A type-theoretic framework for formal reasoning with different logical foundations

Zhaohui Luo*

Dept of Computer Science, Royal Holloway, Univ of London
Egham, Surrey TW20 0EX, U.K.
zhaohui@cs.rhul.ac.uk

Abstract. A type-theoretic framework for formal reasoning with different logical foundations is introduced and studied. With logic-enriched type theories formulated in a logical framework, it allows various logical systems such as classical logic as well as intuitionistic logic to be used effectively alongside inductive data types and type universes. This provides an adequate basis for wider applications of type theory based theorem proving technology. Two notions of set are introduced in the framework and used in two case studies of classical reasoning: a predicative one in the formalisation of Weyl's predicative mathematics and an impredicative one in the verification of security protocols.

1 Introduction

Dependent type theories, or type theories for short, are powerful calculi for logical reasoning that provide solid foundations for the associated theorem proving technology as implemented in various 'proof assistants'. These type theories include Martin-Löf's predicative type theory [NPS90,ML84], as implemented in ALF/Agda [MN94,Agd00] and NuPRL [C$^+$86], and the impredicative type theories [CH88,Luo94], as implemented in Coq [Coq04] and Lego/Plastic [LP92,CL01]. The proof assistants have been successfully used in formalisation of mathematics (e.g., the formalisations in Coq of the four-colour theorem [Gon05]) and in reasoning about programs (e.g., the analysis of security protocols).

The current type theories as found in the proof assistants are all based on intuitionistic logic. As a consequence, the type theory based proof assistants are so far mainly used for constructive reasoning. Examples that require or use other methods of reasoning, say classical reasoning, would have to be done by 'extending' the underlying type theory with classical laws by brute force and praying for such an extension to be OK.

We believe that the type theory based theorem proving technology is not (and should not be) limited to constructive reasoning. In particular, it should adequately support classical reasoning as well as constructive reasoning. To this end, what one needs is a type-theoretic framework that supports the use of a

* This work is partially supported by the following research grants: UK EPSRC grant GR/R84092, Leverhulme Trust grant F/07 537/AA and EU TYPES grant 510996.

wider class of logical systems and, at the same time, keeps the power of type theory in formal reasoning such as inductive reasoning based on inductive types.

In this paper, we introduce such a type-theoretic framework where logic-enriched type theories are formulated in a logical framework such as LF [Luo94] and PAL$^+$ [Luo03]. Logic-enriched type theories with intuitionistic logic have been proposed by Aczel and Gambino in studying type-theoretic interpretations of constructive set theory [AG02,GA06]. They are studied here from the angle of supporting formal reasoning with different logical foundations. We show that it is possible to provide a uniform framework so that type theories can be formulated properly to support different methods of reasoning. The structure of our framework promotes a complete separation between logical propositions and data types. It provides an adequate support to classical inference as well as intuitionistic inference, in the presence of inductive types and type universes.

Two notions of set are introduced in the type-theoretic framework: a predicative one and an impredicative one. They are used in two case studies: one in the formalisation of Weyl's predicative mathematics [Wey18,AL06] and the other in the formalisation and analysis of security protocols. Both case studies use classical reasoning – we have chosen to do so partly because how to use type theory in constructive reasoning has been extensively studied and partly because we want to show how classical logic can be employed in a type-theoretic setting.

The proof assistant Plastic [CL01] has been extended (in an easy and straightforward way) by Paul Callaghan to implement the type-theoretic framework as described in this paper. The case studies have been done in the extended Plastic and help to demonstrate that type theory and the associated proof assistants can be used to support formal reasoning with different logical foundations.

2 Logic-enriched type theories in a logical framework

The type-theoretic framework formulates logic-enriched type theories (LTTs for short) in a logical framework. It consists of two parts, *the part of logical propositions* and *the part of data types*. Both parts are formulated in a logical framework and linked by the induction rules (see below).

We start with the logical framework LF [Luo94] or PAL$^+$ [Luo03], where the kind $Type$ represents the world of types. Now, we extend the logical framework with a new kind $Prop$ that stands for the world of logical propositions and, for every $P : Prop$, a kind $Prf(P)$ of proofs of P.

