# Some open(?) problems concerning Dependent Type Theories*

Peter Aczel
The University of Manchester

May 29, 2012

## 1 Introduction

The aim of this note is to state some problems that are open, as far as I am aware, concerning the logical strength of some dependent type theories. In section 2 I will first state the problems in a rather informal way and then give two more precise versions concerning specific type theories. A selection of type theories will be described in sections 3-7 and their rules will be presented in more detail in the appendix, section 8.

## 2 The problems

**Q1:** Is the type theory implemented in the Coq proof assistant, see [Coq], logically weaker or stronger than **ZF**?

**Q2:** Does the addition of Voevodsky's Univalence Axiom, see [Voevodsky], and possibly other rules such as rules for higher dimenensional inductive definitions, to a standard dependent type theory increase the logical strength of the type theory?

*These problems were raised at a seminar, devoted to the raising of open problems, of the Syntax and Semantics programme of the Cambridge Newton Institute. I am grateful to the Institute for providing me the excellent facilities in which this note was prepared.

I now state more precise versions of the problems using specific type theories which will be described in the rest of this note.

**Q1′:** Is $\mathbf{MLW}\mathbb{PSU}_{<\omega}$ logically weaker or stronger than $\mathbf{ZF}$ ?

**Q2′:** Is $\mathbf{MLW}\mathbb{U}$ logically weaker than $\mathbf{MLW}\mathbb{U} + \mathbf{UA}(\mathbb{U})$, where $\mathbf{UA}(\mathbb{U})$ is Voevodsky's Univalence Axiom for the type universe $\mathbb{U}$.

In the rest of this note we first give an outline description of a selection of type theories obtained from a base type theory **ML** by adding a variety of additional forms of type. In our description we assume that the reader has some familiarity with the various forms of type in the literature. A more detailed presentation of the rules of the type theories has been placed in the appendix.

# 3 The standard forms of judgement of a dependent type theory

A type theory will be a system of rules for deriving judgements, each having the form

$$\Gamma \vdash \mathcal{B}$$

where $\Gamma$ is its **context**

$$x_1 : A_1, \ldots, x_n : A_n$$

of $n \geq 0$ **variable declarations** $x_i : A_i$, for $i = 1, \ldots, n$, of distinct variables $x_1, \ldots, x_n$ and $\mathcal{B}$ is its **body**, which has one of the forms

$$A \ \mathsf{type} \qquad A_1 = A_2 \qquad a : A \qquad a_1 = a_2 : A.$$

The body $A\ \mathsf{type}$ expresses that $A$ is a type, $A_1 = A_2$ expresses that $A_1, A_2$ are judgementally equal types, $a : A$ expresses that $a$ is a term of type $A$ and $a_1 = a_2 : A$ expresses that $a_1, a_2$ are judgementally equal terms of type $A$.

When $\Gamma$ is the empty context we will usually just write $\mathcal{B}$ rather than $\vdash \mathcal{B}$. Each rule will have instances, each of the form

$$\frac{J_1 \cdots J_m}{J}$$

where the $J_1, \ldots, J_m, J$ are forms of judgement. The $J_1, \ldots, J_m$ above the line are the **premisses** of the rule instance and $J$ is its **conclusion**. The rules will be presented schematically, using conventions that we hope will mostly be obvious. Usually, a rule will allow a parametric list of variable declarations to appear in the premisses and conclusion of a rule. In presenting the rule the parametric list will be left implicit. For example a rule for forming function types $A \to B$ will be presented

$$\frac{A \ \mathsf{type} \quad B \ \mathsf{type}}{(A \to B) \ \mathsf{type}}$$

and will have instances

$$\frac{\Gamma \vdash A \ \mathsf{type} \quad \Gamma \vdash B \ \mathsf{type}}{\Gamma \vdash (A \to B) \ \mathsf{type}}.$$

where $\Gamma$ can be any context.

A type theory will be presented by listing its rules, and the theorems of the type theory will form the smallest collection of judgements with the property that whenever the premisses of an instance of a rule are in the collection then so is the conclusion. We will first specify each theory by only presenting the rules that are relevent to the formation of the types of the type theory. A more detailed presentation of all the rules may be found in the appendix.

