Annals of Pure and Applied Logic 160 (2009) 1–12

Contents lists available at ScienceDirect

Annals of Pure and Applied Logic

journal homepage: www.elsevier.com/locate/apal

On the strength of dependent products in the type theory of Martin-Löf

Richard Garner

Department of Mathematics, Uppsala University, Box 480, S-751 06 Uppsala, Sweden

ARTICLE INFO

Article history:
Received 31 March 2008
Received in revised form 1 October 2008
Accepted 2 December 2008
Available online 1 February 2009
Communicated by I. Moerdijk

MSC:
primary 03B15

Keywords:
Dependent type theory
Dependent products
Function extensionality

ABSTRACT

One may formulate the dependent product types of Martin-Löf type theory either in terms of abstraction and application operators like those for the lambda-calculus; or in terms of introduction and elimination rules like those for the other constructors of type theory. It is known that the latter rules are at least as strong as the former: we show that they are in fact strictly stronger. We also show, in the presence of the identity types, that the elimination rule for dependent products – which is a "higher-order" inference rule in the sense of Schroeder-Heister – can be reformulated in a first-order manner. Finally, we consider the principle of function extensionality in type theory, which asserts that two elements of a dependent product type which are pointwise propositionally equal, are themselves propositionally equal. We demonstrate that the usual formulation of this principle fails to verify a number of very natural propositional equalities; and suggest an alternative formulation which rectifies this deficiency.

1. Introduction

This is the first in a series of papers recording the author's investigations into the semantics of Martin-Löf's dependent type theory; by which we mean the type theory set out in the expository volume [9]. The main body of these investigations concerns what the author is calling *two-dimensional* models of dependent type theory. Recall that one typically divides the models of Martin-Löf's type theory into *extensional* and *intensional* ones, the former differentiating themselves from the latter by their admission of an equality reflection rule which collapses the propositional and definitional equalities of the language into a single, *judgemental*, equality. The two-dimensional models that the author is studying are of the intensional kind, but are not wholly intensional: they admit instances of the equality reflection rule at just those types which are themselves identity types.

In the process of making his investigations, the author has discovered certain unresolved issues concerning the dependent product types of Martin-Löf type theory; and since these issues exist beyond the domain of two-dimensional models, it seemed worthwhile to collect his conclusions into this preliminary paper.

The first of these issues concerns how we formulate the rules for the dependent product types. There are two accepted ways of doing this. In both cases, we begin with a *formation* rule which, given a type A and a type $B(x)$ dependent on $x : A$, asserts the existence of a type $\Pi(A, B)$; and an *abstraction* rule which says that, from an element $f(x) : B(x)$ dependent on $x : A$, we may deduce the existence of an element $\lambda(f) : \Pi(A, B)$. We may then complement these rules with either an *application* rule, which tells us that, from $m : \Pi(A, B)$ and $a : A$, we may infer an element $\mathsf{app}(m, x) : B(x)$; or an *elimination* rule, which essentially tells us that any (dependent) function out of $\Pi(A, B)$ is determined, up-to-propositional-equality, by its values on those elements of the form $\lambda(f)$ for some dependent element $x : A \vdash f(x) : B(x)$.

There are two problematic features here. The first concerns the nature of the elimination rule, which is a *higher-order inference rule* in the sense of Schroeder-Heister [10]. In order to formulate this rule rigorously, we must situate our type

E-mail address: rhgg2@cam.ac.uk.

0168-0072/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.apal.2008.12.003

theory within an ambient calculus possessing higher-order features; a suitable choice being the *Logical Framework* described in Part III of [9], and recalled in Section 2 below. Yet it may be that we do not wish to do this: one reason being that the categorical semantics of Martin-Löf type theory looks rather different when it is formulated within the Logical Framework. Hence our first task in this paper is to give a first-order reformulation of the elimination rule in terms of the application rule and a propositional form of the η-rule; a reformulation that may be stated without recourse to the Logical Framework.

The second problematic feature concerns the precise relationship between the application and elimination rules for dependent products. We know that the application rule may be defined in terms of the elimination rule, so that the elimination rule is stronger; yet it is not known whether it is *strictly* stronger. Our second task is to show that this is in fact the case; we do this by describing a non-standard interpretation of the Π-types for which the application rule obtains, yet not the elimination rule.

We then move on to another issue, namely the formulation of the principle of *function extensionality* in Martin-Löf type theory. This principle asserts that if m and n are elements of $\Pi(A, B)$ and we can affirm a propositional equality between $\mathsf{app}(m, x)$ and $\mathsf{app}(n, x)$ whenever $x : A$, then we may deduce the existence of a propositional equality between m and n. One result of the author's investigations into two-dimensional models has been that, if we are to obtain a notion of model which is reasonably urbane from a category-theoretic perspective, then we must impose some kind of function extensionality. Yet the principle of function extensionality just stated has been found wanting in this regard, since it fails to provide witnesses for a number of very natural propositional equalities which are demanded by the semantics; some of which are detailed in Examples 5.6 below. From a category-theoretic perspective, we might say that the principle of function extensionality fails to be *coherent*. Our third task in this paper, therefore, is to propose a suitably coherent replacement for function extensionality.

2. Martin-Löf type theory

2.1 We begin with a brief summary of the two principal ways in which one may present Martin-Löf type theory. The more straightforward is the "polymorphic" presentation of [7,8]. This is given by a reasoning system with four basic forms of judgement:

- $\Gamma \vdash A\ \mathsf{type}$ ("A is a type under the hypothesis Γ");
- $\Gamma \vdash a : A$ ("a is an element of A under the hypothesis Γ");
- $\Gamma \vdash A = B\ \mathsf{type}$ ("A and B are equal types under the hypothesis Γ");
- $\Gamma \vdash a = b : A$ ("a and b are equal elements of A under the hypothesis Γ").

Here, Γ is to be a *context* of assumptions, $\Gamma = (x_1 : A_1,\ x_2 : A_2,\ \ldots,\ x_n : A_n)$, subject to a requirement of well-formedness which affirms that each A_i is a type under the assumptions $(x_1 : A_1,\ \ldots,\ x_{i-1} : A_{i-1})$. The polymorphic presentation of Martin-Löf type theory is now given by specifying a sequent calculus over these four forms of judgement: so a number of *axiom* judgements, together with a number of *inference rules*

$$\frac{\mathcal{J}_1 \qquad \cdots \qquad \mathcal{J}_n}{\mathcal{J}}$$

allowing us to derive the validity of the judgement $\mathcal{J}$ from that of the $\mathcal{J}_i$'s. As usual, these inference rules separate into a group of *structural rules* which deal with the contextual book-keeping of weakening, contraction, exchange and substitution; and a group of *logical rules*, which describe the constructions we wish to be able to perform inside our type theory: constructions such as cartesian product of types, disjoint union of types, or formation of identity types.

2.2 However, the polymorphic presentation of type theory is inadequate for our purposes, because the elimination rule for dependent products we wish to study requires the use of a *second-order judgement* $\Gamma \vdash \mathcal{J}$, in which the context of assumptions Γ itself contains a judgement under hypotheses. One solution to this problem is suggested by Troelstra and van Dalen in [11, Chapter 11]: we extend our system with explicit second-order judgement forms expressing that "B is a family of types over A under the hypothesis Γ", and so on, and express the elimination rule in terms of these. A second solution to this problem – and the one we adopt here – makes use of the "monomorphic" presentation of Martin-Löf type theory. This is given in terms of the *Logical Framework*, which is essentially a formalisation of the meta-theory we use to reason about the calculus of types. The basic judgements of this meta-theory look rather like those of type theory:

$$\Gamma \vdash A\ \mathsf{sort}; \qquad \Gamma \vdash a : A; \qquad \Gamma \vdash A = B\ \mathsf{sort}; \quad \text{and} \quad \Gamma \vdash a = b : A.$$

