Proof Certification in Zenon Modulo: When Achilles Uses Deduction Modulo to Outrun the Tortoise with Shorter Steps*

David Delahaye[1], Damien Doligez[2], Frédéric Gilbert[2], Pierre Halmagrand[1] and Olivier Hermant[3]

[1] Cedric/Cnam/Inria, Paris, France,
David.Delahaye@cnam.fr
Pierre.Halmagrand@inria.fr
[2] Inria, Paris, France,
Damien.Doligez@inria.fr
Frederic.Charles.Gilbert@inria.fr
[3] CRI, MINES ParisTech, Fontainebleau, France,
Olivier.Hermant@mines-paristech.fr

Abstract

We present the certifying part of the Zenon Modulo automated theorem prover, which is an extension of the Zenon tableau-based first order automated theorem prover to deduction modulo. The theory of deduction modulo is an extension of predicate calculus, which allows us to rewrite terms as well as propositions, and which is well suited for proof search in axiomatic theories, as it turns axioms into rewrite rules. In addition, deduction modulo allows Zenon Modulo to compress proofs by making computations implicit in proofs. To certify these proofs, we use Dedukti, an external proof checker for the $\lambda\Pi$-calculus modulo, which can deal natively with proofs in deduction modulo. To assess our approach, we rely on some experimental results obtained on the benchmarks provided by the TPTP library.

1 Introduction

When dealing with automated theorem provers, the critical point is to ensure the soundness of the corresponding implementation: completeness is a minor concern and is often dismissed in favor of some strategies that focus efficiently on some categories of problems. To do so, two options can be considered: either formally proving the soundness of the implemented proof search method, or requiring the implementation to produce proof certificates, which can be checked to reach an appropriate degree of confidence.

The first option is clearly a difficult task, even though successful experiments have been carried out in this domain, in particular for SAT solving algorithms. This task is difficult because the tools used to formalize the soundness of algorithms provide constrained environments (frequently, functions must be total and must terminate), where it is arduous to encode the given algorithm that needs to be certified. In addition, it should be noted that the more sophisticated the algorithm is, the more difficult the soundness proof will be. As a consequence, the soundness proofs are generally made over variants of the initial algorithm, which may be far from the actual code, omitting the most error-prone parts of the algorithm.

The second option alleviates some of the difficulties inherent to the previous option, as it imposes no constraint over the proof search algorithm, which can be as clever as needed.

*This work is supported by the BWare project [7] (ANR-12-INSE-0010) funded by the INS programme of the French National Research Agency (ANR).

However, we must prove the soundness for each result produced by the algorithm, which must therefore provide enough information to build these soundness proofs. This information is generally called proof certificate, and it has become fashionable these days that certifying automated theorem provers satisfy the De Bruijn criterion (formulated by Barendregt in [1]), i.e. they generate proof certificates (or even directly proofs) in a format that can be independently checked by external proof tools. In addition, proof certificates must be of good quality and some principles tend to emerge in this domain. One of them is the Poincaré principle (also formulated by Barendregt in [1]), which is directly related to the size of proofs and which states that traces of computation should not be included in proof certificates. In this case, the external proof checker is expected to redo computations and must be therefore adapted.

In this paper, we present the certifying part of the Zenon Modulo automated theorem prover [5], which is an extension of the Zenon tableau-based first order automated theorem prover [3] to deduction modulo. The theory of deduction modulo is an extension of predicate calculus, which allows us to rewrite terms as well as propositions, and which is well suited for proof search in axiomatic theories, as it turns axioms into rewrite rules. This way, we turn proof search among the axioms into computations, avoiding unnecessary blowups, and we shrink the size of proofs by recording only their meaningful steps (see [4]). Zenon Modulo, just like Zenon, adopts a certifying approach, and is able to produce proofs to be checked by an external proof tool (in this sense, Zenon Modulo satisfies the De Bruijn criterion). In addition, deduction modulo allows Zenon Modulo to compress proofs by producing proofs with no trace of computation (in this sense, Zenon Modulo verifies the Poincaré principle). However, the external proof checker must be able to deal with these proofs in deduction modulo. For that purpose, we use Dedukti [2], which is a proof checker based on the $\lambda\Pi$-calculus modulo.