## 4 The Type Theories **ML**, **ML**$^-$ and **MLW**

We start with the formation rules for the basic type theory **ML**. The theory **ML**$^-$ is obtained from **ML** by leaving out the rules for the type $N$ of natural numbers

| **Type Formation Rules for ML** | | |
|---|---|---|
| $\dfrac{}{N_n \ \mathsf{type}} \quad (n = 0, 1, \ldots)$ | $\dfrac{A \ \mathsf{type} \quad x : A \vdash B[x] \ \mathsf{type}}{(\Pi x : A)B[x] \ \mathsf{type}}$ | $\dfrac{A \ \mathsf{type} \quad a_1, a_2 : A}{Id_A(a_1, a_2) \ \mathsf{type}}$ |
| $\dfrac{}{N \ \mathsf{type}}$ | $\dfrac{A \ \mathsf{type} \quad x : A \vdash B[x] \ \mathsf{type}}{(\Sigma x : A)B[x] \ \mathsf{type}}$ | $\dfrac{A_1 \ \mathsf{type} \quad A_2 \ \mathsf{type}}{A_1 + A_2 \ \mathsf{type}}$ |

There are many more rules needed for the type theory. A complete set of rules is in the appendix. Here we offer the following remarks that may help reader's intuition. The type $N_n$ is the $n$-element type having the $n$ canonical elements $1_n, \ldots, n_n$ and $N$ is the type of natural numbers, having the canonical elements 0 and $s(e)$ for $e : N$. The type $Id_A(a_1, a_2)$ is the identity type. When $a_1 = a_2 = a : A$ then it has the canonical element $r_A(a)$. The $\Pi$ and $\Sigma$ types have, as special cases $A \to B = (\Pi - : A)B$ and $A \times B = (\Sigma - : A)B$, where $-$ indicates a variable that does not occur free in $B$. The type $(\Pi x : A)B[x]$ has canonical elements $(\lambda x : A)b[x]$ where $b[x] : B[x]$ for $x : A$ and $(\Sigma x : A)B[x]$ has the canonical elements $\mathsf{pair}(a, b)$ where $a : A$ and $b : B[a]$. Finally the sum type $A_1 + A_2$ has canonical elements $\mathsf{in}_1(a)$ for $a : A_1$ and $\mathsf{in}_2(a)$ for $a : A_2$.

We obtain the type theory **MLW** by adding the following rule.

$$\frac{A \ \mathsf{type} \qquad x : A \vdash B[x] \ \mathsf{type}}{(Wx : A)B[x] \ \mathsf{type}}$$

This type $W = (Wx : A)B[x]$ is the inductive type whose canonical elements have the form $\mathsf{sup}(y : B[a])c[y]$ where $a : A$ and $c[y] : W$ for $y : B[a]$. Here it should be noted that free occurences of $y$ in $c[y]$ become bound in the term $\mathsf{sup}(y : B[a])c[y]$.

## 5 Adding a Type Universe and an impredicative type of propositions

Another possibility is to add a type universe to **ML** that reflects the forms of type of **ML**. Here, by a type universe we shall mean a type $\mathbb{U}$ whose elements are themselves types. So we add the following rules.

$$\frac{}{\mathbb{U} \ \mathsf{type}} \qquad \frac{A : \mathbb{U}}{A \ \mathsf{type}}$$

That $\mathbb{U}$ reflects the forms of type of **ML** is obtained by also adding the following rules.

| $\mathbb{U}$ **Term Formation Rules for ML**$\mathbb{U}$ | | |
|---|---|---|
| $\dfrac{}{N_n : \mathbb{U}} \ (n = 0, 1, \ldots)$ | $\dfrac{}{N : \mathbb{U}}$ | $\dfrac{A : \mathbb{U} \quad a_1, a_2 : A}{Id_A(a_1, a_2) : \mathbb{U}}$ |
| $\dfrac{A : \mathbb{U} \quad x : A \vdash B[x] : \mathbb{U}}{(\Pi x : A)B[x] : \mathbb{U}}$ | $\dfrac{A : \mathbb{U} \quad x : A \vdash B[x] : \mathbb{U}}{(\Sigma x : A)B[x] : \mathbb{U}}$ | $\dfrac{A_1, A_2 : \mathbb{U}}{A_1 + A_2 : \mathbb{U}}$ |

By adding these rules to **ML** we obtain the type theory **ML**$\mathbb{U}$. We obtain **MLW**$\mathbb{U}$ by adding to **MLW** the above rules together with the obvious rule to reflect the formation rule for $W$-types.