However, the meaning is somewhat different. We think of a sort of the Logical Framework as being a category of *judgements about type theory*. In particular, the Logical Framework has rules

$$\frac{}{\vdash \mathsf{type}\ \mathsf{sort}} \quad \text{and} \quad \frac{}{A : \mathsf{type} \vdash \mathsf{el}\, A\ \mathsf{sort}},$$

which express the existence of the category of judgements "— is a type"; and, under the assumption that "A is a type", of the category of judgements "— is an element of A". Using these, we may interpret more complex judgements of type theory;

for example, if we know that "A is a type", then we can interpret the judgement $\mathcal{J}$ that "$B(x)$ is a type under the hypothesis that x is an element of A" as

$$x : \mathrm{el}\,A \vdash B(x) : \mathrm{type}.$$

Yet this is not an entirely faithful rendition of $\mathcal{J}$, since strictly speaking, the displayed sequent asserts the judgement "$B(x)$ is a type" under the hypothesis that "x is an element of A". To resolve this, we introduce the other key aspect of the Logical Framework, namely the *function sorts*. These are specified by rules of formation, abstraction and application:

$$\frac{\Gamma,\ x : A \vdash B(x)\ \mathrm{sort}}{\Gamma \vdash (x : A)\,B\ \mathrm{sort}}, \qquad \frac{\Gamma,\ x : A \vdash b(x) : B(x)}{\Gamma \vdash [x : A]\,b(x) : (x : A)\,B(x)}$$

$$\text{and} \qquad \frac{\Gamma \vdash f : (x : A)\,B \qquad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B(a)}$$

subject to the α-, β-, η- and ξ-rules of the lambda-calculus. Using function sorts, we can now render the judgement $\mathcal{J}$ more correctly. We have the sort $(x : \mathrm{el}\,A)$ type, which is the category of judgements "— is a type under the hypothesis that x is an element of A"; and can now interpret $\mathcal{J}$ as the judgement

$$\vdash B : (x : \mathrm{el}\,A)\ \mathrm{type}.$$

2.3 We may translate the polymorphic presentation of Martin-Löf type theory into the monomorphic one by encoding the inference rules of the former as higher-order judgements of the latter. For instance, consider the hypothetical type constructor Φ with rules

$$\frac{A\ \mathrm{type}}{\Phi(A)\ \mathrm{type}} \quad \text{and} \quad \frac{A\ \mathrm{type} \qquad a : A}{\phi_A(a) : \Phi(A)}.$$

We may encode this in the Logical Framework by terms

$$\vdash \Phi : (A : \mathrm{type})\ \mathrm{type} \quad \text{and} \quad \vdash \phi : (A : \mathrm{type},\ a : \mathrm{el}\,A)\ \mathrm{el}\,\Phi(A),$$

where for the sake of readability we write iterated function spaces as $(A : \mathrm{type},\ a : \mathrm{el}\,A)\ \mathrm{el}\,\Phi(A)$ instead of the more correct $(A : \mathrm{type})(a : \mathrm{el}\,A)\ \mathrm{el}\,\Phi(A)$. Note that this encoding says more than the original, by affirming a certain insensitivity to ambient context; since from the constants Φ and ϕ, we obtain a whole family of inference rules

$$\frac{\Gamma \vdash A\ \mathrm{type}}{\Gamma \vdash \Phi(A)\ \mathrm{type}} \quad \text{and} \quad \frac{\Gamma \vdash A\ \mathrm{type} \qquad \Gamma \vdash a : A}{\Gamma \vdash \phi_A(a) : \Phi(A)},$$

together with further rules expressing stability under substitution in Γ. However, this is no bad thing, since any acceptable inference rule of the polymorphic theory will possess this "naturality" in the context Γ. In the remainder of this paper we work in the monomorphic presentation of type theory, but will take advantage of the above encoding process in order to present the rules of our type theory in the more readable polymorphic style. For more on the relationship between the monomorphic and polymorphic presentations, see [5].

3. A first-order reformulation of the Π-elimination rule

3.1 Our main concern in this paper is with the dependent product types of Martin-Löf type theory: but in this analysis, we will from time to time make use of the *identity types*, which are a reflection at the type level of the equality judgements $a = b : A$. We begin, therefore, by recalling the rules for the identity types:

$$\frac{A\ \mathrm{type} \qquad a, b : A}{\mathrm{Id}_A(a, b)\ \mathrm{type}}\ \textsc{Id-form}; \qquad \frac{A\ \mathrm{type} \qquad a : A}{r(a) : \mathrm{Id}_A(a, a)}\ \textsc{Id-intro};$$

$$\frac{\begin{array}{c}A\ \mathrm{type} \qquad x, y : A,\ z : \mathrm{Id}_A(x, y) \vdash C(x, y, z)\ \mathrm{type} \\ x : A \vdash d(x) : C(x, x, r(x)) \qquad a, b : A \qquad p : \mathrm{Id}_A(a, b)\end{array}}{J(d, a, b, p) : C(a, b, p)}\ \textsc{Id-elim};$$

$$\frac{\begin{array}{c}A\ \mathrm{type} \qquad x, y : A,\ z : \mathrm{Id}_A(x, y) \vdash C(x, y, z)\ \mathrm{type} \\ x : A \vdash d(x) : C(x, x, r(x)) \qquad a : A\end{array}}{J(d, a, a, r(a)) = d(a) : C(a, a, r(a))}\ \textsc{Id-comp}.$$

The notion of equality captured by the identity types is known as *propositional equality*: to say that a and b are propositionally equal as elements of A is to say that we may affirm a judgement $p : \mathrm{Id}_A(a, b)$. We think of Id_A as being a type inductively generated by the elements $r(a) : \mathsf{Id}_A(a, a)$, with the elimination rule and computation rules expressing that any dependent function out of Id_A is determined up-to-propositional-equality by its value on elements of the form $r(a)$.

3.2 We are now ready to describe the two standard formulations of dependent product types in Martin-Löf type theory. The first, which we will refer to as the app-formulation, is analogous to the lambda-calculus with the β-rule but no η-rule:

$$\frac{A\ \mathsf{type} \qquad x : A \vdash B(x)\ \mathsf{type}}{\Pi(A, B)\ \mathsf{type}}\ \Pi\text{-FORM}; \qquad \frac{x : A \vdash f(x) : B(x)}{\lambda(f) : \Pi(A, B)}\ \Pi\text{-ABS};$$

$$\frac{m : \Pi(A, B) \qquad a : A}{\mathsf{app}(m, a) : B(a)}\ \Pi\text{-APP}; \qquad \frac{x : A \vdash f(x) : B(x) \qquad a : A}{\mathsf{app}(\lambda(f), a) = f(a) : B(a)}\ \Pi\text{-}\beta.$$

Note that, for the sake of readability we omit the hypotheses A type and $x : A \vdash B(x)$ type from the last three of these rules; and in future, we may omit any such hypotheses that are reconstructible from the context. To further reduce syntactic clutter, we may also write $\Pi x : A.\, B(x)$ instead of $\Pi(A, [x : A]\, B(x))$; write $\lambda x.\, f(x)$ instead of $\lambda([x : A]\, f(x))$; and write $m \cdot a$ instead of $\mathsf{app}(m, a)$.

3.3 As we noted in the Introduction, the second formulation of dependent products – which we will refer to as the funsplit-formulation – has the same introduction and abstraction rules but replaces the application and β-rules with elimination and computation rules which mirror those for the other constructors of type theory: they assert that each type $\Pi(A, B)$ is inductively generated by the elements of the form $\lambda(f)$.

$$\frac{\begin{array}{c} y : \Pi(A, B) \vdash C(y)\ \mathsf{type} \\ f : (x : A)\, B(x) \vdash d(f) : C(\lambda(f)) \qquad m : \Pi(A, B) \end{array}}{\mathsf{funsplit}(d, m) : C(m)}\ \Pi\text{-ELIM};$$

$$\frac{\begin{array}{c} y : \Pi(A, B) \vdash C(y)\ \mathsf{type} \\ f : (x : A)\, B(x) \vdash d(f) : C(\lambda(f)) \qquad x : A \vdash g(x) : B(x) \end{array}}{\mathsf{funsplit}(d,\ \lambda(g)) = d(g) : C(\lambda(g))}\ \Pi\text{-COMP}.$$

3.4 Observe that the assumption $f : (x : A)\, B(x) \vdash d(f) : C(\lambda(f))$ makes the funsplit rules into higher-order inference rules, which as such are inexpressible in the "polymorphic" formulation of type theory. Our task in the remainder of this section will be to reformulate these rules in a first-order fashion. Our treatment is a generalisation of that given by Martin-Löf in his introduction to [7], with the major difference that we are working in the theory with *intensional* identity types, as opposed to the *extensional* equality types of [7].

