Zenon Modulo: When Achilles Outruns the Tortoise using Deduction Modulo

David Delahaye, Damien Doligez, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant

Zenon Modulo: When Achilles Outruns the Tortoise using Deduction Modulo*

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 propose an extension of the tableau-based first order automated theorem prover Zenon 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. We also present a heuristic to perform this latter step automatically, and assess our approach by providing some experimental results obtained on the benchmarks provided by the TPTP library, where this heuristic is able to prove difficult problems in set theory in particular. Finally, we describe an additional backend for Zenon that outputs proof certificates for Dedukti, which is a proof checker based on the $\lambda\Pi$-calculus modulo.

Keywords: Tableaux, Deduction Modulo, Rewriting, Automated Theorem Proving, Proof Checking, Zenon, Dedukti.

1 Introduction

Proof search in axiomatic theories, such as Peano arithmetic and set theory, or decidable fragments (Presburger arithmetic, arrays and pointers, axiomatizations of memory models, etc.) is receiving increasing attention, driven by the applications of formal methods in industrial settings. Leaving axioms wandering among the hypotheses is not a reasonable option, as it induces a combinatorial explosion in the proof search space. Moreover, axioms themselves generally do not bear any specific meaning that could be used by automated theorem provers.

A solution to address this problem is to use a cutting-edge combination of a first order automated theorem proving method (resolution) with theory-specific decision procedures. This approach has drawbacks, namely the need for a specific

* This work is supported by the BWare project [13] (ANR-12-INSE-0010) funded by the INS programme of the French National Research Agency (ANR).

decision procedure for each given theory. This imposes a decidability constraint on the theories that we can work with, as well as a lack of automatability. As a consequence, we lose genericity over the theories. However, SMT solvers are well-suited for industrial applications, where those problems are not a concern.

Our approach is to make use of the advances of deduction modulo [9], which allows us to transform axioms into rewrite rules. For example, Peano arithmetic or Zermelo set theory can be expressed without axioms. 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. Deduction modulo has already been experimented within first order automated theorem provers. This is the case of iProver Modulo [7], where a resolution-based automated theorem prover has been extended to deduction modulo. This is also the case of Super Zenon [10], which is an extension of the Zenon tableau-based automated theorem prover [4] to superdeduction [5], a variant of deduction modulo.

In this paper, we go further along this path by adapting Zenon to deduction modulo itself, and following some of the ideas of [3]. Compared to the approach of Super Zenon, this new tool, called Zenon Modulo, allows us to capture more computational aspects of theories, since deduction modulo also adds the possibility to rewrite over terms, while superdeduction only considers rewrite rules over propositions. Moreover, it will also allow us to compare this extension with that of iProver Modulo, and assess the impact of the integration of deduction modulo into different proof search techniques (i.e. resolution and tableaux).

Another contribution introduced in this paper is a heuristic that automatically transforms any set of axioms (and therefore any theory) into a set of rewrite rules, which can be used during the proof search step of Zenon. With this heuristic, we observe significant improvements over the pure axiomatic proof search of Zenon, as can be seen in the experimental results obtained on the set of problems provided by the TPTP library [12]. In particular, this heuristic appears to be quite appropriate for set theory, where we are able to prove difficult problems.

It should be noted that in the short term, we also plan to work on the dual approach, which consists in building theories modulo manually. In particular, we aim to consider the set theory of the B method [1], in order to apply Zenon Modulo to the verification of proof obligations coming from industrial applications, which is one of the goals of the BWare project [13].

One of the major interests of Zenon to experiment deduction modulo resides in its certifying approach, i.e. its ability to produce proof certificates that can be skeptically checked by other proof assistants such as Coq or Isabelle. Extending Zenon to deduction modulo means to also provide a backend able to check proofs in deduction modulo. To do so, we have provided Zenon with a backend that outputs proofs for Dedukti [2], a proof checker based on the $\lambda\Pi$-calculus modulo.

This paper is organized as follows: in Sec. 2, we first introduce the principles of deduction modulo; we then present, in Sec. 3, the rules of Zenon for deduction modulo, and describe, in Sec. 4, the corresponding implementation and the experimental results obtained on the benchmarks provided by the TPTP library; finally, in Sec. 5, we provide an overview of the Dedukti backend.

2 From Axioms to Deduction Modulo

Deduction modulo [9] focuses on the computational part of a theory, where axioms are transformed into rewrite rules, which induces a congruence over propositions, and where reasoning is performed modulo this congruence. For example, considering the inclusion in set theory $\forall X, Y\ (X \subseteq Y \Leftrightarrow \forall x\ (x \in X \Rightarrow x \in Y))$, the proof of $A \subseteq A$ in sequent calculus has the following form:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\ldots, x \in A \vdash A \subseteq A, x \in A}\,\text{Ax}}{\ldots \vdash A \subseteq A, x \in A \Rightarrow x \in A}\,\Rightarrow\text{R}}{\ldots \vdash A \subseteq A, \forall x\ (x \in A \Rightarrow x \in A)}\,\forall\text{R} \qquad \dfrac{}{\ldots, A \subseteq A \vdash A \subseteq A}\,\text{Ax}}{\dfrac{\ldots, \forall x\ (x \in A \Rightarrow x \in A) \Rightarrow A \subseteq A \vdash A \subseteq A}{A \subseteq A \Leftrightarrow \forall x\ (x \in A \Rightarrow x \in A) \vdash A \subseteq A}\,\wedge\text{L}}{\forall X, Y\ (X \subseteq Y \Leftrightarrow \forall x\ (x \in X \Rightarrow x \in Y)) \vdash A \subseteq A}\,\forall\text{L} \times 2}\,\Rightarrow\text{L}$$

In deduction modulo, the axiom of inclusion can be seen as a computation rule, and therefore replaced by the rewrite rule $X \subseteq Y \longrightarrow \forall x\ (x \in X \Rightarrow x \in Y)$. The previous proof is then transformed as follows:

$$\dfrac{\dfrac{\dfrac{}{x \in A \vdash x \in A}\,\text{Ax}}{\vdash x \in A \Rightarrow x \in A}\,\Rightarrow\text{R}}{\vdash A \subseteq A}\,\forall\text{R},\ A \subseteq A \longrightarrow \forall x\ (x \in A \Rightarrow x \in A)$$

where it can be seen that computations are interleaved with the deduction rules. It can be noticed that the proof is much simpler than the one completed using sequent calculus. In addition to simplicity, deduction modulo also allows for unbounded proof size reduction [6].