The logic of an LTT is specified in $Prop$ by declaring constants for the logical operators and the associated rules (as logics are introduced in Edinburgh LF [HHP93]). The data types are introduced in $Type$ as in type theories such as Martin-Löf's type theory [NPS90] and UTT [Luo94]. Different LTTs can be formulated in the framework for formal reasoning with different logical foundations. Instead of considering LTTs in general, we shall present and study a typical example – the LTT with the classical first-order logic, abbreviated as LTT_1, to illustrate how an LTT is formulated in our type-theoretic framework.

2.1 LTT_1: an example

The system LTT_1 consists of the classical first-order logic, the inductive data types, and type universes. Each of the components is described below.

Logic of LTT_1 The logical operators such as $\supset$, $\neg$ and $\forall$ are introduced by declaring as constants the operators and the direct proofs of the associated inference rules. For instance, for universal quantification $\forall$, we declare (where we write '$f[a_1, ..., a_n]$' for applications and '$f[x_1, ..., x_n] : A$ where $x_i : A_i$' for $f : (x_1{:}A_1, ..., x_n{:}A_n)A$):

$$\forall[A, P] : Prop, \quad \forall_I[A, P, f] : Prf(\forall[A, P]) \quad \texttt{and} \quad \forall_E[A, P, a, p] : Prf(P[a])$$

where $A : Type$, $P[x{:}A] : Prop$, $f[x{:}A] : Prf(P[x])$ and $p : Prf(\forall[A, P])$.

Note that $\forall$ can only quantify over types; that is, for a formula $\forall[A, P]$, or $\forall x{:}A, P[x]$ in the usual notation, A must be a type (of kind $Type$). Since $Prop$ is not a type (it is a kind), one cannot form a proposition by quantifying over $Prop$. Higher-order logical quantifications such as $\forall X{:}Prop.X$, as found in impredicative type theories, are not allowed. Similarly, since propositions are not types ($Prf(P)$ is a kind, not a type), one cannot quantify over propositions, either.

As another example, we declare the classical negation operator $\neg P : Prop$ for $P : Prop$ and the corresponding double negation rule $DN[P, p] : Prf(P)$ where $P : Prop$ and $p : Prf(\neg\neg P)]$. Other logical operators can be introduced in a similar way or defined as usual. For instance, an equality operator can be introduced to form propositions $a =_A b$, for $A : Type$ and $a, b : A$.

Inductive data types in LTT_1 The system LTT_1 (and every LTT) contains (some or all of) the inductive types as found in Martin-Löf's type theory [NPS90] or UTT [Luo94], which include those of natural numbers, dependent pairs, lists, trees, ordinals, etc. For example, the type $N : Type$ of natural numbers can be introduced by first declaring its constructors $0 : N$ and $s[n] : N$ where $n : N$, and then its elimination operator $\mathcal{E}_T[C, c, f, n] : C[n]$, for $C[n] : Type$ with $n : N$, and the associated computation rules

$$\begin{aligned} \mathcal{E}_T[C, c, f, 0] &= c \quad : \quad C[0] \\ \mathcal{E}_T[C, c, f, s[n]] &= f[n, \mathcal{E}_T[C, c, f, n]] \quad : \quad C[s[n]] \end{aligned}$$

For each inductive type, there is an associated *induction rule* for proving properties of the objects of that type. For example, the induction rule for N is

$$\mathcal{E}_P[P, c, f, n] : P[n] \qquad for\ P[n] : Prop \quad [n : N]$$

Note that the elimination operator over types, $\mathcal{E}_T$, has associated computational rules, while the elimination operator over propositions, $\mathcal{E}_P$, does not.

The induction rules are crucial in connecting the world of logical propositions (formally represented by $Prop$) and that of the data types (formally represented by $Type$). Quantifications over types allow one to form propositions to express logical properties of data and the induction rules to prove those properties.

Type universes in LTT_1 The system LTT_1 (and every LTT) may contain type universes, types consisting of (names of) types as objects. For example, a universe of 'small types' can be introduced as

$$\mathit{type} : \mathit{Type} \quad \mathit{and} \quad T[x] : \mathit{Type} \quad [x : \mathit{type}].$$

Some of the inductive types have names in a type universe. For example, we can have *nat* as a name of N in *type* by declaring $\mathit{nat} : \mathit{type}$ and $T[\mathit{nat}] = N : \mathit{Type}$. The general way of introducing type universes can be found in [ML84] and see, e.g., [Luo94] for universes containing inductive types generated by schemata.

We remark that, if we have introduced a type universe that contains the names of N and $\emptyset$ (the empty type), we can prove Peano's fourth axiom for natural numbers $(\forall x{:}N.(s[x] \neq_N 0))$ internally in the type-theoretic framework. This is similar to Martin-Löf's type theory, where Peano's fourth axiom is not provable internally without a type universe [Smi88].