Proposition 3.5 (*Cf. [9, p. 52]*). *In the presence of the rules* Π-FORM, Π-INTRO, Π-ELIM *and* Π-COMP, *the rules* Π-APP *and* Π-β *are definable.*

Proof. Suppose that $m : \Pi(A, B)$ and $a : A$. We define a type $y : \Pi(A, B) \vdash C(y)$ type by taking $C(y) := B(a)$; and a term $f : (x : A)\, B(x) \vdash d(f) : C(\lambda(f))$ by taking $d(f) := f(a)$. Applying Π-elimination, we define $\mathsf{app}(m, a) := \mathsf{funsplit}(d, m) : B(a)$. Moreover, when $m = \lambda(f)$ we have by Π-COMP that $\mathsf{app}(\lambda(f), a) = d(f) = f(a)$, which gives us Π-$\beta$ as required. □

Proposition 3.6 (*Cf. [9, p. 62]*). *In the presence of the identity types and the rules* Π-FORM, Π-INTRO, Π-ELIM *and* Π-COMP, *the following rules are definable:*

$$\frac{m : \Pi(A, B)}{\eta(m) : \mathsf{Id}_{\Pi(A,B)}(m,\ \lambda x.\, m \cdot x)}\ \Pi\text{-PROP-}\eta;$$

$$\frac{x : A \vdash f(x) : B(x)}{\eta(\lambda(f)) = r(\lambda(f)) : \mathsf{Id}_{\Pi(A,B)}(\lambda(f),\ \lambda(f))}\ \Pi\text{-PROP-}\eta\text{-COMP}.$$

Proof. Given $y : \Pi(A, B)$, we define a type $C(y) := \mathsf{Id}_{\Pi(A,B)}(y,\ \lambda x.\, y \cdot x)$. In the case where $y = \lambda(f)$ for some $f : (x : A)\, B(x)$, we have $C(y) = \mathsf{Id}_{\Pi(A,B)}(\lambda(f),\ \lambda x.\, \lambda(f) \cdot x) = \mathsf{Id}_{\Pi(A,B)}(\lambda(f),\ \lambda x.\, f(x)) = \mathsf{Id}_{\Pi(A,B)}(\lambda(f),\ \lambda(f))$ so that we may define an element $d(f) : C(\lambda(f))$ by $d(f) := r(\lambda(f))$. Using Π-elimination, we define $\eta(m) := \mathsf{funsplit}(d, m)$; and when $m = \lambda(f)$, we have by Π-COMP that $\eta(\lambda(f)) = d(f) = r(\lambda(f))$ as required. □

Proposition 3.7. *In the presence of identity types, the rules* Π-FORM, Π-INTRO, Π-APP *and* Π-β, *and the rules* Π-PROP-η *and* Π-PROP-η-COMP *of Proposition 3.6, the rules* Π-ELIM *and* Π-COMP *are definable.*

Proof. We first recall that in the presence of identity types, Π-FORM, Π-INTRO, Π-APP and Π-β, we may derive the following "Leibniz rules", which assuming A type and $x : A \vdash B(x)$ type, say that

$$\frac{a_1, a_2 : A \qquad p : \mathsf{Id}_A(a_1, a_2) \qquad b_2 : B(a_2)}{\mathsf{subst}(p, b_2) : B(a_1)} \ \mathsf{Id}\text{-SUBST};$$

$$\frac{a : A \qquad b : B(a)}{\mathsf{subst}(r(a), b) = b : B(a)} \ \mathsf{Id}\text{-SUBST-COMP};$$

see [9, p. 59], for example.
So, suppose given judgements A type, $x : A \vdash B(x)$ type and $y : \Pi(A, B) \vdash C(y)$ type and terms $f : (x : A)\, B(x) \vdash d(f) : C(\lambda(f))$ and $m : \Pi(A, B)$. We are required to define a term $\mathsf{funsplit}(d, m) : C(m)$ satisfying $\mathsf{funsplit}(d, \lambda(f)) = d(f)$. We begin by forming the term

$$T(d, m) := d\big([x : A]\, m \cdot x\big) : C(\lambda x.\, m \cdot x).$$

Now by Π-PROP-η, we have a term $\eta(m) : \mathsf{Id}_{\Pi(A,B)}(m, \ \lambda x.\, m \cdot x)$: and so by substituting $T(d, m)$ along $\eta(m)$ we obtain a term $\mathsf{funsplit}(d, m) := \mathsf{subst}\big(\eta(m), \ T(d, m)\big) : C(m)$ as required. Moreover, when $m = \lambda(f)$, we obtain from Π-β that $T(d, \ \lambda(f)) = d(f)$, and from Π-PROP-η-COMP that $\eta(\lambda(f)) = r(\lambda(f))$; and so from Id-SUBST-COMP, we deduce that

$$\mathsf{funsplit}(d, \ \lambda(f)) = \mathsf{subst}\big(r(\lambda(f)), \ d(f)\big) = d(f) : C(\lambda(f))$$

as required. □

3.8 Thus, in the presence of identity types, the funsplit-formulation of dependent products is equivalent with the app-formulation extended with the propositional η-rule. Note carefully that this equivalence is a *propositional*, rather than *definitional* one; which is to say that if we are given a funsplit term, to which we apply Propositions 3.5 and 3.6 to obtain terms app and η, and then Proposition 3.7 to obtain a new term funsplit′, we should not expect $\mathsf{funsplit}(d, m) = \mathsf{funsplit}'(d, m)$ to hold, but rather only that

$$\frac{y : \Pi(A, B) \vdash C(y) \text{ type} \qquad f : (x : A)\, B(x) \vdash d(f) : C(\lambda(f)) \qquad m : \Pi(A, B)}{\psi(d, m) : \mathsf{Id}_{C(m)}\big(\mathsf{funsplit}(d, m), \mathsf{funsplit}'(d, m)\big)}$$

should be derivable. We may prove this by an application of Π-ELIM.

4. Π-application does not entail Π-elimination

4.1 We saw in Proposition 3.5 that the funsplit-formulation of dependent products subsumes the app-formulation; and the task of this section is to show that the converse does not obtain. In the previous section we were proving a positive derivability result, and so worked in a minimal fragment of Martin-Löf type theory in order to make our result as strong as possible. In this section, we are proving a negative derivability result: and to make this as strong as possible, we work in full Martin-Löf type theory. So in addition to identity types and the app-formulation of dependent products we assume the presence of dependent sums $\Sigma x : A.\, B(x)$, the unit type 1, pairwise disjoint unions $A + B$, the empty type 0, the W-types, and the first universe U. We refer to the type theory with these constructors as $\mathbf{ML}_{\mathsf{app}}$. Our main result will be:

Theorem 4.2. *Relative to the theory* $\mathbf{ML}_{\mathsf{app}}$*, the* funsplit *rules* Π-ELIM *and* Π-COMP *are not definable.*

Now, if we could define Π-ELIM and Π-COMP relative to $\mathbf{ML}_{\mathsf{app}}$, then by Proposition 3.6 we would also be able to derive Π-PROP-η and Π-PROP-η-COMP. Consequently, we may prove Theorem 4.2 by proving:

Theorem 4.2′. *Relative to the theory* $\mathbf{ML}_{\mathsf{app}}$*, the rules* Π-PROP-η *and* Π-PROP-η-COMP *of Proposition 3.6 are not definable.*

4.3 Our method of proving Theorem 4.2′ will be as follows. We first define the following rules relative to the theory $\mathbf{ML}_{\mathsf{app}}$:

$$\frac{A \text{ type} \qquad x : A \vdash B(x) \text{ type}}{\Pi'(A, B) \text{ type}} \ \Pi'\text{-FORM}; \qquad \frac{x : A \vdash f(x) : B(x)}{\lambda'(f) : \Pi'(A, B)} \ \Pi'\text{-ABS};$$

$$\frac{m : \Pi(A, B) \qquad a : A}{\mathsf{app}'(m, a) : B(a)} \ \Pi'\text{-APP}; \qquad \frac{x : A \vdash f(x) : B(x) \qquad a : A}{\mathsf{app}'(\lambda'(f), a) = f(a) : B(a)} \ \Pi'\text{-}\beta.$$

We then show that the corresponding rule Π'-PROP-η is not definable; and from this we deduce that the rule Π-PROP-η cannot be definable either, since if it were then by replacing each Π, λ or app in its derivation with a Π', λ' or app′, we would obtain a derivation of Π'-PROP-η, which would give a contradiction.