There exist some other approaches, which can be considered as variants of deduction modulo. This is the case of superdeduction [5], the formalism at the origin of Super Zenon [10], which proposes to use axioms to enrich the deduction system with new deduction rules (called superdeduction rules). Thus, while deduction modulo integrates some axioms of the theory as computations, superdeduction integrates them as deduction rules, following Prawitz's ideas [11].

However, in contrast with superdeduction, deduction modulo can also capture some computational aspects that are modeled by means of equational axioms. For instance, if we consider an equational sequent calculus with the theory of Peano arithmetic, the proof of $\exists x\ (x + s(0) = s(s(0)))$ is the following without deduction modulo (in the proof context, we only provide the two axioms of Peano arithmetic required to complete the proof, referring to them as $\mathcal{P}$):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{P}, s(0) + s(0) = s(s(0) + 0) \vdash s(0) + s(0) = s(s(0) + 0)}\,\text{Ax}}{\mathcal{P} \vdash s(0) + s(0) = s(s(0) + 0)}\,\forall\text{L} \times 2 \qquad \Pi}{\mathcal{P} \vdash s(0) + s(0) = s(s(0))}\,\text{Subst}_\text{P}}{\begin{cases} \forall x\ (x + 0 = x) \\ \forall x, y\ (x + s(y) = s(x + y)) \end{cases} \vdash \exists x\ (x + s(0) = s(s(0)))}\,\exists\text{R}$$

where Π is the proof expressed as follows:

$$\dfrac{\dfrac{\dfrac{}{\mathcal{P}, s(0)+0=s(0) \vdash s(0)+0=s(0)}\,\mathrm{Ax}}{\mathcal{P} \vdash s(0)+0=s(0)}\,\forall\mathrm{L} \qquad \dfrac{}{\mathcal{P} \vdash s(s(0))=s(s(0))}\,\mathrm{Refl}}{\mathcal{P} \vdash s(s(0)+0)=s(s(0))}\,\mathrm{Subst_P}$$

In deduction modulo, the two axioms are transformed into computation rules on terms, and therefore replaced by the two rewrite rules $x + 0 \longrightarrow x$ and $x + s(y) \longrightarrow s(x + y)$. The corresponding proof is then the following:

$$\dfrac{\dfrac{}{\vdash s(0)+s(0)=s(s(0))}\,\mathrm{Refl},\ s(0)+s(0) \longrightarrow^* s(s(0))}{\vdash \exists x\ (x+s(0)=s(s(0)))}\,\exists\mathrm{R}$$

As previously, it can be noticed that the proof in deduction modulo is much simpler and shorter than the one obtained using the equational sequent calculus.

3 Deduction Modulo Rules for Zenon

In this section, we provide the adaptation of the proof search rules of Zenon to deduction modulo. This mainly consists in extending the usual rules of Zenon [4] by allowing them to work modulo a congruence relation over propositions induced by a set of rewrite rules over propositions and a set of equational axioms and rewrite rules over terms (this extension is partially inspired by the presentation of tableaux modulo presented in [3]).

In the following, we borrow some of the notations, definitions, and propositions of [9], and we call FV the function that returns the set of free variables of a formula. In particular, we introduce the notion of class rewrite system:

Definition 1 (Class Rewrite System). *A term rewrite rule is a pair of terms denoted by $l \longrightarrow r$, where $\mathrm{FV}(r) \subseteq \mathrm{FV}(l)$. An equational axiom is a pair of terms denoted by $l = r$. A proposition rewrite rule is a pair of propositions denoted by $l \longrightarrow r$, where l is an atomic proposition and r is an arbitrary proposition, and where $\mathrm{FV}(r) \subseteq \mathrm{FV}(l)$.*

A class rewrite system is a pair, denoted by $\mathcal{RE}$, consisting of:

- *$\mathcal{R}$: a set of proposition rewrite rules;*
- *$\mathcal{E}$: a set of term rewrite rules and equational axioms.*

Given a class rewrite system $\mathcal{RE}$, the relations $=_{\mathcal{E}}$ and $=_{\mathcal{RE}}$ are the congruences generated respectively by the sets $\mathcal{E}$ and $\mathcal{R} \cup \mathcal{E}$. We then define the notion of $\mathcal{RE}$-rewriting. In the definition below, we use the standard concepts of subterm and term replacement: given an occurrence ω in a proposition P, we write $P_{|\omega}$ for the term or proposition at ω, and $P[t]_\omega$ for the proposition obtained by replacing $P_{|\omega}$ by t in P at ω.

Definition 2 ($\mathcal{RE}$-Rewriting). *Given a class rewrite system $\mathcal{RE}$, the proposition P $\mathcal{RE}$-rewrites to P', denoted by $P \longrightarrow_{\mathcal{RE}} P'$, if $P =_{\mathcal{E}} Q$, $Q_{|\omega} = \sigma(l)$, and $P' =_{\mathcal{E}} Q[\sigma(r)]_{\omega}$, for some rule $l \longrightarrow r \in \mathcal{R}$, some proposition Q, some occurrence ω in Q, and some substitution σ.*

The relation $=_{\mathcal{RE}}$ is not decidable in general, but there are some cases where this relation is decidable depending on the class rewrite system $\mathcal{RE}$ and the rewrite relation $\longrightarrow_{\mathcal{RE}}$, as identified by the following proposition:

Proposition 1 (Decidability of $=_{\mathcal{RE}}$). *If the rewrite relation $\longrightarrow_{\mathcal{RE}}$ is confluent and (weakly) terminating, then the relation $=_{\mathcal{RE}}$ is decidable.*

Given a class rewrite system $\mathcal{RE}$, the proof search rules of Zenon adapted to deduction modulo are summarized in Figs. 1 and 2 (for the sake of simplification, the unfolding and extension rules are omitted), where the "|" symbol is used to separate the formulas of two distinct nodes to be created, ϵ is Hilbert's operator ($\epsilon(x).P(x)$ means some x that satisfies $P(x)$, if it exists, and is considered as a term), capital letters are used for metavariables, and R_r, R_s, R_t, and R_{ts} are respectively reflexive, symmetric, transitive, and transitive-symmetric relations (the corresponding rules also apply to equality). As hinted by the use of Hilbert's operator, the δ-rules are handled by means of ϵ-terms rather than using Skolemization. What we call here metavariables are often named free variables in the tableau-related literature. However, metavariables are not used as variables as they are never substituted, and do not even help to generate a global constraint closing all the branches of the tableau at once; metavariables are instead used as clues (through unification attempts) for the "real" instantiation rules $\gamma_{\forall_{inst}}/\gamma_{\neg\exists_{inst}}$. The proof search rules are applied with the usual tableau method: starting from the negation of the goal, apply the rules in a top-down fashion to build a tree. When all branches are closed (i.e. end with a closure rule), the tree is closed, and this closed tree is a proof of the goal. This algorithm is applied in strict depth-first order: we close the current branch before starting working on another branch. Moreover, we work in a non-destructive way: working on a branch will never change the formulas of another branch.

