Checking Zenon Modulo Proofs in Dedukti

Raphaël Cauderlier, Pierre Halmagrand

Checking Zenon Modulo Proofs in Dedukti *

Raphaël Cauderlier Pierre Halmagrand
Cnam - Inria
Paris, France
raphael.cauderlier@inria.fr pierre.halmagrand@inria.fr

Dedukti has been proposed as a universal proof checker. It is a logical framework based on the $\lambda\Pi$-calculus modulo that is used as a backend to verify proofs coming from theorem provers, especially those implementing some form of rewriting. We present a shallow embedding into Dedukti of proofs produced by Zenon Modulo, an extension of the tableau-based first-order theorem prover Zenon to deduction modulo and typing. Zenon Modulo is applied to the verification of programs in both academic and industrial projects. The purpose of our embedding is to increase the confidence in automatically generated proofs by separating untrusted proof search from trusted proof verification.

1 Introduction

Program verification using deductive methods has become a valued technique among formal methods, with practical applications in industry. It guarantees a high level of confidence regarding the correctness of the developed software with respect to its specification. This certification process is generally based on the verification of a set of proof obligations, generated by deductive verification tools. Unfortunately, the number of proof obligations generated may be very high. To address this issue, deductive verification tools often rely on automated deduction tools such as first-order Automated Theorem Provers (ATP) or Satisfiability Modulo Theories solvers (SMT) to automatically discharge a large number of those proof obligations. For instance, Boogie is distributed with the SMT Z3 [4] and the Why3 platform with Alt-Ergo [16]. After decades of constant work, ATP and SMT have reached a high level of efficiency and now discharge more proof obligations than ever. At the end, many of these program verification tools use their corresponding ATP or SMT as oracles. The main concern here is the level of confidence users give to them. These programs are generally large software, consisting of dozens of thousands of lines of code, and using some elaborate heuristics, with some ad hoc proof traces at best, and with a simple "yes or no" binary answer at worst.

A solution, stated by Barendregt and Barendsen [3] and pursued by Miller [19] among others, relies on the concept of proof certificates. ATP and SMT should be seen as proof-certificate generators. The final "yes or no" answer is therefore left to an external proof checker. In addition, Barendregt and Barendsen proposed that proof checkers should satisfy two principles called the De Bruijn criterion and the Poincaré principle. The former states that proof checkers have to be built on a light and auditable kernel. The latter recommends that they distinguish reasoning and computing and that it should not be necessary to record pure computational steps.

Relying on an external proof checker to verify proofs strongly increases the trust we give them, but it also provides a common framework to express proofs. A profit made by using this common framework is the possibility to share proofs coming from different theorem provers, relying on different proof systems. But nothing comes for free, and using the same proof checker does not guarantee in general that we can

*This work has received funding from the BWare project (ANR-12-INSE-0010) funded by the INS programme of the French National Research Agency (ANR).

Submitted to:
PxTP 2015

share proofs because formulæ and proofs can be translated in incompatible ways. Translation of proofs must rely on a shallow embedding in the sense proposed by Burel [10]: it reuses the features of the target language. It does not introduce new axioms and constants for logical symbols and inference rules. Connectives and binders of the underlying logic of ATP are translated to their corresponding connectives and binders in the target language. In addition, a shallow embedding preserves the computational behavior of the original ATP and the underlying type system of the logic.

In this paper, we present a shallow embedding of Zenon Modulo proofs into the proof checker Dedukti, consisting of an encoding of a typed classical sequent calculus modulo into the $\lambda\Pi$-calculus modulo ($\lambda\Pi^{\equiv}$ for short). Zenon Modulo [13] is an extension to deduction modulo [15] of the first-order tableau-based ATP Zenon [8]. It has also been extended to support ML polymorphism by implementing the TFF1 format [5]. Dedukti [7] is a proof checker that implements $\lambda\Pi^{\equiv}$, a proof language that has been proposed as a proof standard for proof checking and interoperability. This embedding is used to certify proofs in two different projects: FoCaLiZe [17], a programming environment to develop certified programs and based on a functional programming language with object-oriented features, and BWare [14], an industrial research project that aims at providing a framework for the automated verification of proof obligations coming from the B method [1]. The main benefit of Zenon Modulo and Dedukti relies on deduction modulo. Deduction modulo is an extension of first-order logic that allows reasoning modulo a congruence relation over propositions. It is well suited for automated theorem proving when dealing with theories since it turns axioms into rewrite rules. Using rewrite rules during proof search instead of reasoning on axioms lets provers focus on the challenging part of proofs, speeds up the tool and reduces the size of final proof trees [12].

Zenon was designed to support FoCaLiZe as its dedicated deductive tool and to generate proof certificates for Coq. Extension to deduction modulo constrains us to use a proof checker that can easily reason modulo rewriting. Dedukti is a good candidate to meet this specification. A previous embedding of Zenon Modulo proofs into Dedukti, based on a $\neg\neg$ translation [13], was implemented as a tool to translate classical proofs into constructive ones. This tool has the benefit to be shallower since it does not need add the excluded middle as an axiom into the target logic defined in Dedukti, but in return this transformation may be very time-consuming [12] and was not scalable to large proofs like those produced in BWare. The closest related work is the shallow embedding of resolution and superposition proofs into Dedukti proposed by Burel [10] and implemented in iProver Modulo [9]. Our embedding is close enough to easily share proofs of Zenon Modulo and iProver Modulo in Dedukti, at least for the subset of untyped formulæ.

The first contribution presented in this paper consists in the encoding into $\lambda\Pi^{\equiv}$ of typed deduction modulo and a set of translation functions into $\lambda\Pi^{\equiv}$ of theories expressed in this logic. Another contribution of this paper is the extension to deduction modulo and types of the sequent-like proof system LLproof which is the output format of Zenon Modulo proofs. The latter contribution is the embedding of this proof system into $\lambda\Pi^{\equiv}$ and the associated translation function for proofs coming from this system.

This paper is organized as follows: in Sec. 2, we introduce typed deduction modulo; in Sec. 3, we present $\lambda\Pi^{\equiv}$, its proof checker Dedukti, and a canonical encoding of typed deduction modulo in $\lambda\Pi^{\equiv}$; Sec. 4 introduces the ATP Zenon Modulo, the proof system LLproof used by Zenon Modulo to output proofs; and the translation scheme implemented as the new output of Zenon Modulo; finally, in Sec. 5, we present some examples and results to assess our implementation.

2 Typed Deduction Modulo

The Poincaré principle, as stated by Barendregt and Barendsen [3], makes a distinction between deduction and computation. Deduction may be defined using a set of inference rules and axioms, while computation consists mainly in simplification and unfolding of definitions. When dealing with axiomatic theories, keeping all axioms on the deduction side leads to inefficient proof search since the proof-search space grows with the theory. For instance, proving the following statement:

$$\mathsf{fst}(a,a) = \mathsf{snd}(a,a)$$

where a is a constant, and fst and snd are defined by:

$$\forall x,y.\ \mathsf{fst}(x,y) = x \qquad\qquad \forall x,y.\ \mathsf{snd}(x,y) = y$$

and with the reflexivity axiom:

$$\forall x.\ x = x$$

using a usual automated theorem proving method such as tableau, will generate some useless boilerplate proof steps, whereas a simple unfolding of definitions of fst and snd directly leads to the formula $a = a$.

Deduction modulo was introduced by Dowek, Hardin and Kirchner [15] as a logical formalism to deal with axiomatic theories in automated theorem proving. The proposed solution is to remove computational arguments from proofs by reasoning modulo a decidable congruence relation $\equiv$ on propositions. Such a congruence may be generated by a confluent and terminating system of rewrite rules (sometimes extended by equational axioms).

In our example, the two definitions may be replaced by the rewrite rules:

$$\mathsf{fst}(x,y) \longrightarrow x \qquad\qquad \mathsf{snd}(x,y) \longrightarrow y$$

And we obtain the following equivalence between propositions:

$$(\mathsf{fst}(a,a) = \mathsf{snd}(a,a)) \ \equiv\ (a = a)$$

Reasoning with several theories at the same time is often necessary in practice. For instance, in the BWare project, almost all proof obligations combine the theory of booleans, arithmetic and set theory. In this case, we have to introduce an expressive enough type system to ensure that an axiom about booleans, for instance $\forall x.\ x = \mathsf{true} \vee x = \mathsf{false}$, will not be used with a term that has another type. An input format for ATP called TFF1 [5] has been proposed recently by Blanchette and Paskevich to deal with first-order problems with polymorphic types. We propose to extend this format to deduction modulo.

We now introduce the notion of typed rewrite system, extending notations of Dowek et al. [15]. In the following, $\mathrm{FV}(t)$ stands for the set of free variables of t where t is either a TFF1 term or a TFF1 formula.

Definition (Typed Rewrite System)
A term rewrite rule is a pair of TFF1 terms l and r together with a TFF1 typing context Δ denoted by $l \longrightarrow_\Delta r$, where $\mathrm{FV}(r) \subseteq \mathrm{FV}(l) \subseteq \Delta$. It is well-typed in a theory $\mathscr{T}$ if l and r can be given the same type A in $\mathscr{T}$ using Δ to type free variables. A proposition rewrite rule is a pair of TFF1 formulæ l and r together with a typing context Δ denoted by $l \longrightarrow_\Delta r$, where l is an atomic formula and r is an arbitrary formula,

Types	τ	$::=$	α	(type variable)
		$\mid$	$T(\tau_1,\ldots,\tau_m)$	(type constructor)
Terms	e	$::=$	x	(term variable)
		$\mid$	$f(\tau_1,\ldots,\tau_m;e_1,\ldots,e_n)$	(function)
Formulæ	φ	$::=$	$\top \mid \bot$	(true, false)
		$\mid$	$\neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \Rightarrow \varphi_2 \mid \varphi_1 \Leftrightarrow \varphi_2$	(logical connectors)
		$\mid$	$e_1 =_\tau e_2$	(term equality)
		$\mid$	$P(\tau_1,\ldots,\tau_m;e_1,\ldots,e_n)$	(predicate)
		$\mid$	$\forall x:\tau.\ \varphi(x) \mid \exists x:\tau.\ \varphi(x)$	(term quantifiers)
		$\mid$	$\forall_{\mathsf{type}}\alpha : \mathsf{type}.\ \varphi(\alpha) \mid \exists_{\mathsf{type}}\alpha : \mathsf{type}.\ \varphi(\alpha)$	(type quantifiers)
Context	Δ	$::=$	$\emptyset$	(empty context)
		$\mid$	$\Delta, x:\tau$	(declaration)
Theory	$\mathscr{T}$	$::=$	$\emptyset$	(empty theory)
		$\mid$	$\mathscr{T}, T/m$	(m-ary type constructor declaration
		$\mid$	$\mathscr{T}, f : \Pi\vec{\alpha}.\ \vec{\tau} \to \tau$	(function declaration)
		$\mid$	$\mathscr{T}, P : \Pi\vec{\alpha}.\ \vec{\tau} \to o$	(predicate declaration)
		$\mid$	$\mathscr{T}, \mathrm{name} : \varphi$	(axiom)
		$\mid$	$\mathscr{T}, l \longrightarrow_\Delta r$	(rewrite rule)

Figure 1: Syntax of $\mathsf{TFF1}^{\equiv}$

and where $\mathrm{FV}(r) \subseteq \mathrm{FV}(l) \subseteq \Delta$. It is well-typed in a theory $\mathscr{T}$ if both l and r are well-formed formulæ in $\mathscr{T}$ using Δ to type free variables.

