Expressing theories in the $\lambda\Pi$-calculus modulo theory and in the DEDUKTI system

Ali Assaf[1], Guillaume Burel[2], Raphal Cauderlier[3], David Delahaye[4], Gilles Dowek[5], Catherine Dubois[2], Frédéric Gilbert[6], Pierre Halmagrand[3], Olivier Hermant[7], and Ronan Saillard[7]

[1] Inria and École polytechnique, ali.assaf@inria.fr
[2] ENSIIE, {guillaume.burel,catherine.dubois}@ensiie.fr
[3] Cnam and Inria, {raphal.cauderlier,pierre.halmagrand}@inria.fr
[4] Cnam and Université de Montpellier, david.delahaye@umontpellier.fr
[5] Inria and École Normale de Supérieure de Cachan, gilles.dowek@ens-cachan.fr
[6] École des Ponts, Inria, and CEA, frederic.gilbert@inria.fr
[7] MINES Paristech, {olivier.hermant,ronan.saillard}@mines-paristech.fr

Defining a theory, such as arithmetic, geometry, or set theory, in predicate logic just requires to chose function and predicate symbols and axioms, that express the meaning of these symbols. Using, this way, a single logical framework, to define all these theories, has many advantages.

First, it requires less efforts, as the logical connectives, $\wedge$, $\vee$, $\forall$... and their associated deduction rules are defined once and for all, in the framework and need not be redefined for each theory. Similarly, the notions of proof, model... are defined once and for all. And general theorems, such as the soundness and the completeness theorems, can be proved once and for all.

Another advantage of using such a logical framework is that this induces a partial order between theories. For instance, Zermelo-Fraenkel set theory with the axiom of choice (ZFC) is an extension of Zermelo-Fraenkel set theory (ZF), as it contains the same axioms, plus the axiom of choice. It is thus obvious that any theorem of ZF is provable in ZFC, and for each theorem of ZFC, we can ask the question of its provability in ZF. Several theorems of ZFC, that are provable in ZF have been identified, and these theorems can be used in extensions of ZF that are inconsistent with the axiom of choice.

Finally, using such a common framework permits to combine, in a proof, lemmas proved in different theories: if $\mathcal{T}$ is a theory expressed in a language $\mathcal{L}$ and $\mathcal{T}'$ a theory expressed in a language $\mathcal{L}'$, if A is expressed in $\mathcal{L} \cap \mathcal{L}'$, $A \Rightarrow B$ is provable in $\mathcal{T}$, and A is provable in $\mathcal{T}'$, then B is provable in $\mathcal{T} \cup \mathcal{T}'$.

Despite these advantages, several logical systems have been defined, not as theories in predicate logic, but as independent systems: Simple type theory, also known as Higher-order logic, is defined as an independent system—although it is also possible to express it as a theory in predicate logic. Similarly, Intuitionistic type theory, the Calculus of constructions, the Calculus of inductive constructions... are defined as independent systems. As a consequence, it is difficult to reuse a formal proof developed in an automated or interactive theorem prover based on one of these formalisms in another, without redeveloping it. It is also difficult to combine lemmas proved in different systems: the realm of formal proofs is today a tower of Babel, just like the realm of theories was, before the design of predicate logic.

The reason why these formalisms have not been defined as theories in predicate logic is that predicate logic, as a logical framework, has several limitations, that make it difficult to express modern logical systems.

1. Predicate logic does not allow the use of bound variables, except those bound by the quantifiers $\forall$ and $\exists$. For instance, it is not possible to define, in predicate logic, a unary function symbol $\mapsto$, that would bind a variable in its argument.

2. Predicate logic ignores the propositions-as-types principle, according to which a proof π of a proposition A is a term of type A.

3. Predicate logic ignores the difference between reasoning and computation. For example, when Peano arithmetic is presented in predicate logic, there is no natural way to compute the term 2×2 into 4. To prove the theorem $2 \times 2 = 4$, several derivation steps need to be used while a simple computation would have sufficed.

4. Unlike the notions of proof and model, it is not possible to define, once and for all, the notion of cut in predicate logic and to apply it to all theories expressed in predicate logic: a specific notion of cut must be defined for each theory.

5. Predicate logic is classical and not constructive. Constructive theories must be defined in another logical framework: constructive predicate logic.

This has justified the development of other logical frameworks, that address some of these problems. Problem 1 has been solved in extensions of predicate logic such as λ-Prolog and Isabelle. Problems 1 and 2 have been solved in an extension of predicate logic, called the Logical Framework, also known as the $\lambda\Pi$-calculus, and the λ-calculus with dependent types. Problems 3 and 4 have been solved in an extension of predicate logic, called Deduction modulo theory. Combining the $\lambda\Pi$-calculus and Deduction modulo theory yields the $\lambda\Pi$-calculus modulo theory, a variant of Martin-Löf's logical framework, which solves problems 1, 2, 3, and 4.

Problem 5 can also be solved in this logical framework.

In previous work, we have shown that all functional Pure type systems, in particular the Calculus of Constructions, could be expressed in this framework.

Our goal in this talk is twofold: first, we want to go further and show that other systems can be expressed in the $\lambda\Pi$-calculus modulo theory, in particular classical systems, logical systems containing a programming language as a subsystem such as FoCaliZe, simple type theory, and extensions of the Calculus of constructions with universes and inductive types. Second, we want to demonstrate this expressivity, not just with adequacy theorems, but also by showing that large libraries of formal proofs coming from automated and interactive theorem provers can be translated to and checked in Dedukti, our implementation of the $\lambda\Pi$-calculus modulo theory. To do so, we shall translate to Dedukti proofs developed in various systems: the automated theorem proving systems Zenon, Zenon modulo, iProver, and iProverModulo, the system containing a programming language as a subsystem, FoCaLiZe, and the interactive theorem provers HOL and Matita.

We show this way that the logical framework approach scales up and that it is mature enough, so that we can start imagining a single library of formal proofs expressed in different theories, developed in different systems, but formulated in a single framework.

The talk presents several examples of theories that have been expressed in Dedukti, and several large libraries of proofs expressed in these theories and translated to Dedukti. It presents ongoing work to express more theories, and to reverse engineer these proofs. Before that, it presents the $\lambda\Pi$-calculus modulo theory and the Dedukti system.