Weyl's predicative classical mathematics as a logic-enriched type theory

Weyl's Predicative Classical Mathematics as a Logic-Enriched Type Theory

ROBIN ADAMS and ZHAOHUI LUO
Dept of Computer Science, Royal Holloway, Univ of London

We construct a logic-enriched type theory $\mathrm{LTT_w}$ that corresponds closely to the predicative system of foundations presented by Hermann Weyl in *Das Kontinuum*. We formalise many results from that book in $\mathrm{LTT_w}$, including Weyl's definition of the cardinality of a set and several results from real analysis, using the proof assistant Plastic that implements the logical framework LF. This case study shows how type theory can be used to represent a non-constructive foundation for mathematics.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Lambda Calculus and Related Systems*; *Mechanical Theorem Proving*

General Terms: Theory

Additional Key Words and Phrases: Logic-enriched type theories, predicativism, formalisation of mathematics

1. INTRODUCTION

Since the start of the 20th century, many different formal logical systems have been proposed as foundations for mathematics. Each of these foundational systems provides a formal language in which the statements of mathematics can be written, together with axioms and rules of deduction by which, it is claimed, the theorems of mathematics can be deduced. These different systems were often motivated by different philosophical schools of thought on the question of which proofs and constructions in mathematics are valid. These schools include *constructivism*, which holds that the use of the axiom of excluded middle is illegitimate, and *predicativism*, which holds that definitions that involve a certain form of circularity are illegitimate.

Proof assistants or *proof checkers* are tools that help the user to construct formal proofs of theorems in these formal logical systems, and provide a guarantee of each proof's correctness. The logical systems known as *type theories* have proven particularly successful for this purpose. However, so far, the type theories that have been used have almost exclusively been *constructive*. Examples include the proof checker Coq [The Coq development team 2006] based on the type theory CIC [Bertot and Castéran 2004], and the proof checker Agda [Norell 2007] based

Authors' email addresses: {robin,zhaohui}@cs.rhul.ac.uk
This work was supported by the UK EPSRC research grants GR/R84092, GR/R72259, EP/D066638/1, the UK Leverhulme grant F/07-537/AA, and EU TYPES grant 510996.

on Martin-Löf Type Theory [Nordström et al. 1990].

We would like to investigate how the other schools in the foundations of mathematics may be formalised with type theories. As a first case study, we chose the classical, predicative system presented in Hermann Weyl's 1918 text *Das Kontinuum* [Weyl 1918]. In this book, Weyl investigates how much of the mathematical corpus can be retained if we restrict ourselves to predicative definitions and methods of proof. He presents a foundational system in which it is impossible to perform an impredicative definition. He proceeds to construct the real numbers and prove many theorems of mathematics within this system. It is an excellent example of a fully developed non-mainstream foundational system for mathematics.

In the system Weyl presents in *Das Kontinuum*, only *arithmetic* sets may be constructed. We can form the set

$$\{x \mid \phi[x]\}$$

only if the proposition $\phi[x]$ is *arithmetic* — that is, it does not involve quantification over sets. Weyl's aim was to investigate how much mathematics we can reconstruct while restricting ourselves to the arithmetic sets. He shows that many results that are usually proven impredicatively can be proven predicatively; and that, even for those results that cannot, one can often prove a weaker result which in practice is just as useful.

We have constructed a system $\mathrm{LTT_w}$ that corresponds very closely to the semi-formal system that Weyl presented. Our system $\mathrm{LTT_w}$ is a *logic-enriched type theory* (LTT), a type theory extended with a separate mechanism for forming and proving propositions. It contains types of natural numbers, ordered pairs and functions; classical predicate logic with equality, together with the ability to prove propositions by induction; and the ability to form sets by predicative definition.

We have used the proof assistant Plastic to formalise in $\mathrm{LTT_w}$ many of the theorems and proofs presented in Weyl [1918]. The work presented here forms a case study in how type theory — specifically logic-enriched type theories — may be used outside the realm of constructive mathematics, to construct foundational systems in such a way that it is practicable to carry out machine-supported formalisation of proofs.

1.1 Outline

In Section 2, we give some background on logic-enriched type theories and the historical context to Weyl's work. In Section 3, we describe in detail the version of Weyl's foundational system we shall be using. We proceed in Section 4 to describe a logic-enriched type theory within a modified version of the logical framework LF[1] [Luo 1994]. We claim that this logic-enriched type theory faithfully corresponds to the system presented in the preceding section. In Section 5, we describe the results proven in the formalisation, which was carried out in a modified version of the proof assistant Plastic [Callaghan and Luo 2001], an implementation of LF. In Section 6,

[1]The logical framework LF here is a Church-typed version of Martin-Löf's logical framework [Nordström et al. 1990], and is not to be confused with the Edinburgh LF [Harper et al. 1993]. Among other differences, LF allows the user to declare computation rules, and hence to specify type theories such as Martin-Löf's type theory [Nordström et al. 1990] and UTT [Luo 1994].

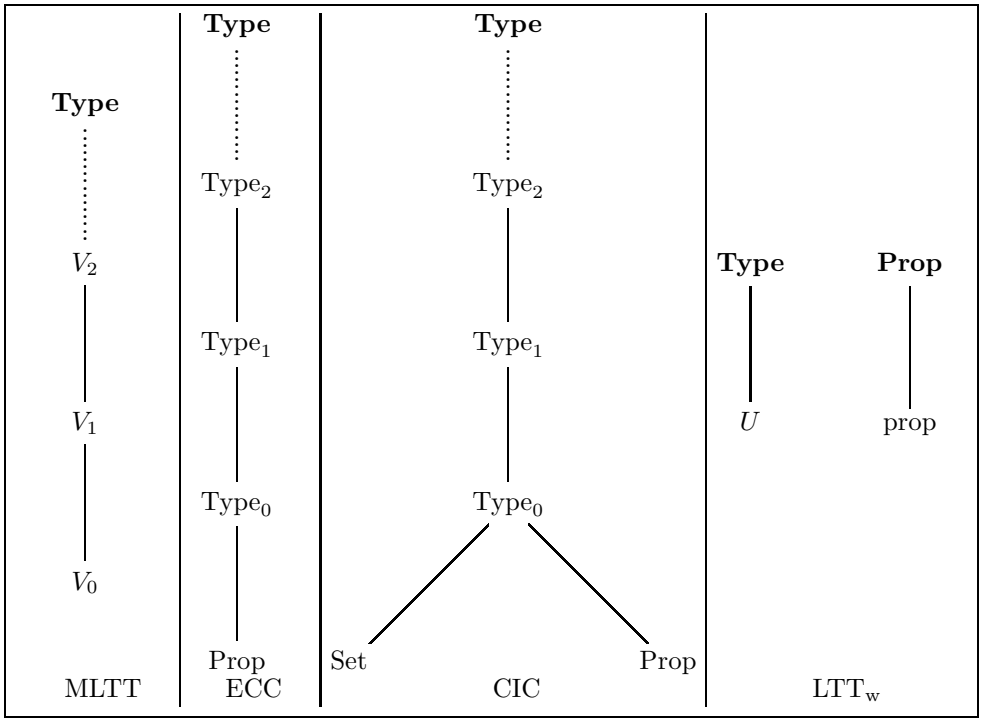

Fig. 1: The division of types into propositions and datatypes in several different type theories. A universe is placed under another, joined by a line, if the first is an object of the second. The kind **Type** is placed above the type universes, and the kind **Prop** above the propositional universe of $\mathrm{LTT_w}$.

we describe some of the other approaches to predicativity that have been followed, and discuss how they might be formalised in a similar way.

A preliminary version of this paper appeared in the proceedings of TYPES 2006 [Adams and Luo 2007]. The source code of the formalisation, together with a list of all the definitions and results in *Das Kontinuum*, is available at
`http://www.cs.rhul.ac.uk/~robin/weyl`

2. BACKGROUND

2.1 Type Theories for Non-constructive Mathematics

A type theory divides mathematical objects into *types*. The types themselves are often collected into *universes*. The usual method for using a type theory as a logical system is known as *propositions as types*: some or all of the types are identified with propositions, and the objects of each type with proofs of that proposition. We prove a theorem by constructing an object of the appropriate type. In Martin-Löf Type Theory (MLTT) [Nordström et al. 1990], every type is considered a proposition. In other type theories, such as ECC [Luo 1994] or CIC, the basis for the proof checker Coq [The Coq development team 2006], only some of the types are considered propositions, and these are collected into a universe, usually denoted by Prop. The other types are often called *datatypes* to distinguish them. Figure 1

shows the universe structure of several type theories.

When types are identified with propositions in this way, many natural type constructions correspond to the connectives of intuitionistic logic. For example, the non-dependent product $A \times B$ corresponds to conjunction ($A \wedge B$), and the dependent product $\Pi x : A.B$ corresponds to universal quantification ($\forall x : A.B$). That is, the introduction and elimination rules for $A \times B$ mirror the introduction and elimination rules for conjunction in intuitionistic logic; similarly, the rules for dependent product mirror those for universal quantification. Type theories are thus very well suited for formalising intuitionistic mathematics.

There are several ways in which a type theory may be modified so as to be appropriate for formalising classical mathematics. This cannot however be done without changing the structure of the datatypes, because the two interact so strongly. In MLTT, they are one and the same; in ECC or CIC, the universes Type_0, Type_1,... contain both propositions and datatypes.

It is possible to introduce constructions into the type theory so that the theory's rules now mirror the rules of deduction of classical logic, such as the 'freeze' and 'unfreeze' operations of the $\lambda\mu$-calculus [Parigot 1992]. However, doing so allows new objects to be formed in the datatypes.

There have also been several formalisations of classical proofs which used an intuitionistic type theory with additional axioms, such as Gonthier's proof of the Four Colour Theorem [Gonthier 2005], which extended the proof checker Coq with axioms for the real numbers that imply the axiom of excluded middle. However, this approach introduces non-canonical objects into the datatypes. (Further discussion on these points can be found in [Luo 2006].)

This problem does not arise in the systems known as *logic-enriched type theories* (LTTs) introduced by Aczel and Gambino [Aczel and Gambino 2002; Gambino and Aczel 2006]. These are type theories in which the propositions and the datatypes are completely separate. It is thus possible to introduce axioms or new rules of deduction without affecting the datatypes. We shall construct such an LTT in this paper, which we shall call $\mathrm{LTT}_{\mathrm{w}}$. Its universe structure is also shown in Figure 1.