4.4 In order to define Π', λ' and app', we will make use of the *disjoint union* types. Given types A and B, their disjoint union is a type $A+B$ with the following introduction and elimination rules:

$$\frac{a : A}{\amalg_1(a) : A+B}\ +\text{-INTRO}_1; \qquad \frac{b : B}{\amalg_2(b) : A+B}\ +\text{-INTRO}_2;$$

$$\frac{\begin{array}{c} z : A+B \vdash C(z)\ \mathsf{type} \\ x : A \vdash f(x) : C(\amalg_1(x)) \qquad y : B \vdash g(y) : C(\amalg_2(y)) \qquad c : A+B \end{array}}{\mathsf{case}(f, g, c) : C(c)}\ +\text{-ELIM},$$

subject to the computation rules $\mathsf{case}\big(f, g, \amalg_1(a)\big) = f(a)$ and $\mathsf{case}\big(f, g, \amalg_2(b)\big) = g(b)$. We use disjoint unions to define the Π'-rules as follows.

$$\frac{A\ \mathsf{type} \qquad x : A \vdash B(x)\ \mathsf{type}}{\Pi'(A, B) := \Pi(A, B) + \Pi(A, B)\ \mathsf{type}}\ \Pi'\text{-FORM};$$

$$\frac{x : A \vdash f(x) : B(x)}{\lambda'(f) := \amalg_1(\lambda(f)) : \Pi(A, B) + \Pi(A, B)}\ \Pi'\text{-ABS};$$

$$\frac{m : \Pi(A, B) + \Pi(A, B) \qquad a : A}{\mathsf{app}'(m, a) := \mathsf{case}(\mathsf{app}(-, a),\ \mathsf{app}(-, a),\ m) : B(a)}\ \Pi'\text{-APP}$$

where we write $\mathsf{app}(-, a)$ as an abbreviation for the term $[x : \Pi(A, B)]\,\mathsf{app}(x, a)$. To see that these definitions validate Π'-β, we suppose that $f : (x : A)\, B(x)$ and $a : A$; then by the first computation rule for disjoint unions and Π-β we have that

$$\mathsf{app}'(\lambda'(f), a) = \mathsf{case}\big(\mathsf{app}(-, a),\ \mathsf{app}(-, a),\ \amalg_1(\lambda(f))\big) = \mathsf{app}(\lambda(f), a) = f(a)$$

as required.

4.5 It remains to show that with respect to the above definitions, the rule

$$\frac{m : \Pi'(A, B)}{\eta'(m) : \mathsf{Id}_{\Pi'(A,B)}\big(m,\ \lambda' x.\,\mathsf{app}'(m, x)\big)}\ \Pi'\text{-PROP-}\eta$$

cannot be derived. So suppose that it could. Since for each judgement $x : A \vdash f(x) : B(x)$ we have a term $\amalg_2(\lambda(f)) : \Pi'(A, B)$, we would obtain from this a derivation of

$$\frac{x : A \vdash f(x) : B(x)}{\eta'(\amalg_2(\lambda(f))) : \mathsf{Id}_{\Pi'(A,B)}\big(\amalg_2(\lambda(f)),\ \lambda' x.\,\mathsf{app}'(\amalg_2(\lambda(f)), x)\big).}$$

But now by the definition of app' we have that

$$\mathsf{app}'(\amalg_2(\lambda(f)), x) = \mathsf{case}(\mathsf{app}(-, x),\ \mathsf{app}(-, x),\ \amalg_2(\lambda(f))) = \mathsf{app}(\lambda(f), x) = f(x);$$

and hence $\lambda' x.\,\mathsf{app}'(\amalg_2(\lambda(f)), x) = \lambda' x.\, f(x) = \amalg_1(\lambda(f))$, so that we may view the above derivation as a derivation of

$$\frac{A\ \mathsf{type} \qquad x : A \vdash B(x)\ \mathsf{type} \qquad x : A \vdash f(x) : B(x)}{\eta'(\amalg_2(\lambda(f))) : \mathsf{Id}_{\Pi(A,B)+\Pi(A,B)}\big(\amalg_2(\lambda(f)),\ \amalg_1(\lambda(f))\big).} \qquad (\star)$$

To complete the proof, it suffices to show that no such derivation can exist. The key to doing so will be the following *disjointness rule*:

$$\frac{C\ \mathsf{type} \qquad c : C \qquad p : \mathsf{Id}_{C+C}\big(\amalg_2(c),\ \amalg_1(c)\big)}{\theta(c, p) : 0}, \qquad (\dagger)$$

where 0 is the empty type. If we can prove that this holds relative to $\mathbf{ML}_{\mathsf{app}}$, then we will be able to deduce the underivability of ($\star$). Indeed, suppose that ($\star$) holds. Then from this and ($\dagger$) we can derive the following rule:

$$\frac{x : A \vdash f(x) : B(x)}{\theta_{\Pi(A,B)}\big(\lambda(f),\ \eta'(\amalg_2(\lambda(f)))\big) : 0};$$

and by instantiating this derivation at some particular A, B and f – a suitable choice being $A := 1$, $B := 1$ and $f := [x : 1]\, x$ – we obtain a global element of 0. But this is impossible, because $\mathbf{ML}_{\mathsf{app}}$ is known to be *consistent*, in the sense that 0 has *no* global elements. An easy way of seeing that this is the case is by exhibiting a consistent model for $\mathbf{ML}_{\mathsf{app}}$ using the sets in our meta-theory. We interpret types as sets; dependent sums and products as indexed sums and products; identity types as meta-theoretic equality; the terminal type as a one-element set and the empty type as the empty set. The interpretation of W-types and the first universe is a little more complex, and depends upon the existence of inductive datatypes in our meta-theory, but is essentially unproblematic.

4.6 All that remains to complete the proof of Theorem 4.2′ is to show that the disjointness rule (†) is derivable in $\mathbf{ML}_{\mathsf{app}}$. This follows by a standard argument (cf. [9, p. 86]). Recall that one of the type constructors in $\mathbf{ML}_{\mathsf{app}}$ was that for the *universe type* [9, Chapter 14]. This is a type U containing "codes" for each of the other type formers of $\mathbf{ML}_{\mathsf{app}}$. In particular, we have rules

$$\frac{}{\hat{0} : \mathsf{U}}\ \text{U-INTRO}_1 \quad \text{and} \quad \frac{}{\hat{1} : \mathsf{U}}\ \text{U-INTRO}_2$$

introducing codes for the empty type and the terminal type. Recall also that U comes equipped with a *decoding function* D which is given by an indexed family of types

$$x : \mathsf{U} \vdash \mathsf{D}(x)\ \mathsf{type}$$

together with computation rules which determine the value of D on canonical elements of U. In particular, we have rules

$$\frac{}{\mathsf{D}(\hat{0}) = 0\ \mathsf{type}}\ \text{U-COMP}_1 \quad \text{and} \quad \frac{}{\mathsf{D}(\hat{1}) = 1\ \mathsf{type}}\ \text{U-COMP}_2.$$

So suppose now that C type, $c : C$ and $p : \mathsf{Id}_{C+C}\big(\amalg_2(c),\ \amalg_1(c)\big)$ as in the premisses of (†). We are required to derive an element of 0. We begin by defining functions

$$x : C \vdash f(x) := \hat{0} : \mathsf{U} \quad \text{and} \quad x : C \vdash g(x) := \hat{1} : \mathsf{U}.$$

Applying +-elimination to these we obtain a function $\mathsf{case}(f, g, -) \colon C + C \to \mathsf{U}$; and using the decoding function D on this we obtain a family

$$z : C + C \vdash T(z) := \mathsf{D}\big(\mathsf{case}(f, g, z)\big)\ \mathsf{type}.$$

Now from the rule Id-SUBST defined in Proposition 3.7, together with the given proof $p : \mathsf{Id}_{C+C}\big(\amalg_2(c),\ \amalg_1(c)\big)$ we obtain the term

$$x : T(\amalg_1(c)) \vdash \mathsf{subst}(p, x) : T(\amalg_2(c)).$$

But we have that $T(\amalg_1(c)) = \mathsf{D}(f(c)) = \mathsf{D}(\hat{0}) = 0$ and that $T(\amalg_2(c)) = \mathsf{D}(g(c)) = \mathsf{D}(\hat{1}) = 1$, so that we may view this as a function $x : 1 \vdash \mathsf{subst}(p, x) : 0$. In particular, by evaluating this function at the canonical element $\star : 1$ we obtain an element $\mathsf{subst}(p, \star) : 0$ as required. This completes the verification of the disjointness rule (†), and hence the proof of Theorem 4.2′.