So far our type theories are generalised predicative and are well below the logical strength of full second order arithmetic. We get a fully impredicative type theory, **ML**$\mathbb{P}$, by adding a calculus of constructions type universe $\mathbb{P}$ having the following rules.

$$\frac{}{\mathbb{P}\ \mathsf{type}} \qquad \frac{A : \mathbb{P}}{A\ \mathsf{type}} \qquad \frac{A\ \mathsf{type} \quad x : A \vdash B[x] : \mathbb{P}}{(\Pi x : A)B[x] : \mathbb{P}}$$

The first two rules just express that $\mathbb{P}$ is a type universe. The impredicativity comes in the third rule which allows the formation of the type $(\Pi x : A)B[x] : \mathbb{P}$ even though the type $A$ might be $\mathbb{P}$ itself or might be a type, such as $(N \to \mathbb{P}) \to \mathbb{P}$, that has been formed using $\mathbb{P}$.

We obtain the type theory **ML**$\mathbb{PU}$ by combining the rules of **ML**$\mathbb{P}$ with those of **ML**$\mathbb{U}$ and adding the following reflection rules.

$$\frac{}{\mathbb{P} : \mathbb{U}} \qquad \frac{A : \mathbb{P}}{A : \mathbb{U}}$$

# 6 Adding a Hierarchy of Type Universes

Instead of adding just one type universe we can add an infinite increasing cumulative hierarchy of type universes $\mathbb{U}_1, \mathbb{U}_2, \ldots$, each reflecting the previous universes. So each $\mathbb{U}_n$ has all the rules we have given to $\mathbb{U}$ and in addition the rules

$$\frac{}{\mathbb{U}_n : \mathbb{U}_{n+1}} \qquad \frac{A : \mathbb{U}_n}{A : \mathbb{U}_{n+1}} \qquad (n = 1, 2, \ldots)$$

In this way we obtain the type theories **ML**$\mathbb{U}_{<\omega}$ and **MLW**$\mathbb{U}_{<\omega}$.

# 7 An Approximation to the Coq Type Theory

We now turn to a type theory close to the type theory implemented in the proof assistant Coq. This type theory is thoroughly impredicative. It is obtained from **MLW**$\mathbb{P}$ by adding a type universe $\mathbb{S}$ that also reflects **MLW**, and has the rule

$$\frac{A : \mathbb{P}}{A : \mathbb{S}}$$

but not the rule $\frac{}{\mathbb{P} : \mathbb{S}}$. and then adding a hierarchy of type universes $\mathbb{U}_0, \mathbb{U}_1, \ldots$ that reflects **MLW**$\mathbb{PS}$, the resulting type theory being **MLW**$\mathbb{PSU}_{<\omega}$. In Coq the types $\mathbb{P}$ and $\mathbb{S}$ are called *Prop* and *Set* respectively.

# 8 Appendix

In this appendix we try to give a reasonably complete and accurate presentation of rules for the type theory **MLW**. Missing are the congruence rules for the constructors as they are very numerous and are determined according to a specific algorithm applied to the explicitly or implicitly given formation rule for each constructor. In fact the congruence rules for the constructors that do not bind any variables can be derived from the Subsitution rule below. Rather than try to precisely state the procedure for the constructors that bind variables we will illustrate it by giving the congruence rules for the constructors $\Sigma$ and $\mathsf{split}$. To understand the general pattern it should be noted that the expression $(\Sigma x : A)B[x]$ should be thought of as prefered notation for $\Sigma(A, (x)B[x])$ where it is indicated that free occurences of $x$ in $B[x]$ become bound in $(\Sigma x : A)B[x]$. The formation rule for this type is

$$\frac{A \;\mathsf{type} \qquad x : A \vdash B[x] \;\mathsf{type}}{(\Sigma x : A)B[x] \;\mathsf{type}}.$$

So its congruence rule is

$$\frac{A_1 = A_2 \qquad x : A_1 \vdash B_1[x] = B_2[x]}{(\Sigma x : A_1)B_1[x] = (\Sigma x : A_2)B_2[x]}.$$