There are several features of type theory that are of especial benefit for proof assistants: each object carries a type which gives information about that object, and the type theory itself has a primitive notion of computation. We contend that the intuitions behind type theory apply outside of intuitionistic mathematics, and that these advantages would prove beneficial when applied to other forms of proof. It is equally natural in classical mathematics to divide mathematical objects into types, and it would be of as much benefit to take advantage of the information provided by an object's type in a classical proof. The notion of computation is an important part of classical mathematics. When formally proving a property of a program, we may be perfectly satisfied with a classical proof, which could well be shorter or easier to find.

We further contend that it is worth developing and studying type theories specifically designed for non-constructive mathematical foundations. For this purpose, logic-enriched type theories would seem to be particularly appropriate.

2.2 Logic-Enriched Type Theories

The concept of an LTT, an extension of the notion of type theory, was proposed by Aczel and Gambino in their study of type-theoretic interpretations of constructive set theory [Aczel and Gambino 2002; Gambino and Aczel 2006]. A *type-theoretic framework*, which formulates LTTs in a logical framework, has been proposed in [Luo 2006] to support formal reasoning with different logical foundations. In particular, it adequately supports classical inference with a predicative notion of set, as described below.

An LTT consists of a type theory augmented with a separate, primitive mechanism for forming and proving propositions. We introduce a new syntactic class of *formulas*, and new judgement forms for a formula being a well-formed proposition, and for a proposition being provable from given hypotheses.

An LTT thus has two rigidly separated 'worlds': the *datatype* world of terms and types, and the *logical* world of proofs and propositions, for describing and reasoning about the datatype world. This provides two advantages over traditional type theories:

— We have separated the datatypes from the propositions. This allows us to add axioms without changing the datatype world. We can, for example, add the axiom of excluded middle without thereby causing all the datatypes of the form $A + (A \to \emptyset)$ to be inhabited.

— We do not have any computation rules on proofs. Further, a proof cannot occur inside a term, type or proposition. We are thus free to add any axioms we like to the logic: we know that, by adding the axiom of excluded middle (say), we shall not affect any of the properties of the reduction relation, such as decidability of convertibility or strong normalisation.

2.2.1 *Remark.* The clear separation between logical propositions and data types is an important salient feature of LTTs [Aczel and Gambino 2002; Gambino and Aczel 2006] and the associated logical framework [Luo 2006]. In Martin-Löf's type theory, for example, types and propositions are identified. The second author has argued, in the development of ECC/UTT [Luo 1994] as implemented in Lego/Plastic [Luo and Pollack 1992; Callaghan and Luo 2001], that it is unnatural to identify logical propositions with data types and there should be a clear distinction between the two. This philosophical idea was behind the development of ECC/UTT, where propositions are types, but not all types are propositions. LTTs have gone one step further – propositions and types are separate syntactic categories.

2.3 Foundations of Mathematics

When building a foundational system for mathematics, two of the decisions that must be made are:

(1) Whether the logic shall be *classical* or *intuitionistic*. In intuitionistic logic, principles such as excluded middle ($\phi \vee \neg\phi$) or $\neg\forall x\phi(x) \to \exists x\neg\phi(x)$ are not universally valid. A proof of $\phi \vee \psi$ must provide a way of deciding which of ϕ or ψ holds, and a proof of $\exists x\phi(x)$ must provide a way of constructing an x for which $\phi(x)$ holds.

(2) Whether *impredicative* definitions are allowed, or only *predicative*. A definition is *impredicative* if it involves a certain kind of 'vicious circle', in which an object is defined in terms of a collection of which it is a member. (A detailed discussion of the concept of predicativity and its history is given in Section 6.)

Each of the four possible combinations of these options has been advocated as a foundation for mathematics at some point in history.

— **Impredicative classical mathematics**. This is arguably the way in which the vast majority of practising mathematicians work. Zermelo-Fraenkel Set Theory (ZF) is one such foundation. The proof checker Mizar [Muzalewski 1993] has been used to formalise a very large body of impredicative classical mathematics.

— **Impredicative constructive mathematics**. Impredicative type theories such as CC [Coquand and Huet 1988], UTT [Luo 1994], and CIC [Bertot and Castéran 2004] are examples of such foundations. These have been implemented by the proof checkers LEGO [Pollack et al. 2001] and Coq [The Coq development team 2006].

— **Predicative classical mathematics**. This was the approach taken by Weyl in his influential monograph of 1918, *Das Kontinuum* [Weyl 1918]. Stronger predicative classical systems have been investigated by Feferman [Feferman 1964] and Schütte [Schütte 1965].

— **Predicative constructive mathematics**. Its foundations are provided, for example, by Martin-Löf's type theory [Nordström et al. 1990; Martin-Löf 1984].

LTTs may provide a uniform type-theoretic framework that can support formal reasoning with these four different logical foundations and others. This idea is discussed further in [Luo 2006].

In this paper, we present a case study in the type-theoretic framework: to construct an LTT to represent the predicative, classical foundational system of mathematics developed by Weyl in his monograph *Das Kontinuum* [Weyl 1918], and to formalise in that LTT several of the results proven in the book.

The system presented in the book has since attracted interest, inspiring for example the second-order system ACA_0 [Feferman 2000], which plays an important role in the project of Reverse Mathematics [Simpson 1999]. It is a prominent example of a fully developed non-mainstream mathematical foundation, and so a formalisation should be of quite some interest.

3. WEYL'S PREDICATIVE FOUNDATIONS FOR MATHEMATICS

Hermann Weyl (1885–1955) contributed to many branches of mathematics in his lifetime. His greatest contribution to the foundations of mathematics was the book *Das Kontinuum* [Weyl 1918] in 1918, in which he presented a predicative foundation which he showed was adequate for a large body of mathematics.

The semi-formal presentation of the foundational system in *Das Kontinuum* would not be acceptable by modern standards. Weyl does not give a rigorous formal definition of his syntax, axioms or rules of deduction. In this section, we shall give a formal definiton of a modern reconstruction of Weyl's foundational system.

The notation of our system shall differ considerably from Weyl's own. We shall also include several features not present in Weyl's system which are redundant in theory, but very convenient practically. The differences between our system and Weyl's shall be discussed under Section 3.1.6 below.

3.1 Weyl's Foundational System

Weyl's system is constructed according to these principles:

(1) The natural numbers are accepted as a primitive concept.

(2) Sets and relations can be introduced by two methods: explicit definitions, which must be *predicative*; and definition by recursion over the natural numbers.

(3) Statements about these objects have a definite truth value; they are either true or false.

Regarding point 2, we are going to provide ourselves with the ability to define sets by *abstraction*: given a formula $\phi[x]$ of the system, to form the set

$$S = \{x \mid \phi[x]\} \ . \tag{1}$$

In order to ensure that every such definition is predicative, we restrict which quantifiers can occur in the formula $\phi[x]$ that can appear in (1): we may quantify over natural numbers, but we may not quantify over sets or functions. In modern terminology, we would say that $\phi[x]$ must be *arithmetic*; that is, it must contain only first-order quantifiers.

3.1.1 *Components of Weyl's System.* Weyl divides the universe of mathematical objects into collections which he calls *categories*. The categories are divided into *basic* categories and *ideal* categories. Each category has *objects*. There are also *propositions*, which are divided into the *arithmetic*[2] propositions, and the *large* propositions.

3.1.2 *Categories*

(1) There is a basic category $\mathbb{N}$, whose objects are called *natural numbers*.

(2) For any two categories A and B, there is a category $A \times B$, whose objects are called *pairs*. If A and B are basic categories, then $A \times B$ is a basic category; otherwise, $A \times B$ is ideal.

(3) For any two categories A and B, there is a category $A \Rightarrow B$, whose objects are called *functions*. The category $A \Rightarrow B$ is always an ideal category.

(4) For any category A, there is a category $\mathrm{Set}\,(A)$, whose objects are called *sets*. The category $\mathrm{Set}\,(A)$ is always an ideal category.

For applications to other branches of mathematics, the system may be extended with other basic categories. For example, when formalising geometry, we may include a basic category of points and a basic category of lines.

[2]Weyl chose the German word *finite*, which in other contexts is usually translated as 'finite'; however, we agree with Pollard and Bole [Weyl 1994] that this would be misleading.

3.1.3 *Objects.* For each category A, there is a collection of *variables associated with* A.

(1) Every variable associated with a category A is an object of category A.

(2) There is an object 0 of the category $\mathbb{N}$.

(3) For every object n of the category $\mathbb{N}$, there is an object $\mathrm{s}(n)$ of the category $\mathbb{N}$, the *successor* of n.

(4) For every object a of category A and b of category B, there is an object (a, b) of category $A \times B$.

(5) For every object p of category $A \times B$, there is an object $\pi_1(p)$ of category A and an object $\pi_2(b)$ of category B.

(6) Let x be a variable associated with category A, and b an object of category B. Then there is an object $\lambda x.b$ of category $A \Rightarrow B$.

(7) For every object f of category $A \Rightarrow B$ and object a of category A, there is an object $f(a)$ of category B.

(8) Let x be a variable associated with category A, and ϕ an *arithmetic* proposition. Then there is an object $\{x \mid \phi\}$ of category $\mathrm{Set}\,(A)$.

(9) Let f be an object of category $A \Rightarrow A$. Then there is an object $\overline{f} : A \times \mathbb{N} \Rightarrow A$, the *iteration* of f. (Intuitively, $\overline{f}((x, n))$ is the result of applying f to x, n times.)

3.1.4 *Propositions*

(1) If A is a *basic* category, and a and b are objects of A, then there is an arithmetic proposition $a = b$.

(2) If a is an object of category A, and S an object of category $\mathrm{Set}\,(A)$, then there is an arithmetic proposition $a \in S$.

(3) If ϕ is a proposition, then $\neg\phi$ is a proposition. If ϕ is arithmetic, then $\neg\phi$ is arithmetic; if ϕ is large, then $\neg\phi$ is large.

(4) If ϕ and ψ are propositions, then $\phi \wedge \psi$, $\phi \vee \psi$ and $\phi \supset \psi$ are propositions. If ϕ and ψ are arithmetic, then these three propositions are arithmetic; if either ϕ or ψ is large, then these three propositions are large.

(5) If x is a variable associated with A and ϕ is a proposition, then $\forall x\phi$ and $\exists x\phi$ are propositions. If A is basic and ϕ is arithmetic, then these two propositions are arithmetic. If A is ideal or ϕ is large, then these two propositions are large.

We define an operation of substitution $[a/x]E$ that avoids variable capture. Here a is an object and x a variable of the same category, and E is either an object or a proposition. We omit the details of the definition.

We write $\phi \leftrightarrow \psi$ for $(\phi \supset \psi) \wedge (\psi \supset \phi)$.