5. Function extensionality

5.1 In this final section, we investigate the principle of *function extensionality* in Martin-Löf type theory, which asserts that two elements of a dependent product type which are pointwise propositionally equal, are themselves propositionally equal. Explicitly, it is given by the following two inference rules:

$$\frac{m,\ n : \Pi(A, B) \qquad k : \Pi x : A.\, \mathsf{Id}_{B(x)}(m \cdot x,\ n \cdot x)}{\mathsf{ext}(m, n, k) : \mathsf{Id}_{\Pi(A,B)}(m, n)}\ \Pi\text{-EXT};$$

$$\frac{f : (x : A)\, B(x)}{\mathsf{ext}\big(\lambda(f), \lambda(f), \lambda(rf)\big) = r(\lambda(f)) : \mathsf{Id}_{\Pi(A,B)}(\lambda(f), \lambda(f))}\ \Pi\text{-EXT-COMP}$$

where we write rf as an abbreviation for the term $[x : A]\, rf(x)$. These rules were considered first by Turner in [12] and then more extensively by Hofmann [4]. If one views Martin-Löf type theory as a *computational* system, in which terms are thought of as algorithms – an idea made precise in [9, Appendix B], for example – then these rules are hard to justify, since two extensionally equal functions can have quite different algorithmic content. From a proof-theoretic perspective they are also problematic, since they destroy one of the more pleasant properties of Martin-Löf type theory, namely that in the syntactic model, every global element of a closed type is definitionally equal to a canonical element.[1] On the other hand, they do not break strong normalisation, so that if we view type theory merely as a *computable* system – one for which the correctness of derivations is decidable – then their addition is unproblematic, and in fact produces a system which is closer to the "everyday" mathematics described by extensional type theory. Indeed, Hofmann [3] shows that augmenting intensional type theory with function extensionality and the principle of *uniqueness of identity proofs* – which asserts that any two proof-terms $p, q : \mathsf{Id}_A(a, b)$ are themselves propositionally equal – yields a system which, whilst still decidable, is propositionally equivalent (in the sense of Section 3.8) to extensional type theory.

[1] Though a construction of Altenkirch [1] shows that there are models validating both function extensionality and the canonicity property.

The author's motivations for studying the principle of function extensionality are somewhat different from those of [3]; they are informed by his investigations [2] into two-dimensional semantics for type theory. In this semantics, dependent product formation is required to be a (suitably weak) two-dimensional right adjoint to reindexing; and in order for the semantics to be complete, we must verify that the syntactic model has this property—which requires the imposition of some form of function extensionality. However, whilst preparing [2], it became apparent to the author that the usual formulation of function extensionality is insufficient for this purpose because it fails to verify a number of necessary propositional equalities between identity proofs. In the setting of [3], the existence of these propositional equalities is assured by the principle of uniqueness of identity proofs; yet for a higher-dimensional semantics it is crucial that we do *not* have uniqueness of identity proofs, whose imposition would allow only trivial, posetal, models to be formed. Thus it is of interest to determine how function extensionality should correctly be formulated when we do not have uniqueness of identity proofs; and it is this that we shall now do. We work in the fragment of type theory given by the identity types and the app-formulation of dependent products. In order to minimize clutter, we also allow ourselves the notational convenience of writing function application $f(x)$ simply as fx, and λ-abstraction $\lambda(f)$ simply as λf. We begin by recording some useful consequences of function extensionality.

Proposition 5.2. *In the presence of* Π-EXT *and* Π-EXT-COMP*, the rules* Π-PROP-η *and* Π-PROP-η-COMP *of Proposition 3.6 are definable.*

Proof. Given $m : \Pi(A, B)$, we must exhibit a term $\eta(m) : \mathsf{Id}_{\Pi(A,B)}(m,\ \lambda x.\, m \cdot x)$. So we define $n = \lambda x.\, m \cdot x : \Pi(A, B)$; and by Π-β have that $n \cdot x = (\lambda x.\, m \cdot x) \cdot x = m \cdot x$ whenever $x : A$. We may now define $\eta(m) := \mathsf{ext}(m,\ n,\ \lambda x.\, r(m \cdot x))$; and moreover, when $m = \lambda f$ for some $f : (x : A)\, B(x)$, the β-rule implies that $m = n$, so that $\eta(\lambda f) = \mathsf{ext}(\lambda f, \lambda f, \lambda(rf)) = r(\lambda f)$ as required. $\square$

Proposition 5.3. *In the presence of* Π-EXT *and* Π-EXT-COMP*, the following* propositional ξ-rules *are definable:*

$$\frac{f,\ g : (x : A)\, B(x) \qquad p : (x : A)\, \mathsf{Id}_{B(x)}(fx, gx)}{\xi(f, g, p) : \mathsf{Id}_{\Pi(A,B)}(\lambda f, \lambda g)}\ \Pi\text{-PROP-}\xi;$$

$$\frac{f : (x : A)\, B(x)}{\xi(f, f, rf) = r(\lambda f) : \mathsf{Id}_{\Pi(A,B)}(\lambda f, \lambda f)}\ \Pi\text{-PROP-}\xi\text{-COMP}.$$

Proof. Given f, g and p as in the hypotheses of Π-PROP-ξ, we consider $m = \lambda f$ and $n = \lambda g$ in $\Pi(A, B)$. By the β-rule, we may view p as a term

$$x : A \vdash p(x) : \mathsf{Id}_{B(x)}(m \cdot x,\ n \cdot x);$$

and hence may define $\xi(f, g, p) = \mathsf{ext}(\lambda f, \lambda g, \lambda p)$. Moreover, we have that $\xi(f, f, rf) = \mathsf{ext}(\lambda f, \lambda f, \lambda(rf)) = r(\lambda f)$ as required. $\square$

In fact, we have a converse to the previous two propositions:

Proposition 5.4. *In the presence of the rules* Π-PROP-η *and* Π-PROP-η-COMP *of Proposition 3.6 and the rules* Π-PROP-ξ *and* Π-PROP-ξ-COMP *of Proposition 5.3, the function extensionality rules* Π-EXT *and* Π-EXT-COMP *are definable.*

Proof. Recall from [6] that, in the presence of dependent products, the identity types admit an operation which one may think of as either *transitivity* or *composition*:

$$\frac{p : \mathsf{Id}_A(a_1, a_2) \qquad q : \mathsf{Id}_A(a_2, a_3)}{q \circ p : \mathsf{Id}_A(a_1, a_3)}\ \mathsf{Id}\text{-TRANS};$$

$$\frac{p : \mathsf{Id}_A(a_1, a_2)}{p \circ r(a_1) = p : \mathsf{Id}_A(a_1, a_2)}\ \mathsf{Id}\text{-TRANS-COMP};$$

and also an operation which one may think of as either *symmetry* or *inverse*:

$$\frac{p : \mathsf{Id}_A(a_1, a_2)}{p^{-1} : \mathsf{Id}_A(a_2, a_1)}\ \mathsf{Id}\text{-SYMM}; \qquad \frac{a : A}{r(a)^{-1} = r(a) : \mathsf{Id}_A(a, a)}\ \mathsf{Id}\text{-SYMM-COMP}.$$

Now suppose that we are given terms m, n and k as in the hypotheses of Π-EXT. We begin by defining terms

$$f := [x : A]\, m \cdot x : (x : A)\, B(x)$$
$$g := [x : A]\, n \cdot x : (x : A)\, B(x)$$
$$\text{and} \quad p := [x : A]\, k \cdot x : (x : A)\, \mathsf{Id}_{B(x)}(fx, gx).$$

Observe that the third of these is well-typed by virtue of the first two. Applying the propositional ξ-rule, we obtain a term

$$\xi(f, g, p) : \mathsf{Id}_{\Pi(A,B)}(\lambda f, \lambda g) = \mathsf{Id}_{\Pi(A,B)}(\lambda x.\, m \cdot x,\ \lambda x.\, n \cdot x).$$