This paper is organized as follows: in Sec. 2, we present the proof generation of Zenon Modulo, and particularly focus on the proof compression aspect (compared to Zenon); in Sec. 3, we describe the proof verification of Zenon Modulo using Dedukti; for both sections, we rely on some experimental results obtained on the benchmarks provided by the TPTP library [6].

2 Proof Compression in Zenon Modulo

In order to validate our implementation of Zenon Modulo, we compare proofs found by both Zenon and Zenon Modulo where rewriting is used at least once in the latter (otherwise, proofs are identical). This leads us to examine a subset of 624 first order problems (FOF category) of the TPTP library (v5.5.0) out of the 1,446 problems proved by both Zenon and Zenon Modulo.

We compare the number of nodes of the resulting proof trees after pruning of the useless formulas (pruning is a technique that allows us to minimize the size of the proof search tree by determining the useful formulas to close the tree; see [3] for more information). The average proof size reduction obtained by Zenon Modulo over the 624 FOF problems is 6.8% (with a maximum of 91.4%). Over the 110 problems from the SET category (problems of set theory), the average proof size reduction reaches 21.6% (with a maximum of 84.6%). In Fig. 1, we present the average proof size reduction depending on the size of the initial proofs in Zenon. We cluster the proofs of Zenon in 10 deciles and compute the associated average proof size reductions. The results for the first decile (i.e. the smallest Zenon proofs, between 1 and 3 proof nodes for FOF and between 1 to 6 proof nodes for SET) are negative: about -42% for FOF and -65% for SET. These results are not shown in Fig. 1, since it is of dubious interest to turn small proofs into bigger yet still small proofs. We observe a positive correlation between the average proof size reduction and the size of Zenon proofs, i.e. the bigger Zenon proofs are, the higher the average proof size reduction is. We obtain the highest reduction for the last decile

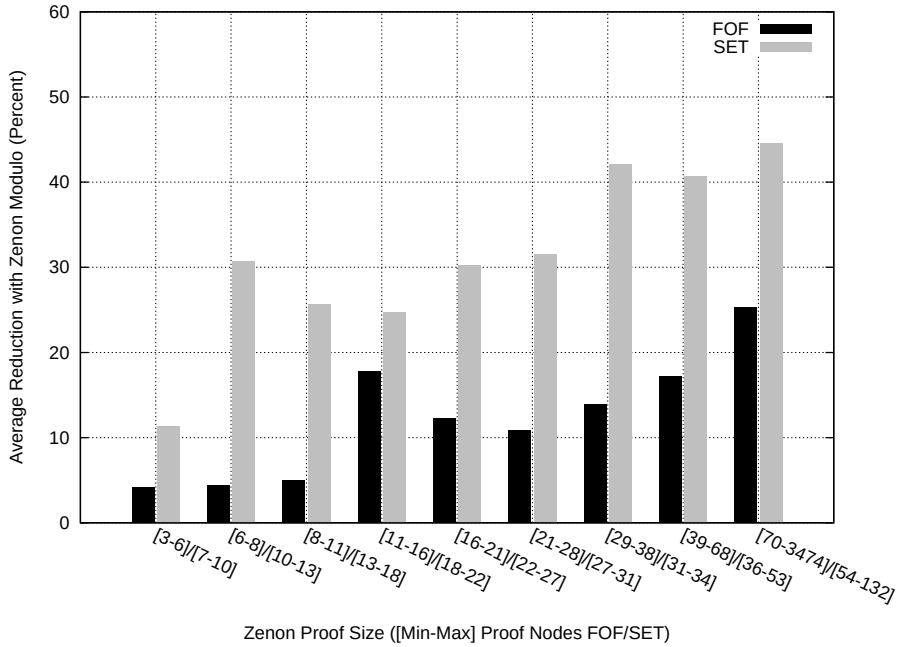

Figure 1: Proof Compression in Zenon Modulo over the FOF and SET Categories

(i.e. the biggest Zenon proofs, between 70 and 3,474 proof nodes in FOF and between 54 and 132 proof nodes in SET): about 25% for FOF and 45% for SET.