We also define an equality relation on every category. For any category A and objects a and b of A, we define the proposition $a =_A b$ as follows.

—If A is a basic category, then $a =_A b$ is the proposition $a = b$.
—If A is $B \times C$ and either B or C is ideal, then $a =_A b$ is the proposition

$$\pi_1(a) =_B \pi_1(b) \wedge \pi_2(a) =_C \pi_2(b) \ .$$

—If A is $B \Rightarrow C$, then $a =_A b$ is the proposition

$$\forall x.a(x) =_C b(x)$$

where x is associated with B.
—If A is $\mathrm{Set}\,(B)$, then $a =_A b$ is the proposition

$$\forall x(x \in a \leftrightarrow x \in b)$$

where x is associated with B.

3.1.5 *Axioms.* The theorems of Weyl's system are those that can be derived via *classical* predicate logic from the following axioms:

(1) For any basic category A,

$$\forall x.x = x$$
$$\forall x \forall y(x = y \supset \phi \supset [y/x]\phi)$$

where ϕ is any proposition, and x and y are associated with A.

(2) Peano's axioms for the natural numbers:

$$\forall x \neg(\mathrm{s}(x) = 0)$$
$$\forall x \forall y(\mathrm{s}(x) = \mathrm{s}(y) \supset x = y)$$
$$[0/x]\phi \supset \forall x(\phi \supset [\mathrm{s}(x)/x]\phi) \supset \forall x\phi$$

for any proposition ϕ, arithmetic or large.

(3) For any categories A and B,

$$\forall x \forall y.\pi_1((x, y)) =_A x$$
$$\forall x \forall y.\pi_2((x, y)) =_B y$$

where x is associated with A and y with B.

(4) For any categories A and B, and any object b of category B,

$$\forall y((\lambda x.b)(y) =_B [y/x]b)$$

where x and y are associated with A.

(5) For any category A and arithmetic proposition ϕ,

$$\forall y(y \in \{x \mid \phi\} \leftrightarrow [y/x]\phi)$$

where x and y are associated with A.

(6) For any category A and object f of category $A \Rightarrow A$,

$$\forall x(\overline{f}(x, 0) =_A x)$$
$$\forall x \forall n(\overline{f}((x, \mathrm{s}(n))) =_A \overline{f}((f(x), n)))$$

3.1.5.1 *Definition by Recursion.* The operation of iteration allows us to define functions by primitive recursion. Let $f : A \Rightarrow A$ and $g : A \times A \times \mathbb{N} \Rightarrow A$. Suppose we want to define the function $h : A \times \mathbb{N} \Rightarrow A$ such that

$$\begin{aligned} h(a, 0) &= f(a) \\ h(a, n+1) &= g(h(a, n), a, n) \end{aligned}$$

This can be done as follows. Define $k : A \times A \times \mathbb{N} \Rightarrow A \times A \times \mathbb{N}$ by

$$k(x, y, n) = (g(x, y, n), y, n + 1) \ .$$

Then

$$h(a, n) = \pi_1^3(\overline{k}((f(a), a, 0), n)) \ ,$$

where $\pi_1^3(a, b, c) = a$.

3.1.6 *Extensions to Weyl's System.* There are two features in our system which were not explicitly present in Weyl's system, but which can justifiably be seen as conservative extensions of the same.

Weyl did not have the categories $A \times B$ of pairs, and did not have all of the function categories $A \Rightarrow B$ of our system. Instead, apart from the basic categories, the categories of Weyl's system were those of the form

$$\mathrm{Set}\,(B_1 \times \cdots \times B_n) \text{ and } A_1 \times \cdots \times A_m \Rightarrow \mathrm{Set}\,(B_1 \times \cdots \times B_n)$$

in our notation.

Instead of a category $\mathbb{N} \Rightarrow \mathbb{N}$, functions from $\mathbb{N}$ to $\mathbb{N}$ in *Das Kontinuum* are sets of ordered pairs, of category $\mathrm{Set}\,(\mathbb{N} \times \mathbb{N})$.

Weyl allowed the iteration only of functions from a category $\mathrm{Set}\,(A_1 \times \cdots \times A_n)$ to itself (the 'Principle of Iteration' [Weyl 1994, p. 36]). He showed by example how definition by recursion is then possible: addition is defined by iterating a suitable function from $\mathrm{Set}\,(\mathbb{N} \times \mathbb{N})$ to $\mathrm{Set}\,(\mathbb{N} \times \mathbb{N})$ [Weyl 1994, p. 51], and multiplication is defined in an 'entirely analogous' manner [Weyl 1994, p. 53].

We have also deviated from Weyl's system in two more minor ways. We have used a primitive operation $\mathrm{s}(x)$ for successor, where Weyl used a binary relation Sxy. We choose to start the natural numbers at 0, where Weyl begins at 1.

4. WEYL'S FOUNDATION AS A LOGIC-ENRICHED TYPE THEORY

A modern eye reading *Das Kontinuum* is immediately struck by how similar the system presented there is to what we now know as a type theory; almost the only change needed is to replace the word 'category' with 'type'. In particular, Weyl's system is very similar to a *logic-enriched type theory* (LTT).

The LTT we shall construct, which we shall call $\mathrm{LTT}_{\mathrm{w}}$, must involve:

—natural numbers, ordered pairs and functions;

—predicate logic;

—the ability to prove propositions by induction;

—the formation of *sets*.

We shall need to divide our types into *small* and *large* types, corresponding to Weyl's basic and ideal categories. We shall also need to divide our propositions

(1) Rules of Deduction for **Type** and El

$$\frac{\Gamma \text{ valid}}{\Gamma \vdash \mathbf{Type} \text{ kind}} \qquad \frac{\Gamma \vdash A : \mathbf{Type}}{\Gamma \vdash El\,(A) \text{ kind}} \qquad \frac{\Gamma \vdash A = B : \mathbf{Type}}{\Gamma \vdash El\,(A) = El\,(B)}$$

(2) Rules of Deduction for **Prop** and Prf

$$\frac{\Gamma \text{ valid}}{\Gamma \vdash \mathbf{Prop} \text{ kind}} \qquad \frac{\Gamma \vdash P : \mathbf{Prop}}{\Gamma \vdash Prf\,(P) \text{ kind}} \qquad \frac{\Gamma \vdash P = Q : \mathbf{Prop}}{\Gamma \vdash Prf\,(P) = Prf\,(Q)}$$

Fig. 2: Kinds $Type$ and $Prop$ in LF′

into *arithmetic* and *large* propositions. To make these divisions, we shall use a *type universe* and a *propositional universe*.

There exist today many *logical frameworks*, designed as systems for representing many different type theories. It requires only a small change to make a logical framework capable of representing LTTs as well. We have constructed LTT_{w} within a variant of the logical framework LF [Luo 1994], which is implemented by the proof checker Plastic [Callaghan and Luo 2001].

4.1 Logic-Enriched Type Theories in Logical Frameworks

Recall that a logical framework, such as LF, is intended as a metalanguage for constructing various type theories, the *object systems*. The frameworks consist of *kinds* and *objects*. An object system is constructed in a framework by making certain *declarations*, which extend the framework with new constants and rules of deduction.

A type theory divides mathematical objects, or *terms*, into *types*. The framework LF provides a kind **Type** and a kind constructor El. The intention is that LF be used for constructing a type theory, with the types represented by the objects of kind **Type**, and the terms of the type A by the objects of kind $El\,(A)$. Further details can be found in [Luo 1994].

A *logic-enriched type theory* has two new syntactic categories: besides terms and types, there are *propositions* and *proofs*. To make LF capable of representing LTTs, we add a kind **Prop** and a kind constructor Prf. We shall refer to this extended framework as LF′. The rules of deduction for these new kinds **Prop** and $Prf\,(\cdots)$ are given in Figure 2, along with the rules those for **Type** and El, for comparison. The full syntax and rules of deduction of LF′ are given in Appendix A.

We construct an LTT in LF′ by representing:

—the *types* by the objects of kind **Type**;

—the *terms* of type A by the objects of kind $El\,(A)$;

—the *propositions* by the objects of kind **Prop**;

—the *proofs* of the proposition ϕ by the objects of kind $Prf\,(\phi)$.

Natural Numbers

$$
\begin{aligned}
\mathbb{N} &: \mathbf{Type} \\
0 &: \mathbb{N} \\
\mathrm{s} &: \mathbb{N} \to \mathbb{N} \\
\mathrm{E}_{\mathbb{N}} &: (C : \mathbb{N} \to \mathbf{Type})C0 \to ((x : \mathbb{N})Cx \to C(\mathrm{s}x)) \to (n : \mathbb{N})Cn \\
\mathrm{Ind}_{\mathbb{N}} &: (P : \mathbb{N} \to \mathbf{Prop})Prf\,(P0) \to ((x : \mathbb{N})Prf\,(Px) \to Prf\,(P(\mathrm{s}x))) \to \\
&\qquad (n : \mathbb{N})Prf\,(Pn)
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}_{\mathbb{N}}\,C\,a\,b\,0 &= a : El\,(C0) \\
\mathrm{E}_{\mathbb{N}}\,C\,a\,b\,(\mathrm{s}n) &= b\,n\,(E_{\mathbb{N}}\,C\,a\,b\,n) : El\,(C(\mathrm{s}n))
\end{aligned}
$$

Implication

$$
\begin{aligned}
\supset &: \mathbf{Prop} \to \mathbf{Prop} \to \mathbf{Prop} \\
\supset\mathrm{I} &: (P : \mathbf{Prop})(Q : \mathbf{Prop})(Prf\,(P) \to Prf\,(Q)) \to Prf\,(P \supset Q) \\
\supset\mathrm{E} &: (P : \mathbf{Prop})(Q : \mathbf{Prop})Prf\,(P \supset Q) \to Prf\,(P) \to Prf\,(Q)
\end{aligned}
$$

Peirce's Law

$$\mathrm{Peirce} : (P : \mathbf{Prop})(Q : \mathbf{Prop})((Prf\,(P) \to Prf\,(Q)) \to Prf\,(P)) \to Prf\,(P)$$

Fig. 3: Some of the constants involved in the declaration of $\mathrm{LTT_w}$ in $\mathrm{LF'}$

4.1.1 *Example.* When constructing an LTT in $\mathrm{LF'}$, we can include conjunction by making the following declarations:

$$
\begin{aligned}
\wedge &: \mathbf{Prop} \to \mathbf{Prop} \to \mathbf{Prop} \\
\wedge I &: (p, q : \mathbf{Prop})Prf\,(p) \to Prf\,(q) \to Prf\,(\wedge pq) \\
\wedge E1 &: (p, q : \mathbf{Prop})Prf\,(\wedge pq) \to Prf\,(p) \\
\wedge E2 &: (p, q : \mathbf{Prop})Prf\,(\wedge pq) \to Prf\,(q)
\end{aligned}
$$

This has the effect of extending the logical framework with the constants $\wedge$, $\wedge I$, $\wedge E1$ and $\wedge E2$. The first allows propositions of the form $\phi \wedge \psi$ to be formed, and the last three are the introduction and elimination rules.

