Recovering Intuition from Automated Formal Proofs using Tableaux with Superdeduction

David Delahaye
David.Delahaye@cnam.fr
Cedric/Cnam, Paris, France

Mélanie Jacquel
Melanie.Jacquel@siemens.com
Siemens IC-MOL, Châtillon, France

Abstract

We propose an automated deduction method which allows us to produce proofs close to the human intuition and practice. This method is based on tableaux, which generate more natural proofs than similar methods relying on clausal forms, and uses the principles of superdeduction, among which the theory is used to enrich the deduction system with new deduction rules. We present two implementations of this method, which consist of extensions of the Zenon automated theorem prover. The first implementation is a version dedicated to the set theory of the B formal method, while the second implementation is a generic version able to deal with any first order theory. We also provide several examples of problems, which can be handled by these tools and which come from different theories, such as the B set theory or theories of the TPTP library.

1 Introduction

These days, theorem proving appears as an appropriate support for education in subjects such as mathematics and more generally logic, where proofs play a significant role. This can be explained by the fact that some of the existing theorem prover based systems have a long history of development, and constantly provide technical innovations not only in terms of design, but also in terms of theory. Among these theorem prover based systems, interactive theorem provers, such as Coq [20] for example, appear to be quite appropriate tools, since they offer special environments dedicated to proving. In particular, these special environments offer syntax and type checking, as well as a bounded set of tactics, i.e. commands building proofs when applied to proof goals. These environments also provide some assistance in the way of building proofs, since tactics are able to automatically and incrementally produce proofs when applied to goals. This assistance does not only concern the application of tactics, but may also be related to other aspects regarding modeling, such as the automated generation of induction schemes from inductive types for instance. However, these mechanized frameworks do not offer any guidance in the way of finding the right proof of a theorem, and if the user does not have the intuition of this proof (which may be acquired by thinking of the proof on paper), it is likely that he/she would experience some difficulties in building the corresponding proof, even with an interactive proof loop (even worse, he/she would probably get lost by unnecessarily applying some tactics in an endless way, like induction tactics for example). To deal with this problem of finding proofs,

we may consider the use of automated theorem provers as long as they at least provide proof traces which are comprehensible enough to recover the intuition of the corresponding proofs.

Automated theorem proving is a quite wide and still very active domain of research. In automated theorem proving, we generally distinguish the semantic methods from the syntactic methods. The semantic methods, such as the Davis-Putnam algorithm [9] or the Binary Decision Diagrams [5] (BDDs) for instance, have the advantage to be quite intuitive, but are limited to propositional calculus. To deal with first order logic, we preferably rely on syntactic methods, which may be split into two large families of methods. The first family of methods is the saturation-based theorem proving, which was actually introduced by Robinson with the resolution calculus [18]. Resolution is a complete method working by refutation: a contradiction (i.e. the empty clause) has to be deduced from an unsatisfiable set of clauses. The search for a contradiction proceeds by saturating the given set of clauses, that is, systematically (and exhaustively) applying all applicable inference rules. The principle of resolution is general enough to allow many calculi to be seen as resolution-based calculi (binary resolution, positive resolution, semantic resolution, hyper-resolution, the inverse method, etc). However, a proof produced by resolution is not appropriate to get the intuition of the proof, since resolution actually works on a formula in clausal form (a preliminary step therefore consists in clausifying the initial formula), and there is little chance to understand the proof of the initial formula from the resolution proof (unless the initial formula is already in clausal form). The second family of syntactic methods tend to palliate this difficulty and are called tableau-based methods. Tableaux are actually much older than resolution-based methods and were introduced by pioneers Hintikka [12] and Beth [2] from the cut-free version of Gentzen's sequent calculus [11]. The tableau method still works by refutation but over the initial formula contrary to resolution, and by case distinction. More precisely, it allows us to systematically generate subcases until elementary contradictions are reached, building a kind of tree from which it is possible to almost directly produce a proof. Compared to resolution, tableaux therefore offer the possibility to build comprehensible proofs which are directly related to the corresponding initial formulas.

If tableaux allow us to produce more comprehensible proofs, some recent deduction techniques have been developed and tend to improve the presentation of proofs in the usual deductive systems, in particular when reasoning modulo a theory. Among these new deduction techniques, there are, for example, deduction modulo [10] and superdeduction [4], which respectively focus on the computational and deductive parts of a theory, and which can be considered as steps toward high-level deductive languages. If deduction modulo and superdeduction are equivalent when reasoning modulo a theory, superdeduction appears to be more appropriate to produce proofs close to the human intuition as it allows us to naturally encode custom deduction schemes. In addition, the principle of superdeduction relies on the generation of specific deduction rules (called superdeduction rules) from the axioms of the theory, and in practice, it is quite easier to extend existing tools with ad hoc deduction rules than with a congruence over the formulas (coming from the computational rules of deduction modulo).

In this paper, we propose an automated deduction method based both on tableaux and superdeduction. As said previously, the main motivation is to build a system able to provide a significant help in matter of education by automatically producing proofs comprehensible enough to recover the intuition of these proofs. To show that such a system is actually effective in practice, we also propose to implement this system by realizing an extension of an existing automated theorem prover called Zenon [3], and which relies on classical first order logic with equality and applies the tableau method

as proof search. In this context, the choice of Zenon is strongly influenced by its ability of producing comprehensible proof traces (with several levels of details). In addition, Zenon offers an extension mechanism, which allows us to extend its core of deductive rules to match specific requirements, and which is therefore quite appropriate to integrate superdeduction. Two extensions of Zenon with superdeduction have been implemented and will be considered in this paper. The first implementation is dedicated to the set theory of the B method [1] (or B for short), which is a formal method and allows engineers to develop software with high guarantees of confidence. This implementation is used by Siemens IC-MOL to automatically verify B proof rules coming from a database which is built adding rules from their several projects and applications, such as driverless metro systems for instance (see [13, 14] for more details). The second implementation is generic and works over any first order theory, which allows us to use it to prove problems from the TPTP library [19] (which is a library of test problems for automated theorem proving systems).

The paper is organized as follows: in Section 2, we first introduce the principles of superdeduction; we then present, in Section 3, the computation of superdeduction rules from axioms in the framework of the tableau method used by Zenon; finally, in Sections 4 and 5, we respectively describe the implementation of our extensions of Zenon for the B set theory and for any first order theory, and also provide some examples respectively coming from the database of B proof rules maintained by Siemens IC-MOL and the TPTP library.

2 Principles of Superdeduction

In this section, we present the principles of superdeduction, which is a variant of deduction modulo, and which allows us to describe proofs in a more compact format in particular. In addition, we show that proofs in superdeduction are not only shorter, but also follow a more natural human reasoning scheme, and that custom deduction schemes, such as structural induction over Peano natural numbers for example, can be naturally encoded using superdeduction.