But from the propositional η-rule and Id-symmetry rule, we have terms

$$\eta(m) : \mathsf{Id}_{\Pi(A,B)}(m,\ \lambda x.\, m \cdot x) \quad \text{and} \quad \eta(n)^{-1} : \mathsf{Id}_{\Pi(A,B)}(\lambda x.\, n \cdot x,\ n)$$

and now can define $\mathsf{ext}(m, n, p) := \eta(n)^{-1} \circ \big(\xi(f, g, p) \circ \eta(m)\big) : \mathsf{Id}_{\Pi(A,B)}(m, n)$. In the case where $m = n = \lambda h$ and $p = \lambda(rh)$ we have by the β-rule that $f = g = h$, and so we may calculate that

$$\begin{aligned} \mathsf{ext}(\lambda h, \lambda h, \lambda(rh)) &= \eta(\lambda h)^{-1} \circ \big(\xi(h, h, rh) \circ \eta(\lambda h)\big) \\ &= r(\lambda h)^{-1} \circ \big(r(\lambda h) \circ r(\lambda h)\big) = r(\lambda h) \end{aligned}$$

as required. □

Thus relative to the theory with identity types plus the app-formulation of dependent products, the function extensionality principle is equivalent[2] with the conjunction of the propositional η- and propositional ξ-rules; and relative to the theory with identity types plus the funsplit formulation of dependent products, function extensionality is equivalent with the propositional ξ-rule.

5.5 We now wish to describe the inadequacies of function extensionality in the absence of uniqueness of identity proofs. These arise from its failure to continue a characteristic trend in intensional type theory, namely that nearly every statement that one may think should hold about the identity types, does hold. For instance, in the proof of Proposition 5.4, we saw that the identity types Id_A come equipped with operations which we called *composition* and *inverse*. We would hope for this composition to be associative and unital, and for the inverse operation to really provide compositional inverses; and a straightforward application of Id-elimination shows this to be the case, at least when we interpret associativity, unitality and invertibility in an "up-to-propositional-equality" sense. Similarly, each judgement $x : A \vdash f(x) : B(x)$ induces a judgement $x, y : A,\ p : \mathsf{Id}_A(x, y) \vdash \tilde{f}(p) : \mathsf{Id}_{B(x)}(fx, fy)$ which we would expect to be suitably "functorial" in p: and again, an application of Id-elimination confirms this, providing us with canonical propositional equalities between $\tilde{f}(q \circ p)$ and $\tilde{f}(q) \circ \tilde{f}(p)$. However, when it comes to function extensionality, there are a number of statements which intuitively should be true but which seem to be impossible to prove. Here are two typical examples.

Examples 5.6. (1) Using Id-elimination we can derive a rule

$$\frac{m,\ n : \Pi(A, B) \qquad p : \mathsf{Id}_{\Pi(A,B)}(m, n) \qquad a : A}{p * a : \mathsf{Id}_{B(a)}(m \cdot a,\ n \cdot a)}$$

satisfying $r(m) * a = r(m \cdot a)$, which expresses that any two propositionally equal elements of a Π-type are pointwise propositionally equal. We would expect that for $k : \Pi x : A.\, \mathsf{Id}_{B(x)}(m \cdot x, n \cdot x)$ and $a : A$, we should have $k \cdot a$ propositionally equal to $\mathsf{ext}(m, n, k) * a$; yet this seems impossible to prove.

(2) Suppose given terms $\ell, m, n : \Pi(A, B)$ and proofs $f : (x : A)\, \mathsf{Id}_{B(x)}(\ell \cdot x,\ m \cdot x)$ and $g : (x : A)\, \mathsf{Id}_{B(x)}(m \cdot x,\ n \cdot x)$. Let us write $g \circ f$ for the term $[x : A]\, gx \circ fx$. It now seems to be impossible to verify a propositional equality between the elements

$$\mathsf{ext}(m, n, \lambda g) \circ \mathsf{ext}(\ell, m, \lambda f) \quad \text{and} \quad \mathsf{ext}(\ell, n, \lambda(g \circ f))$$

of $\mathsf{Id}_{\Pi(A,B)}(\ell, n)$.

5.7 The reason that we encounter these problems is essentially the following. We would like to construct the desired propositional equalities by eliminating over the type $u, v : \Pi(A, B) \vdash \Pi x : A.\, \mathsf{Id}_{B(x)}(u \cdot x,\ v \cdot x)$. To do this we need an elimination rule that we do not have, one which says that this type is generated by elements of the form $(\lambda f, \lambda f, \lambda(rf))$. In light of this, we propose that function extensionality should be replaced with just such an elimination rule. We consider the following two rules:

$$\frac{\begin{array}{c} u, v : \Pi(A, B),\ \ w : \Pi x : A.\, \mathsf{Id}_{B(x)}(u \cdot x,\ v \cdot x) \vdash C(u, v, w)\ \text{type} \\ f : (x : A)\, B(x) \vdash d(f) : C(\lambda f, \lambda f, \lambda(rf)) \\ m, n : \Pi(A, B) \qquad k : \Pi x : A.\, \mathsf{Id}_{B(x)}(m \cdot x,\ n \cdot x) \end{array}}{L(d, m, n, k) : C(m, n, k)}\ \Pi\text{-Id-ELIM},$$

$$\frac{\begin{array}{c} u, v : \Pi(A, B),\ \ w : \Pi x : A.\, \mathsf{Id}_{B(x)}(u \cdot x,\ v \cdot x) \vdash C(u, v, w)\ \text{type} \\ f : (x : A)\, B(x) \vdash d(f) : C(\lambda f, \lambda f, \lambda(rf)) \qquad h : (x : A)\, B(x) \end{array}}{L(d, \lambda h, \lambda h, \lambda(rh)) = d(h) : C(\lambda h, \lambda h, \lambda(rh))}\ \Pi\text{-Id-COMP}.$$

Observe that these two rules are once again higher-order inference rules. We will return to this point in Section 5.11 below. Let us first show that these rules entail function extensionality.

[2] Again, in a propositional, rather than definitional, sense.

Proposition 5.8. *In the presence of identity types, the* app*-formulation of* Π*-types and the rules* Π-Id-ELIM *and* Π-Id-COMP *of Section* 5.7*, it is possible to define the function extensionality rules* Π-EXT *and* Π-EXT-COMP*.*

Proof. For each $u, v : \Pi(A, B)$ and $w : \Pi x : A.\, \mathsf{Id}_{B(x)}(u \cdot x,\ v \cdot x)$ we define a type $C(u, v, w) := \mathsf{Id}_{\Pi(A,B)}(u, v)$; and for each $f : (x : A)\, B(x)$, we define an element $d(f) := r(\lambda f) : C(\lambda f, \lambda f, \lambda(rf))$. Applying Π-Id-elimination, we obtain the desired judgement

$$\frac{m,\ n : \Pi(A,B) \qquad k : \Pi x : A.\, \mathsf{Id}_{B(x)}(m \cdot x,\ n \cdot x)}{\mathsf{ext}(m, n, k) := L(d, m, n, k) : \mathsf{Id}_{\Pi(A,B)}(m, n)}\,;$$

and calculate that $\mathsf{ext}(\lambda f, \lambda f, \lambda(rf)) = L(d, \lambda f, \lambda f, \lambda(rf)) = d(f) = r(\lambda f)$ as required. □

Thus the Π-Id-elimination rules are stronger than the function extensionality rules, and we conjecture that they are *strictly* stronger. To prove this would require either finding a new model of type theory that supports function extensionality but not Π-Id-elimination – new, because every semantic model of which the author is aware that supports the former, also supports the latter – or giving a syntactic proof along the lines of that of Theorem 4.2. In both cases, the author's efforts have been unsuccessful. Leaving this aside, let us now show that the Π-Id-elimination rules allow us to give positive answers to the question posed in Examples 5.6.

Proposition 5.9. *In the presence of identity types, the* app*-formulation of* Π*-types and the rules* Π-Id-ELIM *and* Π-Id-COMP *of Section* 5.7*, the following rules are definable:*

$$\frac{m,\ n : \Pi(A,B) \qquad k : \Pi x : A.\, \mathsf{Id}_{B(x)}(m \cdot x,\ n \cdot x) \qquad a : A}{\mu(m, n, k, a) : \mathsf{Id}_{\mathsf{Id}_{B(a)}(m \cdot a,\ n \cdot a)}\big(\mathsf{ext}(m, n, k) * a,\ \ k \cdot a\big)}\ \Pi\text{-EXT-APP};$$

$$\frac{f : (x : A)\, B(x)}{\mu(\lambda f, \lambda f, \lambda(rf), a) = r(rfa) : \mathsf{Id}_{\mathsf{Id}_{B(a)}(fa, fa)}\big(rfa,\ \ rfa\big)}\ \Pi\text{-EXT-APP-COMP};$$

where $*$ *is the operation defined in Examples* 5.6(1).