4.2 Natural Numbers, Products, Functions and Predicate Logic

We can now proceed to construct a logic-enriched type theory $\mathrm{LTT_w}$ that corresponds to the foundational system Weyl presents in *Das Kontinuum*. In the body of this paper, we shall describe a few of the declarations that comprise the specification of $\mathrm{LTT_w}$ in $\mathrm{LF'}$. The full list of declarations is given in Appendix B.

Our starting point is an LTT that contains, in its datatype world, a type $\mathbb{N}$ of natural numbers, as well as non-dependent product and function types $A \times B$ and $A \Rightarrow B$; and, in its logical world, classical predicate logic. We present some of the declarations involved in its specification in Figure 3, namely those involving natural numbers and implication. The rules for natural numbers include the elimination rule, which allows the definition of functions by recursion over $\mathbb{N}$, and the rule for proof by induction.

4.3 Type Universes and Propositional Universes

We have introduced our collection of types. We now divide them into the small and large types.

We use a familiar device to do this: a *type universe*. A type universe U is a type whose objects are names of types[3]. The types that have a name in U are the *small* types, and those that do not (such as U itself) as the *large* types. We also introduce a constructor T. For each name $a : U$, the type $T(a)$ is the type named by a.

For our system, we need a universe U that contains a name for $\mathbb{N}$, and a method for constructing a name for $A \times B$ out of a name for A and a name for B. We also need to introduce a relation of equality for every small type. These are both done by the constants declared in Figure 4.

We also need to divide our propositions into the arithmetic propositions and the large propositions. To do so, we use the notion in the logical world which is analagous to a type universe: a *propositional universe*.

We wish to introduce the collection 'prop' of names of the *arithmetic* propositions; that is, the propositions that only involve quantification over small types. We also introduce the constructor V; given P : prop, $V(P)$ shall be the small proposition named by P.

It is not immediately obvious where the collection 'prop' should live. We choose to declare prop : **Prop**, and use the objects of kind Prf (prop) for the names of the small propositions.

Now, it must be admitted that prop is not conceptually a proposition; it does not assert any relation to hold between any mathematical objects. We could have declared prop : **Type** instead. It makes no practical difference for this formalisation which choice is made.

We chose to declare prop : **Prop** as this provides a pleasing symmetry with U and **Type**, and prop seems to belong more to the logical world than the datatype world. Until more foundational work on LTTs has been done, we accept this compromise: prop is a 'proposition', each of whose 'proofs' is a name of an arithmetic proposition.[4] We discuss this matter further in Section 4.5.

The declarations associated with prop are given in Figure 4(3). Note that the propositional universe provides us with our first examples of computation rules for propositions.

4.4 The Predicative Notion of Set

We now have all the machinery necessary to be able to introduce *typed sets*. For any type A, we wish to introduce the type Set (A) consisting of all the sets that can be formed, each of whose members is an object of type A. (Thus we do not have any sets of mixed type.) We take a set to be introduced by an *arithmetic predicate* over A; that is, an object of kind $A \to Prf$ (prop), a function which takes objects of A and returns (a name of) an arithmetic proposition.

[3]Such a universe is called a universe *à la Tarski*, as opposed to a universe *à la Russell*, where the objects of the universe are themselves types.

[4]Other alternatives would be to introduce a new top-kind to hold prop, or to make prop itself a top-kind. We do not discuss these here.

(1) **The Type Universe**

$$\begin{aligned} U &: \mathbf{Type} \\ T &: U \to \mathbf{Type} \\ \hat{\mathbb{N}} &: U \\ \hat{\times} &: U \to U \to U \\ T\hat{\mathbb{N}} &= \mathbb{N} : \mathbf{Type} \\ T(a\hat{\times}b) &= Ta \times Tb : \mathbf{Type} \end{aligned}$$

(2) **Propositional Equality**

$$\begin{aligned} \simeq &: (A:U)TA \to TA \to \mathbf{Prop} \\ \simeq I &: (A:U)(a:TA)Prf\,(a \simeq_A a) \\ \simeq E &: (A:U)(P:TA \to \mathbf{Prop})(a,b:TA) \\ &\qquad Prf\,(a \simeq_A b) \to Prf\,(Pa) \to Prf\,(Pb) \end{aligned}$$

(3) **The Propositional Universe**

$$\begin{aligned} \mathrm{prop} &: \mathbf{Prop} \\ V &: Prf\,(\mathrm{prop}) \to \mathbf{Prop} \\ \hat{\bot} &: Prf\,(\mathrm{prop}) \\ \hat{\supset} &: Prf\,(\mathrm{prop}) \to Prf\,(\mathrm{prop}) \to Prf\,(\mathrm{prop}) \\ \hat{\forall} &: (a:U)(Ta \to Prf\,(\mathrm{prop})) \to Prf\,(\mathrm{prop}) \\ \hat{\simeq} &: (a:U)Ta \to Ta \to Prf\,(\mathrm{prop}) \\ V(\hat{\bot}) &= \bot : \mathbf{Prop} \\ V(p\hat{\supset}q) &= Vp \supset Vq : \mathbf{Prop} \\ V(\hat{\forall}a\,p) &= \forall(Ta)\,[x:Ta]V(px) : \mathbf{Prop} \\ V(s\hat{\simeq}_a t) &= s \simeq_a t : \mathbf{Prop} \end{aligned}$$

Fig. 4: A type universe and a propositional universe

We therefore include the following in our LTT:

— Given any type A, we can form the type $\mathrm{Set}\,(A)$. The terms of $\mathrm{Set}\,(A)$ are all the sets that can be formed whose elements are terms of type A.

— Given a *arithmetic* proposition $\phi[x]$ with variable x of type A, we can form the set $\{x : A \mid \phi[x]\} : \mathrm{Set}\,(A)$.

— If $a : A$ and $X : \mathrm{Set}\,(A)$, we can form the proposition $a \in X$, which is arithmetic.

— Finally, we want to ensure that the elements of the set $\{x : A \mid \phi[x]\}$ are precisely the terms a such that $\phi[a]$ is true. This is achieved by adding a *computation rule on propositions*:

$$a \in \{x : A \mid \phi[x]\} \text{ computes to } \phi[a] \ . \tag{2}$$

$$\begin{aligned} \mathrm{Set} &: \mathbf{Type} \to \mathbf{Type} \\ \mathrm{set} &: (A : \mathbf{Type})(A \to Prf\,(\mathrm{prop})) \to \mathrm{Set}\,(A) \\ \in &: (A : \mathbf{Type})A \to \mathrm{Set}\,(A) \to Prf\,(\mathrm{prop}) \\ \in A\,a\,(\mathrm{set}\,A\,P) &= Pa : Prf\,(\mathrm{prop}) \end{aligned}$$

Fig. 5: The Predicative Notion of Set

The type $\mathrm{Set}\,(A)$ is to be a large type; we do not provide any means for forming a name of $\mathrm{Set}\,(A)$ in U.

We therefore make the declarations in Figure 5. In particular, the constants listed there allow us to form the following objects:

—For every type A, the type $\mathrm{Set}\,(A)$.

—For every type A and name $p[x] : Prf\,(\mathrm{prop})$ of an arithmetic proposition, the object $\mathrm{set}\,A\,([x : A]p[x])$, which represents the set $\{x : A \mid V(p[x])\}$ in $\mathrm{Set}\,(A)$.

—Given objects $a : A$ and $X : \mathrm{Set}\,(A)$, the object $\in A\,a\,X$, which represents a name of the proposition $a \in X$.

The computation rule (2) is represented by declaring: $\in A\,a\,(\mathrm{set}\,A\,P)$ computes to $P\,a$.

4.4.1 *Remarks*

(1) We have introduced here a *predicative* notion of set. It is impossible in $\mathrm{LTT_w}$ to perform a definition that is impredicative in Weyl's sense. This restriction on which sets can be formed, proposed by Weyl and formalised in $\mathrm{LTT_w}$, is by no means the only way of defining a predicative notion of set. Historically, there have been a number of different ways of interpreting the concept of 'predicativity', and a number of different predicative formal systems that all impose different restrictions on which sets and functions may be defined. We discuss this matter further in Section 6.

(2) It would be easy to modify the system to use instead the *impredicative* notion of set. This would involve changing the declaration of either prop, U or Set; the details are given in Section 5.2. When we declare both the predicative and the impredicative systems in this way, all the proofs in the predicative system can be reused without change in the impredicative. Further discussion of all these points may be found in [Luo 2006].

4.5 Anomalies in $\mathrm{LTT_w}$

The system $\mathrm{LTT_w}$ has certain features that are anomalies, unwanted side-effects of our design choices.

We chose to make the type universe U an object of kind **Type**. This means that the objects of kind **Type** no longer correspond to the categories of Weyl's system. There are certain extra objects in **Type**: U itself, as well as $U \times U$, $U \Rightarrow U$, $\mathbb{N} \Rightarrow U$, and so forth. We can therefore do more in $\mathrm{LTT_w}$ than we can in

Weyl's system, such as define functions $\mathbb{N} \Rightarrow U$ by recursion, and hence define meta-functions $\mathbb{N} \to \mathbf{Type}$ by recursion. For example, we can define the meta-function $f : (\mathbb{N} \to \mathbf{Type}$ where

$$f(n) = \overbrace{\mathbb{N} \times \mathbb{N} \times \cdots \times \mathbb{N}}^{n} \ .$$

We also chose to make prop an object of kind **Prop**. This results in the objects of kind **Prop** no longer corresponding to the propositions of Weyl's system. There are extra objects in **Prop**: prop itself, as well as prop $\supset$ prop, $\forall x : \mathbb{N}$.prop, and so forth. If we had placed prop in **Type** instead, we would get more anomalous types, and the ability to define functions into prop by recursion.

These anomalies would not arise if we made U and prop top-kinds, instead of placing them in **Type** and **Prop**. The choice of where to place the universes makes no practical difference for this formalisation, because we did not make use of any of these anomalous objects in our formalisation. We have checked that our proof scripts still parse if we declare prop : **Type**.