2.1 Deduction Modulo and Superdeduction

Deduction modulo [10] 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. Superdeduction [4] is a variant of deduction modulo, where axioms are used to enrich the deduction system with new deduction rules, which are called superdeduction rules. For instance, considering the inclusion in set theory $\forall a, b\ (a \subseteq b \Leftrightarrow \forall x\ (x \in a \Rightarrow x \in b))$, the proof of $A \subseteq A$ in sequent calculus has the following form:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\ldots, x \in A \vdash A \subseteq A, x \in A}\,\mathrm{Ax}}{\ldots \vdash A \subseteq A, x \in A \Rightarrow x \in A}\,\Rightarrow\!\mathrm{R}}{\ldots \vdash A \subseteq A, \forall x\ (x \in A \Rightarrow x \in A)}\,\forall\mathrm{R} \qquad \dfrac{}{\ldots, A \subseteq A \vdash A \subseteq A}\,\mathrm{Ax}}{\ldots, \forall x\ (x \in A \Rightarrow x \in A) \Rightarrow A \subseteq A \vdash A \subseteq A}\,\Rightarrow\!\mathrm{L}}{A \subseteq A \Leftrightarrow \forall x\ (x \in A \Rightarrow x \in A) \vdash A \subseteq A}\,\wedge\mathrm{L}}{\forall a, b\ (a \subseteq b \Leftrightarrow \forall x\ (x \in a \Rightarrow x \in b)) \vdash A \subseteq A}\,\forall\mathrm{L} \times 2$$

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

$$\dfrac{\dfrac{\dfrac{}{x \in A \vdash x \in A}\,\mathrm{Ax}}{\vdash x \in A \Rightarrow x \in A}\,\Rightarrow\!\mathrm{R}}{\vdash A \subseteq A}\,\forall\mathrm{R},\ A \subseteq A \rightarrow \forall x\ (x \in A \Rightarrow x \in A)$$

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 us for unbounded proof size speed-up [6].

Superdeduction proposes to go further than deduction modulo precisely when the considered axiom defines a predicate P with an equivalence $\forall \bar{x}\ (P \Leftrightarrow \varphi)$. While deduction modulo replaces the axiom by a rewrite rule, superdeduction adds to this transformation the decomposition of the connectives occurring in this definition. This corresponds to an extension of Prawitz's folding and unfolding rules [17] (called introduction and elimination rules by Prawitz), where the connectives of the definition are introduced and eliminated. The proposed (right) superdeduction rule is then the following (there is also a corresponding left rule):

$$\dfrac{\Gamma, x \in a \vdash x \in b, \Delta}{\Gamma \vdash a \subseteq b, \Delta}\,\mathrm{IncR},\ x \notin \Gamma, \Delta$$

Hence, proving $A \subseteq A$ with this new rule can be performed as follows:

$$\dfrac{\dfrac{}{x \in A \vdash x \in A}\,\mathrm{Ax}}{\vdash A \subseteq A}\,\mathrm{IncR}$$

This new proof is not only simpler and shorter than in deduction modulo, but also follows a natural human reasoning scheme usually used in mathematics as shown more precisely in the next subsection.

2.2 Human Reasoning with Superdeduction

Considering the previous example of inclusion in set theory, we can notice that the superdeduction rule is more natural and intuitive than a simple folding rule à la Prawitz. Given two sets A and B, if we aim to prove $A \subseteq B$, it seems a little unusual to propose to prove $\forall x\ (x \in A \Rightarrow x \in B)$, instead we propose to prove $x \in B$ given x s.t. $x \in A$, which amounts to implicitly introducing the connectives of the unfolded proposition. This implicit introduction of connectives is precisely proposed by the previous superdeduction rule, which can be read as "if any element of a is an element of b, then $a \subseteq b$".

Similarly, superdeduction can also be used to naturally encode custom deduction schemes. For example, let us consider the structural induction scheme over Peano natural numbers (i.e. non-negative integers). This scheme can be defined as follows (i.e. the natural numbers are seen as the set of terms verifying the inductive predicate):

$$\forall n\ (n \in \mathbb{N} \Leftrightarrow \forall P\ (0 \in P \Rightarrow \forall m\ (m \in P \Rightarrow S(m) \in P) \Rightarrow n \in P))$$

In sequent calculus, this scheme can be encoded by the two following (right) superdeduction rules (there are also two corresponding left rules):

$$\frac{\Gamma, 0 \in P, H(P) \vdash n \in P, \Delta}{\Gamma \vdash n \in \mathbb{N}, \Delta}\ \mathrm{IndR}, P \notin \Gamma, \Delta$$

$$\frac{\Gamma, m \in P \vdash S(m) \in P, \Delta}{\Gamma \vdash H(P), \Delta}\ \mathrm{HeredR}, m \notin \Gamma, \Delta$$

Let us notice that the induction scheme actually requires two superdeduction rules, whose one of the rules (the rule HeredR) focuses on the heredity part of the scheme in particular. This focus is motivated by the need of avoiding permutability problems (between Skolemization and instantiation), which may occur when computing superdeduction rules. These permutability problems are quite common in automated proof search, and appear here since superdeduction systems are in fact embedding a part of compiled automated deduction. In [4] and in order to deal with these permutability problems, the authors use a method inspired by focusing techniques in the framework of sequent calculus. It is worth noting that these permutability problems are managed in different ways by automated deduction methods, and in particular, we will therefore not have to use focusing techniques when integrating superdeduction to the tableau method in Section 3.

3 Tableaux with Superdeduction

In this section, we present the tableau method used by the Zenon automated theorem prover, which deals with classical first order logic with equality and a specific support for equivalence relations. Once the rules of this method have been described, we show how it is possible to compute superdeduction rules from axiomatic theories, and how these new rules extend the kernel of rules of Zenon.

3.1 The Tableau Method

The proof search rules of Zenon are described in detail in [3] and summarized in Figure 1 (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 $P(x)$, 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 the equality in particular). 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; they are not used as variables as they are never substituted. The proof search rules are applied with the normal 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 an application of 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 work on another branch. Moreover, we work in a non-destructive way: working on one branch will never change the formulas of another branch.

3.2 From Axioms to Superdeduction Rules

As mentioned in Section 2, reasoning modulo a theory in a tableau method using superdeduction requires to generate new deduction rules from some axioms of the theory. The axioms which can be

Closure and Cut Rules

$$\frac{\bot}{\odot}\,\odot_{\bot} \qquad \frac{\neg\top}{\odot}\,\odot_{\neg\top} \qquad \frac{}{P \mid \neg P}\,\text{cut}$$

$$\frac{\neg R_r(t,t)}{\odot}\,\odot_r \qquad \frac{P \qquad \neg P}{\odot}\,\odot \qquad \frac{R_s(a,b) \qquad \neg R_s(b,a)}{\odot}\,\odot_s$$

Analytic Rules

$$\frac{\neg\neg P}{P}\,\alpha_{\neg\neg} \qquad \frac{P \Leftrightarrow Q}{\neg P, \neg Q \mid P, Q}\,\beta_{\Leftrightarrow} \qquad \frac{\neg(P \Leftrightarrow Q)}{\neg P, Q \mid P, \neg Q}\,\beta_{\neg\Leftrightarrow}$$

$$\frac{P \wedge Q}{P, Q}\,\alpha_{\wedge} \qquad \frac{\neg(P \vee Q)}{\neg P, \neg Q}\,\alpha_{\neg\vee} \qquad \frac{\neg(P \Rightarrow Q)}{P, \neg Q}\,\alpha_{\neg\Rightarrow}$$

$$\frac{P \vee Q}{P \mid Q}\,\beta_{\vee} \qquad \frac{\neg(P \wedge Q)}{\neg P \mid \neg Q}\,\beta_{\neg\wedge} \qquad \frac{P \Rightarrow Q}{\neg P \mid Q}\,\beta_{\Rightarrow}$$

$$\frac{\exists x\ P(x)}{P(\epsilon(x).P(x))}\,\delta_{\exists} \qquad \frac{\neg\forall x\ P(x)}{\neg P(\epsilon(x).\neg P(x))}\,\delta_{\neg\forall}$$