To illustrate the proof compression in Zenon Modulo, we present, in Fig. 2, the proofs found by Zenon and Zenon Modulo for the problem SET705+4, which states that every set is included in its powerset, i.e. $\forall A\ A \in \mathcal{P}(A)$. The axioms used to complete this proof are the following:

$$\begin{array}{lll} \forall X, A\ (X \in \mathcal{P}(A) & \Leftrightarrow & X \subseteq A) \qquad (powerset) \\ \forall A, B \quad (A \subseteq B & \Leftrightarrow & \forall X\ (X \in A \Rightarrow X \in B)) \qquad (subset) \end{array}$$

Zenon uses the *powerset* axiom as an hypothesis and the *subset* axiom as a definition, called by the p-unfold$_\neg$ rule. In contrast, Zenon Modulo transforms these two axioms into propositional rewrite rules as follows:

$$\begin{array}{rcll} X \in \mathcal{P}(A) & \longrightarrow & X \subseteq A & (powerset) \\ A \subseteq B & \longrightarrow & \forall X\ (X \in A \Rightarrow X \in B) & (subset) \end{array}$$

We observe a reduction of 55% of proof nodes in Zenon Modulo. The resulting proof tree is also much simpler since there is no branch creation.

3 Proof Verification with Dedukti

Integrating some rewriting capabilities on top of the two existing outputs of Zenon for Coq and Isabelle would lose compression of Sec. 2 by providing all the rewrite steps. We chose instead to use a proof checker with rewriting capabilities: Dedukti, based on the $\lambda\Pi$-calculus modulo.

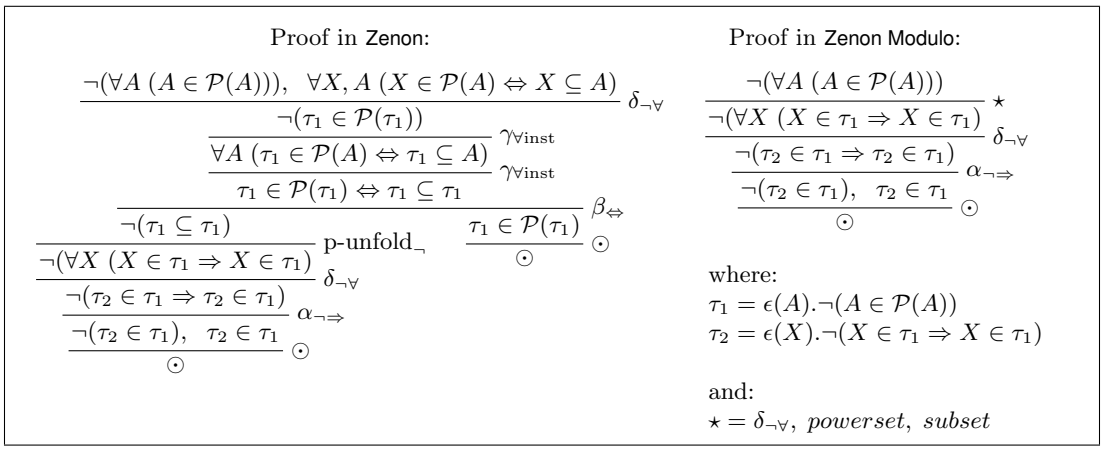

Figure 2: Proofs of Problem SET705+4 in Zenon and Zenon Modulo

As the target logic is constructive, a translation from classical logic must be therefore implemented. Since Dedukti offers many facilities to define shallow embeddings (thanks to rewrite rules), we opted for a double-negation translation rather than for a non-computational axiomatization. Introducing double-negations at each place à la Kolmogorov [8] is a linear solution with respect to proof size, but appeared impracticable. This is why we forged a translation that introduces as few double-negation as possible, improving over previously existing work by Gödel, Gentzen, and Kuroda [8]. Moreover, we treat differently the formulas along the side of the sequent they appear, following the remark that left-rules are identical in classical and intuitionistic sequent calculi. Combined with the use of rewrite rules, this has an important impact on the size of the generated certificates, and on the time it takes to check them.