It seems likely that these anomalies are harmless. We conjecture that $\mathrm{LTT_w}$ is a conservative extension of the system in which U and prop are top-kinds. However, it also seems likely that this would not be true for stronger LTTs. What effect the placement of universes has in LTTs is a question that needs further investigation.

5. FORMALISATION IN PLASTIC

We have formalised this work in a version of the proof assistant Plastic [Callaghan and Luo 2001], modified by Paul Callaghan to be an implementation of LF'. We have produced a formalisation which includes all the definitions and proofs of several of the results from Weyl's book.

In Plastic, all lines that are to be parsed begin with the character `>`; any line that does not is a comment line. A constant c may be declared to have kind $(x_1 : K_1) \cdots (x_n : K_n)K$ by the input line

$$\texttt{> [}c\texttt{[}x_1\texttt{:}K_1\texttt{]} \cdots \texttt{[}x_n\texttt{:}K_n\texttt{]} \ \texttt{:} \ \ K\texttt{];}$$

We can define the constant c to be the object $[x_1 : K_1] \cdots [x_n : K_n]k$ of kind $(x_1 : K_1) \cdots (x_n : K_n)K$ by writing

$$\texttt{> [}c\texttt{[}x_1\texttt{:}K_1\texttt{]} \cdots \texttt{[}x_n\texttt{:}K_n\texttt{]} \ \texttt{=} \ k \ \texttt{:} \ \ K\texttt{];}$$

In both these lines, the kind indicator $:K$ is optional, and is usually omitted.

We can make any argument implicit by replacing it with a 'meta-variable' `?`, indicating that we wish Plastic to infer its value.

5.1 Results Proven

5.1.1 *Peano's Fourth Axiom.* Peano's fourth axiom is the proposition

$$\forall x : \mathbb{N}.\mathrm{s}(x) \neq 0 \ .$$

This can be proven in $\mathrm{LTT_w}$ by taking advantage of the fact that we can define functions from $\mathbb{N}$ to Set $(\mathbb{N})$ by recursion.

Define the meta-function $f : \mathbb{N} \to \mathrm{Set}\,(\mathbb{N})$ as follows.

$$\begin{aligned} f(0) &= \emptyset \\ f(n+1) &= \{x : \mathbb{N} \mid \top\} \end{aligned}$$

Now, we have $0 \in f(n+1)$. If $n+1 = 0$, we would have $0 \in f(0) = \emptyset$, which is a contradiction. Therefore, $n+1 \neq 0$.

Peano's fourth axiom is often surprisingly difficult to prove in a type theory, requiring a universe or something equally strong. This is true in logic-enriched type theories, too; if we remove U, prop and Set from $\mathrm{LTT_w}$, then Peano's fourth axiom cannot be proven in the resulting system. (Proof: the resulting system has a model in which every type has exactly one object.)

5.1.2 *Cardinality of Sets.* In Weyl's system, we can define the predicate 'the set X has exactly n members' in the following manner.

Given a basic category A, define the function $K : \mathbb{N} \Rightarrow \mathrm{Set}\,(\mathrm{Set}\,(A))$ by recursion as follows. The intention is that $K(n)$ is the set of all sets $X : \mathrm{Set}\,(A)$ that have at least n members.

$$\begin{aligned} K(0) &= \{X : \mathrm{Set}\,(A) \mid \top\} \\ K(n+1) &= \{X : \mathrm{Set}\,(A) \mid \exists a : A(a \in X \wedge X \setminus \{a\} \in K(n))\} \end{aligned}$$

In Plastic, this is done as follows:

```
> [at_least_set [tau : U] = E_Nat ([_ : Nat] Set (Set (T tau)))
>     (full (Set (T tau)))
>     [n : Nat] [Kn : Set (Set (T tau))] set (Set (T tau))
>        [X : Set (T tau)] ex tau [a : T tau]
>           and (in (T tau) a X) (in ? (setminus' tau X a) Kn)];
```

We define the proposition 'X has at least n members' to be $X \in K(n)$.

```
> [At_Least [tau : U] [X : Set (T tau)] [n : Nat]
>    = In ? X (at_least_set tau n)];
```

For n a natural number, define the *cardinal number* $\overline{n}$ to be $\{x : \mathbb{N} \mid x < n\}$.

```
> [card [n : Nat] = set Nat [x : Nat] lt x n];
```

Define the *cardinality* of a set A to be $|A| = \{n : \mathbb{N} \mid A \text{ has at least } \mathrm{s}(n) \text{ members}\}$.

```
> [cardinality [tau : U] [A : Set (T tau)]
>    = set Nat [n : Nat] at_least tau A (succ n)];
```

We can prove the following result:

The cardinality $|X|$ of a set X is either $\{x : \mathbb{N} \mid \top\}$ or $\overline{n}$ for some n.

We thus have two classes of cardinal numbers: $\overline{n}$, for measuring the size of finite sets, and $\{x : \mathbb{N} \mid \top\}$, which we denote by ∞, for measuring the size of infinite sets. (There is thus only one infinite cardinality in *Das Kontinuum*.) We define 'X has exactly n members' to be $|X| =_{\mathrm{Set}(\mathbb{N})} \overline{n}$.

```
> [infty = full Nat];
> [Exactly [tau : U] [A : Set (T tau)] [n : Nat]
>    = Seteq Nat (cardinality tau A) (card n)];
```

With these definitions, we can prove results such as the following:

(1) If A has at least n elements and $m \leq n$, then A has at least m elements.
(2) If A has exactly n elements, then $m \leq n$ iff A has at least m elements.
(3) If A has exactly m elements, B has exactly n elements, and A and B are disjoint, then $A \cup B$ has exactly $m + n$ elements.

We have thus provided definitions of the concepts 'having at least n members' and 'having exactly n members' in such a way that the sets $\{X : \mathrm{Set}\,(A) \mid X \text{ has at least } n \text{ members}\}$ and $\{X : \mathrm{Set}\,(A) \mid X \text{ has exactly } n \text{ members}\}$ are definable predicatively. This would not be possible if, for example, we defined 'X has exactly n elements' as the existence of a bijection between X and $\overline{n}$; we would have to quantify over the ideal category $A \Rightarrow \mathbb{N}$. It also cannot be done as directly in a predicative system of second order arithmetic such as ACA_0 [Simpson 1999].

5.1.3 *Construction of the Reals.* The set of real numbers is constructed by the following process. We first define the type of integers $\mathbb{Z}$, with a defined relation of equality $\approx_{\mathbb{Z}}$. We then define a *rational* to be a pair of integers, the second of which is non-zero. That is, for $q : \mathbb{Z} \times \mathbb{Z}$, we define '$q$ is rational' by

$$\langle x, y\rangle \text{ is rational} \equiv y \not\approx_{\mathbb{Z}} 0 \ .$$

We proceed to define equality of rationals ($q \approx_{\mathbb{Q}} q'$), addition, multiplication and ordering on the rationals.

A *real* is a Dedekind cut of rationals; that is, an object R of the category $\mathrm{Set}\,(\mathbb{Z} \times \mathbb{Z})$ that:

—is a *domain of rationals*; if q and q' are rationals, $q \in R$, and $q \approx_{\mathbb{Q}} q'$, then $q' \in R$;
—is *closed downwards*; if q and q' are rationals, $q \in R$, and $q' < q$, then $q' \in R$;
—has no maximal element; for every rational $q \in R$, there exists a rational $q' \in R$ such that $q < q'$;
—and is neither empty nor full; there exists a rational q such that $q \in R$, and a rational q' such that $q' \notin R$.

Equality of reals is defined to be extensional equality restricted to the rationals:

$$R \approx_{\mathbb{R}} S \equiv \forall q(q \text{ is rational} \supset (q \in R \leftrightarrow q \in S))$$

We note that, in this formalisation, we could define the collection of integers as a type, because every pair of natural numbers is an integer. In contrast, there was no way to define the collection of rationals as a type, say as the 'sigma-type' '$(\Sigma q : \mathbb{Z} \times \mathbb{Z})q$ is rational'. This is because our LTT offers no way to form a type from a type $\mathbb{Z} \times \mathbb{Z}$ and a proposition 'q is rational'. We are, however, able to form the *set* $\mathbb{Q} = \{q : \mathbb{Z} \times \mathbb{Z} \mid q \text{ is rational}\} : \mathrm{Set}\,(\mathbb{Z} \times \mathbb{Z})$. Likewise, we cannot form the type of reals, but we can form the set of reals $\mathbb{R} : \mathrm{Set}\,(\mathrm{Set}\,(\mathbb{Z} \times \mathbb{Z}))$.

5.1.4 *Real Analysis.* Weyl was keen to show that his predicative system was strong enough to be used for mathematical work by demonstrating that, while several traditional theorems cannot be proven within it, we can usually prove a version of the theorem that is only slightly weaker.

For example, we cannot predicatively prove the *least upper bound principle*: that every set A of real numbers bounded above has a least upper bound l. Impredicatively, we would define l to be the union of A. This cannot be done predicatively, as it involves quantification over real numbers. However, we can prove the following two statements, which are enough for most practical purposes:

(1) Every set S of *rational* numbers bounded above has a unique (real) least upper bound l. Take $l = \{q \in \mathbb{Q} \mid (\exists q' \in S)q < q'\}$.

(2) Every *sequence* $r_1, r_2, \ldots$ of real numbers bounded above has a unique least upper bound l. Take $l = \{q \in \mathbb{Q} \mid (\exists n : \mathbb{N})q \in r_n\}$.

These involve only quantification over the rationals and the natural numbers, respectively. (We note that either of these is equivalent to the least upper bound principle in an impredicative setting.)

The first is enough to prove the classical Intermediate Value Theorem:

> If $f : \mathrm{Set}\,(\mathbb{Z} \times \mathbb{Z}) \Rightarrow \mathrm{Set}\,(\mathbb{Z} \times \mathbb{Z})$ is a continuous function from the reals to the reals, and $f(a) < v < f(b)$ for some reals a, b, v with $a < b$, then there exists a real c such that $a < c < b$ and $f(c) = v$.

The predicative proof of this theorem takes advantage of the fact that a continuous function is determined by its values on the rationals. Weyl defined c to be the least upper bound of the set of all *rationals* q such that $a < q < b$ and $f(q) < v$. In the formalisation, we defined directly: $c = \{q \in \mathbb{Q} \mid (\exists q' \in \mathbb{Q})(q < q' < b \wedge f(q') < v)\}$.

5.2 An Impredicative Development

As mentioned in Section 4.4, it is not difficult to modify this formulation to get a development of the same theorems in an impredicative system. All we have to do is remove the distinction between large and arithmetic propositions.