Compared to [9] and [3], it should be noticed that there is no explicit rule of extended narrowing in the proposed deduction modulo rules for Zenon, since the relation $=_{\mathcal{RE}}$ is actually disseminated in all the initial rules of Zenon. However, the extended narrowing rule is not only a rule that allows us to apply rewrite rules, but also a rule that may suggest instantiations for metavariables. The technique used by Zenon to find those instantiations must therefore be extended as well. Initially, Zenon tries to close a branch by looking for two formulas P and $\neg P$ that can be unified by a substitution σ (over metavariables), and this substitution σ is then used in the $\gamma_{\forall_{inst}}/\gamma_{\neg\exists_{inst}}$ rules corresponding to the unified metavariables. In deduction modulo, this method must be extended as follows: we look for two formulas P and Q s.t. $P =_{\mathcal{RE}} P'$, $Q =_{\mathcal{RE}} \neg Q'$, and there exists a substitution σ s.t. $\sigma(P') =_{\mathcal{E}} \sigma(Q')$. To be complete, we must also extend this metavariable instantiation search to any propositional narrowing (even if we are

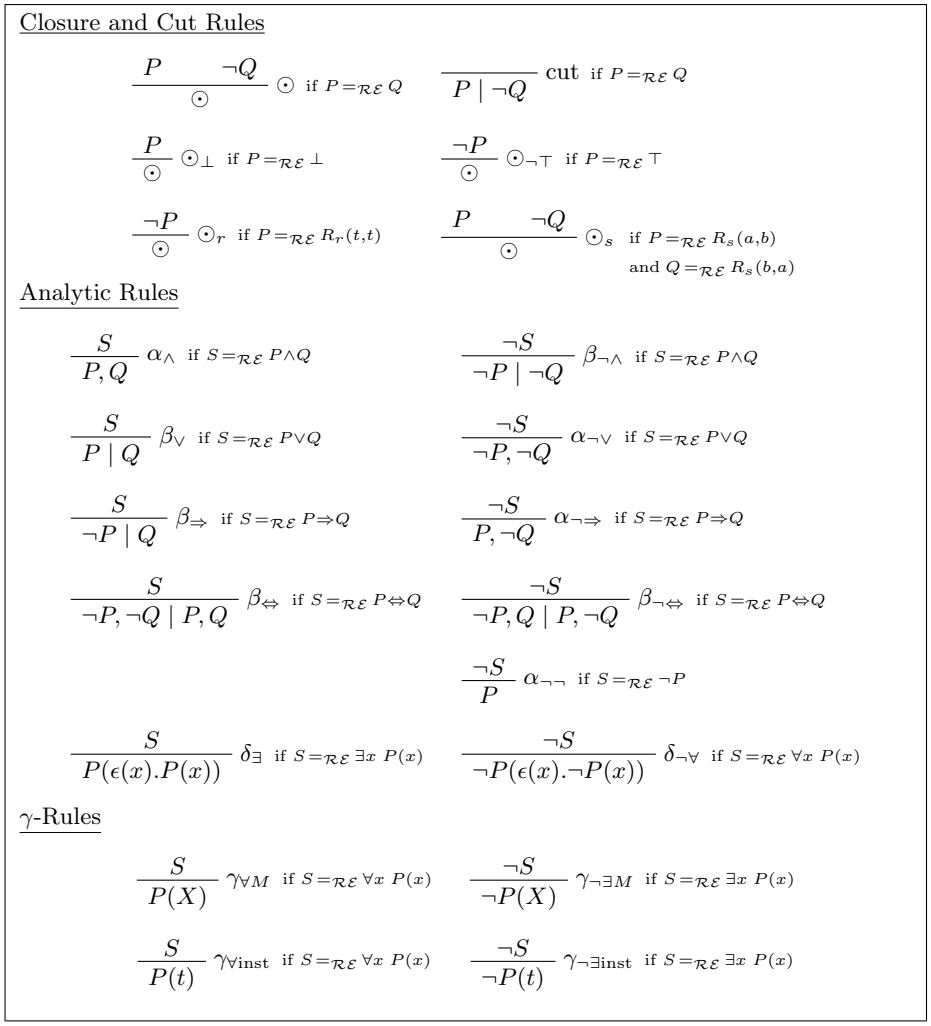

Fig. 1. Deduction Modulo Rules for Zenon (Part 1)

not trying to close a branch): we look for a formula P and a substitution σ s.t. $P =_{\mathcal{RE}} P'$, and there exist $P'_{|\omega}$ and a rule $l \longrightarrow r$ of $\mathcal{R} \cup \mathcal{E}$ s.t. $\sigma(P'_{|\omega}) =_{\mathcal{E}} \sigma(l)$.

4 Implementation and Experimental Results

In this section, we present our implementation of the extension of Zenon to deduction modulo, as well as a heuristic to transform an axiomatic theory into a theory modulo automatically. We also discuss the results obtained on the benchmarks provided by the TPTP library, and we detail a problem that is difficult according to the TPTP ranking, and whose proof is found by our extension.