Proof. For each $u, v : \Pi(A, B)$ and $w : \Pi x : A.\, \mathsf{Id}_{B(x)}(u \cdot x,\ v \cdot x)$ we define a type

$$C(u, v, w) := \Pi x : A.\, \mathsf{Id}_{\mathsf{Id}_{B(x)}(u \cdot x, v \cdot x)}(\mathsf{ext}(u, v, w) * x,\ \ w \cdot x).$$

Now for $f : (x : A)\, B(x)$, we calculate that

$$\begin{aligned} C(\lambda f, \lambda f, \lambda(rf)) &= \Pi x : A.\, \mathsf{Id}_{\mathsf{Id}_{B(x)}(fx, fx)}(\mathsf{ext}(\lambda f, \lambda f, \lambda(rf)) * x,\ \ rfx) \\ &= \Pi x : A.\, \mathsf{Id}_{\mathsf{Id}_{B(x)}(fx, fx)}(r(\lambda f) * x,\ \ rfx) \\ &= \Pi x : A.\, \mathsf{Id}_{\mathsf{Id}_{B(x)}(fx, fx)}(rfx,\ \ rfx) \end{aligned}$$

so that we may define $d(f) := \lambda x.\, r(rfx) : C(\lambda f, \lambda f, \lambda(rf))$. An application of Π-Id-elimination now yields the judgement Π-EXT-APP by taking

$$\mu(m, n, k, a) := L(d, m, n, k) \cdot a : \mathsf{Id}_{\mathsf{Id}_{B(a)}(m \cdot a, n \cdot a)}(\mathsf{ext}(m, n, k) * a,\ \ k \cdot a).$$

Finally, we compute that $\mu(\lambda f, \lambda f, \lambda(rf), a) = \lambda x.\, r(rfx) \cdot a = r(rfa)$ as required. □

Proposition 5.10. *In the presence of identity types, the* app*-formulation of* Π*-types and the rules* Π-Id-ELIM *and* Π-Id-COMP *of Section* 5.7*, the following rule is definable:*

$$\frac{\ell,\ m,\ n : \Pi(A,B) \qquad f : (x : A)\, \mathsf{Id}_{B(x)}(\ell \cdot x, m \cdot x) \qquad g : (x : A)\, \mathsf{Id}_{B(x)}(m \cdot x, n \cdot x)}{\nu(f, g) : \mathsf{Id}_{\mathsf{Id}_{\Pi(A,B)}(\ell, n)}\big(\mathsf{ext}(m, n, \lambda g) \circ \mathsf{ext}(\ell, m, \lambda f),\ \ \mathsf{ext}(\ell, n, \lambda(g \circ f))\big)}$$

Proof. It suffices to derive the rule:

$$\frac{\ell,\ m,\ n : \Pi(A,B) \qquad j : \Pi x : A.\, \mathsf{Id}_{B(x)}(\ell \cdot x, m \cdot x) \qquad k : \Pi x : A.\, \mathsf{Id}_{B(x)}(m \cdot x, n \cdot x)}{\nu'(j, k) : \mathsf{Id}_{\mathsf{Id}_{\Pi(A,B)}(\ell, n)}\big(\mathsf{ext}(m, n, k) \circ \mathsf{ext}(\ell, m, j),\ \ \mathsf{ext}(\ell, n, \lambda x.\, k \cdot x \circ j \cdot x)\big)}$$

since the required result then follows by taking $j := \lambda f$ and $k := \lambda g$. But by Π-Id-elimination on k, it suffices to derive this rule in the case where $m = n = \lambda h$ and $k = \lambda(rh)$; which is to show that

$$\frac{\ell : \Pi(A,B) \qquad h : (x : A)\, B(x) \qquad j : \Pi x : A.\, \mathsf{Id}_{B(x)}(\ell \cdot x, hx)}{\nu'(j, \lambda(rh)) : \mathsf{Id}_{\mathsf{Id}_{\Pi(A,B)}(\lambda h, n)}\big(\mathsf{ext}(\lambda h,\ \lambda h,\ \lambda(rh)) \circ \mathsf{ext}(\ell,\ \lambda h,\ j),\ \ \mathsf{ext}(\ell,\ \lambda h,\ \lambda x.\, r(hx) \circ j \cdot x)\big)}$$

is derivable. But we have that $r(hx) \circ j \cdot x = j \cdot x$ and that $\mathsf{ext}(\lambda h, \lambda h, \lambda(rh)) = r(\lambda h)$ so that $\mathsf{ext}(\lambda h, \lambda h, \lambda(rh)) \circ \mathsf{ext}(\ell, \lambda h, j) = \mathsf{ext}(\ell, \lambda h, j)$: so that it suffices to show that

$$\frac{\ell : \Pi(A,B) \qquad h : (x : A)\, B(x) \qquad j : \Pi x : A.\, \mathsf{Id}_{B(x)}(\ell \cdot x, hx)}{\nu'(j, \lambda(rh)) : \mathsf{Id}_{\mathsf{Id}_{\Pi(A,B)}(\lambda h, n)}\big(\mathsf{ext}(\ell,\ \lambda h,\ j),\ \ \mathsf{ext}(\ell,\ \lambda h,\ \lambda x.\, j \cdot x)\big)}$$

is derivable. Now, using the propositional η-rule, we can derive a term $\eta(j)$ witnessing the propositional equality of j and $\lambda x.\, j \cdot x$; and we will be done if we can lift this to a propositional equality between $\mathsf{ext}(\ell, \lambda h, j)$ and $\mathsf{ext}(\ell, \lambda h, \lambda x.\, j \cdot x)$. But we may do this using the following rule:

$$\frac{a, b : \Pi(A, B) \qquad c, d : \Pi x : A.\, \mathsf{Id}_{B(x)}(a \cdot x, b \cdot x) \qquad p : \mathsf{Id}_{\Pi x : A.\, \mathsf{Id}_{B(x)}(a \cdot x, b \cdot x)}(c, d)}{\widetilde{\mathsf{ext}}(p) : \mathsf{Id}_{\mathsf{Id}_{\Pi(A,B)}}\big(\mathsf{ext}(a, b, c),\ \ \mathsf{ext}(a, b, d)\big)},$$

which is derivable by Id-elimination on p. $\square$

In Section 4, we saw that the higher-order formulation of Π-types can be restated in a first-order manner; and the final result of this paper will do something similar for the Π-Id-elimination rule.

Proposition 5.11. *In the presence of the identity types; the* app*-formulation of* Π*-types; the function extensionality rules* Π-EXT *and* Π-EXT-COMP*; and the rules* Π-EXT-APP *and* Π-EXT-APP-COMP *of Proposition 5.9, we can define the rules* Π-Id-ELIM *and* Π-Id-COMP *of Section 5.7.*

Proof. Suppose that we are given terms

$$\begin{aligned} u, v : \Pi(A, B),\ \ w : \Pi x : A.\, \mathsf{Id}_{B(x)}(u \cdot x,\ v \cdot x) &\vdash C(u, v, w)\ \mathsf{type} \\ f : (x : A)\, B(x) &\vdash d(f) : C(\lambda f, \lambda f, \lambda(rf)) \\ m, n : \Pi(A, B) \qquad k &: \Pi x : A.\, \mathsf{Id}_{B(x)}(m \cdot x,\ n \cdot x) \end{aligned}$$

as in the premisses of Π-Id-ELIM. We must find an element $L(d, m, n, k) : C(m, n, k)$. We will employ much the same method as we did in the proof of Proposition 3.7, though the details will be a little more complicated. We begin by constructing the element $d([x : A]\, m \cdot x) : C(\lambda x.\, m \cdot x,\ \ \lambda x.\, m \cdot x,\ \ \lambda x.\, r(m \cdot x))$; and the remainder of the proof will involve applying various substitutions to this element until we obtain the required element of $C(m, n, k)$. The key result we need is the following lemma.