γ-Rules

$$\frac{\forall x\ P(x)}{P(X)}\,\gamma_{\forall M} \qquad \frac{\neg\exists x\ P(x)}{\neg P(X)}\,\gamma_{\neg\exists M} \qquad \frac{\forall x\ P(x)}{P(t)}\,\gamma_{\forall\text{inst}} \qquad \frac{\neg\exists x\ P(x)}{\neg P(t)}\,\gamma_{\neg\exists\text{inst}}$$

Relational Rules

$$\frac{P(t_1,\ldots,t_n) \qquad \neg P(s_1,\ldots,s_n)}{t_1 \neq s_1 \mid \ldots \mid t_n \neq s_n}\,\text{pred} \qquad \frac{f(t_1,\ldots,t_n) \neq f(s_1,\ldots,s_n)}{t_1 \neq s_1 \mid \ldots \mid t_n \neq s_n}\,\text{fun}$$

$$\frac{R_s(s,t) \qquad \neg R_s(u,v)}{t \neq u \mid s \neq v}\,\text{sym} \qquad \frac{\neg R_r(s,t)}{s \neq t}\,\neg_{\text{refl}}$$

$$\frac{R_t(s,t) \qquad \neg R_t(u,v)}{u \neq s, \neg R_t(u,s) \mid t \neq v, \neg R_t(t,v)}\,\text{trans}$$

$$\frac{R_{ts}(s,t) \qquad \neg R_{ts}(u,v)}{v \neq s, \neg R_{ts}(v,s) \mid t \neq u, \neg R_{ts}(t,u)}\,\text{transsym}$$

$$\frac{s = t \qquad \neg R_t(u,v)}{u \neq s, \neg R_t(u,s) \mid \neg R_t(u,s), \neg R_t(t,v) \mid t \neq v, \neg R_t(t,v)}\,\text{transeq}$$

$$\frac{s = t \qquad \neg R_{ts}(u,v)}{v \neq s, \neg R_{ts}(v,s) \mid \neg R_{ts}(v,s), \neg R_{ts}(t,u) \mid t \neq u, \neg R_{ts}(t,u)}\,\text{transeqsym}$$

Figure 1: Proof Search Rules of Zenon

considered for superdeduction are of the form $\forall \bar{x}\ (P \Leftrightarrow \varphi)$, where P is atomic. This specific form of axiom allows us to introduce an orientation of the axiom from P to φ, and we introduce the notion of proposition rewrite rule (this notion appears in [4], from which we borrow the following definition and notation):

Definition 1 (Proposition Rewrite Rule) *The notation $R : P \to \varphi$ denotes the axiom $\forall \bar{x}\ (P \Leftrightarrow \varphi)$, where R is the name of the rule, P an atomic proposition, φ a proposition, and $\bar{x}$ the free variables of P and φ.*

It should be noted that P may contain first order terms and therefore that such an axiom is not just a definition. For instance, $x \in \{\ y \mid y \in a \wedge y \in b\ \} \to x \in a \wedge x \in b$ (where the comprehension set is a first order term) is a proposition rewrite rule.

Let us now describe how the computation of superdeduction rules for Zenon is performed from a given proposition rewrite rule.

Definition 2 (Computation of Superdeduction Rules) *Let $\mathcal{S}$ be a set of rules composed by the subset of the proof search rules of Zenon formed of the closure rules, the analytic rules, as well as the $\gamma_{\forall M}$ and $\gamma_{\neg\exists M}$ rules. Given a proposition rewrite rule $R : P \to \varphi$, two superdeduction rules (a positive one R and a negative one $\neg R$) are generated in the following way:*

1. *To get the positive rule R, initialize the procedure with the formula φ. Next, apply the rules of $\mathcal{S}$ until there is no open leaf anymore on which they can be applied. Then, collect the premises and the conclusion, and replace φ by P to obtain the positive rule R.*
2. *To get the negative rule $\neg R$, initialize the procedure with the formula $\neg\varphi$. Next, apply the rules of $\mathcal{S}$ until there is no open leaf anymore on which they can be applied. Then, collect the premises and the conclusion, and replace $\neg\varphi$ by $\neg P$ to obtain the negative rule $\neg R$.*

If the rule R (resp. $\neg R$) involves metavariables, an instantiation rule R_{inst} (resp. $\neg R_{\text{inst}}$) is added, where one or several metavariables can be instantiated.

Integrating these new deduction rules to the proof search rules of Zenon is sound as they are generated from a subset of rules of Zenon, while cut-free completeness cannot be preserved in general (i.e. for any theory). In practice, soundness can be ensured by the ability of Zenon of producing proofs for some proof assistants, such as Coq and Isabelle, which can be used as proof checkers.

Let us illustrate the computation of superdeduction rules from a proposition rewrite rule with the example of the set inclusion.

Example 3 (Set Inclusion) *From the definition of the set inclusion, we introduce the proposition rewrite rule* $\text{Inc} : a \subseteq b \to \forall x\ (x \in a \Rightarrow x \in b)$*, and the corresponding superdeduction rules* Inc *and* $\neg\text{Inc}$ *are generated as follows:*

$$\dfrac{\dfrac{\forall x\ (x \in a \Rightarrow x \in b)}{X \in a \Rightarrow X \in b}\,\gamma_{\forall M}}{X \notin a \mid X \in b}\,\beta_{\Rightarrow} \qquad \dfrac{\dfrac{\neg\forall x\ (x \in a \Rightarrow x \in b)}{\neg(\epsilon_x \in a \Rightarrow \epsilon_x \in b)}\,\delta_{\neg\forall}}{\epsilon_x \in a, \epsilon_x \notin b}\,\alpha_{\neg\Rightarrow}$$

where $\epsilon_x = \epsilon(x).\neg(x \in a \Rightarrow x \in b)$.

The resulting superdeduction rules are then the following:

$$\dfrac{a \subseteq b}{X \notin a \mid X \in b}\,\text{Inc} \qquad \dfrac{a \subseteq b}{t \notin a \mid t \in b}\,\text{Inc}_{\text{inst}} \qquad \dfrac{a \not\subseteq b}{\epsilon_x \in a, \epsilon_x \notin b}\,\neg\text{Inc}$$

4 An Implementation for the B Set Theory

In this section, we describe our first extension of Zenon with superdeduction in the case of the B set theory, which is the underlying theory of the B formal method, and where superdeduction rules are hard-coded from the initial axiomatic theory. We also propose an example of B proof rule coming from the database maintained by Siemens IC-MOL, and that can be verified by this tool producing a quite comprehensible proof.

4.1 Superdeduction Rules for the B Set Theory

This extension of Zenon for the B set theory is actually motivated by an experiment which is managed by Siemens IC-MOL regarding the verification of B proof rules [13, 14]. The B method [1], or B for short, allows engineers to develop software with high guarantees of confidence; more precisely it allows them to build correct by design software. B is a formal method based on set theory and theorem proving, and which relies on a refinement-based development process. The Atelier B environment [7] is a platform that supports B and offers, among other tools, both automated and interactive provers. In practice, to ensure the global correctness of formalized applications, the user must discharge proof obligations. These proof obligations may be proved automatically, but otherwise, they have to be handled manually either by using the interactive prover, or by adding new proof rules that the automated prover can exploit. These new proof rules can be seen as axioms and must be verified by other means, otherwise the global correctness may be endangered.