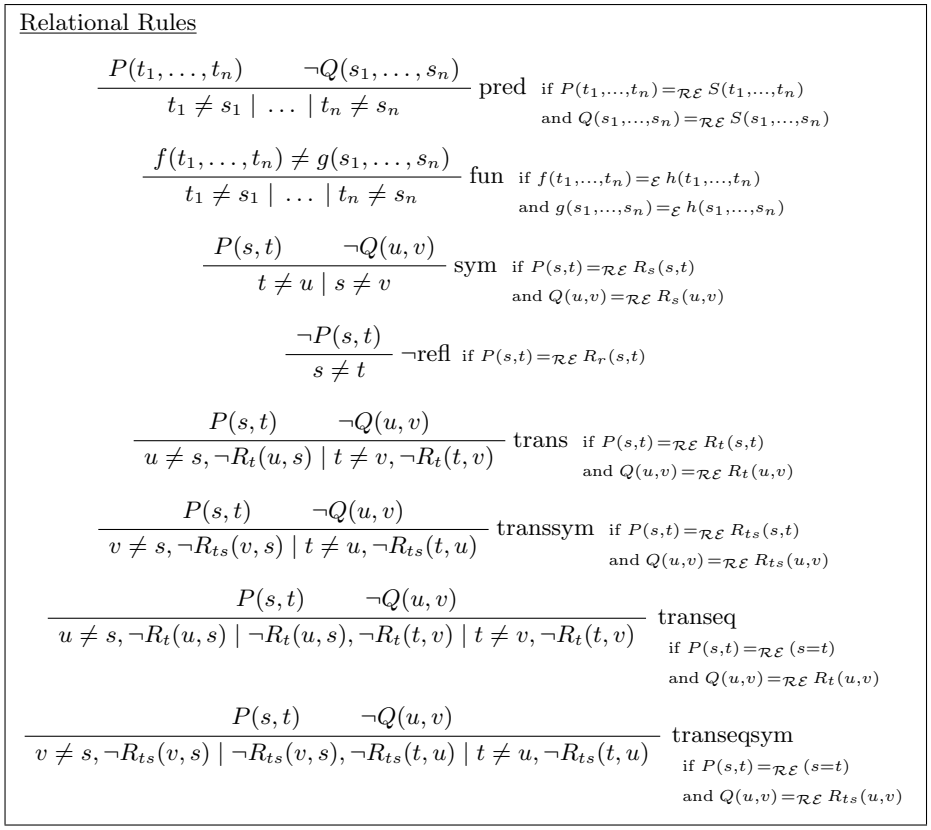

Fig. 2. Deduction Modulo Rules for Zenon (Part 2)

4.1 Implementation

The extension of Zenon to deduction modulo described in Sec. 3 has been implemented in a tool called Zenon Modulo[4]. In this implementation, the class rewrite system $\mathcal{RE}$ is assumed to be a pure rewrite system, i.e. there are only rewrite rules and no equational axiom in $\mathcal{E}$. In addition, the rewrite relation $\longrightarrow_{\mathcal{RE}}$ is assumed to be confluent and (weakly) terminating, and the relation $=_{\mathcal{RE}}$ is therefore decidable (see Prop. 1 in Sec. 3). Thus, given two propositions P and Q, it is sufficient to compare their normal forms (w.r.t. $\longrightarrow_{\mathcal{RE}}$) to decide whether $P =_{\mathcal{RE}} Q$. A solution to deal with the relation $=_{\mathcal{RE}}$ is then to normalize all the formulas of the proof search tree. However, this solution is not efficient in general, as it may perform many useless rewritings. To alleviate this problem, we use an alternate (but equivalent) solution, which consists in performing rewriting only if the formula is a literal. In this case, the terms of the formula are normalized,

[4] Available on demand (sending a mail to the authors).

and one step of proposition rewriting is then applied; if the obtained formula is still a literal, the process is reiterated.

To generate the rewrite system $\mathcal{R} \cup \mathcal{E}$, we have implemented two options. With the first one, the user builds a theory modulo (with axioms and rewrite rules) and provides this theory to Zenon Modulo through an extension of the TPTP input syntax [12], which is one of the input formats used by Zenon, to natively support rewrite rules. With the second option, the user provides a purely axiomatic theory and Zenon Modulo transforms it into a theory modulo automatically. This transformation relies on a heuristic described in Subsec. 4.2.

Zenon Modulo is still in an early stage of development and some features have not been implemented yet. In particular, this is the case of the narrowing for terms and propositions. As a consequence, this leads to incompleteness cases, some of which arise in the benchmarks presented in this paper.

4.2 A Heuristic to Build Theories Modulo

To obtain a theory modulo from an axiomatic theory automatically, we propose a heuristic that generates rewrite rules from axioms based on the shape of the latter. In general, this heuristic does not preserve cut-free completeness. Here are the shapes of axioms that can be handled by our heuristic, as well as the rewrite rules that are generated from them (in the following P is an atomic formula that is not an equation, φ an arbitrary formula, and s and t two terms):

- Axiom of the form $\forall \bar{x}\ (P \Leftrightarrow \varphi)$: the proposition rewrite rule $P \longrightarrow \varphi$ is generated if $\mathrm{FV}(\varphi) \subseteq \mathrm{FV}(P)$, otherwise if φ is a literal and $\mathrm{FV}(P) \subset \mathrm{FV}(\varphi)$ then we apply the heuristic to the formula $\forall \bar{x}\ (\varphi \Leftrightarrow P)$;
- Axiom of the form $\forall \bar{x}\ (\neg P \Leftrightarrow \varphi)$: the proposition rewrite rule $P \longrightarrow \neg\varphi$ is generated if $\mathrm{FV}(\varphi) \subseteq \mathrm{FV}(P)$, otherwise if φ is a literal and $\mathrm{FV}(P) \subset \mathrm{FV}(\varphi)$ then we apply the heuristic to the formula $\forall \bar{x}\ (\varphi \Leftrightarrow \neg P)$;
- Axiom of the form $\forall \bar{x}\ s = t$: the term rewrite rule $s \longrightarrow t$ is generated if $\mathrm{FV}(t) \subseteq \mathrm{FV}(s)$, otherwise the term rewrite rule $t \longrightarrow s$ if $\mathrm{FV}(s) \subset \mathrm{FV}(t)$. In addition, all the axioms expressing the commutativity of a given symbol are excluded from this rule of our heuristic.

In this heuristic, it should be noticed that we exclude the axioms where P is an equation in order to benefit from the equational reasoning of Zenon. To illustrate this heuristic, an example is provided in Subsec. 4.4, where it is shown how a part of the theory is transformed into rewrite rules automatically.