Logical consistency of LTT_1 The system LTT_1 is logically consistent in the sense that there are unprovable propositions. If one is not satisfied with the 'simple minded consistency' [ML84], a meta-mathematical consistency can be proved – it can be shown that LTT_1 is relatively consistent w.r.t. ZF.

Theorem 1 (consistency). *The type system* LTT_1 *is logically consistent.*

Proof sketch Let T be Martin-Löf's intensional type theory extended with Excluded Middle (i.e., extending it with assumed proofs of $A + (A \to \emptyset)$ for all types A). Then T is logically consistent w.r.t. ZF. Now, consider the mapping $\sharp : \mathrm{LTT}_1 \to T$ that maps the types and propositions of LTT_1 to types of T so that $A^\sharp = A$ for $A : \mathit{Type}$ and, e.g., $\forall[A, P]^\sharp = \Pi[A^\sharp, P^\sharp]$ and $(\neg P)^\sharp = P^\sharp \to \emptyset$. Then, by proving a more general lemma by induction, we can show that if $\Gamma \vdash a : A$ in LTT_1, then $\Gamma^\sharp \vdash a^\sharp : A^\sharp$ in T. The logical consistency follows. □

Although there are other ways to prove the meta-mathematical consistency, the above proof sketch raises an interesting question one may ask: if Martin-Löf's type theory extended with a classical law (say Excluded Middle) is consistent, why does one prefer to use LTT_1 rather than such an extension directly?

One of the reasons for such a preference is that the LTT approach preserves the meaning-theoretic understanding of types as consisting of their canonical objects (e.g., N consists of zero and the successors). Such an *adequacy property* would be destroyed by a direct extension of Martin-Löf's type theory with a classical law, where every inductive type contains (infinitely many) non-canonical objects. Therefore, in this sense, it is inadequate to introduce classical laws directly to Martin-Löf's type theory or other type theories.

In our type-theoretic framework, there is a clear distinction between logical propositions and data types. For example, the classical law in LTT_1 does not affect the data types such as N. It hence provides an adequate treatment of classical reasoning on the one hand and a clean meaning-theoretic understanding of the inductive types on the other.

This clear separation between logical propositions and data types is an important salient feature of the type-theoretic framework in general. In Martin-Löf's type theory, for example, types and propositions are identified. The author has argued, for instance in the development of ECC/UTT [Luo94] as implemented in Lego/Plastic and the current version of Coq[1], 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 data types are not propositions, although logical propositions are types.

Logic-enriched type theories, and hence our framework as presented in this paper, have gone one step further – there is a complete separation between propositions and types. Logical propositions and their totality $Prop$ are not regarded as types. This has led to a more flexible treatment of logics in our framework.

2.2 Implementation

The type-theoretic framework has been implemented by Callaghan by extending the proof assistant Plastic [CL01]. Plastic implements the logical framework LF. The extension is to add the kind $Prop$ and the operator Prf, as described at the beginning of this section. Logical operators such as those of LTT_1 are introduced by the user. Plastic already supports the inductive types in the kind $Type$. However, it does not automatically generate the induction rules (represented above by $\mathcal{E}_P$ for N) which, at the moment, are entered by the user (this is possible because $\mathcal{E}_P$ does not have associated computation rules.) We should also mention that Plastic supports adding computation rules of certain form and this allows one to add universes and the associated computation rules. The system LTT_1 has been completely implemented in Plastic and so have the notions of set and the case studies to be described in the following two sections.

3 Typed sets

When we consider sets in our type-theoretic framework, the objects of a set are all 'similar' in the sense that every set is a *typed set*. In other words, every set has a *base type* from which every element of the set comes. For instance, a set with N as base type contains only natural numbers as its elements. We believe that such a notion of typed set is natural and much closer to the practice of everyday mathematical reasoning than that in the traditional set theory where there is no distinction between types. Sometimes, mathematicians use the word 'category' for what we call types and consider sets with elements of the same category (see, e.g., [Wey18]). Here, types are formal representations of categories.

In the following, we consider two notions of (typed) set: an impredicative notion and a predicative notion.

[1] The current type structure of Coq (version 8.0) [Coq04], after the universe Set becomes predicative, is very similar to (if not exactly the same as) that of ECC/UTT.