A typed rewrite system is a set $\mathscr{R}$ of proposition rewrite rules along with a set $\mathscr{E}$ of term rewrite rules. Given a rewrite system $\mathscr{R}\mathscr{E}$, the relation $=_{\mathscr{R}\mathscr{E}}$ denotes the congruence generated by $\mathscr{R}\mathscr{E}$. It is well-formed in a theory $\mathscr{T}$, if all its rewrite rules are well-typed in $\mathscr{T}$.

The notion of TFF1 theory can be extended with rewrite rules; we call the resulting logic $\mathsf{TFF1}^{\equiv}$. Its syntax is given in Fig. 1.

3 Dedukti

The $\lambda\Pi$-calculus [2] is the simplest Pure Type System featuring dependent types. It is commonly used as a logical framework for encoding logics [18]. The $\lambda\Pi$-calculus modulo, presented in Fig. 2, is an extension of the $\lambda\Pi$-calculus with rewriting. The $\lambda\Pi$-calculus modulo (abbreviated as $\lambda\Pi^{\equiv}$) has successfully been used to encode many logical systems (Coq [6], HOL, iProver Modulo [10], FoCaLiZe) using shallow embeddings.

In $\lambda\Pi^{\equiv}$, conversion goes beyond simple β-equivalence since it is extended by a custom rewrite system. When this rewrite system is both strongly normalizing and confluent, each term gets a unique (up to α-conversion) normal form and both conversion and type-checking become decidable. Dedukti is an implementation of this decision procedure.

Burel [10] defines two encodings of deduction modulo in Dedukti: a deep encoding $|\varphi|$ in which logical connectives are simply declared as Dedukti constants and a shallow encoding $\|\varphi\| := \mathsf{prf}\ |\varphi|$ using a decoding function prf for translating connectives to their impredicative encodings. In Sec. 3.1 and Sec. 3.2, we extend these encodings to $\mathsf{TFF1}^{\equiv}$.

Syntax

$$\begin{array}{lcl} s & ::= & \mathsf{Type} \mid \mathsf{Kind} \\ t & ::= & x \mid t\ t \mid \lambda x : t.t \mid \Pi x : t.t \mid s \\ \Delta & ::= & \emptyset \mid \Delta, x : t \\ \Gamma & ::= & \emptyset \mid \Gamma, x : t \mid \Gamma, t \hookrightarrow_\Delta t \end{array}$$

Well-formdness

$$\frac{}{\emptyset \vdash}\ (\text{Empty}) \qquad \frac{\Gamma \vdash \qquad \Gamma \vdash A : s \qquad x \notin \Gamma}{\Gamma, x : A \vdash}\ (\text{Decl}) \qquad \frac{\Gamma, \Delta \vdash l : A \qquad \Gamma, \Delta \vdash A : \mathsf{Type} \qquad \Gamma, \Delta \vdash r : A \qquad FV(r) \subseteq FV(l) \subseteq \Delta}{\Gamma, l \hookrightarrow_\Delta r \vdash}\ (\text{Rew})$$

Typing

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathsf{Type} : \mathsf{Kind}}\ (\text{Sort}) \qquad \frac{\Gamma \vdash \qquad x : A \in \Gamma}{\Gamma \vdash x : A}\ (\text{Var})$$

$$\frac{\Gamma \vdash t_1 : \Pi x : A.B(x) \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1\ t_2 : B(t_1)}\ (\text{App}) \qquad \frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x : A \vdash t : B(x) \qquad \Gamma, x : A \vdash B(x) : s}{\Gamma \vdash \lambda x : A.t(x) : \Pi x : A.B(x)}\ (\text{Abs})$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x : A \vdash B(x) : s}{\Gamma \vdash \Pi x : A.B(x) : s}\ (\text{Prod}) \qquad \frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : s \qquad A \equiv_{\beta\Gamma} B}{\Gamma \vdash t : B}\ (\text{Conv})$$

Figure 2: The $\lambda\Pi$-calculus modulo

3.1 Deep Embedding of Typed Deduction Modulo in Dedukti

In Fig. 3, for each symbol of our first-order typed logic, we declare its corresponding symbol into $\lambda\Pi^{\equiv}$. In $\lambda\Pi^{\equiv}$, types cannot be passed as arguments (no polymorphism) so we have to translate $\mathsf{TFF1}^{\equiv}$ types as Dedukti terms. The Dedukti type of translated $\mathsf{TFF1}^{\equiv}$ types is type and we can see an inhabitant of type as a Dedukti type thanks to the term function.

In Fig. 4, we define a direct translation of $\mathsf{TFF1}^{\equiv}$ in Dedukti. It is correct in the following sense:

- if the theory $\mathscr{T}$ is well-formed in $\mathsf{TFF1}^{\equiv}$, then $|\mathscr{T}| \vdash$.
- if τ is a well-formed $\mathsf{TFF1}^{\equiv}$ type in a theory $\mathscr{T}$, then $|\mathscr{T}| \vdash |\tau| : \mathsf{type}$.
- if t is a $\mathsf{TFF1}^{\equiv}$ term of type τ in a theory $\mathscr{T}$, then $|\mathscr{T}| \vdash |t| : \mathsf{term}\ |\tau|$.
- if φ is a well-formed $\mathsf{TFF1}^{\equiv}$ formula in a theory $\mathscr{T}$, then $|\mathscr{T}| \vdash |\varphi| : \mathsf{Prop}$.

3.2 From Deep to Shallow

Following Burel [10], we add rewrite rules defining the decoding function prf in Fig. 5 using the usual impredicative encoding of connectives. This transforms our deep encoding of $\mathsf{TFF1}^{\equiv}$ into a shallow encoding in which all connectives are defined by the built-in constructions of $\lambda\Pi^{\equiv}$.

This encoding is better suited for sharing proofs with other ATP because it is less sensible to small modifications of the logic. Any proof found, for example, by iProver Modulo is directly usable as an (untyped) proof in the shallow encoding.

Primitive Types

$$\mathsf{Prop} : \mathsf{Type} \qquad \mathsf{prf} : \mathsf{Prop} \to \mathsf{Type} \qquad \mathsf{type} : \mathsf{Type} \qquad \mathsf{term} : \mathsf{type} \to \mathsf{Type}$$

Primitive Connectives

$$\begin{array}{ll}
\top : \mathsf{Prop} & \bot : \mathsf{Prop} \\
\neg\text{-} : \mathsf{Prop} \to \mathsf{Prop} & \text{-} \wedge \text{-} : \mathsf{Prop} \to \mathsf{Prop} \to \mathsf{Prop} \\
\text{-} \vee \text{-} : \mathsf{Prop} \to \mathsf{Prop} \to \mathsf{Prop} & \text{-} \Rightarrow \text{-} : \mathsf{Prop} \to \mathsf{Prop} \to \mathsf{Prop} \\
\text{-} \Leftrightarrow \text{-} : \mathsf{Prop} \to \mathsf{Prop} \to \mathsf{Prop} & \forall \text{--} : \Pi\alpha : \mathsf{type}.(\mathsf{term}\ \alpha \to \mathsf{Prop}) \to \mathsf{Prop} \\
\forall_{\mathsf{type}}\text{-} : (\mathsf{type} \to \mathsf{Prop}) \to \mathsf{Prop} & \exists \text{--} : \Pi\alpha : \mathsf{type}.(\mathsf{term}\ \alpha \to \mathsf{Prop}) \to \mathsf{Prop} \\
\exists_{\mathsf{type}}\text{-} : (\mathsf{type} \to \mathsf{Prop}) \to \mathsf{Prop} & \text{-} =_{\text{-}} \text{-} : \Pi\alpha : \mathsf{type}.\mathsf{term}\ \alpha \to \mathsf{term}\ \alpha \to \mathsf{Prop}
\end{array}$$

Figure 3: Dedukti Declarations of TFF1$^{\equiv}$ Symbols

Translation Function for Types

$$|\alpha| := \alpha \qquad |T(\tau_1,\ldots,\tau_m)| := T\ |\tau_1|\ \ldots\ |\tau_m|$$

Translation Function for Terms

$$|x| := x \qquad |f(\tau_1,\ldots,\tau_m;e_1,\ldots,e_n)| := f\ |\tau_1|\ \ldots\ |\tau_m|\ |e_1|\ \ldots\ |e_n|$$

Translation Function for Formulæ

$$\begin{array}{rr}
|\top| := \top & |\bot| := \bot \\
|\neg\varphi| := \neg|\varphi| & |\varphi_1 \wedge \varphi_2| := |\varphi_1| \wedge |\varphi_2| \\
|\varphi_1 \vee \varphi_2| := |\varphi_1| \vee |\varphi_2| & |\varphi_1 \Rightarrow \varphi_2| := |\varphi_1| \Rightarrow |\varphi_2| \\
|\varphi_1 \Leftrightarrow \varphi_2| := |\varphi_1| \Leftrightarrow |\varphi_2| & |e_1 =_\tau e_2| := |e_1| =_{|\tau|} |e_2| \\
|\forall x : \tau.\ \varphi| := \forall\ |\tau|\ (\lambda x : \mathsf{term}\ |\tau|.\ |\varphi|) & |\exists x : \tau.\ \varphi| := \exists\ |\tau|\ (\lambda x : \mathsf{term}\ |\tau|.\ |\varphi|) \\
|\forall_{\mathsf{type}}\alpha : \mathsf{type}.\ \varphi| := \forall_{\mathsf{type}}\ (\lambda\alpha : \mathsf{type}\ .\ |\varphi|) & |\exists_{\mathsf{type}}\alpha : \mathsf{type}.\ \varphi| := \exists_{\mathsf{type}}\ (\lambda\alpha : \mathsf{type}\ .\ |\varphi|)
\end{array}$$

$$|P(\tau_1,\ldots,\tau_m;e_1,\ldots,e_n)| := P\ |\tau_1|\ \ldots\ |\tau_m|\ |e_1|\ \ldots\ |e_n|$$

Translation Function for Typing Contexts

$$|\emptyset| := \emptyset \qquad |\Delta, x : \tau| := |\Delta|, x : \mathsf{term}\ |\tau|$$

Translation Function for Theories

$$|\emptyset| := \Gamma_0 \text{ where } \Gamma_0 \text{ is the Dedukti context of Fig. 3}$$

$$|\mathscr{T}, T/m| := |\mathscr{T}|, T : \overbrace{\mathsf{type} \to \ldots \to \mathsf{type}}^{m \text{ times}} \to \mathsf{type}$$

$$|\mathscr{T}, f : \Pi(\alpha_1,\ldots,\alpha_m).\ (\tau_1,\ldots,\tau_n) \to \tau| := |\mathscr{T}|, f : \Pi\alpha_1 : \mathsf{type}.\ldots\Pi\alpha_m : \mathsf{type}.\ \mathsf{term}\ |\tau_1| \to \ldots \to \mathsf{term}\ |\tau_n| \to |\tau|$$

$$|\mathscr{T}, P : \Pi(\alpha_1,\ldots,\alpha_m).\ (\tau_1,\ldots,\tau_n) \to o| := |\mathscr{T}|, P : \Pi\alpha_1 : \mathsf{type}.\ldots\Pi\alpha_m : \mathsf{type}.\ \mathsf{term}\ |\tau_1| \to \ldots \to \mathsf{term}\ |\tau_n| \to \mathsf{Prop}$$

$$|\mathscr{T}, \mathrm{name} : \varphi| := |\mathscr{T}|, \mathrm{name} : \mathsf{prf}\ |\varphi|$$

$$|\mathscr{T}, l \longrightarrow_\Delta r| := |\mathscr{T}|, |l| \hookrightarrow_\Delta |r|$$