In [13], we develop an approach based on the use of Zenon to verify B proof rules. The method used in this approach consists in first normalizing the formulas to be proved, in order to obtain first order formulas containing only the membership set operator, and then calling Zenon on these new formulas. This experiment gives satisfactory results in the sense that it can prove a significant part of the rules coming from the database maintained by Siemens IC-MOL (we can deal with about 1,400 rules, 1,100 of which can be proved automatically, over a total of 5,300 rules). However, this approach is not complete (after the normalization, Zenon proves the formulas without any axiom of set theory, while some instantiations may require to be normalized), and suffers from efficiency issues (due to the preliminary normalization). To deal with these problems, the idea developed in [14] is to integrate the axioms and constructs of the B set theory into the Zenon proof search method by means of superdeduction rules. This integration is concretely achieved thanks to the extension mechanism offered by Zenon, which allows us to extend its core of deductive rules to match specific requirements. This new tool has emphasized significant speed-ups both in terms of proof time and proof size compared to the previous approach (see [14] for more details).

The B method is based on a typed set theory. There are two rule systems: one for demonstrating that a formula is well-typed, and one for demonstrating that a formula is a logical consequence of a set of axioms. The main aim of the type system is to avoid inconsistent formulas, such as Russell's paradox for example. The B proof system is based on a sequent calculus with equality. Six axiom schemes define the basic operators and the extensionality which, in turn, defines the equality of two sets. In addition, the other operators ($\cup$, $\cap$, etc.) are defined using the previous basic ones. To generate the superdeduction rules corresponding to the axioms and constructs, we use the algorithm described in Section 2, and we must therefore identify the several proposition rewrite rules. Regarding the axioms, they are all of the form $P_i \Leftrightarrow Q_i$, and the associated proposition rewrite rules are therefore

$R_i : P_i \rightarrow Q_i$, where each axiom is oriented from left to right. For instance, let us consider the equality of two sets, which is defined by the following axiom:

$$a = b \Leftrightarrow \forall x \ (x \in a \Leftrightarrow x \in b)$$

From this axiom, we can compute two superdeduction rules as follows (a third rule dealing with instantiation is also implicitly computed since one of the generated rules involves metavariables):

$$\frac{a = b}{X \notin a, X \notin b \mid X \in a, X \in b} = \qquad \frac{a \neq b}{\epsilon_x \notin a, \epsilon_x \in b \mid \epsilon_x \in a, \epsilon_x \notin b} \neq$$
$$\text{with } \epsilon_x = \epsilon(x).\neg(x \in a \Leftrightarrow x \in b)$$

Concerning the constructs, they are expressed by definitions of the form $E_i \triangleq F_i$, where E_i and F_i are expressions, and the corresponding proposition rewrite rules are $R_i : x \in E_i \rightarrow x \in F_i$. Let us illustrate the computation of superdeduction rules for constructs with the example of domain restriction, which is defined in the following way:

$$a \lhd b \triangleq \mathrm{id}(a); b$$

where:

$$a; b \triangleq \{ \ (x, z) \mid \exists y \ ((x, y) \in a \wedge (y, z) \in b \ \}$$
$$\mathrm{id}(a) \triangleq \{ \ (x, y) \mid (x, y) \in a \times a \wedge x = y \ \}$$

The corresponding superdeduction rules are computed as follows:

$$\frac{(x, y) \in a \lhd b}{(x, y) \in b, x \in a} \lhd \qquad \frac{(x, y) \notin a \lhd b}{(x, y) \notin b \mid x \notin a} \neg\lhd$$

For further details regarding the computation of superdeduction rules for the B set theory, as well as the corresponding implementation using Zenon, the reader can refer to [14].

4.2 Verification of a B Proof Rule

To assess our extension of Zenon for the B set theory using superdeduction and to show that it can produce proofs comprehensible enough to recover the intuition of these proofs, we propose to consider the example of a B proof rule coming from the database maintained by Siemens IC-MOL. The rule being considered is the rule named "SimplifyRelDorXY.27" (this rule is actually part of the Atelier B set of rules), and whose proof is small enough to be understood easily and described in the space restrictions for this paper. The definition of this rule is the following:

$$\emptyset \lhd a = \emptyset$$

When applied to this rule, our extension of Zenon produces the proof of Figure 2 (Zenon proposes several proof formats, and the proof presented in this figure uses the format with the highest level of abstraction). The statement of the rule, i.e. the command starting with "fof", is provided using the TPTP syntax [19], and the proof (if found) is displayed after this statement. The proof

```
fof(simplifyRelDorXY_27, conjecture,
  b_eq (b_drest (b_empty, a), b_empty)).
(* PROOF-FOUND *)
   1. H0: (-. (b_eq (b_drest (b_empty) (a)) (b_empty)))
      ### [Extension/b/b_not_eq H0 H1 H2 H3 H4 H5 H6] --> 2 3
   2. H2: (b_in T_7 (b_empty))
      ### [Extension/b/b_in_empty H2 H8 H9 H7] --> 4
   4. H9: (-. (b_in T_7 (b_BIG)))
      H8: (b_in T_7 (b_BIG))
      ### [Axiom H8 H9]
   3. H3: (b_in T_7 (b_drest (b_empty) (a)))
      ### [Extension/b/b_in_drest H3 H10 H11 H12 H7 H6 H13] --> 5
   5. H12: (b_in T_14 (b_empty))
      ### [Extension/b/b_in_empty H12 H15 H16 H14] --> 6
   6. H16: (-. (b_in T_14 (b_BIG)))
      H15: (b_in T_14 (b_BIG))
      ### [Axiom H15 H16]
```

Figure 2: Proof of Rule "SimplifyRelDorXY.27" of Atelier B

consists of several numbered steps, where each of them is a set of formulas together with a proof rule which has been applied to the considered proof step. Formulas of a proof step are signed formulas, and formulas starting with "-." are negative formulas. A proof rule appears at the end of a proof step and after the string "###", and also provides, after the string "–>", the other proof steps to which it is connected (these other proof steps represent the result of the application of this proof rule to the set of formulas of the considered proof step). For example, in this proof, Step 1 is connected to Steps 2 and 3. This connection between proof steps provides the proof with a tree-like structure, where proof steps with axiomatic rules, i.e. starting with "Axiom", are leaves, while the other proof steps are nodes. Among these other proof steps, there are in particular superdeduction rules, which start with "Extension". In this proof, the B constructs are prefixed by "b_", and "b_empty", "b_BIG", "b_in", "b_eq", and "b_drest" respectively represent the empty set $\emptyset$, the set BIG (which is an infinite set, mostly only used to build natural numbers in the foundational theory), the membership operator "$\in$", the (extensional) equality "$=$", and the domain restriction construct "$\lhd$".

