbf{type}$ and $\Gamma \vdash B : \mathbf{type}$. Both are in β_P-nf, so we are done by using the induction hypothesis.

□

Proposition 4.2. If $\Gamma \vdash M : A$, Γ and M are in β_P-nf and contain no λ_P, then $\Gamma \vdash^- M : B$ for some $B =_\beta A$.

Proof. By induction on the length of $\Gamma + M$.

$M \equiv x$ Then $\Gamma \equiv \Gamma_1, x{:}B, \Gamma_2$ with $A =_\beta B$. Now $\Gamma_1 \vdash B : \mathbf{type/kind}$ and the induction hypothesis applies, so $\Gamma \vdash^- B : \mathbf{type/kind}$, so $\Gamma_1, x{:}B \vdash^- x : B$. Similarly for the declarations in Γ_2, so we find $\Gamma \vdash^- x : B$.

$M \equiv \lambda_0 x{:}C.N$ Then $\Gamma, x{:}C \vdash N : D$, so the induction hypothesis applies and we find that $\Gamma, x{:}C \vdash^- N : D'$ for some $D =_\beta D'$. Now it must be the case that $\Gamma \vdash^- C : \mathbf{type}$ and $\Gamma, x{:}C \vdash^- D' : \mathbf{type}$, so $\Gamma \vdash^- \lambda_0 x{:}C.N : \Pi x{:}C.D'$.

$M \equiv PN$ Then $\Gamma \vdash P : \Pi x{:}C.D$ and $\Gamma \vdash N : C$, so by induction hypothesis we find that $\Gamma \vdash^- P : \Pi x{:}C'.D'$ and $\Gamma \vdash^- N : C''$ with $C =_\beta C' =_\beta C''$. But then $\Gamma \vdash^- PN : D'[N/x]$ and $D'[N/x] =_\beta A$.

$M \equiv \Pi x{:}C.N$ Easy.

□

Corollary 4.3. If $\Gamma \vdash M : A(: \textbf{type})$, all in β_P-normal form, then $\Gamma \vdash^- M : A(: \textbf{type})$.

Proof. Suppose $\Gamma \vdash M : A$ with $\Gamma \vdash A : \textbf{type}$, all in β_P-normal-form. Then Γ, M and A do not contain any λ_P (Lemma 4.1), so we can apply the Proposition and find a $B =_\beta A$ such that $\Gamma \vdash^- M : B$. Moreover, $\Gamma \vdash^- A : \textbf{type}$, so $\Gamma \vdash^- M : A$ by the conversion rule.

□

Now, if Γ is a λP 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 λP, 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 $\Pi x{:}A.B : \textbf{kind}$ 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 Automath systems and λP (certainly for logical systems) is the declaration of $T : \mathrm{prop}{\to}\textbf{type}$, where $\mathrm{prop} : \textbf{type}$ is another declaration.

We could even be more ‘minimal’ and not allow function types of the form $\Pi x_1{:}A_1.\ldots\Pi x_n{:}A_n.\textbf{type}$, but instead add rules for ‘parametric constants’:

$$\frac{\Gamma, x_1{:}A_1, \ldots, x_n{:}A_n \vdash}{\Gamma, c{:}\Pi(x_1{:}A_1, \ldots, x_n{:}A_n).\textbf{type} \vdash c : \Pi(x_1{:}A_1, \ldots, x_n{:}A_n).\textbf{type}}$$

$$\frac{\Gamma \vdash c : \Pi(x_1{:}A_1, \ldots, x_n{:}A_n).\textbf{type} \quad \Gamma \vdash t_i : A_i[t_1/x_1]\ldots[t_{i-1}/x_{i-1}]\ (\forall i \leq n)}{\Gamma \vdash c\vec{t} : \textbf{type}}$$

Then a declared constant of type $\Pi x_1{:}A_1.\ldots\Pi x_n{:}A_n.\textbf{type}$ can only be used in its ‘fully applied’ form, i.e. by applying it to n values.