Figure 4: Translation Functions from TFF1$^{\equiv}$ to $\lambda\Pi^{\equiv}$

4 Zenon Modulo

Zenon Modulo [13] is an extension to deduction modulo [15] of the first-order tableau-based automated theorem prover Zenon [8]. It has also been improved to deal with typed formulæ and TFF1 input files. In this paper, we focus on the output format of Zenon Modulo. After finding a proof using its tableau-based proof-search algorithm [8], Zenon translates its proof tree into a *low level* format called LLproof, which

$$
\begin{array}{ll}
\mathsf{prf}\ \top & \hookrightarrow \Pi P : \mathsf{Prop}.\ \mathsf{prf}\ P \to \mathsf{prf}\ P \\
\mathsf{prf}\ \bot & \hookrightarrow \Pi P : \mathsf{Prop}.\ \mathsf{prf}\ P \\
\mathsf{prf}\ (\neg A) & \hookrightarrow \mathsf{prf}\ A \to \mathsf{prf}\ \bot \\
\mathsf{prf}\ (A \wedge B) & \hookrightarrow \Pi P : \mathsf{Prop}.\ (\mathsf{prf}\ A \to \mathsf{prf}\ B \to \mathsf{prf}\ P) \to \mathsf{prf}\ P \\
\mathsf{prf}\ (A \vee B) & \hookrightarrow \Pi P : \mathsf{Prop}.\ (\mathsf{prf}\ A \to \mathsf{prf}\ P) \to (\mathsf{prf}\ B \to \mathsf{prf}\ P) \to \mathsf{prf}\ P \\
\mathsf{prf}\ (A \Rightarrow B) & \hookrightarrow \mathsf{prf}\ A \to \mathsf{prf}\ B \\
\mathsf{prf}\ (A \Leftrightarrow B) & \hookrightarrow \mathsf{prf}\ ((A \Rightarrow B) \wedge (B \Rightarrow A)) \\
\mathsf{prf}\ (\forall\ \tau\ P) & \hookrightarrow \Pi x : \mathsf{term}\ \tau.\ \mathsf{prf}\ (P\ x) \\
\mathsf{prf}\ (\forall_{\mathsf{type}}\ P) & \hookrightarrow \Pi \alpha : \mathsf{type}.\ \mathsf{prf}\ (P\ \alpha) \\
\mathsf{prf}\ (\exists\ \tau\ P) & \hookrightarrow \Pi P : \mathsf{Prop}.\ (\Pi x : \mathsf{term}\ \tau.\ \mathsf{prf}\ (P\ x) \to \mathsf{prf}\ P) \to \mathsf{prf}\ P \\
\mathsf{prf}\ (\exists_{\mathsf{type}}\ P) & \hookrightarrow \Pi P : \mathsf{Prop}.\ (\Pi \alpha : \mathsf{type}.\ \mathsf{prf}\ (P\ \alpha) \to \mathsf{prf}\ P) \to \mathsf{prf}\ P \\
\mathsf{prf}\ (x =_\tau y) & \hookrightarrow \Pi P : (\mathsf{term}\ \tau \to \mathsf{Prop}).\ \mathsf{prf}\ (P\ x) \to \mathsf{prf}\ (P\ y)
\end{array}
$$

Figure 5: Shallow Definition of Logical Connectives in Dedukti

is a classical sequent-like proof system. This format is used for Zenon proofs before their automatic translation to Coq. LLproof is a one-sided sequent calculus with explicit contractions in every inference rule, which is close to an upside-down non-destructive tableau method.

We present in Figs. 6 and 7 the new proof system LLproof$^{\equiv}$, an adaptation of Zenon output format LLproof [8] to deduction modulo and TFF1 typing.

Normalization and deduction steps may interleave anywhere in the final proof tree. This leads to the introduction of the congruence relation $=_{\mathscr{RE}}$ inside rules of Figs. 6 and 7: if the formula P is in normal form (with respect to $\mathscr{RE}$), we denote by $[P]$ any formula congruent to P modulo $=_{\mathscr{RE}}$.

Extension of LLproof to TFF1 typing leads to the introduction of four new rules for quantification over type variables $\exists_{\mathsf{type}}$, $\neg\forall_{\mathsf{type}}$, $\forall_{\mathsf{type}}$ and $\neg\exists_{\mathsf{type}}$, and also to introduce some type information into

Closure and Quantifier-free Rules

$$\frac{}{\Gamma, [\bot] \vdash \bot}\ \bot \qquad \frac{}{\Gamma, [\neg\top] \vdash \bot}\ \neg\top \qquad \frac{}{\Gamma, [P], [\neg P] \vdash \bot}\ \mathrm{Ax} \qquad \frac{\Gamma, P \vdash \bot \qquad \Gamma, \neg P \vdash \bot}{\Gamma \vdash \bot}\ \mathrm{Cut}$$

$$\frac{}{\Gamma, [t \neq_\tau t] \vdash \bot}\ \neq \qquad \frac{}{\Gamma, [t =_\tau u], [u \neq_\tau t] \vdash \bot}\ \mathrm{Sym} \qquad \frac{\Gamma, \neg\neg P, P \vdash \bot}{\Gamma, [\neg\neg P] \vdash \bot}\ \neg\neg \qquad \frac{\Gamma, P \wedge Q, P, Q \vdash \bot}{\Gamma, [P \wedge Q] \vdash \bot}\ \wedge$$

$$\frac{\Gamma, P \vee Q, P \vdash \bot \qquad \Gamma, P \vee Q, Q \vdash \bot}{\Gamma, [P \vee Q] \vdash \bot}\ \vee \qquad \frac{\Gamma, P \Rightarrow Q, \neg P \vdash \bot \qquad \Gamma, P \Rightarrow Q, Q \vdash \bot}{\Gamma, [P \Rightarrow Q] \vdash \bot}\ \Rightarrow$$

$$\frac{\Gamma, P \Leftrightarrow Q, \neg P, \neg Q \vdash \bot \qquad \Gamma, P \Leftrightarrow Q, P, Q \vdash \bot}{\Gamma, [P \Leftrightarrow Q] \vdash \bot}\ \Leftrightarrow \qquad \frac{\Gamma, \neg(P \wedge Q), \neg P \vdash \bot \qquad \Gamma, \neg(P \wedge Q), \neg Q \vdash \bot}{\Gamma, [\neg(P \wedge Q)] \vdash \bot}\ \neg\wedge$$

$$\frac{\Gamma, \neg(P \vee Q), \neg P, \neg Q \vdash \bot}{\Gamma, [\neg(P \vee Q)] \vdash \bot}\ \neg\vee \qquad \frac{\Gamma, \neg(P \Rightarrow Q), P, \neg Q \vdash \bot}{\Gamma, [\neg(P \Rightarrow Q)] \vdash \bot}\ \neg\Rightarrow$$

$$\frac{\Gamma, \neg(P \Leftrightarrow Q), \neg P, Q \vdash \bot \qquad \Gamma, \neg(P \Leftrightarrow Q), P, \neg Q \vdash \bot}{\Gamma, [\neg(P \Leftrightarrow Q)] \vdash \bot}\ \neg\Leftrightarrow$$

Figure 6: LLproof$^{\equiv}$ Inference Rules of Zenon Modulo (part 1)

Quantifier Rules

$$\frac{\Gamma, \exists_{\mathsf{type}}\alpha : \mathsf{type}.\ P(\alpha), P(\tau) \vdash \bot}{\Gamma, [\exists_{\mathsf{type}}\alpha : \mathsf{type}.\ P(\alpha)] \vdash \bot}\ \exists_{\mathsf{type}} \qquad \frac{\Gamma, \neg\forall_{\mathsf{type}}\alpha : \mathsf{type}.\ P(\alpha), \neg P(\tau) \vdash \bot}{\Gamma, [\neg\forall_{\mathsf{type}}\alpha : \mathsf{type}.\ P(\alpha)] \vdash \bot}\ \neg\forall_{\mathsf{type}}$$

where τ is a fresh type constant

$$\frac{\Gamma, \forall_{\mathsf{type}}\alpha : \mathsf{type}.\ P(\alpha), P(\beta) \vdash \bot}{\Gamma, [\forall_{\mathsf{type}}\alpha : \mathsf{type}.\ P(\alpha)] \vdash \bot}\ \forall_{\mathsf{type}} \qquad \frac{\Gamma, \neg\exists_{\mathsf{type}}\alpha : \mathsf{type}.\ P(\alpha), \neg P(\beta) \vdash \bot}{\Gamma, [\neg\exists_{\mathsf{type}}\alpha : \mathsf{type}.\ P(\alpha)] \vdash \bot}\ \neg\exists_{\mathsf{type}}$$

where β is any closed type

$$\frac{\Gamma, \exists x : \tau.\ P(x), P(c) \vdash \bot}{\Gamma, [\exists x : \tau.\ P(x)] \vdash \bot}\ \exists \qquad \frac{\Gamma, \neg\forall x : \tau.\ P(x), \neg P(c) \vdash \bot}{\Gamma, [\neg\forall x : \tau.\ P(x)] \vdash \bot}\ \neg\forall$$

where $c : \tau$ is a fresh constant

$$\frac{\Gamma, \forall x : \tau.\ P(x), P(t) \vdash \bot}{\Gamma, [\forall x : \tau.\ P(x)] \vdash \bot}\ \forall \qquad \frac{\Gamma, \neg\exists x : \tau.\ P(x), \neg P(t) \vdash \bot}{\Gamma, [\neg\exists x : \tau.\ P(x)] \vdash \bot}\ \neg\exists$$

where $t : \tau$ is any closed term

Special Rules

$$\frac{\Delta, t_1 \neq_{\tau'_1} u_1 \vdash \bot \quad \dots \quad \Delta, t_n \neq_{\tau'_n} u_n \vdash \bot}{\Gamma, [P(\tau_1, \dots, \tau_m; t_1, \dots, t_n)], [\neg P(\tau_1, \dots, \tau_m; u_1, \dots, u_n)] \vdash \bot}\ \text{Pred}$$

where $\Delta = \Gamma \cup \{P(\tau_1, \dots, \tau_m; t_1, \dots, t_n), \neg P(\tau_1, \dots, \tau_m; u_1, \dots, u_n)\}$

$$\frac{\Delta, t_1 \neq_{\tau'_1} u_1 \vdash \bot \quad \dots \quad \Delta, t_n \neq_{\tau'_n} u_n \vdash \bot}{\Gamma, [f(\tau_1, \dots, \tau_m; t_1, \dots, t_n) \neq_\tau f(\tau_1, \dots, \tau_m; u_1, \dots, u_n)] \vdash \bot}\ \text{Fun}$$

where $\Delta = \Gamma \cup \{f(\tau_1, \dots, \tau_m; t_1, \dots, t_n) \neq_\tau f(\tau_1, \dots, \tau_m; u_1, \dots, u_n)\}$

$$\frac{\Delta, H_{11}, \dots, H_{1m} \vdash \bot \quad \dots \quad \Delta, H_{n1}, \dots, H_{nq} \vdash \bot}{\Gamma, [C_1], \dots, [C_p] \vdash \bot}\ \text{Ext}(\text{name}, \text{args}, C_1, \dots, C_p, H_{11}, \dots, H_{nq})$$

where $\Delta = \Gamma \cup \{C_1, \dots, C_p\}$

Figure 7: LLproof$^{\equiv}$ Inference Rules of Zenon Modulo (part 2)

other rules dealing with equality or quantification. For instance, equality of two closed terms t and u, both of type τ, is denoted by $t =_\tau u$. For predicate and function symbols, we first list types, then terms, separated by a semi-colon.