As can be observed, this proof expressed in this format can be easily understood not only thanks to the tableau method which follows a natural way to find the proof in this case, but also thanks to superdeduction rules which allow us to shorten the proof removing formal details useless for the comprehension of the proof. To justify this claim, let us describe the formal proof sketch which can be extracted from this formal proof, and which is appropriate to provide the intuition of the proof. This formal proof sketch is built as follows:

1. The proof starts from the sequent "$\vdash \emptyset \lhd a = \emptyset$", which corresponds to Hypothesis H0 in Step 1 in the formal proof (where the initial formula has been negated since the tableau method works by refutation). The proof rule applied to this sequent is a superdeduction rule which deals with the equality, and which corresponds to the superdeduction rule named "b_not_eq" in the formal

proof, i.e. the negation of the equality still because the initial formula has been negated. In sequent calculus, this superdeduction rule is the following:

$$\frac{\Gamma, x \in a \vdash x \in b, \Delta \qquad \Gamma, x \in b \vdash x \in a, \Delta}{\Gamma \vdash a = b, \Delta} =\text{R}, x \notin \Gamma, \Delta$$

2. Applying the superdeduction rule for equality, we obtain two cases to prove (as shown by the rule above). The formal proof focuses on the right-hand side of the rule at first, and we therefore have to prove the sequent "$x \in \emptyset \vdash x \in \emptyset \lhd a$", which corresponds to Step 2. As can be seen, in the set of formulas of each proof step, the formal proof only displays the formulas which are useful to complete the proof. For instance, in Step 2, the formula "$x \in \emptyset \lhd a$" is not displayed (even though it is present in the set of formulas), because it is not used in the following of the proof. The formal proof therefore focuses on the hypothesis "$x \in \emptyset$" and applies the superdeduction rule named "b_in_empty" and corresponding to the empty set. In B, the empty set is defined as follows: $\emptyset \triangleq \text{BIG} - \text{BIG}$. In sequent calculus, the corresponding superdeduction rule is then computed as follows:

$$\frac{\Gamma, x \in \text{BIG}, x \notin \text{BIG} \vdash \Delta}{\Gamma, x \in \emptyset \vdash \Delta} \emptyset\text{L}$$

Once this rule has been applied, we obtain the sequent "$x \in \text{BIG}, x \notin \text{BIG} \vdash x \in \emptyset \lhd a$", which is proved by reductio ad absurdum and corresponds to Step 4 in the formal proof.

3. The second case following the application of the superdeduction rule for equality corresponds to the sequent "$x \in \emptyset \lhd a \vdash x \in \emptyset$", which appears to be Step 3 in the formal proof. In this step, the formal proof focuses on the hypothesis "$x \in \emptyset \lhd a$", and applies the superdeduction rule named "b_in_drest" and corresponding to the domain restriction. This superdeduction rule is the following in sequent calculus:

$$\frac{\Gamma, x = (y, z), (y, z) \in b, y \in a \vdash \Delta}{\Gamma, x \in a \lhd b \vdash \Delta} \lhd\text{L}, y, z \notin \Gamma, \Delta$$

Once this rule has been applied, we obtain the sequent "$x = (y, z), (y, z) \in a, y \in \emptyset \vdash x \in \emptyset$", where the formal proof can again focus on the hypothesis $y \in \emptyset$ in Step 5 and close the proof as previously in Step 6.

From this formal proof sketch, it is now quite easy to produce an informal and short proof (as it would have been done in a textbook) as follows:

- To show $\emptyset \lhd a = \emptyset$, we have to consider two cases: given $x \in \emptyset$, we must show $x \in \emptyset \lhd a$, and given $x \in \emptyset \lhd a$, we must show $x \in \emptyset$;
  1. If $x \in \emptyset$ then $x \in \text{BIG}$ and $x \notin \text{BIG}$, which is therefore absurd;
  2. If $x \in \emptyset \lhd a$ then $x = (y, z)$, $(y, z) \in a$, and $y \in \emptyset$, which is absurd as previously.

5 A Generic Implementation for First Order Theories

In this section, we present our second extension of Zenon with superdeduction, which is able to deal with any first order theory. In this extension, the theory is analyzed to determine the axioms which can be turned into superdeduction rules, and these superdeduction rules are automatically computed on the fly to enrich the deductive kernel of Zenon. We also describe the proofs of two examples coming from the TPTP library and produced by this tool, and which are quite comprehensible as well.

5.1 From Theories to Superdeduction Systems

This extension of Zenon is actually a generalization of the previous one dedicated to the B set theory, where superdeduction rules are henceforth automatically computed on the fly. In the previous extension, superdeduction rules are hard-coded since the B set theory is a higher order theory due to one of the axioms of the theory (the comprehension scheme), and we have to deal with this axiom specifically in the implementation of Zenon. Even though some techniques exist to handle higher order theories as first order theories (like the theory of classes, for example), a hard-coding of these theories may be preferred as these techniques unfortunately tend to increase the entropy of the proof search. In addition, in the previous extension, some of the superdeduction rules must be manually generated as they must be shrewdly tuned (ordering the several branches of the rules, for instance) to make the tool efficient. The new extension of Zenon dealing with any first order theory has been developed as a tool called Super Zenon [15], where each theory is analyzed to determine the axioms which are candidates to be turned into superdeduction rules. As said in Section 3, axioms of the form $\forall \bar{x}\ (P \Leftrightarrow \varphi)$, where P is atomic, can be transformed, but we can actually deal with more axioms. Here is the exhaustive list of axioms that can be handled, as well as the corresponding superdeduction rules that can be generated (in the following, P and P' are atomic, and φ is an arbitrary formula):

- Axiom of the form $\forall \bar{x}\ (P \Leftrightarrow \varphi)$: we consider the proposition rewrite rule $R : P \rightarrow \varphi$, and the two superdeduction rules R and $\neg R$ are generated;
- Axiom of the form $\forall \bar{x}\ (P \Rightarrow P')$: we consider the proposition rewrite rules $R : P \rightarrow P'$ and $R' : \neg P' \rightarrow \neg P$, and only the superdeduction rules R and R' are generated;
- Axiom of the form $\forall \bar{x}\ (P \Rightarrow \varphi)$: we consider the proposition rewrite rule $R : P \rightarrow \varphi$, and only the superdeduction rule R is generated;
- Axiom of the form $\forall \bar{x}\ (\varphi \Rightarrow P)$: we consider the proposition rewrite rule $R : \neg P \rightarrow \neg\varphi$, and only the superdeduction rule R is generated;
- Axiom of the form $\forall \bar{x}\ P$: we consider the proposition rewrite rule $R : \neg P \rightarrow \bot$, and only the superdeduction rule R is generated.

The axioms of the theory, which are not of these forms, are left as regular axioms. An axiom, which is of one of these forms, is also left as a regular axiom if the conclusion of one of the generated superdeduction rules (i.e. the top formula of one of these rules) unifies with the conclusion of an already computed superdeduction rule (in this case, the theory is actually non-deterministic and we try to minimize this source of non-determinism by dividing these incriminated axioms among the sets

of superdeduction rules and regular axioms). An axiom, which is of one of these forms, is still left as a regular axiom if P is an equality (as we do not want to interfere with the specific management of equality by the kernel of Zenon). Finally, it should be noted that for axioms of the form $\forall \bar{x}\ (P \Rightarrow P')$, we also consider the proposition rewrite rule which corresponds to the converse of the initial formula; this actually allows us to keep cut-free completeness in this particular case.