The formation rule for $\mathsf{split}$ is

$$\boxed{\frac{\begin{cases} z : (\Sigma x : A)B[x] \vdash C[z] \;\mathsf{type} \\ x : A, y : B[x] \vdash c[x, y] : C[\mathsf{pair}(x, y)] \end{cases}}{\begin{cases} z : (\Sigma x : A)B[x] \vdash \mathsf{split}[z] : C[z] \\ x : A, y : B[x] \vdash \mathsf{split}[\mathsf{pair}(x, y)] = c[x, y] : C[\mathsf{pair}(x, y)] \end{cases}}}$$

where $\mathsf{split}[e]$ abbreviates $\mathsf{split}(e, (x, y)c[x, y])$. So its congruence rule is

$$\boxed{\frac{\begin{cases} z : (\Sigma x : A)B[x] \vdash C[z] \;\mathsf{type} \\ x : A, y : B[x] \vdash c_1[x, y] = c_2[x, y] : C[\mathsf{pair}(x, y)] \end{cases}}{z : (\Sigma x : A)B[x] \vdash \mathsf{split}(z, (x, y)c_1[x, y]) = \mathsf{split}(z, (x, y)c_2[x, y]) : C[z]}}$$

## 8.1 Assumption and Substitution Rules

**Assumption** $\dfrac{\Gamma \vdash \mathcal{B} \qquad A \;\mathsf{type}}{x : A, \Gamma \vdash \mathcal{B}}$ and $\dfrac{\Gamma \vdash \mathcal{B} \qquad A \;\mathsf{type}}{x : A, \Gamma \vdash x : A}$, where it is assumed that $x$ is not declared in $\Gamma$ or in any implicit parametric context.)

**Substitution** $\dfrac{x : A, \Gamma[x] \vdash \mathcal{B}[x] \qquad a : A}{\Gamma[a] \vdash \mathcal{B}[a]}$

## 8.2 Equality Rules

$$\frac{A\ \mathsf{type}}{A = A} \qquad \frac{A_1 = A_2}{A_2 = A_1} \qquad \frac{A_1 = A_2 \quad A_2 = A_3}{A_1 = A_3}$$

$$\frac{a : A}{a = a : A} \qquad \frac{a_1 = a_2 : A}{a_2 = a_1 : A} \qquad \frac{a_1 = a_2 : A \quad a_2 = a_3 : A}{a_1 = a_3 : A}$$

$$\frac{a : A_1 \quad A_1 = A_2}{a : A_2} \qquad \frac{a_1 = a_2 : A_1 \quad A_1 = A_2}{a_1 = a_2 : A_2}$$

## 8.3 Finite Type Rules

For $n = 0, 1, 2, \ldots$ and $k = 1, \ldots, n$

$$\frac{}{N_n} \qquad \frac{}{k_n : N_n} \qquad \frac{\left\{\begin{array}{l} z : N_n \vdash C[z]\ \mathsf{type} \\ c_i : C[i_n] \ \ (i = 1, \ldots, n) \end{array}\right.}{\left\{\begin{array}{l} z : N_n \vdash \mathsf{R}_n(z) : C[z] \\ \mathsf{R}_n(k_n) = c_k : C[k_n] \end{array}\right.}$$

where $\mathsf{R}_n[e]$ abbreviates $\mathsf{R}_n(e, c_1, \ldots, c_n)$.

## 8.4 Natural Number Rules

$$\frac{}{N\ \mathsf{type}} \qquad \frac{}{0 : N} \qquad \frac{e : N}{succ(e) : N}$$

$$\frac{\left\{\begin{array}{l} z : N \vdash C[z]\ \mathsf{type} \\ c_0 : C[0] \\ z : N, x : C[z] \vdash d[z,x] : C[succ(z)] \end{array}\right.}{\left\{\begin{array}{l} z : N \vdash \mathsf{R}_N[z] : C[z] \\ \mathsf{R}_N[0] = c_0 : C[0] \\ z : N, x : C[z] \vdash \mathsf{R}_N[succ(z)] = d[z, \mathsf{R}_N[z]] : C[succ(z)] \end{array}\right.}$$

where $\mathsf{R}_N[e]$ abbreviates $\mathsf{R}_N(e, c_0, (z,x)d[z,x])$.

## 8.5 Identity Type Rules

$$\frac{A\ \mathsf{type}}{x_1, x_2 : A \vdash Id_A(x_1, x_2)} \qquad \frac{a : A}{r_A(a) : Id_A(a,a)}$$

$$\frac{\begin{cases} a : A \\ y : A, z : Id_A(a, y) \vdash C[y, z] \ \mathsf{type} \\ e : C[a, r_A(a)] \end{cases}}{\begin{cases} y : A, z : Id_A(a, y) \vdash J(a, e, y, z) : C[y, z] \\ J(a, e, a, r_A(a)) = e : C[a, r_A(a)] \end{cases}}$$