Finally, last difference regarding rules presented in [8] is the removal of rules "definition" and "lemma". Zenon Modulo, unlike Zenon, does not need to explicitly unfold definitions and the lemma constructions have been removed.

4.1 Translation of Zenon Modulo Proofs into $\lambda\Pi^{\equiv}$

We present in Fig. 8 a deep embedding of LLproof$^{\equiv}$ into $\lambda\Pi^{\equiv}$. We declare a constant for each inference rule, except for special rules Pred and Fun which have a dependency on the arity n of their underlying predicate and function. Fortunately, they can be expressed with the following Subst inference rule which corresponds to the substitution in a predicate P of a subterm $t : \tau'$ by another $u : \tau'$:

$$\frac{\Gamma, P(\vec{\tau}; t), t \neq_{\tau'} u \vdash \bot \qquad \Gamma, P(\vec{\tau}; t), P(\vec{\tau}; u) \vdash \bot}{\Gamma, P(\vec{\tau}; t) \vdash \bot}\ \text{Subst}$$

Zenon Modulo Rules

$\mathsf{R}_{\bot}$: $\mathsf{prf}\ \bot \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\neg\top}$: $\mathsf{prf}\ (\neg\top) \to \mathsf{prf}\ \bot$
R_{Ax} : $\Pi P : \mathsf{Prop}.\ \mathsf{prf}\ P \to \mathsf{prf}\ (\neg P) \to \mathsf{prf}\ \bot$
R_{Cut} : $\Pi P : \mathsf{Prop}.\ (\mathsf{prf}\ P \to \mathsf{prf}\ \bot) \to (\mathsf{prf}\ (\neg P) \to \mathsf{prf}\ \bot) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\neq}$: $\Pi\alpha : \mathsf{type}.\ \Pi t : \mathsf{term}\ \alpha.\ \mathsf{prf}\ (t \neq_\alpha t) \to \mathsf{prf}\ \bot$
R_{Sym} : $\Pi\alpha : \mathsf{type}.\ \Pi t,u : \mathsf{term}\ \alpha.\ \mathsf{prf}\ (t =_\alpha u) \to \mathsf{prf}\ (u \neq_\alpha t) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\neg\neg}$: $\Pi P : \mathsf{Prop}.\ (\mathsf{prf}\ P \to \mathsf{prf}\ \bot) \to \mathsf{prf}\ (\neg\neg P) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\wedge}$: $\Pi P,Q : \mathsf{Prop}.\ (\mathsf{prf}\ P \to \mathsf{prf}\ Q \to \mathsf{prf}\ \bot) \to \mathsf{prf}\ (P \wedge Q) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\vee}$: $\Pi P,Q : \mathsf{Prop}.\ (\mathsf{prf}\ P \to \mathsf{prf}\ \bot) \to (\mathsf{prf}\ Q \to \mathsf{prf}\ \bot) \to \mathsf{prf}\ (P \vee Q) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\Rightarrow}$: $\Pi P,Q : \mathsf{Prop}.\ (\mathsf{prf}\ (\neg P) \to \mathsf{prf}\ \bot) \to (\mathsf{prf}\ Q \to \mathsf{prf}\ \bot) \to \mathsf{prf}\ (P \Rightarrow Q) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\Leftrightarrow}$: $\Pi P,Q : \mathsf{Prop}.\ (\mathsf{prf}\ (\neg P) \to \mathsf{prf}\ (\neg Q) \to \mathsf{prf}\ \bot) \to (\mathsf{prf}\ P \to \mathsf{prf}\ Q \to \mathsf{prf}\ \bot) \to \mathsf{prf}\ (P \Leftrightarrow Q) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\neg\wedge}$: $\Pi P,Q : \mathsf{Prop}.\ (\mathsf{prf}\ (\neg P) \to \mathsf{prf}\ \bot) \to (\mathsf{prf}\ (\neg Q) \to \mathsf{prf}\ \bot) \to \mathsf{prf}\ (\neg(P \wedge Q)) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\neg\vee}$: $\Pi P,Q : \mathsf{Prop}.\ (\mathsf{prf}\ (\neg P) \to \mathsf{prf}\ (\neg Q) \to \mathsf{prf}\ \bot) \to \mathsf{prf}\ (\neg(P \vee Q)) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\neg\Rightarrow}$: $\Pi P,Q : \mathsf{Prop}.\ (\mathsf{prf}\ P \to \mathsf{prf}\ (\neg Q) \to \mathsf{prf}\ \bot) \to \mathsf{prf}\ (\neg(P \Rightarrow Q)) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\neg\Leftrightarrow}$: $\Pi P,Q : \mathsf{Prop}.\ (\mathsf{prf}\ (\neg P) \to \mathsf{prf}\ Q \to \mathsf{prf}\ \bot) \to (\mathsf{prf}\ P \to \mathsf{prf}\ (\neg Q) \to \mathsf{prf}\ \bot) \to \mathsf{prf}\ (\neg(P \Leftrightarrow Q)) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\exists}$: $\Pi\alpha : \mathsf{type}.\ \Pi P : (\mathsf{term}\ \alpha \to \mathsf{Prop}).\ (\Pi t : \mathsf{term}\ \alpha.\ (\mathsf{prf}\ (P\ t) \to \mathsf{prf}\ \bot)) \to \mathsf{prf}\ (\exists\ \alpha\ P) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\forall}$: $\Pi\alpha : \mathsf{type}.\ \Pi P : (\mathsf{term}\ \alpha \to \mathsf{Prop}).\ \Pi t : \mathsf{term}\ \alpha.\ (\mathsf{prf}\ (P\ t) \to \mathsf{prf}\ \bot) \to \mathsf{prf}\ (\forall\ \alpha\ P) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\neg\exists}$: $\Pi\alpha : \mathsf{type}.\ \Pi P : (\mathsf{term}\ \alpha \to \mathsf{Prop}).\ \Pi t : \mathsf{term}\ \alpha.\ (\mathsf{prf}\ (\neg(P\ t)) \to \mathsf{prf}\ \bot) \to \mathsf{prf}\ (\neg(\exists\ \alpha\ P) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\neg\forall}$: $\Pi\alpha : \mathsf{type}.\ \Pi P : (\mathsf{term}\ \alpha \to \mathsf{Prop}).\ (\Pi t : \mathsf{term}\ \alpha.\ (\mathsf{prf}\ (\neg(P\ t)) \to \mathsf{prf}\ \bot)) \to \mathsf{prf}\ (\neg\forall\ \alpha\ P) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\exists_{\mathsf{type}}}$: $\Pi P : (\mathsf{type} \to \mathsf{Prop}).\ (\Pi\alpha : \mathsf{type}.\ (\mathsf{prf}\ (P\ \alpha) \to \mathsf{prf}\ \bot)) \to \mathsf{prf}\ (\exists_{\mathsf{type}}\ P) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\forall_{\mathsf{type}}}$: $\Pi P : (\mathsf{type} \to \mathsf{Prop}).\ \Pi\alpha : \mathsf{type}.\ (\mathsf{prf}\ (P\ \alpha) \to \mathsf{prf}\ \bot) \to \mathsf{prf}\ (\forall_{\mathsf{type}}\ f) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\neg\exists_{\mathsf{type}}}$: $\Pi P : (\mathsf{type} \to \mathsf{Prop}).\ \Pi\alpha : \mathsf{type}.\ (\mathsf{prf}\ (\neg(P\ \alpha)) \to \mathsf{prf}\ \bot) \to \mathsf{prf}\ (\neg(\exists_{\mathsf{type}}\ P)) \to \mathsf{prf}\ \bot$
$\mathsf{R}_{\neg\forall_{\mathsf{type}}}$: $\Pi P : (\mathsf{type} \to \mathsf{Prop}).\ (\Pi\alpha : \mathsf{type}.\ (\mathsf{prf}\ (\neg(P\ \alpha)) \to \mathsf{prf}\ \bot)) \to \mathsf{prf}\ (\neg(\forall_{\mathsf{type}}\ P)) \to \mathsf{prf}\ \bot$
R_{Subst} : $\Pi\alpha : \mathsf{type}.\ \Pi P : (\mathsf{term}\ \alpha \to \mathsf{Prop}).\ \Pi t,u : \mathsf{term}\ \alpha.\ (\mathsf{prf}\ (t \neq_\alpha u) \to \mathsf{prf}\ \bot) \to (\mathsf{prf}\ (P\ u) \to \mathsf{prf}\ \bot) \to \mathsf{prf}\ (P\ t) \to \mathsf{prf}\ \bot$

Figure 8: $\mathsf{LLproof}^{\equiv}$ in $\lambda\Pi^{\equiv}$

The special rules Pred and Fun can be easily decomposed into n applications of the Subst rule. For instance, for a binary predicate P, from (we omit to repeat the context Γ)

$$\dfrac{\dfrac{\Pi_1}{t_1 \neq_{\tau'} u_1 \vdash \bot} \quad \dfrac{\Pi_2}{t_2 \neq_{\tau''} u_2 \vdash \bot}}{P(\vec{\tau};t_1,t_2), \neg P(\vec{\tau};u_1,u_2) \vdash \bot}\ \text{Pred}$$

we obtain

$$\dfrac{\dfrac{\Pi_1}{t_1 \neq_{\tau'} u_1 \vdash \bot} \quad \dfrac{\dfrac{\Pi_2}{t_2 \neq_{\tau''} u_2 \vdash \bot} \quad \dfrac{}{P(\vec{\tau};u_1,u_2)}\ \text{Ax}}{P(\vec{\tau};u_1,t_2)}\ \text{Subst}}{P(\vec{\tau};t_1,t_2), \neg P(\vec{\tau};u_1,u_2) \vdash \bot}\ \text{Subst}$$

In Fig. 9, we present the translation function for $\mathsf{LLproof}^{\equiv}$ sequents and proofs into $\lambda\Pi^{\equiv}$. Let us present a simple example. We want to translate this proof tree:

$$\Pi := \dfrac{\dfrac{\Pi_P}{\Gamma, P \vee Q, P \vdash \bot} \quad \dfrac{\Pi_Q}{\Gamma, P \vee Q, Q \vdash \bot}}{\Gamma, P \vee Q \vdash \bot}\ \vee(P,Q)$$