Impredicative notion of set Impredicative sets can be introduced as follows.

- $Set[A{:}Type] : Type$ ($Set[A]$ is the type of sets with base type A.)
- $set[A{:}Type, P[x{:}A]{:}Prop] : Set[A]$ (Informally, $set[A, P]$ is $\{ x : A \mid P[x] \}$.)
- $in[A{:}Type, a{:}A, S{:}Set[A]] : Prop$ (Informally, $in[A, a, S]$ is $a \in S$.)
- $in[A, a, set[A, P]] = P[a] : Prop$ (Computational equality)

Note that this notion of set is impredicative. For example, the powerset of $S : Set[A]$ can be defined as $\{ S'{:}Set[A] \mid \forall x{:}A.\, x \in S' \supset x \in S \}$ of type $Set[Set[A]]$.

Predicative notion of set The notion of predicativity has been studied by many people, most notably by Feferman in a series of papers, including [Fef05]. Intuitively, the definition of a set $\{ x \mid p(x) \}$ is predicative if the predicate $p(x)$ does not involve any quantification over the totality that contains the entity being defined. If the set-forming predicate p does not involve any quantification over sets at all, then the definition of the set is predicative.

In our type-theoretic framework, predicative sets can be introduced by first introducing a propositional universe of small propositions:

$$prop : Prop, \qquad V[p : prop] : Prop$$

Intuitively, a small proposition contains only quantifications over small types (with names in *type*). This can be seen from the quantifier rules for *prop*:

$$\bar{\forall}[a, p] : prop \quad [a{:}type,\ p[x{:}T[a]] : prop]$$
$$V[\bar{\forall}[a, p]] = \forall[T[a], [x{:}T[a]]V[p[x]]] : Prop$$

where $p[x]$ must be a small proposition for any object x in the small type a.

Formally, predicative sets can be introduced as follows:

- $Set[a{:}type] : Type$ ($Set[a]$ is a type if a is a small type in universe *type*.)
- $set[a{:}type, p[x{:}T[a]]{:}prop] : Set[a]$ (Informally, p in $\{ x : A \mid p[x] \}$ must be a small propositional function.)
- $in[a{:}type, x{:}T[a], S{:}Set[a]] : prop$ (Informally, $x \in S$ is a small proposition.)
- $in[a, x, set[a, p]] = p[x] : prop$ (Computational equality)

Note that a type of sets is not a small type. Therefore, quantification over sets is not allowed when stating a set-forming condition.

Remark 1. Aczel and Gambino [AG02,GA06] have considered a type universe $\mathbf{P}$ of small propositions in the world of types (formally, $\mathbf{P} : Type$). Such a universe has played an important role in their study of the type-theoretic interpretation of constructive set theory. However, it seems that this has the effect of putting logical propositions directly 'back' to the world of types. In our case, the propositional universe *prop* is of kind $Prop$, not of kind $Type$. □

We argue that the notions of set introduced above are adequate in supporting '*mathematical pluralism*'.[2] With these notions, the type-theoretic framework can be used to formalise, in the classical setting, the ordinary (classical and impredicative) mathematics and Weyl's predicative mathematics and, in the intuitionistic setting, the predicative and impredicative constructive mathematics.

[2] The philosophical view of mathematical pluralism is to be elaborated elsewhere.

4 Case studies

Formalisation of Weyl's predicative mathematics As is known, the ordinary mathematics is *impredicative* in the sense that it allows impredicative definitions that some people might regard as 'circular' and hence problematic. Such people would believe that the so-called *predicative* mathematics is safer, where impredicative or circular definitions are regarded as illegal. For instance, in the early quarter of the last century, the mathematician Hermann Weyl has developed a predicative treatment of the calculus (in classical logic) [Wey18], which has been studied and further developed by Feferman and others [Fef05].

The formalisation of Weyl's work [Wey18] has been done in the type-theoretic framework (more specifically, in LTT_1 with predicative sets) in Plastic [AL06].

Formalisation and analysis of security protocols Security protocols have been extensively studied in the last two to three decades. Besides other interesting research, theorem provers have been used to formalise security protocols and prove their properties. For instance, Paulson has studied the 'inductive approach' [Pau98] in Isabelle [Pau94] to verify properties of security protocols.