4.3 Experimental Results

We propose a test of our approach on a benchmark drawn from the TPTP library [12] (v.5.5.0), which is a large library of standard benchmark examples for automated theorem proving systems. On this benchmark, we compare Zenon with two different heuristics of Zenon Modulo. The first heuristic consists in only selecting the axioms that can be transformed into proposition rewrite

TPTP Category	Zenon	Zenon Modulo (Prop. Rewriting)	Zenon Modulo (Term & Prop. Rewriting)
FOF 6,659 problems	1,586	1,626 (2.5%) +114 (7.2%) -74 (4.7%)	1,616 (1.9%) +170 (10.7%) -140 (8.8%)
SET 462 problems	149	219 (47%) +78 (52.3%) -8 (5.4%)	222 (49%) +86 (57.7%) -13 (8.7%)

Table 1. Experimental Results over the TPTP Library

rules (equational axioms that can be transformed into term rewrite rules are ignored), while the second one is a greedy heuristic that transforms every axiom that matches one of the patterns described above, producing both term and propositional rewrite rules. The results of this experiment (run on an Intel Xeon X5650 2.67GHz computer, with a memory limit of 1GB and a timeout of 300s) are summarized in Tab. 1, where we have considered the first order problems of the whole library (FOF category) and the problems of set theory (SET category). This table has three columns: the first one provides the number of problems proved by Zenon for each category, while the two other columns show the results of Zenon Modulo with each of the heuristics described above. For Zenon Modulo, there are three numbers per category and heuristic: the left-hand side number is the number of problems proved by Zenon Modulo, while the two right-hand side numbers represent, from top to bottom, the number of problems proved by Zenon Modulo but not by Zenon, and the number of problems proved by Zenon but not by Zenon Modulo.

From the results of Tab. 1, we observe that Zenon Modulo always proves more problems than Zenon whatever the considered category and the selected heuristic. If the gain seems to be low for the whole FOF category (less than 3%) in spite of a significant proportion of problems proved by Zenon Modulo but not by Zenon (about 7% and 11% depending on the heuristic), this is essentially due to incompleteness cases of Zenon Modulo, where narrowing is actually required and for which Zenon succeeds in finding a proof (about 5% and 9% of the cases depending the heuristic). Once narrowing will have been implemented, we can reasonably hope to drastically reduce the number of these cases, and obtain a quite higher gain (probably up to about 11% in the best case scenario).

However, even without narrowing, the gain of Zenon Modulo becomes quite significant for the SET category (about 50% for both heuristics). This very promising result in the SET category tends to show that set theory is a good candidate for automated reasoning with deduction modulo, even when using an automated heuristic. Moreover, as said in the introduction, we plan to apply Zenon Modulo in the context of the B method [1], in particular to verify proof

obligations coming from industrial applications. This is one of the tasks of the BWare project [13]. As the modeling technique used by the B method relies on a customized set theory, which will have a hand-tailored expression as a rewrite system, we can therefore be quite confident in the effectiveness of our tool in the verification of these proof obligations.

The results of the two instances of Zenon Modulo are close, but the heuristic based on term and proposition rewriting proves more problems that are not proved by Zenon (about 11% for FOF and 58% for SET), and once narrowing will be implemented, this heuristic should therefore be preferred.

It should be also noticed that among the 86 problems of the SET category proved by Zenon Modulo (with the heuristic based on term and proposition rewriting) but not by Zenon, there are 29 difficult problems according to the TPTP ranking, namely 29 with a ranking greater than 0.7[5], 9 with a ranking greater than 0.8, and 1 with a ranking greater than 0.9.

4.4 A Nontrivial Example from the TPTP Library

To show the effectiveness of Zenon Modulo, let us describe the proof found for the problem SET815+4, which has the highest ranking (0.91) among those solved, and which deals with the theory of ordinal numbers. The conjecture states that any ordinal number is equal to the union of the elements of its successor. The axioms used to complete this proof are the following:

$$
\begin{array}{rll}
\forall A, B & (A \subseteq B \Leftrightarrow \forall X\ (X \in A \Rightarrow X \in B)) & (\mathit{subset}) \\
\forall A, B & (A =_{\mathrm{set}} B \Leftrightarrow A \subseteq B \wedge B \subseteq A) & (\mathit{eqset}) \\
\forall X, A, B & (X \in A \cup B \Leftrightarrow X \in A \vee X \in B) & (\mathit{union}) \\
\forall X, A & (X \in \{A\} \Leftrightarrow X = A) & (\mathit{singleton}) \\
\forall X, A & (X \in \bigcup A \Leftrightarrow \exists Y\ (Y \in A \wedge X \in Y)) & (\mathit{sum}) \\
\forall A & (A \in \mathrm{On} \Leftrightarrow \mathrm{set}(A) \wedge \forall X\ (X \in A \Rightarrow X \subseteq A) \wedge \\
 & \qquad \mathrm{strict\_wo}(\mathrm{mem\_pred}, A)) & (\mathit{ordinal}) \\
\forall X, A & (X \in A + 1 \Leftrightarrow X \in A \cup \{A\}) & (\mathit{successor})
\end{array}
$$

where set is a predicate that requires the argument to be a set, and where On is the "set" of ordinal numbers, mem_pred the membership relation over ordinal numbers (this relation is related to $\in$ by means of an axiom not shown here as it is not required to complete the proof), and strict_wo a formula encoding the strict well-order relation.

According the rules of the heuristic described in Subsec. 4.2, all these axioms can be turned into proposition rewrite rules (we use the first rule of the heuristic, and each axiom is oriented from left to right). Once this theory modulo has been built, we can then try to prove the conjecture, which is expressed as follows:

$$\forall A\ (A \in \mathrm{On} \Rightarrow \bigcup (A + 1) =_{\mathrm{set}} A)$$

[5] It means that at least 70% of the tested automated theorem provers fail in proving the considered problems.