Translation Function for Sequents

$$|[\varphi_1],\ldots,[\varphi_n] \vdash \bot| := x_{\varphi_1} : \mathsf{prf}\ |\varphi_1|,\ldots,x_{\varphi_n} : \mathsf{prf}\ |\varphi_n|$$

Translation Function for Proofs

$$\left|\frac{\dfrac{\Pi_1}{\Delta, H_{11},\ldots,H_{1m} \vdash \bot} \quad \ldots \quad \dfrac{\Pi_n}{\Delta, H_{n1},\ldots,H_{nq} \vdash \bot}}{\Gamma, C_1,\ldots,C_p \vdash \bot}\ \mathrm{Rule}(\mathrm{Arg}_1,\ldots,\mathrm{Arg}_r)\right| :=$$

$$\begin{array}{l} \mathsf{R}_{\mathrm{Rule}}\ |\mathrm{Arg}_1|\ldots|\mathrm{Arg}_r| \\ \quad (\lambda x_{H_{11}} : \mathsf{prf}\ |H_{11}|.\ldots.\lambda x_{H_{1m}} : \mathsf{prf}\ |H|_{1m}.|\Pi_1|) \\ \quad \vdots \\ \quad (\lambda x_{H_{n1}} : \mathsf{prf}\ |H_{n1}|.\ldots.\lambda x_{H_{nq}} : \mathsf{prf}\ \left|H_{nq}\right|.|\Pi_n|) \\ \quad x_{C_1}\ldots x_{C_p} \end{array}$$

Figure 9: Translation Functions for LLproof$^{\equiv}$ Proofs into $\lambda\Pi^{\equiv}$

where Π_P and Π_Q are respectively proofs of sequents $\Gamma, P \vdash \bot$ and $\Gamma, Q \vdash \bot$, and where we annotate rule names with its parameters. Then, by applying the translation procedure of Figs. 4 and 9, we obtain the Dedukti term

$$\mathsf{R}_{\vee}\ |P|\ |Q|\ (\lambda x_P : \mathsf{prf}\ |P|.|\Pi_P|)\ (\lambda x_Q : \mathsf{prf}\ |Q|.|\Pi_Q|)\ x_{P\vee Q}$$

where the notation $|x|$ means the translation of x into $\lambda\Pi^{\equiv}$, and x_P is a variable declared of type $\mathsf{prf}\ |P|$. We then check that Π is a proof of the sequent $\Gamma, P \vee Q \vdash \bot$ in a TFF1$^{\equiv}$ theory $\mathscr{T}$, by checking that $|\mathscr{T}|, |\Gamma, P \vee Q| \vdash |\Pi| : \mathsf{prf}\ \bot$ in $\lambda\Pi^{\equiv}$.

More generally, for any LLproof$^{\equiv}$ proof Π and any sequent $\Gamma \vdash \bot$, we check that Π is a proof of $\Gamma \vdash \bot$ by checking the $\lambda\Pi^{\equiv}$ typing judgment $|\mathscr{T}|, |\Gamma| \vdash |\Pi| : \mathsf{prf}\ \bot$.

4.2 Shallow Embedding of LLproof$^{\equiv}$

The embedding of LLproof$^{\equiv}$ presented in Fig. 8 can also be lifted to a shallow embedding. In Fig. 13 of Appendix A, we present rewrite rules that prove all constants corresponding to LLproof$^{\equiv}$ inference rules into the logic presented in Sec. 3. This has been written in Dedukti syntax and successfully checked by Dedukti (see the file `modulogic.dk` distributed with the source code of Zenon Modulo[1]). The only remaining axiom is the law of excluded middle. This shows the soundness of LLproof$^{\equiv}$ relatively to the consistency of the logic of Sec. 3.

5 Experimental Results

Zenon Modulo helps to automatically discharge proof obligations in particular in the two projects FoCaLiZe [17] and BWare [14]. We present in this section some examples of theories, and simple related properties, that are handled successfully by Zenon Modulo, and its translation to Dedukti.

[1] https://www.rocq.inria.fr/deducteam/ZenonModulo/

Declarations

$$\begin{array}{rl}
\mathsf{bool}/0 & \\
\mathsf{false} & : \mathsf{bool} \\
\mathsf{true} & : \mathsf{bool} \\
\sim & : \mathsf{bool} \to \mathsf{bool} \\
\text{- \&\& -} & : \mathsf{bool} \to \mathsf{bool} \to \mathsf{bool} \\
\text{- || -} & : \mathsf{bool} \to \mathsf{bool} \to \mathsf{bool} \\
\mathsf{if}_ \text{ - then - else -} & : \Pi\alpha.\ \mathsf{bool} \to \alpha \to \alpha \to \alpha
\end{array}$$

Rewrite rules

$$\begin{array}{rlrl}
\mathsf{true} \mathbin{\&\&} a & \longrightarrow a & \mathsf{true} \mathbin{||} a & \longrightarrow \mathsf{true} \\
a \mathbin{\&\&} \mathsf{true} & \longrightarrow a & a \mathbin{||} \mathsf{true} & \longrightarrow \mathsf{true} \\
\mathsf{false} \mathbin{\&\&} a & \longrightarrow \mathsf{false} & \mathsf{false} \mathbin{||} a & \longrightarrow a \\
a \mathbin{\&\&} \mathsf{false} & \longrightarrow \mathsf{false} & a \mathbin{||} \mathsf{false} & \longrightarrow a \\
a \mathbin{\&\&} a & \longrightarrow a & a \mathbin{||} a & \longrightarrow a \\
a \mathbin{\&\&} (b \mathbin{\&\&} c) & \longrightarrow (a \mathbin{\&\&} b) \mathbin{\&\&} c & a \mathbin{||} (b \mathbin{||} c) & \longrightarrow (a \mathbin{||} b) \mathbin{||} c \\
 & & & \\
\sim \mathsf{true} & \longrightarrow \mathsf{false} & a \mathbin{\&\&} (b \mathbin{||} c) & \longrightarrow (a \mathbin{\&\&} b) \mathbin{||} (a \mathbin{\&\&} c) \\
\sim \mathsf{false} & \longrightarrow \mathsf{true} & (a \mathbin{||} b) \mathbin{\&\&} c & \longrightarrow (a \mathbin{\&\&} c) \mathbin{||} (b \mathbin{\&\&} c) \\
\sim (\sim a) & \longrightarrow a & & \\
\sim (a \mathbin{||} b) & \longrightarrow (\sim a) \mathbin{\&\&} (\sim b) & \mathsf{if}_\alpha \ \mathsf{true} \ \mathsf{then}\ t \ \mathsf{else}\ e & \longrightarrow t \\
\sim (a \mathbin{\&\&} b) & \longrightarrow (\sim a) \mathbin{||} (\sim b) & \mathsf{if}_\alpha \ \mathsf{false} \ \mathsf{then}\ t \ \mathsf{else}\ e & \longrightarrow e
\end{array}$$

Extension deduction rules

$$\frac{\Gamma, \neg P(\mathsf{true}) \vdash \bot \qquad \Gamma, \neg P(\mathsf{false}) \vdash \bot}{\Gamma, [\neg \forall b : \mathsf{bool}.\ P(b)] \vdash \bot}\ \text{Ext(bool-case-}\neg\forall, P)$$

$$\frac{\Gamma, P(\mathsf{true}) \vdash \bot \qquad \Gamma, P(\mathsf{false}) \vdash \bot}{\Gamma, [\exists b : \mathsf{bool}.\ P(b)] \vdash \bot}\ \text{Ext(bool-case-}\exists, P)$$

Figure 10: A $\mathsf{TFF1}^{\equiv}$ Theory of Booleans

5.1 Application to FoCaLiZe

FoCaLiZe is a framework for specifying, developing and certifying programs. The specification language is first-order logic and proofs can be discharged to Zenon or Zenon Modulo. The FoCaLiZe compiler produces both a regular program written in OCaml and a certificate written either in Coq or in Dedukti (but only the Dedukti output can be used from Zenon Modulo).

In FoCaLiZe, specifications usually rely a lot on the primitive type bool so it is important that Zenon Modulo deals with booleans efficiently. In order to prove all propositional tautologies, it is enough to add the following rules for reasoning by case on booleans (together with truth tables of connectives):

$$\frac{\Gamma, \neg P(\mathsf{true}) \vdash \bot \qquad \Gamma, \neg P(\mathsf{false}) \vdash \bot}{\Gamma, [\neg \forall b : \mathsf{bool}.\ P(b)] \vdash \bot}\ \text{Ext(bool-case-}\neg\forall, P)$$

$$\frac{\Gamma, P(\mathsf{true}) \vdash \bot \qquad \Gamma, P(\mathsf{false}) \vdash \bot}{\Gamma, [\exists b : \mathsf{bool}.\ P(b)] \vdash \bot}\ \text{Ext(bool-case-}\exists, P)$$

However, we get a much smaller proof-search space and smaller proofs by adding common algebraic laws as rewrite rules. In Fig. 10, we define a theory of booleans in $\mathsf{TFF1}^{\equiv}$. This theory handles idempotency and associativity of conjunction and disjunction but not commutativity because the rule

```
λx1 : prf(¬(∀ bool (λx : term bool. ∀ bool (λy : term bool. x && y =bool y && x)))).
 Rbool-case-¬∀
  (λx : term bool. ∀ bool (λy : term bool. x && y =bool y && x))
  (λx2 : prf(¬(∀ bool (λy : term bool. y =bool y)))).
   R¬∀ bool
    (λy : term bool. y ≠bool y)
    (λa : term bool.
     λx3 : prf(a ≠bool a).
      R≠ bool a x3)
    x2)
  (λx4 : prf(¬(∀ bool (λy : term bool. false =bool false))).
   R¬∀ bool
    (λy : term bool. false =bool false)
    (λa : term bool.
     λx5 : prf(false ≠bool false).
      R≠ bool false x5)
    x4)
  x1
```

Figure 11: Proof Certificate for Commutativity of Conjunction in Dedukti

$a \,\&\&\, b \hookrightarrow b \,\&\&\, a$ would lead to a non terminating rewrite system; therefore, commutativity is a lemma with the following proof:

$$\dfrac{\dfrac{\dfrac{}{a \neq_{\mathsf{bool}} a \vdash \bot}\neq}{\neg\forall y : \mathsf{bool}.\ y =_{\mathsf{bool}} y \vdash \bot}\neg\forall \quad \dfrac{\dfrac{}{\mathsf{false} \neq_{\mathsf{bool}} \mathsf{false} \vdash \bot}\neq}{\neg\forall y : \mathsf{bool}.\ \mathsf{false} =_{\mathsf{bool}} \mathsf{false} \vdash \bot}\neg\forall}{\neg\forall x, y : \mathsf{bool}.\ x \,\&\&\, y =_{\mathsf{bool}} y \,\&\&\, x \vdash \bot}\ \text{Ext(bool-case-}\neg\forall\text{)}$$

The translation of this proof in Dedukti is shown in Fig. 11.