We do this by adding two impredicative quantifiers to prop, together with their computation rules:

$$\overline{\forall} : (A : \mathbf{Type})(A \to Prf\,(\mathrm{prop})) \to Prf\,(\mathrm{prop}) \quad V(\overline{\forall}AP) = \forall A([x : A]V(Px))$$
$$\overline{\exists} : (A : \mathbf{Type})(A \to Prf\,(\mathrm{prop})) \to Prf\,(\mathrm{prop}) \quad V(\overline{\exists}AP) = \exists A([x : A]V(Px))$$

Now prop, which determines the collection of propositions over which sets can be formed, covers the whole of **Prop**. We can form the set $\{x : A \mid \phi[x]\}$ for any well-formed proposition $\phi[x]$. However, all our old proof files written in terms of prop and V still parse.

Once this change has been made, we can go on to prove the statement that every set of real numbers bounded above has a least upper bound.

There are other ways in which we could have made our system impredicative. We could have changed the construction of U, so that every type in **Type** has a name. The simplest way would be to erase the universes U and prop, and to change every instance of prop to **Prop** in the constants associated with Set (Figure 5). However, this would leave us unable to reuse the proofs that we had written for the predicative system. We could also have replaced the declarations of prop and V (Figure 4) with

$$\begin{array}{ll} \text{prop} = \mathbf{Prop} & V = [x : \mathbf{Prop}]\mathbf{Prop} \\ \hat{\bot} = \bot & \hat{\supset} = \supset \\ \hat{\forall} = [A : U]\forall(TA) & \hat{\simeq} = \simeq \end{array}$$

However, at present Plastic becomes unstable when equations are declared at the top-kind level such as prop = **Prop**.

6. RELATED WORK

Weyl's work in *Das Kontinuum* is by no means the only attempt to provide a predicative foundation to mathematics. Historically, there have been several different views on which definitions and proofs are predicatively acceptable. Of these, Weyl's in *Das Kontinuum* is one of the strictest. Many of these have been formalised as systems of second-order arithmetic. The question of to what extent they may be formalised by type theories, and the related question of which type theories may be considered predicative, have not received much attention before now.

6.1 Other Formalisations of *Das Kontinuum*

Feferman [Feferman 2000] has claimed that the system ACA_0 is "a modern formulation of Weyl's system." The system ACA_0 is a subsystem of second-order arithmetic, so it deals only with natural numbers and sets of natural numbers. A set $\{x \mid \phi[x]\}$ may only be introduced in ACA_0 for an *arithmetic* predicate $\phi[x]$: this is the 'arithmetic comprehension axiom' that gives the system its name. The stronger system ACA has also been studied, which has a stronger induction principle: ACA_0 has the induction axiom

$$\forall X(0 \in X \supset \forall x(x \in X \supset \mathrm{s}(x) \in X) \supset \forall x x \in X) \ ,$$

whereas ACA has the induction *schema*

$$\phi[0] \supset \forall x(\phi[x] \supset \phi[\mathrm{s}(x)]) \supset \forall x \phi[x]$$

for every formula $\phi[x]$.

Weyl does go beyond ACA_0 in two places: in his definition of the cardinality of a set ([Weyl 1994, Chapter 1 §7, pp. 38–39]), and the results proven using this definition ([Weyl 1994, Chapter 2 §1, pp.55f]). The definition requires the use of third-order sets (sets of sets), and the proofs use a stronger induction principle than the one found in ACA_0. When formalising these parts of the book (see Section 5.1.2 above), we needed to use the type $\mathrm{Set}\,(\mathrm{Set}\,(A))$, and to use $\mathrm{Ind}_{\mathbb{N}}$ with a large proposition.

The rest of *Das Kontinuum* can be formalised in ACA_0. It is also possible to find another definition of the cardinality of a set that can be formalised in ACA_0, and the results in *Das Kontinuum* can then be proven in ACA_0. However, these two sections of the book show that Weyl did intend his system to be more than second-order.

6.2 Other Approaches to Predicativity

The concept of predicativity originated with Poincaré and Russell in 1906–8, who held that the paradoxes of set theory each invoked a definition that involves a *vicious circle*; in each case, a set A is defined using a concept that presupposes the set A itself. They proposed the *vicious circle principle*, which holds that such definitions are illegitimate. Poincaré wrote [Poincaré 1906]: "A definition containing a vicious circle defines nothing." Russell expressed the vicious circle principle as follows [Russell 1906]: "Whatever involves an apparent [bound] variable must not be among the possible values of that variable."

Russell proceeded to develop the *theory of types* [Russell 1908], a system of logic that strictly adheres to the vicious circle principle. The theory divides mathematical objects into types. These include the type of individuals; the type of 0th-order or *arithmetic* sets of individuals, sets that can be defined using quantifiers that range over individuals only; the type of 1st-order sets, sets that can be defined by quantifying over individuals and 0th-order sets; the type of 2nd-order sets, defined by quantification over individuals, 0th-order sets and 1st-order sets; and so forth[5].

This ramification has some undesirable consequences; for example, if we define real numbers as sets of rationals, we find we have 0th-order reals, 1st-order reals, and so forth. The usual development of analysis cannot be carried out in this framework: for example, the least upper bound property does not hold for any of these reals.

Russell's solution was to introduce the *Axiom of Reducibility*: for every nth-order set, there is a 0th-order set with the same members. This axiom effectively allows impredicative definitions to be made in the theory of types. Weyl's solution, as we have seen, was to allow only 0th-order sets.

Gödel [1944] suggested constructing sets of *transfinite* order; to form sets of order α, where α ranges over some initial segment of the ordinals. The sets of order $\alpha+1$ are those that can be formed by quantifying over the sets of order $\leq \alpha$; for each limit ordinal λ, the sets of order λ are the union of the sets of lower order. This idea was later named the *ramified analytic hierarchy* by Kleene [1959]. Kreisel [1960] that the predicatively acceptable sets should be identified with the sets of order α for α a recursive ordinal. Feferman [1964] and Schütte [1965] independently suggested that the predicatively acceptable sets should be identified with those of order $< \Gamma_0$. This latter idea has dominated the thinking on predicativity since.

[5]Russell's theory of types included many more types than those listed here. For a historical account, we refer to [Kamareddine et al. 2002].

6.3 Predicativity in Type Theory

In general, the type theory community has confined its interest in predicativity to the question of whether a *universe* is predicative.

Roughly, a universe is a type of types. A universe U is called *impredicative* if some of the types in U are constructed using U itself; otherwise, U is *predicative*. A type theory is called *predicative* if it uses no impredicative universes.

Martin-Löf's original type theory [Martin-Löf 1971] used a single universe V, which was highly impredicative as the system used the typing rule $V \in V$. After Girard [1972] showed that the system was inconsistent, Martin-Löf [1972] modified the system so that the universe became predicative. Later he extended the system with an infinite sequence of universes [1975], and with *W-types*, types of well-ordered trees [Martin-Löf 1982].

Apart from the universes and the W-types, all the types in Martin-Löf type theory are simple inductive definitions, and so can be defined by an explicit predicative definition given the natural numbers; in this sense, Martin-Löf type theory without W-types is predicative modulo the natural numbers. It was conjectured by Peter Hancock [Martin-Löf 1975] and proved independently by Peter Aczel and Feferman [Feferman 1982] that the proof-theoretic strength of Martin-Löf type theory, with infinitely many universes but without W-types, is Γ_0.

Including the W-types increases the strength of the system dramatically. Setzer [1993] has shown that the theory with W-types and just one universe has a proof-theoretic ordinal much larger than Γ_0. If one agrees with Feferman and Schütte, then W-types are an impredicative element in the type theory.

It is worth noting that consistent impredicative type theories have also been usefully employed, most notably the Calculus of Constructions [Coquand and Huet 1988] and its variants.

6.4 Predicativity in Logic-Enriched Type Theories

Logic-enriched type theories may provide a useful tool for investigating all these different conceptions of predicativity, as they provide a setting in which the three questions listed at the start of the section are kept separate. When designing a logic-enriched type theory, we are able to adjust three parameters independently.

(1) We can make the propositional universes stronger or weaker. This will determine which predicates may be used to define sets.

(2) We can include more or fewer types in each type universe. This will determine which inductive definitions may be used to define types.

(3) We can make the universes predicative or impredicative.

This flexibility means logic-enriched type theories form a very rich structure. It would likely be fruitful to explore this structure by constructing LTTs corresponding to different approaches to the foundations of mathematics, and investigating correspondences between these LTTs and both traditional type theories and systems of predicate logic. This would provide a setting in which we could experiment by (say) adding and removing generalised inductive types, in a way that is much more suitable for machine-assisted formalisation than are systems of predicate logic.

7. CONCLUSION

We have conducted a case study in Plastic of the use of a type-theoretic framework to construct a logic-enriched type theory as the basis for a formalisation of a non-constructive system of mathematical foundations, namely that presented in Weyl's *Das Kontinuum*. As a representation of Weyl's work, it is arguably better in some ways than such second-order systems as ACA_0 [Feferman 2000], since we can form a definition of the cardinality of a set that is much closer to Weyl's own. The formalisation work required only a minor change to the existing logical framework implemented in Plastic.

7.1 Further Work

We have seen how LTT_w corresponds very closely to Weyl's system. It is possible to embed the system ACA in LTT_w. We can also embed the system ACA_0 in LTT_w with the constant $\mathrm{Ind}_\mathbb{N}$ removed and replaced with

$$\begin{aligned}\mathrm{ind}_\mathbb{N} \;:\; &(P : \mathbb{N} \to Prf\,(\mathrm{prop}))Prf\,(V(P0)) \to \\ &((x : \mathbb{N})Prf\,(V(Px)) \to Prf\,(V(P(\mathrm{s}x)))) \to (n : \mathbb{N})Prf\,(V(Pn)) \quad ,\end{aligned}$$

which allows us to prove only *arithmetic* propositions by induction.

It is also possible to define a translation from LTT_w to Martin-Löf Type Theory with one universe extended by the axiom of excluded middle, and from ACA to LTT_w. Future work will involve investigating these translations further — whether they are conservative, and whether a translation can be defined in the opposite direction in each case.

We also hope to define more logic-enriched type theories corresponding to other schools in the foundations of mathematics; both other approaches to predicativity, and other schools entirely. For example, we anticipate we can construct a system corresponding to the Theory of Types by extending LTT_w with infinitely many type and propositional universes. A stronger universe construction should also allow us to capture Kreisel's construction of the ramified analytic hierarchy over the recursive ordinals. It remains to be seen whether we can capture Feferman-Schütte predicativity, or whether another principle more naturally suited to LTTs will be discovered.