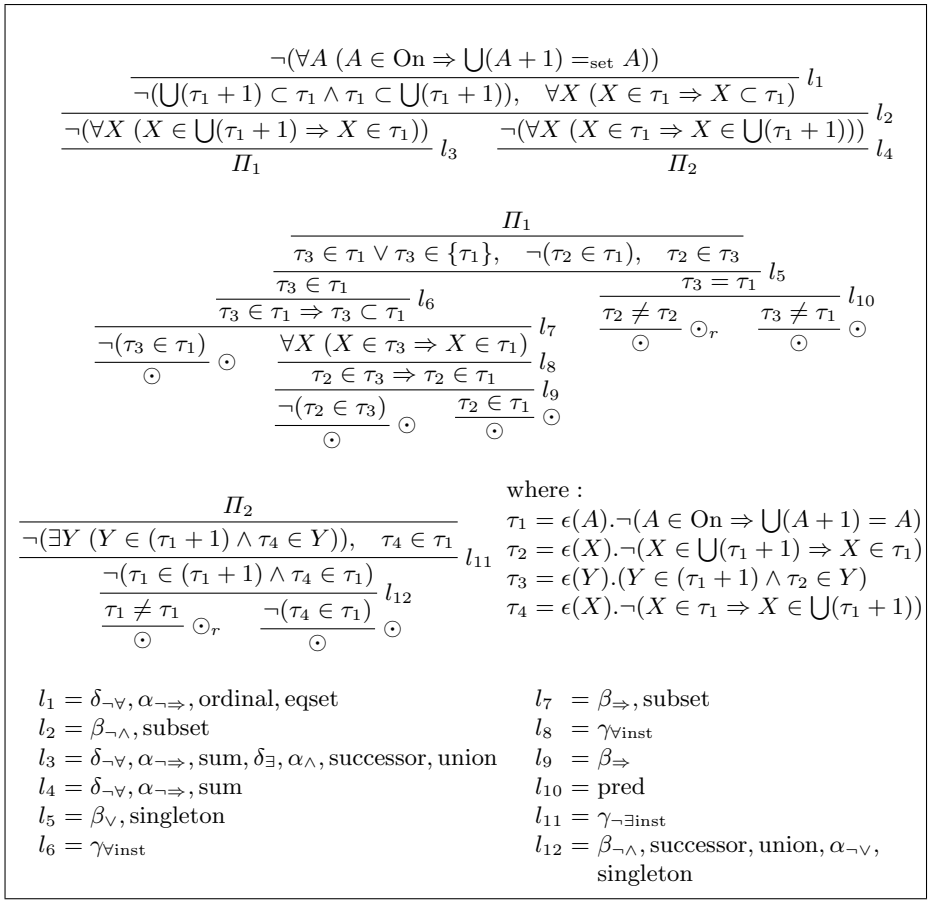

Fig. 3. Proof of Problem SET815+4

When applied to this specification, Zenon Modulo produces the (rather short) proof of Fig. 3. The proof is presented using the rules of Sec. 3, even though these rules are more used for proof search rather than for proof presentation (for that purpose, Zenon actually uses an intermediate format, which is described in detail in [4]). Moreover, to make the presentation more compact, one proof step may consist of several rules. Notice the clever instantiation rule l_6, which cannot be done before the $\delta_\exists$ of l_3.

5 Proof Verification with Dedukti

In this section, we describe the Dedukti backend that has been implemented for Zenon Modulo, and which relies in particular on a proof transformation from classical to constructive logic.

5.1 Dedukti as a Backend for Deduction Modulo

Skeptically checking proof traces produced by an automated theorem prover imposes that the traces contain all the information needed by the proof checker to assert the validity of proofs. A naive way to check proofs performed by Zenon Modulo would be to record rewriting as a special rule, but this method would be very expensive in space, because an arbitrary number of rewrite rules can occur between any consecutive nodes of the proof. To circumvent this problem, the proof checker or the formalization of the proof must not distinguish propositions or terms belonging to the same equivalence class modulo rewriting. The only information needed is the set of rewrite rules, which cannot be bigger than the problem statement itself. Dedukti fits this specific constraint in a simple way: if the set of rewrite rules is translated into Dedukti rewrite rules specified in the header of the proof, any proposition or term used in the proof can be replaced by an equivalent proposition or term modulo rewriting.

Dedukti [2] is a type checker for the $\lambda\Pi$-calculus modulo, which is an extension of the λ-calculus with dependent types and rewrite rules. In order to check proofs of a given logical system, we have to define an embedding of this system into the $\lambda\Pi$-calculus modulo. A logical system is embedded into the $\lambda\Pi$-calculus modulo using declarations of constants and declarations of rewrite rules. As an example, let us consider the implicative fragment of natural deduction and a predicate P. To encode this example, we provide the context of Fig. 4, where the syntax of $\lambda\Pi$-calculus modulo is used. In this context, we can check the trivial proof of $Imp\ P\ P$ by verifying that the term $Intro\ P\ P\ (\lambda t : (Proof\ P).\ t)$ has type $Proof\ (Imp\ P\ P)$. This technique, which consists in defining rules as constants in the $\lambda\Pi$-calculus modulo, is called deep embedding.

However, the $\lambda\Pi$-calculus modulo allows us to define rewrite rules that avoid the definitions of the previous encoding and therefore get shorter proofs. For example, we can replace the *Intro* and *Elim* constants by the rewrite rule $Proof\ (Imp\ A\ B) \longrightarrow (Proof\ A \to Proof\ B)$, where A and B are variables of type *Prop*. The previous term then reduces to $\lambda t : (Proof\ P).t$, which has the same type. This technique, which consists in reusing the language features (here, the $\lambda\Pi$-calculus modulo) is called shallow embedding. In particular, the compu-

$$
\begin{aligned}
Prop &: Type \\
P &: Prop \\
Imp &: Prop \to Prop \to Prop \\
Proof &: Prop \to Type \\
Intro &: \Pi A : Prop.\Pi B : Prop.(Proof\ A \to Proof\ B) \to Proof\ (Imp\ A\ B)) \\
Elim &: \Pi A : Prop.\Pi B : Prop.(Proof\ (Imp\ A\ B)) \to (Proof\ A) \to Proof\ B)
\end{aligned}
$$

Fig. 4. Encoding of Natural Deduction in $\lambda\Pi$-Calculus Modulo

tations (β-reduction) of the initial system are preserved. Notice how deduction modulo allows us to smoothly go from deep to shallow embedding.

The definition of an output to deduction modulo first consists in providing the declaration of the language (terms, predicates, connectives, etc.), and the shallow embedding of the natural deduction rules that is close to $\lambda\Pi$-calculus. We then proceed to the definition of the rewrite rules for propositions and terms, according to the input set of rewrite rules. Once this encoding has been defined, proofs can be checked in this context. As an example, we can extend the previous encoding with a predicate Q and two axioms $Imp\ Q\ (Imp\ P\ P)$ and $Imp\ (Imp\ P\ P)\ Q$. The heuristic of Sec. 4 will replace these two axioms by a single rewrite rule (these two axioms represent an equivalence), and we just have to declare the rewrite rule $Q \longrightarrow Imp\ P\ P$, modulo which Q admits the same proof as before, i.e. $\lambda t : (Proof\ P).t$.

5.2 From Classical to Constructive Proofs

Zenon's logic is classical and expressed in a formalism very close to sequent calculus [14]. As a consequence, using Dedukti as a backend requires two steps: the first one is to translate classical proofs of Zenon into proofs of constructive sequent calculus with equality, which we discuss here, and the second one is a standard translation from sequent calculus to natural deduction.