5.2 Application to Set Theory

The BWare project is an industrial research project that aims to provide a framework to support the automated verification of proof obligations coming from the development of industrial applications using the B method [1]. The B method relies on a particular set theory with types. In the context of the BWare project, this typed set theory has been encoded into WhyML, the native language of Why3 [16]. To call Zenon Modulo, Why3 translates proof obligations and the B theory into TFF1 format. If it succeeds in proving the proof obligation, Zenon Modulo produces a proof certificate containing both the theory and the term, following the model presented in Fig. 12.

The BWare project provides a large benchmark made of 12,876 proof obligations coming from industrial projects. The embedding presented in this paper allowed us to verify with Dedukti all the 10,340 proof obligations that are proved by Zenon Modulo.

Let us present a small subset of this set theory, and a simple example of LLproof$^{\equiv}$ proof produced by Zenon Modulo. The theory consists of three axioms that have been turned into rewrite rules. We define constructors: a type constructor set, the membership predicate $\in$, equality on sets $=_{\mathsf{set}}$, the empty set $\emptyset$ and difference of sets $-$. For readability, we use an infix notation and let type parameters of functions and predicates in subscript. We want to prove the property

$$\forall_{\mathsf{type}}\alpha : \mathsf{type}.\ \forall s : \mathsf{set}\alpha.\ s -_\alpha s =_{\mathsf{set}\alpha} \emptyset_\alpha$$

$$
\begin{array}{rl}
\mathsf{set}\text{-} : & \mathsf{type} \to \mathsf{type} \\
\text{-} \in_{\text{-}} \text{-} : & \Pi\alpha : \mathsf{type}.\ (\mathsf{term}\ \mathsf{set}\alpha) \to (\mathsf{term}\ \mathsf{set}\alpha) \to \mathsf{Prop} \\
\text{-} =_{\mathsf{set}\text{-}} \text{-} : & \Pi\alpha : \mathsf{type}.\ (\mathsf{term}\ \mathsf{set}\alpha) \to (\mathsf{term}\ \mathsf{set}\alpha) \to \mathsf{Prop} \\
\emptyset_{\text{-}} : & \Pi\alpha : \mathsf{type}.\ (\mathsf{term}\ \mathsf{set}\alpha) \\
\text{-} -_{\text{-}} \text{-} : & \Pi\alpha : \mathsf{type}.\ (\mathsf{term}\ \mathsf{set}\alpha) \to (\mathsf{term}\ \mathsf{set}\alpha) \to (\mathsf{term}\ \mathsf{set}\alpha)
\end{array}
$$

$$
\begin{array}{rl}
s =_{\mathsf{set}\alpha} t \hookrightarrow & \forall\ (\alpha)\ (\lambda x : (\mathsf{term}\ \alpha).\ x \in_\alpha s \Leftrightarrow x \in_\alpha t) \\
x \in_\alpha \emptyset_\alpha \hookrightarrow & \bot \\
x \in_\alpha s -_\alpha t \hookrightarrow & x \in_\alpha s \wedge x \notin_\alpha t
\end{array}
$$

$$
\begin{array}{rl}
\mathit{Goal} : & \mathsf{prf}\ (\neg(\forall_{\mathsf{type}}\ (\lambda\alpha : \mathsf{type}.\ (\forall\ (\mathsf{set}\alpha)\ (\lambda s : (\mathsf{term}\ \mathsf{set}\alpha).\ s -_\alpha s =_{\mathsf{set}\alpha} \emptyset_\alpha)))) \to \mathsf{prf}\ \bot) \\
[]\ \mathit{Goal} \hookrightarrow & \lambda x_1 : \mathsf{prf}\ (\neg(\forall_{\mathsf{type}}\ (\lambda\alpha : \mathsf{type}.\ (\forall\ (\mathsf{set}\alpha)\ (\lambda s : (\mathsf{term}\ \mathsf{set}\alpha).\ s -_\alpha s =_{\mathsf{set}\alpha} \emptyset_\alpha))))). \\
& \mathsf{R}_{\neg\forall_{\mathsf{type}}}\ (\lambda\alpha : \mathsf{type}.\ (\forall\ (\mathsf{set}\alpha)\ \lambda s : (\mathsf{term}\ \mathsf{set}\alpha).\ s -_\alpha s =_{\mathsf{set}\alpha} \emptyset_\alpha)) \\
& \qquad (\lambda\tau : \mathsf{type}. \\
& \qquad\ \lambda x_2 : \mathsf{prf}\ (\neg(\forall\ (\mathsf{set}\tau)\ (\lambda s : (\mathsf{term}\ \mathsf{set}\tau).\ s -_\tau s =_{\mathsf{set}\tau} \emptyset_\tau))). \\
& \qquad\ \mathsf{R}_{\neg\forall}\ (\mathsf{set}\tau) \\
& \qquad\qquad (\lambda s : (\mathsf{term}\ \mathsf{set}\tau).\ s -_\tau s =_{\mathsf{set}\tau} \emptyset_\tau) \\
& \qquad\qquad (\lambda c_1 : (\mathsf{term}\ \mathsf{set}\tau). \\
& \qquad\qquad\ \lambda x_3 : \mathsf{prf}\ (c_1 -_\tau c_1 \neq_{\mathsf{set}\tau} \emptyset_\tau). \\
& \qquad\qquad\ \mathsf{R}_{\neg\forall}\ (\tau) \\
& \qquad\qquad\qquad (\lambda x : (\mathsf{term}\ \tau).\ (x \in_\tau c_1 -_\tau c_1) \Leftrightarrow (x \in_\tau \emptyset_\tau)) \\
& \qquad\qquad\qquad (\lambda c_2 : (\mathsf{term}\ \tau). \\
& \qquad\qquad\qquad\ \lambda x_4 : \mathsf{prf}\ (\neg((c_2 \in_\tau c_1 -_\tau c_1) \Leftrightarrow (c_2 \in_\tau \emptyset_\tau))). \\
& \qquad\qquad\qquad\ \mathsf{R}_{\neg\Leftrightarrow}\ (c_2 \in_\tau c_1 -_\tau c_1) \\
& \qquad\qquad\qquad\qquad (c_2 \in_\tau \emptyset_\tau) \\
& \qquad\qquad\qquad\qquad (\lambda x_5 : \mathsf{prf}\ (\neg(c_2 \in_\tau c_1 -_\tau c_1)). \\
& \qquad\qquad\qquad\qquad\ \lambda x_6 : \mathsf{prf}\ (c_2 \in_\tau \emptyset_\tau). \\
& \qquad\qquad\qquad\qquad\ \mathsf{R}_\bot\ x_6) \\
& \qquad\qquad\qquad\qquad (\lambda x_7 : \mathsf{prf}\ (c_2 \in_\tau c_1 -_\tau c_1). \\
& \qquad\qquad\qquad\qquad\ \lambda x_8 : \mathsf{prf}\ (\neg(c_2 \in_\tau \emptyset_\tau)). \\
& \qquad\qquad\qquad\qquad\ \mathsf{R}_\wedge\ (c_2 \in_\tau c_1) \\
& \qquad\qquad\qquad\qquad\qquad (c_2 \notin_\tau c_1) \\
& \qquad\qquad\qquad\qquad\qquad (\lambda x_9 : \mathsf{prf}\ (c_2 \in_\tau c_1). \\
& \qquad\qquad\qquad\qquad\qquad\ \lambda x_9 : \mathsf{prf}\ (c_2 \notin_\tau c_1). \\
& \qquad\qquad\qquad\qquad\qquad\ \mathsf{R}_{Ax}\ (c_2 \notin_\tau c_1) \\
& \qquad\qquad\qquad\qquad\qquad\qquad x_8 \\
& \qquad\qquad\qquad\qquad\qquad\qquad x_9) \\
& \qquad\qquad\qquad\qquad\qquad x_7) \\
& \qquad\qquad\qquad\qquad x_4) \\
& \qquad\qquad\qquad x_3) \\
& \qquad\qquad x_2) \\
& \qquad x_1)
\end{array}
$$

Figure 12: Proof Certificate for a B Set Theory Property in Dedukti

with the theory:

$$\begin{array}{rl}
\mathsf{set}\text{-}\ /1 & \\
\text{-} \in_{\text{-}} \text{-} : \Pi\alpha.\ \mathsf{set}\alpha \to \mathsf{set}\alpha \to \mathsf{Prop} & \qquad s =_{\mathsf{set}\alpha} t \longrightarrow \forall x : \alpha.\ x \in_\alpha s \Leftrightarrow x \in_\alpha t \\
\text{-} =_{\mathsf{set}\text{-}} \text{-} : \Pi\alpha.\ \mathsf{set}\alpha \to \mathsf{set}\alpha \to \mathsf{Prop} & \qquad x \in_\alpha \emptyset_\alpha \longrightarrow \bot \\
\emptyset_{\text{-}} : \Pi\alpha.\ \mathsf{set}\alpha & \qquad x \in_\alpha s -_\alpha t \longrightarrow x \in_\alpha s \wedge x \notin_\alpha t \\
\text{-} -_{\text{-}} \text{-} : \Pi\alpha.\ \mathsf{set}\alpha \to \mathsf{set}\alpha \to \mathsf{set}\alpha &
\end{array}$$

The LLproof proof tree generated by Zenon Modulo is (we omit to repeat context Γ):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(c_2 \in_\tau c_1 -_\tau c_1), c_2 \in_\tau \emptyset_\tau \vdash \bot}\,\bot \quad \dfrac{\dfrac{}{c_2 \in_\tau c_1, c_2 \notin_\tau c_1 \vdash \bot}\,\mathrm{Ax}}{c_2 \in_\tau c_1 -_\tau c_1, \neg(c_2 \in_\tau \emptyset_\tau) \vdash \bot}\,\wedge}{\neg((c_2 \in_\tau c_1 -_\tau c_1) \Leftrightarrow (c_2 \in_\tau \emptyset_\tau)) \vdash \bot}\,\neg\Leftrightarrow}{\neg(c_1 -_\tau c_1 =_{\mathsf{set}\tau} \emptyset_\tau) \vdash \bot}\,\neg\forall}{\neg(\forall s : \mathsf{set}\tau.\ s -_\tau s =_{\mathsf{set}\tau} \emptyset_\tau) \vdash \bot}\,\neg\forall}{\neg(\forall_{\mathsf{type}}\alpha : \mathsf{type}.\ \forall s : \mathsf{set}\alpha.\ s -_\alpha s =_{\mathsf{set}\alpha} \emptyset_\alpha) \vdash \bot}\,\neg\forall_{\mathsf{type}}$$

We obtain the proof certificate of Fig. 12 checkable by Dedukti, using the file `modulogic.dk`, and that is successfully checked.

6 Conclusion

We have presented a shallow embedding of Zenon Modulo proofs into Dedukti. For this encoding, we have needed to embed into $\lambda\Pi^{\equiv}$ an extension to deduction modulo of the underlying logic of the TFF1 format, denoted by TFF1$^{\equiv}$. We then defined LLproof$^{\equiv}$, the extension to TFF1$^{\equiv}$ of the proof system LLproof, which is the output format of Zenon Modulo. Finally, we have embedded LLproof$^{\equiv}$ into $\lambda\Pi^{\equiv}$ by giving the translation function for proofs. This embedding is shallow in the sense that we have reused the features of the target language and have not declared new constants for connectives and inference rules. The only axiom that we have added is the law of excluded middle.