As shown in Tab. 1, this approach is currently able to check with success about 90% of the proofs found by Zenon Modulo, and no certificate has been explicitly rejected. As for failures, there are two main reasons for which some proofs fail to be checked properly. The first one (see the column "Dedukti Failure") is that Dedukti itself does not succeed to check its input. This is mainly due to the produced rewrite system, which is ill-formed according to Dedukti. In particular, this is the case when the rewrite system is non-terminating or contain some non-linear rules (Dedukti may also fail in presence of non-confluent rewrite systems). Since the heuristic that transforms automatically axioms into rewrite rules used by Zenon Modulo does not perform any verification over the produced rewrite system, we can imagine to reduce the proportion of these cases of Dedukti failure by carrying out a (mainly syntactical) study of the rewrite system (possibly using some dedicated external tools) before calling Dedukti.

The second reason of failures (see the column "Backend Issue") is that the backend fails to produce a result (timeout or memory overflow) due to internal inefficiencies. By making the translation depend of the side of the sequent where the formula appears, we lose modularity. In other words, the cut rule is not "directly" admissible. Unfortunately, cuts are introduced by Zenon Modulo during proof search and proof optimization, before the intervention of the backend. To treat them, we have to eliminate them "by hand" up to a point where both left and right translations become equal. This step has not been optimized yet, and may take exponential time and size. Even worse, in deduction modulo, this process may loop whatever the reduction strategy. Another increase in the complexity of the translation is introduced by the analysis of the proof tree as a whole to decide whether the double-negation introduced by the translation of

FOF (624 problems)	Dedukti Success	Dedukti Failure	Backend Issue
Problems	559	5	60
Rate	89.6%	0.8%	9.6%

Table 1: Proof Verification with Dedukti over a Subset of the FOF Category

formulas is necessary: in fact, a double negation corresponds to the use of the excluded-middle principle, and many classical proofs do not rely on this.

4 Conclusion

We have described the certifying part of the Zenon Modulo automated theorem prover, which is an extension of the Zenon tableau-based first order automated theorem prover to deduction modulo. Thanks to deduction modulo, Zenon Modulo turns axioms into rewrite rules and is therefore able to produce smaller proofs (compared to Zenon) with no trace of computation. However, the external proof checker used to certify these proofs must be adapted, and we use Dedukti, an external proof checker based on the $\lambda\Pi$-calculus modulo, which can deal natively with proofs in deduction modulo. To assess our approach, we have presented some experimental results obtained on the benchmarks provided by the TPTP library. These results show an increasing average proof size reduction with the size of the proof, while a large part of the proofs produced by Zenon Modulo are verified by Dedukti. As future work, we plan to improve the Dedukti backend by introducing more algorithmic optimizations, but an alternative solution could be to develop a more "syntactical" backend and unplug some optimizations, in order to avoid dependency on cut-elimination and proof inspection.

References

[1] H. Barendregt and E. Barendsen. Autarkic Computations in Formal Proofs. *Journal of Automated Reasoning (JAR)*, 28(3):321–336, Apr. 2002.

[2] M. Boespflug, Q. Carbonneaux, and O. Hermant. The $\lambda\Pi$-Calculus Modulo as a Universal Proof Language. In *Proof Exchange for Theorem Proving (PxTP)*, pages 28–43, Manchester (UK), June 2012.

[3] R. Bonichon, D. Delahaye, and D. Doligez. Zenon: An Extensible Automated Theorem Prover Producing Checkable Proofs. In *Logic for Programming Artificial Intelligence and Reasoning (LPAR)*, volume 4790 of *LNCS/LNAI*, pages 151–165, Yerevan (Armenia), Oct. 2007. Springer.

[4] G. Burel. Efficiently Simulating Higher-Order Arithmetic by a First-Order Theory Modulo. *Logical Methods in Computer Science (LMCS)*, 7(1):1–31, Mar. 2011.

[5] D. Delahaye, D. Doligez, F. Gilbert, P. Halmagrand, and O. Hermant. Zenon Modulo: When Achilles Outruns the Tortoise using Deduction Modulo. In *Logic for Programming Artificial Intelligence and Reasoning (LPAR)*, volume 8312 of *LNCS/ARCoSS*, pages 274–290, Stellenbosch (South Africa), Dec. 2013. Springer.

[6] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning (JAR)*, 43(4):337–362, Dec. 2009.

[7] The BWare Project, 2012. `http://bware.lri.fr/`.

[8] A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics: An Introduction*. Elsevier, Amsterdam (The Netherlands), 1988. ISBN 0444705066.