5.2 Proof of a Logic Puzzle

As the Super Zenon tool is able to deal with any first order theory, it can be used in many contexts, and in particular, it can be applied to all the first order problems of the TPTP library [19] (about 6,600 problems), which is a library of test problems for automated theorem proving systems. To assess the effectiveness of this tool and to show that it can also produce comprehensible proofs, let us consider an example of the puzzle category of TPTP and called "Crime in Beautiful Washington" (Puzzle #132), which is a problem in the same vein as the "Who Killed Aunt Agatha?" well-known puzzle. This kind of problems is quite appropriate for educational purposes when teaching artificial intelligence and logic for example. The problem being considered consists of the following axioms:

$\forall x\ (\mathrm{capital}(x) \Rightarrow \mathrm{city}(x))$	$(capital\_city\_type)$
$\mathrm{capital}(\mathrm{washington})$	$(washington\_type)$
$\mathrm{country}(\mathrm{usa})$	(usa_type)
$\forall x\ (\mathrm{country}(x) \Rightarrow \mathrm{capital}(\mathrm{capital\_city}(x)))$	$(country\_capital\_type)$
$\forall x\ (\mathrm{city}(x) \Rightarrow \mathrm{has\_crime}(x))$	$(crime\_axiom)$
$\mathrm{capital\_city}(\mathrm{usa}) = \mathrm{washington}$	$(usa\_capital\_axiom)$
$\forall x\ (\mathrm{country}(x) \Rightarrow \mathrm{beautiful}(\mathrm{capital\_city}(x)))$	$(beautiful\_capital\_axiom)$

As can be observed, all the axioms can be turned into superdeduction rules, except the axiom $(usa\_capital\_axiom)$ since it is an equality, and the axiom $(beautiful\_capital\_axiom)$ since one of the generated superdeduction rules for this axiom overlap with one of the superdeduction rules computed previously for the axiom $(country\_capital\_type)$.

The conjecture to be proved is expressed as follows:

$$\mathrm{beautiful}(\mathrm{washington}) \wedge \mathrm{has\_crime}(\mathrm{washington})$$

When applied to this specification, Super Zenon produces the proof of Figure 3 for the previous conjecture (we still use the proof format with the highest level of abstraction). As in the example of verification of a B proof rule in Section 4, it is possible to build quite directly the following informal proof sketch from this formal proof:

- To show $\mathrm{beautiful}(\mathrm{washington}) \wedge \mathrm{has\_crime}(\mathrm{washington})$, we have to consider the two cases $\mathrm{beautiful}(\mathrm{washington})$ and $\mathrm{has\_crime}(\mathrm{washington})$;

    1. To show $\mathrm{beautiful}(\mathrm{washington})$, we apply the axiom $(beautiful\_capital\_axiom)$ instantiated with usa, and we have to consider two cases: we must show $\mathrm{country}(\mathrm{usa})$, and given $\mathrm{beautiful}(\mathrm{capital\_city}(usa))$, we must show $\mathrm{beautiful}(\mathrm{washington})$;

        (a) To show $\mathrm{country}(\mathrm{usa})$, we use the axiom (usa_type);

```
fof(washington_conjecture, conjecture,
  (beautiful (washington) & has_crime (washington))).
(* PROOF-FOUND *)
   1. H0: (-. ((beautiful (washington)) /\ (has_crime (washington))))
      H1: ((capital_city (usa)) = (washington))
      H2: (All X, ((country X) => (beautiful (capital_city X))))
      ### [NotAnd H0] --> 2 3
   2. H3: (-. (beautiful (washington)))
      ### [All H2] --> 4
   4. H4: ((country (usa)) => (beautiful (capital_city (usa))))
      ### [Imply H4] --> 5 6
   5. H5: (-. (country (usa)))
      ### [Extension/szen/usa_type H5]
   6. H6: (beautiful (capital_city (usa)))
      ### [P-NotP H6 H3] --> 7
   7. H7: ((capital_city (usa)) != (washington))
      ### [Axiom H1 H7]
   3. H8: (-. (has_crime (washington)))
      ### [Extension/szen/crime_axiom H8 H9 H10] --> 8
   8. H9: (-. (city (washington)))
      ### [Extension/szen/capital_city_type H9 H11 H10] --> 9
   9. H11: (-. (capital (washington)))
      ### [Extension/szen/washington_type H11]
```

Figure 3: Proof of Puzzle #132 of TPTP

(b) Given $\mathrm{beautiful}(\mathrm{capital\_city}(usa)$, to show $\mathrm{beautiful}(\mathrm{washington})$, it is enough to show $\mathrm{capital\_city}(\mathrm{usa}) = \mathrm{washington}$ using the axiom $(usa\_capital\_axiom)$.

2. To show $\mathrm{has\_crime}(\mathrm{washington})$, we apply the axiom $(crime\_axiom)$, and we must show $\mathrm{city}(\mathrm{washington})$;
   - To show $\mathrm{city}(\mathrm{washington})$, we apply the axiom $(capital\_city\_type)$, and we must show $\mathrm{capital}(\mathrm{washington})$;
   - To show $\mathrm{capital}(\mathrm{washington})$, we use the axiom $(washington\_type)$.

5.3 Proof of a Geometry Problem

As a second example of proof, we consider a problem coming from the geometry category of the TPTP library. This problem (Problem #170+3) states that if two distinct points are incident with a line, then this line is equivalent with the connecting line of these points. The interest of such an example is actually twofold. First, the corresponding proof is larger (but remains reasonably large to be presented in this paper) than for the previous considered examples, which tends to show that our approach is effective even when the proofs require more than 20 steps. Second, the subject of

geometry is quite fundamental in the high school curriculum in the sense that it is generally the only mathematical topic where proofs are explicitly mentioned and where formal reasoning is actually considered. The axioms considered in this example are the axioms of constructive geometry, but Super Zenon uses classical logic and constructive geometry plus classical logic is equivalent to textbook theories. The proof of this example uses the following axioms of this theory (we use the names given in the TPTP files):

$$\forall x, y\ (\text{distinct\_points}(x, y) \Rightarrow \neg\text{apart\_point\_and\_line}(x, \text{line\_connecting}(x, y))) \quad (ci1)$$
$$\forall x, y\ (\text{distinct\_points}(x, y) \Rightarrow \neg\text{apart\_point\_and\_line}(y, \text{line\_connecting}(x, y))) \quad (ci2)$$
$$\forall x, y, u, v\ (\text{distinct\_points}(x, y) \wedge \text{distinct\_lines}(u, v) \Rightarrow \text{apart\_point\_and\_line}(x, u) \vee \text{apart\_point\_and\_line}(x, v) \vee \text{apart\_point\_and\_line}(y, u) \vee \text{apart\_point\_and\_line}(y, v)) \quad (cu1)$$
$$\forall x, y\ (\text{equal\_lines}(x, y) \Leftrightarrow \neg\text{distinct\_lines}(x, y)) \quad (ax2)$$
$$\forall x, y\ (\text{incident\_point\_and\_line}(x, y) \Leftrightarrow \neg\text{apart\_point\_and\_line}(x, y)) \quad (a4)$$

where $\text{distinct\_points}(x, y)$ (resp. $\text{distinct\_lines}(x, y)$) means that x and y are two distinct points (resp. lines), $\text{incident\_point\_and\_line}(x, y)$ (resp. $\text{apart\_point\_and\_line}(x, y)$) means that the point x is (resp. is not) incident with the line y, $\text{equal\_lines}(x, y)$ means that x and y denote the same line, and $\text{line\_connecting}(x, y)$ denotes the line connecting the points x and y.