This embedding has helped us to verify a large set of proof obligations coming from two different projects. FoCaLiZe can now benefit from deduction modulo to improve program verification when dealing with theories. In BWare, this work allowed us to certify all the proofs generated by Zenon Modulo.

Our work is closely related to the embedding of iProver Modulo proofs into Dedukti [10]. The two main differences are the assumption of the excluded middle and the extension of the logic to deal with ML-style polymorphism. Because these shallow encodings are close, we could easily share proofs of untyped formulæ with iProver Modulo.

We do not have to trust the full implementation of Zenon Modulo but only the translation of TFF1$^{\equiv}$ problems to $\lambda\Pi^{\equiv}$ discussed in Sec. 3 and, of course, Dedukti. In the case of FoCaLiZe, we go even further by using an external translator, Focalide [11]. Hence Zenon Modulo requires no confidence in that context. As future work, we want export this model. To achieve that, deduction tools must be able to read Dedukti in addition to write some. This model improves the confidence on automated deduction tools because it is no more possible to introduce inconsistency inside a proof certificate. In addition, in case of the verification of several formulæ, it should be possible to inject terms coming from different tools inside the same Dedukti file. A first experiment with Zenon Modulo and iProver Modulo in FoCaLiZe would be an interesting proof of concept.

References

[1] Jean-Raymond Abrial (1996): *The B-Book, Assigning Programs to Meanings*. Cambridge University Press, doi:10.1017/CBO9780511624162.

[2] Hendrik Pieter Barendregt, Wil Dekkers & Richard Statman (2013): *Lambda calculus with types*. Cambridge University Press, doi:10.1017/CBO9781139032636.

[3] Henk Barendregt & Erik Barendsen (2002): *Autarkic Computations in Formal Proofs*. *Journal of Automated Reasoning (JAR)* 28, doi:10.1023/A:1015761529444.

[4] Mike Barnett, Bor-Yuh Evan Chang, Robert DeLine, Bart Jacobs & K Rustan M Leino (2006): *Boogie: A modular reusable verifier for object-oriented programs*. In: *Formal Methods for Components and Objects*, Springer, doi:10.1007/11804192_17.

[5] Jasmin Christian Blanchette & Andrei Paskevich (2013): *TFF1: The TPTP Typed First-Order Form with Rank-1 Polymorphism*. In: *Conference on Automated Deduction (CADE)*, *LNCS* 7898, Springer, doi:10.1007/978-3-642-38574-2_29.

[6] Mathieu Boespflug & Guillaume Burel (2012): *CoqInE: translating the calculus of inductive constructions into the λΠ-calculus modulo*. In: *Proof Exchange for Theorem Proving (PxTP)*.

[7] Mathieu Boespflug, Quentin Carbonneaux & Olivier Hermant (2012): *The λΠ-Calculus Modulo as a Universal Proof Language*. In: *Proof Exchange for Theorem Proving (PxTP)*.

[8] Richard Bonichon, David Delahaye & Damien Doligez (2007): *Zenon: An Extensible Automated Theorem Prover Producing Checkable Proofs*. In: *Logic for Programming Artificial Intelligence and Reasoning (LPAR)*, *LNCS/LNAI* 4790, Springer, doi:10.1007/978-3-540-75560-9_13.

[9] Guillaume Burel (2011): *Experimenting with Deduction Modulo*. In: *Conference on Automated Deduction (CADE)*, *LNCS/LNAI* 6803, Springer, doi:10.1007/978-3-642-22438-6_14.

[10] Guillaume Burel (2013): *A Shallow Embedding of Resolution and Superposition Proofs into the λΠ-Calculus Modulo*. In: *International Workshop on Proof Exchange for Theorem Proving (PxTP)*.

[11] Raphaël Cauderlier: *Focalide*. `https://www.rocq.inria.fr/deducteam/Focalide`.

[12] David Delahaye, Damien Doligez, Frédéric Gilbert, Pierre Halmagrand & Olivier Hermant (2013): *Proof Certification in Zenon Modulo: When Achilles Uses Deduction Modulo to Outrun the Tortoise with Shorter Steps*. In: *International Workshop on the Implementation of Logics (IWIL)*.

[13] David Delahaye, Damien Doligez, Frédéric Gilbert, Pierre Halmagrand & Olivier Hermant (2013): *Zenon Modulo: When Achilles Outruns the Tortoise using Deduction Modulo*. In: *Logic for Programming Artificial Intelligence and Reasoning (LPAR)*, *LNCS/ARCoSS* 8312, Springer, doi:10.1007/978-3-642-45221-5_20.

[14] David Delahaye, Catherine Dubois, Claude Marché & David Mentré (2014): *The BWare Project: Building a Proof Platform for the Automated Verification of B Proof Obligations*. In: *Abstract State Machines, Alloy, B, VDM, and Z (ABZ)*, LNCS, Springer, doi:10.1007/978-3-662-43652-3_26.

[15] Gilles Dowek, Thérèse Hardin & Claude Kirchner (2003): *Theorem Proving Modulo*. *Journal of Automated Reasoning (JAR)* 31, doi:10.1023/A:1027357912519.

[16] Jean-Christophe Filliâtre & Andrei Paskevich (2013): *Why3 - Where Programs Meet Provers*. In: *European Symposium on Programming (ESOP)*, doi:10.1007/978-3-642-37036-6_8.

[17] Thérèse Hardin, François Pessaux, Pierre Weis & Damien Doligez (2009): *FoCaLiZe reference manual*.

[18] Robert Harper, Furio Honsell & Gordon Plotkin (1993): *A Framework for Defining Logics*. *Journal of the ACM* 40, doi:10.1145/138027.138060.

[19] Dale Miller (2011): *A proposal for broad spectrum proof certificates*. In: *Certified Programs and Proofs (CPP)*, Springer, doi:10.1007/978-3-642-25379-9_6.

A Appendix: Shallow Embedding of LLproof$^{\equiv}$ System into Dedukti

Law of Excluded Middle and Lemmas

$$ExMid(P : \mathsf{Prop}) : \Pi Z : \mathsf{Prop}.\ (\mathsf{prf}\ P \to \mathsf{prf}\ Z) \to (\mathsf{prf}\ (\neg P) \to \mathsf{prf}\ Z) \to \mathsf{prf}\ Z$$

$$NNPP(P : \mathsf{Prop}) : \mathsf{prf}\ (\neg\neg P) \to \mathsf{prf}\ P$$
$$:= \lambda H_1 : \mathsf{prf}\ (\neg\neg P).\ ExMid\ P\ P\ (\lambda H_2 : \mathsf{prf}\ P.\ H_2)\ (\lambda H_3 : \mathsf{prf}\ (\neg P).\ H_1\ H_3\ P)\ \square$$

$$Contr(P : \mathsf{Prop}, Q : \mathsf{Prop}) : (\mathsf{prf}\ (P \Rightarrow Q) \to \mathsf{prf}\ (\neg Q \to \neg P))$$
$$:= \lambda\ H_1 : \mathsf{prf}\ (P \Rightarrow Q).\ \lambda H_2 : \mathsf{prf}\ (\neg Q).\ \lambda H_3 : \mathsf{prf}\ P.\ H_2\ (H_1\ H_3)\ \square$$

LLproof Inference Rules

$$[]\ \mathsf{R}_{\bot} \hookrightarrow \lambda H : \mathsf{prf}\ \bot.\ H\ \square$$

$$[]\ \mathsf{R}_{\neg\top} \hookrightarrow \lambda H_1 : \mathsf{prf}\ (\neg\top).\ H_1\ (\lambda Z : \mathsf{Prop}.\ \lambda H_2 : \mathsf{prf}\ Z.\ H_2)\ \square$$

$$[P : \mathsf{Prop}]\ \mathsf{R}_{Ax}\ P \hookrightarrow \lambda H_1 : \mathsf{prf}\ P.\ \lambda H_2 : \mathsf{prf}\ (\neg P).\ H_2\ H_1\ \square$$

$$[\alpha : \mathsf{type}, t : \mathsf{term}\ \alpha]\ \mathsf{R}_{\neq}\ \alpha\ t \hookrightarrow \lambda H_1 : \mathsf{prf}\ (t \neq_\alpha t).\ H_1\ (\lambda z : (\mathsf{term}\ \alpha \to \mathsf{Prop}).\ \lambda H_2 : \mathsf{prf}\ (z\ t).\ H_2)\ \square$$

$$[\alpha : \mathsf{type}, t : \mathsf{term}\ \alpha, u : \mathsf{term}\ \alpha]\ \mathsf{R}_{Sym}\ \alpha\ t\ u \hookrightarrow \lambda H_1 : \mathsf{prf}\ (t =_\alpha u).\ \lambda H_2 : \mathsf{prf}\ (u \neq_\alpha t).\ H_2$$
$$(\lambda z : (\mathsf{term}\ \alpha \to \mathsf{Prop}).\ \lambda H_3 : \mathsf{prf}\ (z\ u).\ H_1\ (\lambda x : \mathsf{term}\ \alpha.\ (z\ x) \Rightarrow (z\ t))\ (\lambda H_4 : \mathsf{prf}\ (z\ t).\ H_4)\ H_3)\ \square$$

$$[P : \mathsf{Prop}]\ \mathsf{R}_{Cut}\ P \hookrightarrow \lambda H_1 : (\mathsf{prf}\ P \to \mathsf{prf}\ \bot).\ \lambda H_2 : (\mathsf{prf}\ (\neg P) \to \mathsf{prf}\ \bot).\ H_2\ H_1\ \square$$

$$[P : \mathsf{Prop}]\ \mathsf{R}_{\neg\neg}\ P \hookrightarrow \lambda H_1 : (\mathsf{prf}\ P \to \mathsf{prf}\ \bot).\ \lambda H_2 : \mathsf{prf}\ (\neg\neg P).\ H_2\ H_1\ \square$$

$$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\wedge}\ P\ Q \hookrightarrow \lambda H_1 : (\mathsf{prf}\ P \to \mathsf{prf}\ Q \to \mathsf{prf}\bot).\ \lambda H_2 : \mathsf{prf}\ (P \wedge Q).\ H_2\ \bot\ H_1\ \square$$

$$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\vee}\ P\ Q \hookrightarrow \lambda H_1 : (\mathsf{prf}\ P \to \mathsf{prf}\bot).\ \lambda H_2 : (\mathsf{prf}\ Q \to \mathsf{prf}\ \bot).\ \lambda H_3 : \mathsf{prf}\ (P \vee Q).\ H_3\ \bot\ H_1\ H_2\ \square$$

$$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\Rightarrow}\ P\ Q \hookrightarrow \lambda H_1 : (\mathsf{prf}\ (\neg P) \to \mathsf{prf}\ \bot).\ \lambda H_2 : (\mathsf{prf}\ Q \to \mathsf{prf}\ \bot).\ \lambda H_3 : \mathsf{prf}\ (P \Rightarrow Q).\ H_1$$
$$(Contr\ P\ Q\ H_3\ H_2)\ \square$$