It would also be interesting to carry out the impredicative development of analysis in our setting, reusing the code from the predicative development.

A. THE LOGICAL FRAMEWORK LF′

LF′ is an extension of the logical framework LF as described in Chapter 9 of [Luo 1994]. LF is a typed version of Martin-Löf's logical framework and LF′ extends LF by the propositional kind **Prop** and the associated rules.

Formally, the terms of LF′ are of the following forms:

$$\mathbf{Type},\ El(A),\ \mathbf{Prop}, Prf\,(P)\,,\ (x : K)K',\ [x : K]k',\ f(k),$$

where the free occurrences of variable x in K' and k' are bounded by the binding operators $(x : K)$ and $[x : K]$, respectively. We shall usually omit El when writing kinds. We write $K \to K'$ for $(x : K)K'$ when x does not occur free in K'.

Contexts and assumptions

$$\frac{}{\langle\rangle\ valid} \quad \frac{\Gamma \vdash K\ kind \quad x \notin FV(\Gamma)}{\Gamma, x : K\ valid} \quad \frac{\Gamma, x : K, \Gamma'\ valid}{\Gamma, x : K, \Gamma' \vdash x\colon K}$$

General equality rules

$$\frac{\Gamma \vdash K\ kind}{\Gamma \vdash K = K} \quad \frac{\Gamma \vdash K = K'}{\Gamma \vdash K' = K} \quad \frac{\Gamma \vdash K = K' \quad \Gamma \vdash K' = K''}{\Gamma \vdash K = K''}$$

$$\frac{\Gamma \vdash k\colon K}{\Gamma \vdash k = k\colon K} \quad \frac{\Gamma \vdash k = k'\colon K}{\Gamma \vdash k' = k\colon K} \quad \frac{\Gamma \vdash k = k'\colon K \quad \Gamma \vdash k' = k''\colon K}{\Gamma \vdash k = k''\colon K}$$

Equality typing rules

$$\frac{\Gamma \vdash k\colon K \quad \Gamma \vdash K = K'}{\Gamma \vdash k\colon K'} \quad \frac{\Gamma \vdash k = k'\colon K \quad \Gamma \vdash K = K'}{\Gamma \vdash k = k'\colon K'}$$

Substitution rules

$$\frac{\Gamma, x : K, \Gamma'\ valid \quad \Gamma \vdash k\colon K}{\Gamma, [k/x]\Gamma'\ valid}$$

$$\frac{\Gamma, x : K, \Gamma' \vdash K'\ kind \quad \Gamma \vdash k\colon K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K'\ kind} \quad \frac{\Gamma, x : K, \Gamma \vdash K'\ kind \quad \Gamma \vdash k = k'\colon K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K' = [k'/x]K'}$$

$$\frac{\Gamma, x : K, \Gamma' \vdash k'\colon K' \quad \Gamma \vdash k\colon K}{\Gamma, [k/x]\Gamma' \vdash [k/x]k'\colon [k/x]K'} \quad \frac{\Gamma, x : K, \Gamma' \vdash k'\colon K' \quad \Gamma \vdash k_1 = k_2\colon K}{\Gamma, [k_1/x]\Gamma' \vdash [k_1/x]k' = [k_2/x]k'\colon [k_1/x]K'}$$

$$\frac{\Gamma, x : K, \Gamma' \vdash K' = K'' \quad \Gamma \vdash k\colon K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K' = [k/x]K''} \quad \frac{\Gamma, x : K, \Gamma' \vdash k' = k''\colon K' \quad \Gamma \vdash k\colon K}{\Gamma, [k/x]\Gamma' \vdash [k/x]k' = [k/x]k''\colon [k/x]K'}$$

Fig. 6: Rules of deduction of LF′ (I)

There are five forms of judgements:

— Γ valid, which asserts that Γ is a valid context;
— $\Gamma \vdash K$ kind, which asserts that K is a kind;
— $\Gamma \vdash k : K$, which asserts that k is an object of kind K;
— $\Gamma \vdash k = k' : K$, which asserts that k and k' are equal objects of kind K;
— $\Gamma \vdash K = K'$, which asserts that K and K' are equal kinds.

The rules of deduction of LF′ are those of LF [Luo 1994] plus those involving **Prop**. For completeness, they are given in Figures 6 and 7.

B. SPECIFICATION OF $\mathrm{LTT_W}$

The following is the list of all the constant and equation declarations that comprise the specification of $\mathrm{LTT_w}$ within LF′. We write the constants $\supset$, $\hat{\supset}$, $\times$ and $\hat{\times}$ as infix; so we write $\supset\ \phi\,\psi$ as $\phi \supset \psi$. We shall also write $\simeq\ A\,a\,b$ as $a \simeq_A b$, and $\hat{\simeq}\,A\,a\,b$ as $a \hat{\simeq}_A b$.

The kind **Type**

$$\frac{\Gamma \ valid}{\Gamma \vdash \mathbf{Type} \ kind} \quad \frac{\Gamma \vdash A\colon \mathbf{Type}}{\Gamma \vdash El(A) \ kind} \quad \frac{\Gamma \vdash A = B\colon \mathbf{Type}}{\Gamma \vdash El(A) = El(B)}$$

The kind **Prop**

$$\frac{\Gamma \ valid}{\Gamma \vdash \mathbf{Prop} \ kind} \quad \frac{\Gamma \vdash P\colon \mathbf{Prop}}{\Gamma \vdash Prf\,(P) \ \ kind} \quad \frac{\Gamma \vdash P = Q\colon \mathbf{Prop}}{\Gamma \vdash Prf\,(P) = Prf\,(Q)}$$

Dependent product kinds

$$\frac{\Gamma \vdash K \ \ kind \ \ \Gamma, x : K \vdash K' \ \ kind}{\Gamma \vdash (x : K)K' \ \ kind} \quad \frac{\Gamma \vdash K_1 = K_2 \ \ \Gamma, x : K_1 \vdash K_1' = K_2'}{\Gamma \vdash (x : K_1)K_1' = (x : K_2)K_2'}$$

$$\frac{\Gamma, x : K \vdash k\colon K'}{\Gamma \vdash [x : K]k\colon (x : K)K'} \quad \frac{\Gamma \vdash K_1 = K_2 \ \ \Gamma, x : K_1 \vdash k_1 = k_2\colon K}{\Gamma \vdash [x : K_1]k_1 = [x : K_2]k_2\colon (x : K_1)K}$$

$$\frac{\Gamma \vdash f\colon (x : K)K' \ \ \Gamma \vdash k\colon K}{\Gamma \vdash f(k)\colon [k/x]K'} \quad \frac{\Gamma \vdash f = f'\colon (x : K)K' \ \ \Gamma \vdash k_1 = k_2\colon K}{\Gamma \vdash f(k_1) = f'(k_2)\colon [k_1/x]K'}$$

$$\frac{\Gamma, x : K \vdash k'\colon K' \ \ \Gamma \vdash k\colon K}{\Gamma \vdash ([x : K]k')(k) = [k/x]k'\colon [k/x]K'} \quad \frac{\Gamma \vdash f\colon (x : K)K' \ \ x \notin FV(f)}{\Gamma \vdash [x : K]f(x) = f\colon (x : K)K'}$$

Fig. 7: Rules of deduction of LF′ (II)

B.1 Classical Predicate Logic

$$\begin{aligned} \bot \ &: \ \mathbf{Prop} \\ \bot E \ &: \ (p : \mathbf{Prop})Prf\,(\bot) \to Prf\,(p) \end{aligned}$$

$$\begin{aligned} \supset \ &: \ \mathbf{Prop} \to \mathbf{Prop} \to \mathbf{Prop} \\ \supset I \ &: \ (p, q : \mathbf{Prop})(Prf\,(p) \to Prf\,(q)) \to Prf\,(p \supset q) \\ \supset E \ &: \ (p, q : \mathbf{Prop})Prf\,(p \supset q) \to Prf\,(p) \to Prf\,(q) \end{aligned}$$

$$\text{Peirce} \ : \ (p, q : \mathbf{Prop})((Prf\,(p) \to Prf\,(q)) \to Prf\,(p)) \to Prf\,(p)$$

$$\begin{aligned} \forall \ &: \ (A : \mathbf{Type})(A \to \mathbf{Prop}) \to \mathbf{Prop} \\ \forall I \ &: \ (A : \mathbf{Type})(P : A \to \mathbf{Prop})((x : A)Prf\,(Px)) \to Prf\,(\forall AP) \\ \forall E \ &: \ (A : \mathbf{Type})(P : A \to \mathbf{Prop})(a : A)Prf\,(\forall AP) \to Prf\,(Pa) \end{aligned}$$

B.2 Natural Numbers

$$
\begin{aligned}
\mathbb{N} &: \mathbf{Type} \\
0 &: \mathbb{N} \\
\mathrm{s} &: \mathbb{N} \to \mathbb{N} \\
E_{\mathbb{N}} &: (C : \mathbb{N} \to \mathbf{Type})C0 \to ((n : \mathbb{N})Cn \to C(\mathrm{s}n)) \to (n : \mathbb{N})Cn \\
&\quad E_{\mathbb{N}}\, C\, a\, b\, 0 = a : C0 \\
&\quad E_{\mathbb{N}}\, C\, a\, b\, (\mathrm{s}n) = b\, n\, (E_{\mathbb{N}}\, C\, a\, b\, n) : C(\mathrm{s}n) \\
\mathrm{Ind}_{\mathbb{N}} &: (P : \mathbb{N} \to \mathbf{Prop})Prf\,(P0) \to ((n : \mathbb{N})Prf\,(Pn) \to Prf\,(P(\mathrm{s}n))) \to \\
&\qquad (n : \mathbb{N})Prf\,(Pn)
\end{aligned}
$$

B.3 Pairs

$$
\begin{aligned}
\times &: \mathbf{Type} \to \mathbf{Type} \to \mathbf{Type} \\
\mathrm{pair} &: (A, B : \mathbf{Type})A \to B \to A \times B \\
E_{\times} &: (A, B : \mathbf{Type})(C : A \times B \to \mathbf{Type})((a : A)(b : B)C(\mathrm{pair}\, A\, B\, a\, b)) \to \\
&\qquad (p : A \times B)Cp \\
&\quad E_{\times}\, A\, B\, C\, e\, (\mathrm{pair}\, A\, B\, a\, b) = eab : C(\mathrm{pair}\, A\, B\, a\, b) \\
\mathrm{Ind}_{\times} &: (A, B : \mathbf{Type})(P : A \times B \to \mathbf{Type})((a : A)(b : B)Prf\,(P(\mathrm{pair}\, A\, B\, a\, b))) \\
&\qquad \to (p : A \times B)Prf\,(Pp)
\end{aligned}
$$