Among these axioms, the axioms $(ci2)$, $(ax2)$, and $(a4)$ are turned into superdeduction rules. The axiom $(ci1)$ is left as an axiom because one of its superdeduction rules overlap with one of the superdeduction rules computed previously for the axiom $(ci2)$. The axiom $(cu1)$ is also left as an axiom because it has not the right form to be turned into superdeduction rules. It should be noted that for the axiom $(ax2)$, a superdeduction rule corresponding to the converse of this axiom is also generated since both sides of the implication are atomic.

The conjecture given previously is formally expressed as follows:

$$\forall x, y, z\ (\text{distinct\_points}(x, y) \wedge \text{incident\_point\_and\_line}(x, z) \wedge \text{incident\_point\_and\_line}(y, z) \Rightarrow \text{equal\_lines}(z, \text{line\_connecting}(x, y)))$$

When applied to this problem, Super Zenon is able to produce the proof of Figures 4 and 5 (we use the same proof format than for the previous examples), where the preliminary Skolemization steps are compressed (Steps 2 and 3 are left implicit). From this proof, it is possible to build the following informal proof sketch as previously:

- Given the points x, y, and the line z s.t. $\text{distinct\_points}(x, y)$, $\text{incident\_point\_and\_line}(x, z)$, and $\text{incident\_point\_and\_line}(y, z)$, we have to show $\text{equal\_lines}(z, \text{line\_connecting}(x, y))$;
- From the hypotheses $\text{incident\_point\_and\_line}(x, z)$ and $\text{incident\_point\_and\_line}(y, z)$, we have $\neg\text{apart\_point\_and\_line}(x, z)$ and $\neg\text{apart\_point\_and\_line}(y, z)$ using the axiom $(a4)$;
- Using the axiom $(ax2)$, the goal $\text{equal\_lines}(z, \text{line\_connecting}(x, y))$ to be proved is equivalent to $\neg\text{distinct\_lines}(z, \text{line\_connecting}(x, y))$;
- Using the axiom $(cu1)$ with x, y, z, and $\text{line\_connecting}(x, y)$, we have to show the previous goal in the following cases:

```
fof(geometry_conjecture, conjecture,
  (! [X, Y, Z] : ((distinct_points (X, Y) &
    incident_point_and_line (X, Z) &
    incident_point_and_line (Y, Z)) =>
    equal_lines (Z, line_connecting(X, Y))))).
(* PROOF-FOUND *)
   1. H0: (-. (All X, (All Y, (All Z, (((distinct_points X Y) /\
             ((incident_point_and_line X Z) /\
             (incident_point_and_line Y Z))) =>
             (equal_lines Z (line_connecting X Y)))))))
      H1: (All X, (All Y, ((distinct_points X Y) =>
             (-. (apart_point_and_line X (line_connecting X Y))))))
      H2: (All X, (All Y, (All U, (All V, (((distinct_points X Y) /\
             (distinct_lines U V)) => ((apart_point_and_line X U) \/
             ((apart_point_and_line X V) \/
             ((apart_point_and_line Y U) \/
             (apart_point_and_line Y V)))))))))
      ### [NotAllEx H0] --> [...] 4
   4. H7: (incident_point_and_line T_4 T_8)
      H9: (-. (equal_lines T_8 (line_connecting T_4 T_6)))
      H10: (distinct_points T_4 T_6)
      H11: (incident_point_and_line T_6 T_8)
      ### [Extension/szen/a4 H7 H12 H4 H8] --> 5
   5. H12: (-. (apart_point_and_line T_4 T_8))
      ### [Extension/szen/a4 H11 H13 H6 H8] --> 6
   6. H13: (-. (apart_point_and_line T_6 T_8))
      ### [Extension/szen/not_ax2 H9 H14 H8 H15] --> 7
   7. H14: (distinct_lines T_8 (line_connecting T_4 T_6))
      ### [All H2] --> 8
   8. H16: (All Y, (All U, (All V, (((distinct_points T_4 Y) /\
             (distinct_lines U V)) => ((apart_point_and_line T_4 U) \/
             ((apart_point_and_line T_4 V) \/
             ((apart_point_and_line Y U) \/
             (apart_point_and_line Y V)))))))))
      ### [All H16] --> 9
   9. H17: (All U, (All V, (((distinct_points T_4 T_6) /\
             (distinct_lines U V)) => ((apart_point_and_line T_4 U) \/
             ((apart_point_and_line T_4 V) \/
             ((apart_point_and_line T_6 U) \/
             (apart_point_and_line T_6 V)))))))
      ### [All H17] --> 10
```

Figure 4: Proof of Geometry Problem #170+3 of TPTP (Part 1)

```
10. H18: (All V, (((distinct_points T_4 T_6) /\
           (distinct_lines T_8 V)) =>
           ((apart_point_and_line T_4 T_8) \/
           ((apart_point_and_line T_4 V) \/
           ((apart_point_and_line T_6 T_8) \/
           (apart_point_and_line T_6 V))))))
    ### [All H18] --> 11
11. H19: (((distinct_points T_4 T_6) /\
           (distinct_lines T_8 (line_connecting T_4 T_6))) =>
           ((apart_point_and_line T_4 T_8) \/
           ((apart_point_and_line T_4 (line_connecting T_4 T_6)) \/
           ((apart_point_and_line T_6 T_8) \/
           (apart_point_and_line T_6 (line_connecting T_4 T_6))))))
    ### [DisjTree H19] --> 12 13 14 15 16 17
12. H20: (-. (distinct_points T_4 T_6))
    ### [Axiom H10 H20]
13. H21: (-. (distinct_lines T_8 (line_connecting T_4 T_6)))
    ### [Axiom H14 H21]
14. H22: (apart_point_and_line T_4 T_8)
    ### [Axiom H22 H12]
15. H23: (apart_point_and_line T_4 (line_connecting T_4 T_6))
    ### [All H1] --> 18
18. H24: (All Y, ((distinct_points T_4 Y) =>
           (-. (apart_point_and_line T_4 (line_connecting T_4 Y)))))
    ### [All H24] --> 19
19. H25: ((distinct_points T_4 T_6) =>
           (-. (apart_point_and_line T_4 (line_connecting T_4 T_6))))
    ### [Imply H25] --> 20 21
20. H20: (-. (distinct_points T_4 T_6))
    ### [Axiom H10 H20]
21. H26: (-. (apart_point_and_line T_4 (line_connecting T_4 T_6)))
    ### [Axiom H23 H26]
16. H27: (apart_point_and_line T_6 T_8)
    ### [Axiom H27 H13]
17. H28: (apart_point_and_line T_6 (line_connecting T_4 T_6))
    ### [Extension/szen/ci2ctrp H28 H20 H4 H6] --> 22
22. H20: (-. (distinct_points T_4 T_6))
    ### [Axiom H10 H20]
```

Figure 5: Proof of Geometry Problem #170+3 of TPTP (Part 2)

1. Given $\neg\text{distinct\_points}(x, y)$, we have also $\text{distinct\_points}(x, y)$ in hypothesis, which is therefore absurd;
2. Given $\neg\text{distinct\_lines}(z, \text{line\_connecting}(x, y))$, it is exactly the goal to be proved, which is then proved directly by hypothesis;
3. Given $\text{apart\_point\_and\_line}(x, z)$, we have also $\neg\text{apart\_point\_and\_line}(x, z)$ in hypothesis, which is therefore absurd;
4. Given $\text{apart\_point\_and\_line}(x, \text{line\_connecting}(x, y))$, we use the axiom $(ci1)$ with x, y, and $\text{distinct\_points}(x, y)$, to have $\neg\text{apart\_point\_and\_line}(x, \text{line\_connecting}(x, y))$, which is therefore absurd;
5. Given $\text{apart\_point\_and\_line}(y, z)$, we have also $\neg\text{apart\_point\_and\_line}(y, z)$ in hypothesis, which is therefore absurd;
6. Given $\text{apart\_point\_and\_line}(y, \text{line\_connecting}(x, y))$ used with the converse of the axiom $(ci2)$ with x and y, we have $\neg\text{distinct\_points}(x, y)$, which is therefore absurd as we have also $\text{distinct\_points}(x, y)$ in hypothesis.