Lemma. *We may define a rule:*

$$\frac{u, v : \Pi(A, B) \qquad p : \mathsf{Id}_{\Pi(A,B)}(u, v) \qquad c : C(\lambda x.\, u \cdot x,\ \ \lambda x.\, u \cdot x,\ \ \lambda x.\, r(u \cdot x))}{\phi(p, c) : C(u, v, \lambda x.\, p * x)} \qquad (\star)$$

satisfying $\phi(r(\lambda f),\ c) = c$.

Before proving this, let us see how it allows us to derive the required element of $C(m, n, k)$. Using function extensionality we can form $\mathsf{ext}(m, n, k) : \mathsf{Id}_{\Pi(A,B)}(m, n)$; and so by applying ϕ to this and $d([x : A]\, m \cdot x)$ can obtain an element

$$b(m, n, k) := \phi(\mathsf{ext}(m, n, k),\ \ d([x : A]\, m \cdot x)) : C(m, n, \lambda x.\, \mathsf{ext}(m, n, k) * x).$$

We now make use of the rule Π-EXT-APP of Proposition 5.9, which provides us with a term

$$x : A \vdash \mu(m, n, k, x) : \mathsf{Id}_{\mathsf{Id}_{B(x)}(m \cdot x,\, n \cdot x)}\big(\mathsf{ext}(m, n, k) * x,\ \ k \cdot x\big);$$

applying function extensionality to which yields a term

$$p(m, n, k) := \mathsf{ext}(\lambda x.\, \mathsf{ext}(m, n, k) * x,\ \ k,\ \ \lambda x.\, \mu(m, n, k, x)) : \mathsf{Id}_{\Pi x : A.\, \mathsf{Id}_{B(x)}(m \cdot x,\, n \cdot x)}(\lambda x.\, \mathsf{ext}(m, n, k) * x,\ \ k).$$

The final step is to use the Leibniz rule defined in the proof of Proposition 3.7 to form the required term $L(d, m, n, k) := \mathsf{subst}(p(m, n, k), b(m, n, k)) : C(m, n, k)$. We are also required to show that $L(d, \lambda f, \lambda f, \lambda(rf)) = d(f)$. For this, we first note that $b(\lambda f, \lambda f, \lambda(rf)) = \phi(r(\lambda f), d(f)) = d(f) : C(\lambda f, \lambda f, \lambda(rf))$. Next we observe that $\mu(\lambda f, \lambda f, \lambda(rf), x) = r(rfx)$ so that we have

$$\begin{aligned} p(\lambda f, \lambda f, \lambda(rf)) &= \mathsf{ext}\big(\lambda x.\, r(\lambda f) * x,\ \ \lambda(rf),\ \ \lambda x.\, r(rfx)\big) \\ &= \mathsf{ext}\big(\lambda(rf),\ \ \lambda(rf),\ \ \lambda(rrf)\big) = r(\lambda(rf)) \end{aligned}$$

so that $L(d, \lambda f, \lambda f, \lambda(rf)) = \mathsf{subst}(r(\lambda(rf)), d(f)) = d(f)$ as required.

It remains only to prove the Lemma. We will derive ($\star$) by Id-elimination on p, for which it suffices to consider the case where $u = v$ and $p = r(u)$. So we must show that

$$\frac{u : \Pi(A, B) \qquad c : C(\lambda x.\, u \cdot x,\ \ \lambda x.\, u \cdot x,\ \ \lambda x.\, r(u \cdot x))}{\phi(r(u), c) : C(u, u, \lambda x.\, r(u) * x)}$$

is derivable; which in turn we may do by Π-elimination on u. Indeed, when we have $u = \lambda f$ for some $f : (x : A)\, B(x)$, we find that $C(\lambda x.\, u \cdot x,\ \ \lambda x.\, u \cdot x,\ \ \lambda x.\, r(u \cdot x)) = C(\lambda f, \lambda f, \lambda(rf)) = C(u, u, \lambda x.\, r(u) * x)$ so that we may take $\phi(r(\lambda f), c) = c$. $\square$

5.12 We end the paper with an informal discussion of the adequacy of our strengthening of the principle of function extensionality. We have portrayed it as a necessary strengthening, but we have not indicated why we think it sufficient: could there not be yet more exotic propositional equalities of the sort considered in Examples 5.6 which our Π-Id-elimination rule cannot verify? The reason the author believes this not to be the case is essentially semantic. As mentioned in Section 5.1, if we wish to describe higher-dimensional categorical semantics for type theory in which Π-type formation is a (suitably weak) right adjoint to reindexing, then we need a form of function extensionality. As it turns out, the Π-Id-elimination rule given above is just what is needed to make this go through. The author has verified the details of this for two-dimensional models in [2], and has sketched them for a putative theory of three-dimensional models. Moreover, there is a general argument which suggests that this extends to all higher dimensions, which runs as follows.

When we form higher-dimensional models of type theory, we obtain the higher-dimensionality from the identity type structure. In order for Π-type formation to provide a weak right adjoint to pullback, it must respect the higher-dimensionality, and hence the identity type structure. Now, if we are given A type and $x : A \vdash B(x)$ type, then dependent product formation over $x : A$ sends the identity type

$$x : A,\ \ y, z : B(x) \vdash \mathsf{Id}_{B(x)}(y, z)\ \mathsf{type}$$

to the type

$$m, n : \Pi(A, B) \vdash \Pi x : A.\, \mathsf{Id}_{B(x)}(m \cdot x, n \cdot x)\ \mathsf{type};$$

and to say that function space formation preserves the identity type structure is to say that this latter type should act like an identity type for $\Pi(A, B)$; and it precisely this which is expressed by our elimination rule Π-Id-ELIM.

Acknowledgements

The author wishes to thank Johan Granström, Per Martin-Löf, Erik Palmgren, Olov Wilander and other members of the Stockholm–Uppsala Logic Seminar for useful comments and suggestions on earlier drafts of this paper. He also thanks the anonymous referee for several useful comments and suggestions.
He was supported by a Research Fellowship of St. John's College, Cambridge and a Marie Curie Intra-European Fellowship, Project No. 040802.

References

[1] Thorsten Altenkirch, Extensional equality in intensional type theory, in: 14th Symposium on Logic in Computer Science, Trento, 1999, IEEE Computer Soc., Los Alamitos, CA, 1999, pp. 412–420.
[2] Richard Garner, Two-dimensional models of dependent type theory, Mathematical Structures in Computer Science (2009) (in press).
[3] Martin Hofmann, Conservativity of equality reflection over intensional type theory, Types for Proofs and Programs, Torino, 1995, in: Lecture Notes in Comput. Sci., vol. 1158, Springer, Berlin pp. 153–164.
[4] Martin Hofmann, Extensional concepts in intensional type theory, Ph.D. Thesis, University of Edinburgh, 1995.
[5] Martin Hofmann, Syntax and semantics of dependent types, in: Semantics and Logics of Computation, Cambridge, 1995, in: Publications of the Newton Institute, vol. 14, Cambridge University Press, 1997, pp. 79–130.
[6] Martin Hofmann, Thomas Streicher, The groupoid interpretation of type theory, in: Twenty-five Years of Constructive Type Theory, Venice, 1995, in: Oxford Logic Guides, vol. 36, Oxford University Press, 1998, pp. 83–111.
[7] Per Martin-Löf, Intuitionistic type theory, Studies in Proof Theory, Bibliopolis, Naples, 1984.
[8] Per Martin-Löf, An intuitionistic theory of types, in: Twenty-five Years of Constructive Type Theory, Venice, 1995, in: Oxford Logic Guides, vol. 36, Oxford University Press, 1998, pp. 127–172. Reprinted version of an unpublished report from 1972.
[9] Bengt Nordström, Kent Petersson, Jan Smith, Programming in Martin-Löf's type theory, in: International Series of Monographs on Computer Science, vol. 7, Oxford University Press, 1990.
[10] Peter Schroeder-Heister, A natural extension of natural deduction, Journal of Symbolic Logic 49 (4) (1984) 1284–1300.
[11] Anne Troelstra, Dirk van Dalen, Constructivism in Mathematics. Vol. II, in: Studies in Logic and the Foundations of Mathematics, vol. 123, North-Holland Publishing Co., Amsterdam, 1988, An introduction.
[12] David Turner, A new formulation of constructive type theory, in: Proceedings of the Workshop on Programming Logic, Programming Methodology Group, Göteberg, 1989, pp. 258–294.