The translation from classical to constructive logic relies on an optimized double-negation translation [15], presented in Tab. 2, where other connectives ($\neg$ and $\top$) are defined as usual through $\Rightarrow$ and $\bot$. With these definitions, it is possible to show that a formula A has a classical proof if and only if $\varphi(A)$ has a constructive proof. The purpose of defining the three functions of Tab. 2 is that the algorithm introduces a minimal number of double negations. In particular, we improve over both Kuroda's and Gödel's translations [15] by combining their principles: double negation only after universal quantifiers and double negation only in front of disjunctive and existentially quantified propositions, respectively. At top level, we push double negations as far as possible inside the formula, as Gödel, in front of the first encountered disjunction/existential quantifier (the role of φ); at this point, we stop introducing double negations until, as Kuroda, we meet a universal quantifier, in which case we start again the process (the role of ψ). We refine our translation with the polarity of formulas (i.e. the side of the sequent on which the formula appears): if a formula appears on the left-hand side of a sequent, we do not put a double-negation in front of it (the role of χ).

Furthermore, the algorithm analyses the structure of the classical proof in order to remove more double negations. For instance, a disjunction can be proved constructively even in a classical calculus. As a consequence, the statement of many proofs remains unchanged. This is the case of the TPTP problem SET815+4, discussed in Sec. 4, whose proof certificate (expressed in $\lambda\Pi$-calculus modulo), is given in Figs. 5 and 6 of Appx. A. This algorithm of proof transformation has been implemented for Zenon Modulo to produce Dedukti proof certificates. In particular, Zenon Modulo succeeds in producing a proof certificate for the previous problem SET815+4, which is correctly checked by Dedukti.

	φ	χ	ψ
$A \wedge B$	$\varphi(A) \wedge \varphi(B)$	$\chi(A) \wedge \chi(B)$	$\psi(A) \wedge \psi(B)$
$A \vee B$	$\neg\neg(\psi(A) \vee \psi(B))$	$\chi(A) \vee \chi(B)$	$\psi(A) \vee \psi(B)$
$A \Rightarrow B$	$\chi(A) \Rightarrow \varphi(B)$	$\psi(A) \Rightarrow \chi(B)$	$\chi(A) \Rightarrow \psi(B)$
$\forall x\ A$	$\forall x\ \varphi(A)$	$\forall x\ \chi(A)$	$\forall x\ \varphi(A)$
$\exists x\ A$	$\neg\neg\exists x\ \psi(A)$	$\exists x\ \chi(A)$	$\exists x\ \psi(A)$
$\bot$	$\bot$	$\bot$	$\bot$
Atomic P	$\neg\neg P$	P	P

Table 2. Translation from Classical to Constructive Logic

6 Conclusion

We have proposed an extension of the tableau-based first order automated theorem prover Zenon to deduction modulo. This extension essentially consists in considering a part of a given theory as rewrite rules (over terms and propositions), and integrating these rewrite rules into the proof search rules of Zenon. We have also presented an implementation of this extension, called Zenon Modulo, as well as a heuristic to turn axioms of theories into rewrite rules automatically. This new tool significantly improves the proof search of Zenon, as shown by the experimental results obtained on the benchmarks provided by the TPTP library. In particular, this is the case of the SET category, where Zenon Modulo is able to prove difficult problems according to the TPTP ranking. Finally, we have also described an additional backend for Zenon that outputs proof certificates for Dedukti, which is a proof checker based on the $\lambda\Pi$-calculus modulo.

As future work, we first aim to complete our implementation to deal with narrowing when trying to find instantiations for metavariables. This will allow us to ensure completeness for our extension of Zenon, even though narrowing may paradoxically widen the proof search space in some cases (e.g., in set theory, a metavariable representing a set can be "narrowed" using the major part of rewrite rules defining the set operators). To deal with these cases, we will probably implement a switch that will allow us to activate/deactivate the use of narrowing. It might be also desirable to extend our proof search method to polarized deduction modulo [8], which is a refinement of deduction modulo to deal with theories formed with axioms using implications. This extension would allow us to consider more theories, where a significant part of the axioms are expressed neither by equivalences, nor by equations, but only by implications. Finally, in the framework of the BWare project [13], we plan to apply this tool in the context of the B method [1], with in particular the verification of proof obligations coming from industrial applications. To achieve this task, we have to build a theory modulo for the modeling method of B, which is actually a typed set theory, and we should be able to reuse some ideas of [10], where a B set theory is proposed using superdeduction. Given the very promising results

of our tool in the SET category of the TPTP library, we are quite confident in the effectiveness of our tool in the verification of B proof obligations.

Acknowledgement. Many thanks to G. Dowek, F. Irigoin, and P. Jouvelot for their detailed comments on this paper, to G. Burel for helpful discussions, and to the Deducteam Inria research team for the many interactions.

References

1. J.-R. Abrial. *The B-Book, Assigning Programs to Meanings*. Cambridge University Press, Cambridge (UK), 1996. ISBN 0521496195.
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. TaMeD: A Tableau Method for Deduction Modulo. In *International Joint Conference on Automated Reasoning (IJCAR)*, volume 3097 of *LNCS*, pages 445–459, Cork (Ireland), July 2004. Springer.
4. 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.
5. P. Brauner, C. Houtmann, and C. Kirchner. Principles of Superdeduction. In *Logic in Computer Science (LICS)*, pages 41–50, Wrocław (Poland), July 2007. IEEE Computer Society Press.
6. 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.
7. G. Burel. Experimenting with Deduction Modulo. In *Conference on Automated Deduction (CADE)*, volume 6803 of *LNCS/LNAI*, pages 162–176, Wrocław (Poland), July 2011. Springer.
8. G. Dowek. What is a Theory? In *Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2285 of *LNCS*, pages 50–64, Antibes Juan-les-Pins (France), Mar. 2002. Springer.
9. G. Dowek, T. Hardin, and C. Kirchner. Theorem Proving Modulo. *Journal of Automated Reasoning (JAR)*, 31(1):33–72, Sept. 2003.
10. M. Jacquel, K. Berkani, D. Delahaye, and C. Dubois. Tableaux Modulo Theories using Superdeduction: An Application to the Verification of B Proof Rules with the Zenon Automated Theorem Prover. In *International Joint Conference on Automated Reasoning (IJCAR)*, volume 7364 of *LNCS*, pages 332–338, Manchester (UK), June 2012. Springer.
11. D. Prawitz. Natural Deduction. A Proof-Theoretical Study. *Stockholm Studies in Philosophy*, 3, 1965.
12. 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.
13. The BWare Project, 2012. `http://bware.lri.fr/`.
14. A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, Cambridge (UK), 1996. ISBN 0521779111.
15. A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics: An Introduction*. Elsevier, Amsterdam (The Netherlands), 1988. ISBN 0444705066.