5. Concluding Remarks

We have studied first order dependent typed λ-calculus from two perspectives: first as a logical systems itself (via the formulas-as-types embedding) and second as a framework for defining logical systems. From the first perspective, we have proved the completeness of the formulas-as-types embedding of minimal first order predicate logic into λP, which turned out to be remarkably intricate. It is known that this result does not extend to the embedding of higher order predicate into the the Calculus of Constructions (see (Berardi 1990) and (Geuvers 1993)): completeness fails for the third order case and higher; for the second order case, the question of completeness is still open.

We have also seen that the rule that allows to λ-abstract over a type ($A : \textbf{type}$) to create a term of a kind ($B : \textbf{kind}$), does not contribute to the power of the system λP. (This is defined as *λ_P-abstraction* in Definition 2.12.) If we look at the formulas-as-types embedding, this addition is a conservative extension: see Corollary 3.37. If we look at λP as a logical framework, Corollary 4.3 shows that λ_P-abstraction is superfluous.

References

A. Avron, F. Honsell and I. Mason, Using typed lambda calculus to implement formal systems on a machine, Report 87-31, LFCS Edingurgh, UK.

H.P. Barendregt, Typed lambda calculi. In *Handbook of Logic in Computer Science*, eds. Abramski et al., Oxford Univ. Press.

E. Barendsen and H. Geuvers, λP is conservative over first order predicate logic, Manuscript, Faculty of Mathematics and Computer Science, University of Nijmegen, Netherlands,

S. Berardi, Type dependence and constructive mathematics, Ph.D. thesis, Universita di Torino, Italy.

M. Bezem and J. Springintveld, A simple proof of the undecidability of inhabitation in λP. *Journal of Functional Programming*, vol 6 (5), pp. 757–761.

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.

Th. Coquand, An algorithm for testing conversion in type theory. In Huet and Plotkin (eds.), *Logical Frameworks*, Cambridge Univ. Press.

Th. Coquand and G. Huet, The calculus of constructions, *Information and Computation*, 76, pp 95-120.

Th. Coquand and G. Huet, Constructions: a higher order proof system for mechanizing mathematics. *Proceedings of EUROCAL '85, Linz*, LNCS 203.

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.*

G. Dowek, The undecidability of typability in the $\lambda\Pi$-calculus, M. Bezem, J.F. Groote (Eds.), *Typed Lambda calculi and applications*, LNCS 664, Springer-Verlag (1993), pp. 139–145.

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.

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.

J.H. Geuvers, *Logics and Type systems*, PhD. Thesis, University of Nijmegen, Netherlands.

J.H. Geuvers, A short and flexible proof of Strong Normalization for the Calculus of Constructions, in *Types for Proofs and Programs*, Int. Workshop, Bastad, Sweden, 1994, Eds. P. Dybjer, B. Nordström and J. Smith, LNCS 996, Springer, pp. 14–38.

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.

R. Harper, F. Honsell and G. Plotkin, A framework for defining logics, *Journal of the ACM*, pp 143–184.

J.R. Hindley and J.P. Seldin *Introduction to Combinators and λ-Calculus*, London Math. Soc. Student Texts 1, Cambridge University Press, 1986.

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.

P. Martin-Löf, An intuitionistic theory of types: predicative part, in *Logic Colloquium 1973*, eds. H.E. Rose et al., North-Holland, 1975, pp 73–118.

Ch. Mohring, Algorithm development in the Calculus of Constructions, in *Proceedings First Symposium on Logic in Computer Science, (Cambridge, Mass.)* , 1986, IEEE, Washington DC, pp. 84–91.

R.P. Nederpelt, J.H. Geuvers and R.C. de Vrijer (editors), *Selected Papers on Automath*, Volume 133 in Studies in Logic and the Foundations of Mathematics, North-Holland, Amsterdam, 1994, pp 1024.

A. Salvesen, The Church-Rosser Theorem for LF with η reduction. Notes of a talk presented at the BRA-Logical Frameworks meeting, Antibes 1990.

M. Swaen, Weak and strong sum-elimination in intuitionistic type theory, Ph.D. thesis, Faculty of Mathematics and Computer Science, University of Amsterdam, Netherlands, September 1989.