6 Conclusion

In this paper, we have proposed an automated deduction method which allows us to produce proofs close to the human intuition and practice. This method is based on tableaux and uses the principles of superdeduction, among which the theory is used to enrich the deduction system with new deduction rules, called superdeduction rules. We have presented two implementations of this method, which consist of extensions of the Zenon automated theorem prover. The first implementation is a version dedicated to the B set theory, where the superdeduction rules are hard-coded from the initial axiomatic theory. The second implementation is a generic version able to deal with any first order theory, where the theory is analyzed to determine the axioms which can be turned into superdeduction rules, and where these superdeduction rules are automatically computed on the fly to enrich the deductive kernel of Zenon. For information, these two implementations are available as free software at [15]. We have also provided some examples of problems, which can be handled by these tools and which come from different theories, such as the B set theory or theories of the TPTP library (in the puzzle and geometry categories, in particular). In these examples, we have shown that both tools are able to produce formal proofs comprehensible enough to recover the intuition of these proofs, and that the user can therefore easily extract informal proof sketches from these proofs.

As future work, it would be interesting to improve the readability of the produced proofs in order to get more natural proofs and in particular, it might be desirable to turn proofs into pure direct proofs (searching for a proof of the initial formula), rather than refutational proofs (searching to invalidate the negation of the initial formula). In a way, this corresponds to get back to Gentzen's initial purely proof theoretical motivation when trying to find proofs in the cut-free version of sequent calculus, and in particular, this is opposed to Hintikka and Beth's semantic view of tableaux, which consists of a procedure systematically trying to find a counter example for a given formula (i.e. a model in which its negation is true). Such an improvement evokes some similar initiatives in other automated theorem provers, such as Muscadet [16], which is based on natural deduction and which uses methods resembling those used by humans.

To increase the readability of the proofs generated by our extensions, it would also be interesting to export these proofs to other kinds of languages, which appear more appropriate regarding readability. In particular, we could export the proofs to declarative proof languages, such as Isar [22] for Isabelle, which tends to bridge the semantic gap between internal notions of proof given by state-of-the-art interactive theorem proving systems and an appropriate level of abstraction for user-level work. This translation should be automatic, and to be more effective, it should also be probably combined with an interactive layer over the automated deduction tool (see below) in order to produce intelligible proofs, i.e. proofs where a certain number of cuts can be manually introduced. We could even go further and automatically produce proofs in natural languages using, for example, the ideas of [8], where Coq formal proofs are translated in a pseudo natural language.

Finally, in this paper, our extensions only produce proofs automatically without any interaction with the user. In an educational setting, a system able to present sample proofs is already a valuable bonus, but the students must also be involved in the process of building proofs. To do so, the idea is to implement an interactive layer over our extensions in the spirit of [21], which will aim to offer the user the possibility to guide the proof search. This interactive layer would be a benefit for both the user and the automated deduction tool. For the user, this layer could make the interface with the proof engine, which could propose a set of applicable rules or next-step hints. For the automated deduction tool, this layer could be used to find proofs with the help of the user, who could propose to focus on some branches of the proof search, which would allow the tool to find a proof, while the strategy of the tool would have focused on inappropriate branches resulting in an endless proof search.

References

[1] Jean-Raymond Abrial. *The B-Book, Assigning Programs to Meanings*. Cambridge University Press, Cambridge (UK), 1996. ISBN 0521496195.

[2] Evert Willem Beth. Semantic Entailment and Formal Derivability. *Medelingen von de Koninklijke Nederlandse Akademie van Wetenschappen, Afdeling Letterkunde*, 18(13):309–342, 1955.

[3] Richard Bonichon, David Delahaye, and Damien 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), October 2007. Springer.

[4] Paul Brauner, Clément Houtmann, and Claude Kirchner. Principles of Superdeduction. In *Logic in Computer Science (LICS)*, pages 41–50, Wrocław (Poland), July 2007. IEEE Computer Society Press.

[5] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8):677–691, August 1986.

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

[7] ClearSy. *Atelier B 4.0*, February 2009. `http://www.atelierb.eu/`.

[8] Yann Coscoy. A Natural Language Explanation for Formal Proofs. In *Logical Aspects of Computational Linguistics (LACL)*, volume 1328 of *LNCS*, pages 149–167, Nancy (France), September 1996. Springer.

[9] Martin Davis and Hillary Putnam. A Computing Procedure for Quantification Theory. *Journal of the Association for Computing Machinery (JACM)*, 7(3):201–215, 1960.

[10] Gilles Dowek, Therèse Hardin, and Claude Kirchner. Theorem Proving Modulo. *Journal of Automated Reasoning (JAR)*, 31(1):33–72, September 2003.

[11] Gerhard Gentzen. Untersuchungen über das Logische Schließen. *Mathematische Zeitschrift*, 39(2/3):176–210, 405–431, 1935.

[12] K. Jaakko J. Hintikka. Form and Content in Quantification Theory. *Acta Philosophica Fennica*, 8:7–55, 1955.

[13] Mélanie Jacquel, Karim Berkani, David Delahaye, and Catherine Dubois. Verifying B Proof Rules using Deep Embedding and Automated Theorem Proving. In *Software Engineering and Formal Methods (SEFM)*, volume 7041 of *LNCS*, pages 253–268, Montevideo (Uruguay), November 2011. Springer.

[14] Mélanie Jacquel, Karim Berkani, David Delahaye, and Catherine 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.

[15] Mélanie Jacquel and David Delahaye. *Super Zenon, version 0.0.1*. Siemens and Cnam, May 2012. `http://cedric.cnam.fr/~delahaye/super-zenon/`.

[16] Dominique Pastre. *Muscadet 4.1*. Université Paris Descartes (Paris 5), April 2011. `http://www.math-info.univ-paris5.fr/~pastre/muscadet/`.

[17] Dag Prawitz. Natural Deduction. A Proof-Theoretical Study. *Stockholm Studies in Philosophy*, 3, 1965.

[18] John Alan Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the Association for Computing Machinery (JACM)*, 12(1):23–41, January 1965.

[19] Geoff 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, December 2009.

[20] The Coq Development Team. *Coq, version 8.4*. Inria, August 2012. `http://coq.inria.fr/`.

[21] Daniel J. Velleman. *How to Prove It: A Structured Approach*. Cambridge University Press, New York (NY, USA), April 2006. ISBN 9780521675994.

[22] Markus Wenzel. Isar – A Generic Interpretative Approach to Readable Formal Proof Documents. In *Theorem Proving in Higher Order Logics (TPHOLs)*, volume 1690 of *LNCS*, pages 167–184, Nice (France), September 1999. Springer.