A Proof Certificate for Problem SET815+4 in $\lambda\Pi$-Calculus Modulo

Declarations:

$Term, Prop : Type$
$Proof : Prop \to Type$
$\wedge, \vee, \Rightarrow : Prop \to Prop \to Prop$
$\forall, \exists : (Term \to Prop) \to Prop$
$\mathrm{On}, \mathrm{mem\_pred} : Term$
$+1, \{\ \}, \bigcup : Term \to Term$
$\cup : Term \to Term \to Term$
$\mathrm{set} : Term \to Prop$
$=, =_{\mathrm{set}}, \in, \subset, \mathrm{strict\_wo} : Term \to Term \to Prop$

Rewrite Rules:

$[A, B : Prop] Proof\ (A \wedge B) \longrightarrow \Pi P : Prop.$
$\quad (Proof\ A \to Proof\ B \to Proof\ P) \to Proof\ P$
$[A, B : Prop] Proof\ (A \vee B) \longrightarrow \Pi P : Prop.(Proof\ A \to Proof\ P) \to$
$\quad (Proof\ B \to Proof\ P) \to Proof\ P$
$[A, B : Prop] Proof\ (A \Rightarrow B) \longrightarrow Proof\ A \to Proof\ B$
$[A : Term \to Prop] Proof\ (\forall\ A) \longrightarrow \Pi x : Term.Proof\ (A\ x)$
$[A : Term \to Prop] Proof\ (\exists\ A) \longrightarrow \Pi P : Prop.$
$\quad (\Pi x : Term.Proof\ (A\ x) \to Proof\ P) \to Proof\ P$
$[x, y : Term] Proof\ (x = y) \longrightarrow \Pi P : (Term \to Prop).Proof\ ((P\ x) \Rightarrow (P\ y))$
$[A, B : Term] A \subset B \longrightarrow \forall\ (\lambda X : Term.X \in A \Rightarrow X \in B)$
$[A, B : Term] A =_{\mathrm{set}} B \longrightarrow A \subset B \wedge B \subset A$
$[A, B, X : Term] X \in A \cup B \longrightarrow X \in A \vee X \in B$
$[A, X : Term] X \in \{A\} \longrightarrow X = A$
$[A, X : Term] X \in \bigcup(A) \longrightarrow \exists(\lambda Y : Term.Y \in A \wedge X \in Y)$
$[A : Term] A \in \mathrm{On} \longrightarrow \mathrm{set}(A) \wedge \mathrm{strict\_wo}(\mathrm{mem\_pred}, A) \wedge$
$\quad \forall\ (\lambda X : Term.X \in A \Rightarrow X \subset A)$
$[A, X : Term] X \in (A + 1) \longrightarrow X \in A \cup \{A\}$

where $[\cdot]$ is the typing context of a rewrite rule.

Fig. 5. Context of the Proof of Problem SET815+4 in $\lambda\Pi$-Calculus Modulo

Proof (λ-term):

$$
\begin{array}{l}
\lambda A : \mathit{Term}.\lambda H0 : \mathit{Proof}\ (A \in \mathrm{On}).H0\ (\bigcup(A+1) =_{\mathrm{set}} A) \\
\ (\lambda H1 : \mathit{Proof}\ (\mathrm{set}(A)). \\
\ \lambda H2 : \mathit{Proof}\ (\mathrm{strict\_wo}(\mathrm{mem\_pred}, A) \wedge \\
\quad (\forall\ (\lambda X : \mathit{Term}.X \in A \Rightarrow X \subset A))).H2\ (\bigcup(A+1) =_{\mathrm{set}} A) \\
\ (\lambda H3 : \mathit{Proof}\ (\mathrm{strict\_wo}(\mathrm{mem\_pred}, A)). \\
\ \lambda H4 : \mathit{Proof}\ \forall\ (\lambda X : \mathit{Term}.(X \in A \Rightarrow X \subset A)).\lambda P0 : \mathit{Prop}. \\
\ \lambda H6 : (\mathit{Proof}\ (\bigcup(A+1) \subset A) \to \mathit{Proof}\ (A \subset \bigcup(A+1)) \to \mathit{Proof}\ P0).H6 \\
\ (\lambda B : \mathit{Term}.\lambda H7 : \mathit{Proof}\ (B \in \bigcup(A+1)).H7\ (B \in A) \\
\ (\lambda D : \mathit{Term}.\lambda H8 : \mathit{Proof}\ (D \in (A+1) \wedge B \in D).H8\ (B \in A) \\
\ (\lambda H9 : \mathit{Proof}\ (D \in (A+1)).\lambda H10 : \mathit{Proof}\ (B \in D).H9\ (B \in A) \\
\ (\lambda H11 : \mathit{Proof}\ (D \in A).((((H4\ D)\ H11)\ B)\ H10)) \\
\ (\lambda H12 : \mathit{Proof}\ (D = A). \\
\ (\lambda P1 : (\mathit{Term} \to \mathit{Prop}).\lambda H13 : \mathit{Proof}\ (P1\ B).H13)\ (\lambda X : \mathit{Term}.X \in A) \\
\ (H12\ (\lambda X : \mathit{Term}.B \in X)\ H10))))) \\
\ (\lambda C : \mathit{Term}.\lambda H14 : \mathit{Proof}\ (C \in A).\lambda P2 : \mathit{Prop}. \\
\ \lambda H15 : (\Pi Y : \mathit{Term}.\mathit{Proof}\ (Y \in (A+1) \wedge C \in Y) \to \mathit{Proof}\ P2).H15\ A \\
\ (\lambda P3 : \mathit{Prop}.\lambda H16 : (\mathit{Proof}\ (A \in A) \to \mathit{Proof}\ (C \in A) \to \mathit{Proof}\ P3).H16 \\
\ (\lambda P4 : \mathit{Prop}.\lambda H17 : (\mathit{Proof}\ (A \in A) \to \mathit{Proof}\ P4). \\
\ \lambda H18 : (\mathit{Proof}\ (A = A) \to \mathit{Proof}\ P4).H18 \\
\ (\lambda P5 : (\mathit{Term} \to \mathit{Prop}).\lambda H19 : \mathit{Proof}\ (P5\ A).H19))\ H14))))
\end{array}
$$

Fig. 6. Proof of Problem SET815+4 in $\lambda\Pi$-Calculus Modulo