## 8.6 Pi Type Rules

$$\frac{A \ \mathsf{type} \quad x : A \vdash B[x] \ \mathsf{type}}{(\Pi x : A)B[x] \ \mathsf{type}} \qquad \frac{f : (\Pi x : A)B[x] \quad a : A}{\mathsf{app}(f, a) : B[a]}$$

$$\frac{x : A \vdash b[x] : B[x]}{(\lambda x : A)b[x] : (\Pi x : A)B[x]} \qquad \frac{x : A \vdash b[x] : B[x] \qquad a : A}{\mathsf{app}((\lambda x : A)b[x], a) = b[a] : B[a]}$$

## 8.7 Sigma Type Rules

$$\frac{A \ \mathsf{type} \quad x : A \vdash B[x] \ \mathsf{type}}{(\Sigma x : A)B[x] \ \mathsf{type}} \qquad \frac{x : A \vdash B[x] \quad a : A \quad b : B[a]}{\mathsf{pair}(a, b) : (\Sigma x : A)B[x]}$$

$$\frac{\begin{cases} z : (\Sigma x : A)B[x] \vdash C[z] \ \mathsf{type} \\ x : A, y : B[x] \vdash c[x, y] : C[\mathsf{pair}(x, y)] \end{cases}}{\begin{cases} z : (\Sigma x : A)B[x] \vdash \mathsf{split}[z] : C[z] \\ x : A, y : B[x] \vdash \mathsf{split}[\mathsf{pair}(x, y)] = c[x, y] : C[\mathsf{pair}(x, y)] \end{cases}}$$

where $\mathsf{split}[e]$ abbreviates $\mathsf{split}(e, (x, y)c[x, y])$.

## 8.8 Binary Sum Rules

$$\frac{A_1 \ \mathsf{type} \quad A_2 \ \mathsf{type}}{A_1 + A_2 \ \mathsf{type}} \qquad \frac{A_1 \ \mathsf{type} \quad A_2 \ \mathsf{type} \quad a : A_i}{\mathsf{in}_i(a) : A_1 + A_2} \ (i = 1, 2)$$

$$\frac{\begin{cases} z : A_1 + A_2 \vdash C[z] \ \mathsf{type} \\ x : A_j \vdash d_j[x] : C[\mathsf{in}_j(x)] \ \ (j = 1, 2) \end{cases}}{\begin{cases} z : A_1 + A_2 \vdash \mathsf{case}[z] : C[z] \\ x : A_j \vdash \mathsf{case}[\mathsf{in}_j(x)] = d_j[x] : C[\mathsf{in}_j(x)] \ \ (j = 1, 2) \end{cases}} \ (i = 1, 2)$$

where $\mathsf{case}[e]$ abbreviates $\mathsf{case}(e, (x)d_1[x], (x)d_2[x])$.

## 8.9 W Type Rules

$$\frac{A\ \mathsf{type} \qquad x : A \vdash B[x]\ \mathsf{type}}{(Wx : A)B[x]\ \mathsf{type}} \qquad \frac{a : A \qquad y : B[a] \vdash c[y] : W}{(\mathsf{sup}\ y : B[a])c[y] : W}$$

where we use $W$ to abbreviate $(Wx : A)B[x]$.

$$\frac{\begin{cases} z : W \vdash C[z]\ \mathsf{type} \\ x : A, u : (B[x] \to W), v : C'[x,u] \vdash d[x,u,v] : C''[x,u] \end{cases}}{\begin{cases} z : W \vdash \mathsf{R}_W[z] : C[z] \\ x : A, u : (B[x] \to W) \vdash \mathsf{R}_W[\mathbf{sup}'[x,u]] = d'[x,u] : C''[x,u] \end{cases}}$$

where we use the following abbreviations.

- $\mathsf{R}_W[e]$ abbreviates $\mathsf{R}_W(e, (x,u,v)d[x,u,v])$,
- $C'[x,u]$ abbreviates $(\Pi y : B[x])C[\mathsf{app}(u,y)]$,
- $d'[x,u]$ abbreviates $d[x,u,(\lambda y : B[x])\mathsf{R}_W[\mathsf{app}(u,y)]]$,
- $\mathsf{sup}\ '[x,u]$ abbreviates $(\mathsf{sup}\ y : B[x])\mathsf{app}(u,y)$, and
- $C''[x,u]$ abbreviates $C[\mathbf{sup}'[x,u]]$.

# References


[Coq] `http://coq.inria.fr/`

[Voevodsky] `http://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations.html`