$$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\Leftrightarrow}\ P\ Q \hookrightarrow \lambda H_1 : (\mathsf{prf}\ (\neg P) \to \mathsf{prf}\ (\neg Q) \to \mathsf{prf}\ \bot).\ \lambda H_2 : (\mathsf{prf}\ P \to \mathsf{prf}\ Q \to \mathsf{prf}\ \bot).$$
$$\lambda H_3 : \mathsf{prf}\ (P \Leftrightarrow Q).\ H_3\ \bot\ (\lambda H_4 : (\mathsf{prf}\ P \to \mathsf{prf}\ Q).\ \lambda H_5 : (\mathsf{prf}\ Q \to \mathsf{prf}\ P).\ (H_1\ (Contr\ P\ Q\ H_4$$
$$(\lambda H_6 : \mathsf{prf}\ Q.\ (H_2\ (H_5\ H_6))\ H_6)))\ (\lambda H_7 : \mathsf{prf}\ Q.\ (H_2\ (H_5\ H_7))\ H_7))\ \square$$

$$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\neg\wedge}\ P\ Q \hookrightarrow \lambda H_1 : (\mathsf{prf}\ (\neg P) \to \mathsf{prf}\ \bot).\ \lambda H_2 : (\mathsf{prf}\ (\neg Q) \to \mathsf{prf}\ \bot).\ \lambda H_3 : \mathsf{prf}\ (\neg(P \wedge Q)).$$
$$H_1\ (\lambda H_5 : \mathsf{prf}\ P.\ H_2\ (\lambda H_6 : \mathsf{prf}\ Q.\ H_3\ (\lambda Z : \mathsf{Prop}.\ \lambda H_4 : (\mathsf{prf}\ P \to \mathsf{prf}\ Q \to \mathsf{prf}\ Z).\ H_4\ H_5\ H_6)))\ \square$$

$$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\neg\vee}\ P\ Q \hookrightarrow \lambda H_1 : (\mathsf{prf}\ (\neg P)\mathsf{prf}\ (\neg Q) \to \mathsf{prf}\ \bot).\ \lambda H_2 : \mathsf{prf}\ (\neg(P \vee Q)).\ H_1\ (Contr\ P\ (P \vee Q)$$
$$(\lambda H_3 : \mathsf{prf}\ P.\ \lambda Z : \mathsf{Prop}.\ \lambda H_4 : (\mathsf{prf}\ P \to \mathsf{prf}\ Z).\ \lambda H_5 : (\mathsf{prf}\ P \to \mathsf{prf}\ Z).\ H_4\ H_3)\ H_2)\ (Contr\ Q\ (P \vee Q)$$
$$(\lambda H_6 : \mathsf{prf}\ Q.\ \lambda Z : \mathsf{Prop}.\ \lambda H_7 : (\mathsf{prf}\ P \to \mathsf{prf}\ Z).\ \lambda H_8 : (\mathsf{prf}\ Q \to \mathsf{prf}\ Z).\ H_8\ H_6)\ H_2)\ \square$$

$$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\neg\Rightarrow}\ P\ Q \hookrightarrow \lambda H_1 : (\mathsf{prf}\ P \to \mathsf{prf}\ (\neg Q) \to \mathsf{prf}\ \bot).\ \lambda H_2 : \mathsf{prf}\ (\neg(P \Rightarrow Q)).\ H_2\ (\lambda H_3 : \mathsf{prf}\ P.$$
$$(H_1\ H_3)\ (\lambda H_4 : \mathsf{prf}\ Q.\ H_2\ (\lambda H_5 : \mathsf{prf}\ P.\ H_4))\ Q)\ \square$$

$$[P : \mathsf{Prop}, Q : \mathsf{Prop}]\ \mathsf{R}_{\neg\Leftrightarrow}\ P\ Q \hookrightarrow \lambda H_1 : (\mathsf{prf}\ (\neg P) \to \mathsf{prf}\ (\neg Q)).\ \lambda H_2 : (\mathsf{prf}\ P \to \mathsf{prf}\ (\neg\neg Q)).$$
$$\lambda H_3 : \mathsf{prf}\ (\neg(P \Leftrightarrow Q)).\ (\lambda H_4 : \mathsf{prf}\ (\neg P).\ H_3\ (\lambda Z : \mathsf{Prop}.\ \lambda H_5 : (\mathsf{prf}\ (P \Rightarrow Q) \to \mathsf{prf}\ (Q \Rightarrow P) \to \mathsf{prf}\ Z).$$
$$H_5\ (\lambda H_6 : \mathsf{prf}\ P.\ H_4\ H_6\ Q)\ (\lambda H_7 : \mathsf{prf}\ Q.\ H_1\ H_4\ H_7\ P)))\ (\lambda H_8 : \mathsf{prf}\ P.\ H_2\ H_8\ (\lambda H_9 : \mathsf{prf}\ Q.\ H_3\ (\lambda Z : \mathsf{Prop}.$$
$$\lambda H_{10} : (\mathsf{prf}\ (P \Rightarrow Q) \to \mathsf{prf}\ (Q \Rightarrow P) \to \mathsf{prf}\ Z).\ H_{10}\ (\lambda H_{11} : \mathsf{prf}\ P.\ H_9)\ (\lambda H_{12} : \mathsf{prf}\ Q.\ H_8))))\ \square$$

$$[\alpha : \mathsf{type}, P : \mathsf{term}\ \alpha \to \mathsf{Prop}]\ \mathsf{R}_{\exists}\ \alpha\ P \hookrightarrow \lambda H_1 : (t : \mathsf{term}\ \alpha \to \mathsf{prf}\ (P\ t) \to \mathsf{prf}\ \bot).\ \lambda H_2 : \mathsf{prf}\ (\exists\ \alpha\ P).\ H_2\ \bot\ H_1\ \square$$

$$[\alpha : \mathsf{type}, P : \mathsf{term}\ \alpha \to \mathsf{Prop}, t : \mathsf{term}\ \alpha]\ \mathsf{R}_{\forall}\ \alpha\ P\ t \hookrightarrow \lambda H_1 : (\mathsf{prf}\ (P\ t) \to \mathsf{prf}\ \bot).\ \lambda H_2 : \mathsf{prf}\ (\forall\ \alpha\ P).\ H_1\ (H_2\ t)\ \square$$

$$[\alpha : \mathsf{type}, P : \mathsf{term}\ \alpha \to \mathsf{Prop}, t : \mathsf{term}\ \alpha]\ \mathsf{R}_{\neg\exists}\ \alpha\ P\ t \hookrightarrow \lambda H_1 : (\mathsf{prf}\ (\neg(P\ t)) \to \mathsf{prf}\ \bot).\ \lambda H_2 : \mathsf{prf}\ (\neg(\exists\ \alpha\ P)).\ H_1$$
$$(\lambda H_4 : \mathsf{prf}\ (P\ t).\ H_2\ (\lambda Z : \mathsf{Prop}.\ \lambda H_3 : (x : \mathsf{term}\ \alpha \to \mathsf{prf}\ (P\ x) \to \mathsf{prf}\ Z).\ H_3\ t\ H_4))\ \square$$

$$[\alpha : \mathsf{type}, P : \mathsf{term}\ \alpha \to \mathsf{Prop}]\ \mathsf{R}_{\neg\forall}\ \alpha\ P \hookrightarrow \lambda H_1 : (t : \mathsf{term}\ \alpha \to \mathsf{prf}\ (\neg(P\ t)) \to \mathsf{prf}\ \bot).\ \lambda H_2 : \mathsf{prf}\ (\neg(\forall\ \alpha\ P)).$$
$$H_2\ (\lambda t : \mathsf{term}\ \alpha.\ NNPP\ (P\ t)\ (H_1\ t))\ \square$$

$$[P : \mathsf{type} \to \mathsf{Prop}]\ \mathsf{R}_{\exists_{\mathsf{type}}}\ P \hookrightarrow \lambda H_1 : (\alpha : \mathsf{type} \to \mathsf{prf}\ (P\ \alpha) \to \mathsf{prf}\ \bot).\ \lambda H_2 : \mathsf{prf}\ (\exists_{\mathsf{type}}\ P).\ H_2\ \bot\ H_1\ \square$$

$$[P : \mathsf{type} \to \mathsf{Prop}, \alpha : \mathsf{type}]\ \mathsf{R}_{\forall_{\mathsf{type}}}\ P\ \alpha \hookrightarrow \lambda H_1 : (\mathsf{prf}\ (P\ \alpha) \to \mathsf{prf}\ \bot).\ \lambda H_2 : \mathsf{prf}\ (\forall_{\mathsf{type}}\ P).\ H_1\ (H_2\ \alpha)\ \square$$

$$[P : \mathsf{type} \to \mathsf{Prop}, \alpha : \mathsf{type}]\ \mathsf{R}_{\neg\exists_{\mathsf{type}}}\ P\ \alpha \hookrightarrow \lambda H_1 : (\mathsf{prf}\ (\neg(P\ \alpha)) \to \mathsf{prf}\ \bot).\ \lambda H_2 : \mathsf{prf}\ (\neg(\exists_{\mathsf{type}}\ P)).\ H_1$$
$$(\lambda H_4 : \mathsf{prf}\ (P\ \alpha).\ H_2\ \lambda Z : \mathsf{Prop}.\ \lambda H_3 : (\beta : \mathsf{type} \to \mathsf{prf}\ (P\ \beta) \to \mathsf{prf}\ Z).\ H_3\ \alpha\ H_4)\ \square$$

$$[P : \mathsf{type} \to \mathsf{Prop}]\ \mathsf{R}_{\neg\forall_{\mathsf{type}}}\ P \hookrightarrow \lambda H_1 : (\alpha : \mathsf{type} \to \mathsf{prf}\ (\neg(P\ \alpha)) \to \mathsf{prf}\ \bot).\ \lambda H_2 : \mathsf{prf}\ (\neg(\forall_{\mathsf{type}}\ P)).\ H_2$$
$$(\lambda\alpha : \mathsf{type}.\ NNPP\ (P\ \alpha)\ (H_1\ \alpha))\ \square$$

$$[\alpha : \mathsf{type}, P : \mathsf{term}\ \alpha \to \mathsf{Prop}, t_1 : \mathsf{term}\ \alpha, t_2 : \mathsf{term}\ \alpha]\ \mathsf{R}_{Subst}\ \alpha\ P\ t_1\ t_2 \hookrightarrow \lambda H_1 : (\mathsf{prf}\ (t_1 \neq_\alpha t_2) \to \mathsf{prf}\ \bot).$$
$$\lambda H_2 : (\mathsf{prf}\ (P\ t_2) \to \mathsf{prf}\ \bot).\ \lambda H_3 : \mathsf{prf}\ (P\ t_1).\ H_1\ (\lambda H_4 : \mathsf{prf}\ (t_1 \neq_\alpha t_2).\ H_2\ (H_4\ P\ H_3))\ \square$$

Figure 13: Shallow Embedding of LLproof into Dedukti

The deep embedding of LLproof$^{\equiv}$ presented in Sec. 4.1 is well-typed with respect to the deep em-

bedding of typed deduction modulo presented in Sec. 3.1. Using the shallow embedding presented in Sec. 3.2, we can prove all the rules declared in Fig. 8 by rewriting the $\mathsf{R}_{\mathsf{rule}}$ symbols using only one axiom: the law of excluded middle. These proofs are listed in Fig. 13 where the $\square$ symbol is used to delimit proofs.