B.4 Functions

$$
\begin{aligned}
\Rightarrow &: \mathbf{Type} \to \mathbf{Type} \to \mathbf{Type} \\
\lambda &: (A, B : \mathbf{Type})(A \to B) \to (A \Rightarrow B) \\
E_{\Rightarrow} &: (A, B : \mathbf{Type})(C : (A \Rightarrow B) \to \mathbf{Type})((b : A \to B)C(\lambda\, A\, B\, b)) \to \\
&\qquad (f : A \Rightarrow B)Cf \\
&\quad E_{\Rightarrow}\, A\, B\, C\, e\, (\lambda\, A\, B\, b) = eb : C(\lambda\, A\, B\, b) \\
\mathrm{Ind}_{\Rightarrow} &: (A, B : \mathbf{Type})(P : (A \Rightarrow B) \to \mathbf{Prop})((b : A \to B)Prf\,(P(\lambda\, A\, B\, b))) \to \\
&\qquad (f : A \Rightarrow B)Prf\,(Pf)
\end{aligned}
$$

B.5 The Type Universe

$$
\begin{aligned}
U &: \mathbf{Type} \\
T &: U \to \mathbf{Type} \\
\hat{\mathbb{N}} &: U \\
\hat{\times} &: U \to U \to U \\
&\quad T\hat{\mathbb{N}} = \mathbb{N} : \mathbf{Type} \\
&\quad T(A \hat{\times} B) = TA \times TB : \mathbf{Type}
\end{aligned}
$$

B.6 Equality

$$
\begin{aligned}
\simeq &: (A:U)TA \to TA \to \mathbf{Prop}\\
\simeq I &: (A:U)(a:TA)Prf\,(a \simeq_A a)\\
\simeq E &: (A:U)(P:TA \to \mathbf{Prop})(a,b:TA)Prf\,(Pa) \to Prf\,(a \simeq_A b) \to Prf\,(Pb)
\end{aligned}
$$

B.7 The Propositional Universe

$$
\begin{aligned}
\mathrm{prop} &: \mathbf{Prop}\\
V &: Prf\,(\mathrm{prop}) \to \mathbf{Prop}\\
\hat{\bot} &: Prf\,(\mathrm{prop})\\
\hat{\supset} &: Prf\,(\mathrm{prop}) \to Prf\,(\mathrm{prop}) \to Prf\,(\mathrm{prop})\\
\hat{\forall} &: (A:U)(TA \to Prf\,(\mathrm{prop})) \to Prf\,(\mathrm{prop})\\
\hat{\simeq} &: (A:U)TA \to TA \to Prf\,(\mathrm{prop})\\
&V\hat{\bot} = \bot : \mathbf{Prop}\\
&V(p\hat{\supset}q) = Vp \supset Vq : \mathbf{Prop}\\
&V(\hat{\forall}\,A\,P) = \forall(TA)[x:TA]V(Px) : \mathbf{Prop}\\
&V(a\hat{\simeq}_A b) = a \simeq_A b : \mathbf{Prop}
\end{aligned}
$$

B.8 The Predicative Notion of Set

$$
\begin{aligned}
\mathrm{Set} &: \mathbf{Type} \to \mathbf{Type}\\
\mathrm{set} &: (A:\mathbf{Type})(A \to Prf\,(\mathrm{prop})) \to \mathrm{Set}\,(A)\\
\in &: (A:\mathbf{Type})A \to \mathrm{Set}\,(A) \to Prf\,(\mathrm{prop})\\
&\in A\,a\,(\mathrm{set}\,A\,P) = Pa : Prf\,(\mathrm{prop})
\end{aligned}
$$

ACKNOWLEDGMENTS

Thanks go to Paul Callaghan for his efforts in extending Plastic to implement the type-theoretic framework which has made the reported case study possible, and Peter Aczel for his comments during his visit to Royal Holloway. Thanks also go to Thierry Coquand and the anonymous referees, of this paper and the conference version, for their very helpful comments.

REFERENCES

Aczel, P. and Gambino, N. 2002. Collection principles in dependent type theory. In *Types for Proofs and Programs: International Workshop, TYPES 2000, Durham, UK, December 8–12, 2000. Selected Papers*, P. Callaghan, Z. Luo, J. McKinna, and R. Pollack, Eds. LNCS, vol. 2277. Springer-Verlag, 1–23.

Adams, R. and Luo, Z. 2007. Weyl's predicative classical mathematics as a logic-enriched type theory. In *Types for Proofs and Programs: International Workshop, TYPES 2006, Revised Selected Papers*, T. Altenkirch and C. McBride, Eds. LNCS, vol. 4502. Springer, 1–17.

Benacerraf, P. and Putnam, H., Eds. 1983. *Philosophy of Mathematics: Selected Readings*, Second edition ed. Cambridge University Press.

Bertot, Y. and Castéran, P. 2004. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer.

Callaghan, P. and Luo, Z. 2001. An implementation of LF with coercive subtyping and universes. *Journal of Automated Reasoning 27,* 1, 3–27.

Coquand, T. and Huet, G. 1988. The calculus of constructions. *Information and Computation 76*, 95–120.

Feferman, S. 1964. Systems of predicative analysis. *J. Symbolic Logic 29*, 1–30.

Feferman, S. 1982. Iterated inductive fixed-point theories: Application to hancock's conjecture. In *Patras Logic Symposium*, G. Metakides, Ed. North-Holland, 171–196.

Feferman, S. 2000. The significance of hermann weyl's *Das Kontinuum*. In *Proof Theory — Historical and Philosophical Significance*, V. Hendricks, S. A. Pedersen, and K. F. Jørgensen, Eds. Synthese Library, vol. 292. Kluwer Academic Publishers, Dordrecht, Chapter 7, 179–194.

Gambino, N. and Aczel, P. 2006. The generalised type-theoretic interpretation of constructive set theory. *J. Symbolic Logic 71,* 1, 67–103.

Girard, J.-Y. 1972. Interpretation fonctionelle et elimination des coupures de l'arithmetique d'ordre superieur. Ph.D. thesis, Paris VII.

Gödel, K. 1944. Russell's mathematical logic. In *The Philosophy of Bertrand Russell*, P. A. Schilpp, Ed. The Library of Living Philosophers, vol. III. Evanston & Chicago, Northwestern University, 125–153. Reprinted in [Benacerraf and Putnam 1983].

Gonthier, G. 2005. A computer-checked proof of the four colour theorem. Tech. rep., Microsoft Research.

Harper, R., Honsell, F., and Plotkin, G. 1993. A framework for defining logics. *Journal of the Association for Computing Machinery 40,* 1, 143–184.

Kamareddine, F., Laan, T., and Nederpelt, R. 2002. Types in logic and mathematics before 1940. *Bulletin of Symbolic Logic 8,* 2 (June), 185–245.

Kleene, S. C. 1959. Hierarchies of number-theoretic predicates. *Bull. Amer. Math. Soc. 61*, 193–213.

Kreisel, G. 1960. La prédicativité. *Bull. Soc. math. France 88*, 371–391.

Luo, Z. 1994. *Computation and Reasoning: A Type Theory for Computer Science*. Number 11 in International Series of Monographs on Computer Science. Oxford University Press.

Luo, Z. 2006. A type-theoretic framework for formal reasoning with different logical foundations. In *Proc of the 11th Annual Asian Computing Science Conference*, M. Okada and I. Satoh, Eds. LNCS, vol. 4435. Tokyo, 179–194.

Luo, Z. and Pollack, R. 1992. LEGO Proof Development System: User's Manual. LFCS Report ECS-LFCS-92-211, Dept of Computer Science, Univ of Edinburgh.

Martin-Löf, P. 1971. An intuitionistic theory of types. manuscript.

Martin-Löf, P. 1972. An intuitionistic theory of types. Reprinted in [Sambin and Smith 1998, 127–172].

Martin-Löf, P. 1975. An intuitionistic theory of types: Predicative part. In *Logic Colloquium 1973*, H. E. Rose and J. C. Shepherdson, Eds. North-Holland, 73–118.

Martin-Löf, P. 1982. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science, VI, 1979*, L. J. Cohen, J. Los, H. Pfeiffer, and K.-P. Podewski, Eds. North-Holland, 153–175.

Martin-Löf, P. 1984. *Intuitionistic Type Theory*. Bibliopolis.

The Coq development team. 2006. *The Coq proof assistant reference manual*. LogiCal Project. Version 8.1.

Muzalewski, M. 1993. *An Outline of PC Mizar*. Fondation Philippe le Hodey, Brussels.

Nordström, B., Petersson, K., and Smith, J. 1990. *Programming in Martin-Löf's Type Theory: an Introduction*. Oxford University Press.

Norell, U. 2007. Towards a practical programming language based on dependent type theory. Ph.D. thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden.

Parigot, M. 1992. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *Logic Programming and Automated Reasoning*, A. Voronkov, Ed. Lecture Notes in Artificial Intelligence, vol. 624. Springer, 190–201.

Poincaré, H. 1906. Les mathématiques et la logique III. *Revue de Métaphysique et de Morale 14,* 1, 294–317.

Pollack, R. et al. 2001. The LEGO Proof Assistant.

Russell, B. 1906. Les paradoxes de la logique. *Revue de Métaphysique et de Morale 14,* 5, 627–650.

Russell, B. 1908. Mathematical logic as based on the theory of types. *American Journal of Mathematics 30,* 3 (July), 222–262.

Sambin, G. and Smith, J. M., Eds. 1998. *Twenty-Five Years of Constructive Type Theory.* Oxford Logic Guides, vol. 36. Oxford University Press.

Schütte, K. 1965. Predicative well-orderings. In *Formal Systems and Recursive Functions: Proceedings of the Eigth Logic Colloquium Oxford, July 1963*, J. Crossley and M. Dummett, Eds. North-Holland, 279–302.

Setzer, A. 1993. Proof theoretical strength of Martin-Löf Type Theory with W-type and one universe. Ph.D. thesis, Universität München, available via http://www.cs.swan.ac.uk/∼csetzer.

Simpson, S. G. 1999. *Subsystems of Second-Order Arithmetic.* Springer-Verlag.

Weyl, H. 1918. *Das Kontinuum.* Translated as [Weyl 1994].

Weyl, H. 1994. *The Continuum: A Critical Examination of the Foundation of Analysis.* Dover, Kirksville, Missouri. Translated by Stephen Pollard and Thomas Bole.

Received September 2008; accepted December 2008