As a case study of classical impredicative reasoning in the type-theoretic framework, we have formalised in Plastic several security protocols in LTT_1 with impredicative sets. The examples include simple protocols such as the Needham-Schroeder public-key protocol [NS78,Low96] and the Yahalom protocol [BAN89,Pau99]. Our formalisation has followed Paulson's inductive approach closely, in order to examine how well the power of inductive reasoning in the type-theoretic framework matches that in Isabelle. Our experience shows that the answer is positive, although more automation in some cases would be desirable. In our formalisation, agents, messages and events are all modelled as inductive types (rather than as inductive sets as in Isabelle). The operations such as $parts$, $analz$ and $synth$ are defined as maps between sets of messages. A protocol is then modelled as a set of traces (a set of lists of events). One can then show that various properties are satisfied by the protocol concerned, including the secrecy properties such as the session key secrecy theorem.

5 Concluding remarks on future work

Future work includes comparative studies with other existing logical systems and the associated theorem proving technology. For example, it would be interesting to compare the predicative notion of set with that studied by Feferman and others [Fef00,Sim99] and to consider a comparison with that of Martin-Löf's type theory [NPS90,ML84] to study the relationship between the notion of predicative set and that of predicative type.

Acknowledgements Thanks go to Peter Aczel for his comments during his visit to Royal Holloway, Robin Adams for discussions during our joint work on formalisation of Weyl's predicative mathematics and Paul Callaghan for his efforts in extending Plastic to implement the type-theoretic framework.

References

[AG02] P. Aczel and N. Gambino. Collection principles in dependent type theory. Proofs and Programs, (eds) P.Callaghan, Z. Luo, J. McKinna and R. Pollack. LNCS 2277, 2002.

[Agd00] Agda proof assistant. `http://www.cs.chalmers.se/~catarina/agda`, 2000.

[AL06] R. Adams and Z. Luo. Weyl's predicative classical mathematics as a logic-enriched type theory. Submitted to TYPES'06, to appear, 2006.

[BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proc. of the Royal Society of London*, 426:233–271, 1989.

[C$^+$86] R. Constable et al. *Implementing Mathematics with the NuPRL Proof Development System*. Pretice-Hall, 1986.

[CH88] Th. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2/3), 1988.

[CL01] P. C. Callaghan and Z. Luo. An implementation of typed LF with coercive subtyping and universes. *J. of Automated Reasoning*, 27(1):3–27, 2001.

[Coq04] The Coq Development Team. *The Coq Proof Assistant Reference Manual (Version 8.0), INRIA*, 2004.

[Fef00] S. Feferman. The significance of Hermann Weyl's Das Kontinuum. Proof Theory, (eds) V. Hendricks et al., 2000.

[Fef05] S. Feferman. Predicativity. In S. Shapiro, editor, *The Oxford Handbook of Philosophy of Mathematics and Logic*. Oxford Univ Press, 2005.

[GA06] N. Gambino and P. Aczel. The generalised type-theoretic interpretation of constructive set theory. *J. of Symbolic Logic*, 71(1):67–103, 2006.

[Gon05] G. Gonthier. A computer checked proof of the four colour theorem, 2005.

[HHP93] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. *Lecture Notes in Computer Science*, 1055, 1996.

[LP92] Z. Luo and R. Pollack. LEGO Proof Development System: User's Manual. LFCS Report ECS-LFCS-92-211, Dept of Computer Science, Univ of Edinburgh, 1992.

[Luo94] Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science*. Oxford University Press, 1994.

[Luo03] Z. Luo. PAL$^+$: a lambda-free logical framework. *Journal of Functional Programming*, 13(2):317–338, 2003.

[ML84] P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.

[MN94] L. Magnusson and B. Nordström. The ALF proof editor and its proof engine. *LNCS 806*, 1994.

[NPS90] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory: An Introduction*. Oxford University Press, 1990.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Comm. of the ACM*, 21(12):993–999, 1978.

[Pau94] L. Paulson. Isabelle: a generic theorem prover. *LNCS*, 828, 1994.

[Pau98] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[Pau99] L. Paulson. Proving security protocols correct. *LICS*, 1999.

[Sim99] S. Simpson. *Subsystems of Second-Order Arithmetic*. Springer-Verlag, 1999.

[Smi88] J. Smith. The independence of Peano's fourth axiom from Martin-Löf's type theory without universes. *Journal of Symbolic Logic*, 53(3), 1988.

[Wey18] H. Weyl. *The Continuum: a critical examination of the foundation of analysis*. Dover Publ. 1994, (English translation of Das Kontinuum, 1918).