Interpreting Leo-II's proofs in Isabelle/HOL

Nik Sultana[1]* and Christoph Benzmüller[2]

[1] Cambridge University Computer Lab, UK
[2] Freie Universität Berlin, Germany

Abstract. We describe the design and implementation of a proof-reconstruction module that converts TPTP-encoded proofs produced by LEO-II into theorems of Isabelle/HOL. We provide theoretical and practical justification for design decisions, and discuss the generality of the method.

1 Introduction

The case for interfacing logic tools together has been made countless times in the literature, but this has not diluted its importance. Nor are the solutions to this problem reaching saturation: despite there being various logics and tools for carrying out formal developments, practitioners lament the difficulty of reliably exchanging mathematical data between tools.

Writing a proof-reconstruction tool is hard, since usually such tools need to be extended for each source prover they are to handle (since each prover usually implements its own proof calculus). Moreover, system integration is inevitably fraught with difficulties of engineering. This article describes a how to uniformly transform proofs from a class of calculi into a form which can be readily reconstructed in Isabelle/HOL (§3).

We also describe a new bridge between LEO-II and Isabelle/HOL, which allows Isabelle/HOL users to reliably import proofs found by LEO-II. There is already an interface between LEO-II and Isabelle/HOL, forming part of Isabelle/HOL's Sledgehammer tool [1]. Sledgehammer interfaces between Isabelle/HOL and various kinds of proof tools. It prepares and translates Isabelle/HOL problems into the provers' input languages. If a target prover finds a proof, Sledgehammer oversees the reconstruction of the proof in Isabelle/HOL. However, proof reconstruction in Sledgehammer consists of re-finding a proof using another prover whose proofs can be interpreted by Isabelle/HOL: currently Sledgehammer relies on Metis [2] and Z3 [3] for this. We hope that our work can complement these by making LEO-II available to assist with proof-reconstruction. The immediate consequence of our work is that the reliability of reconstruction of LEO-II proofs increases considerably—from about 70% to over 90% in our experiments (§5).

* Thanks to Larry Paulson for regular conversations on this topic, and for Isabelle.

Contributions. We study syntactic transformations (§3) which can map proofs encoded in formalisms belonging to a class of consistency-preserving calculi, into proofs encoded in formalisms in a class of validity-preserving calculi. In particular, we study how this can be used to map LEO-II proofs into Isabelle/HOL theorems. We argue for the adequacy (soundness and completeness) of these transformations. An implementation of a LEO-II proof-reconstruction module for Isabelle/HOL is described (§4) and evaluated (§5).

In the course of this work we have done various improvements to LEO-II's proof output. We also offer suggestions for improvements to the TPTP proof format, arising from our experience in building the proof-reconstruction module. These suggestions are discussed in §7. Related work is addressed in §6.

2 Background

2.1 Leo-II

LEO-II [4] is an implementation of a resolution-style calculus for classical higher-order predicate logic. It works in concert with other provers (usually E [5]) to find refutations. In this article we will focus on the calculus of LEO-II (as reflected in its proof output), not the implementation, and we will describe parts of the calculus as we go along. LEO-II consumes problems encoded in TPTP languages [6], and produces proofs [7] which are also encoded in TPTP languages. In this article we will describe how such proofs can be imported into Isabelle/HOL as theorems.

2.2 Isabelle and Isabelle/HOL

Isabelle [8] is a programmatic framework for building interactive reasoning tools. It provides a meta-logic and associated API to facilitate the encoding of, and reasoning with, various logics. Isabelle/HOL [9] is an encoding of classical higher-order predicate logic (HOL) in Isabelle.

Isabelle's formalism is based on the simply-typed λ-calculus [10], and it uses a linear notation for rules. For instance the rule $\frac{A \wedge B}{B}$ is encoded in Isabelle as $\mathsf{Trueprop}(A \wedge B) \Longrightarrow \mathsf{Trueprop}(B)$. Note that '$A \wedge B$' and '$B$' are terms in the object language (say, propositional logic), whereas '$\mathsf{Trueprop}(A \wedge B) \Longrightarrow \mathsf{Trueprop}(B)$', '$\Longrightarrow$', '$\mathsf{Trueprop}(A \wedge B)$' and '$\mathsf{Trueprop}(B)$' are terms in the meta-language. '$\mathsf{Trueprop}$' is a truth judgement in the meta-language; it maps propositions in the object language (such as HOL) to propositions in the meta-language (Isabelle). Semantically, in HOL propositions take Boolean values. Isabelle is a minimal logic, and its propositions can be interpreted as witnesses to the inhabitation of simple types [11].

Isabelle interprets three primitive constants: '$\equiv$' denotes identity of propositions up to $\alpha\beta\eta$-equivalence, '$\Longrightarrow$' is an implication between propositions, and '$\bigwedge$' is a binder denoting generalisation: it maps a propositional function into a proposition.

It is useful to distinguish the kinds of symbols which appear in Isabelle encodings, together with the typographical conventions we will use to help distinguish them:

Variables $x, y, \ldots$ are symbols which are bound by an object-level binder, and whose scope is determined by the binding.

Abstractions $\lambda x.\ T$ (for T a meta-variable ranging over terms) express λ-functions.

Constants $\mathsf{c}, \mathsf{d}, \ldots$ denote a fixed and arbitrary value. Constant symbols are drawn from a countable signature Σ and are globally scoped (i.e., their scope extends throughout the theory). We write constants (or symbols which are regarded to be constants at the object-level, as described next) in sans.

Logical variables $\hat{x}_{()}, \hat{y}_{(\mathsf{c},\mathsf{d})}, \ldots$ are meta-level unbound variables which denote a non-fixed and arbitrary value. Their scope extends to the end of a proof. Despite being unbound, these variables are conscious to the scopes of enclosing $\bigwedge$-bindings. For instance, in a theory whose signature is Σ, the logical variable $\hat{x}_{()}$ may only be instantiated for a λ-term whose constants are in Σ. On the other hand if the logical variable $\hat{y}$ arises in a context where variables (say c and d) are bound at the meta-level using '$\bigwedge$', then the logical variable is expressed as $\hat{y}_{(\mathsf{c},\mathsf{d})}$, and it may be instantiated for any λ-term whose constants are in $\Sigma \cup \{\mathsf{c}, \mathsf{d}\}$. In other words, the values of logical variables may have an extended range. Careful handling of instantiation is important for soundness and completeness, as discussed in §B.1.

In a local sense — that is, within a formula — $\bigwedge$-bound variables can be regarded to be constants, and '$\bigwedge$' can be used to formalise a local extension of the signature. It was used by Paulson to encode Eigenvariable conditions [14]. Here it serves essentially the same purpose: encoding a name-freshness side-condition wrt the original signature.

Semantically, '$\bigwedge$' names an arbitrary element of the universe. It also asserts that a corresponding element in the domain must exist, but this is acceptable since, in our object logic, domains are non-empty.

For a $\bigwedge$-prefixed statement to be valid, the statement's body must be satisfied by any such named element. This describes a universal quantifier; indeed, for HOL's '$\forall$' combinator it's possible to prove that $\forall P \equiv \bigwedge P$ for any HOL predicate P. (Recall that in HOL, quantifiers are higher-order functions, and we can write $\forall P$, $\forall(\lambda x.Px)$, and $\forall x.Px$ interchangeably since HOL validates η-equivalence.)

At the top-level of a term, there is no such thing as free variables. The object-predicate P in the previous meta-formula is bound in neither the object-level nor the meta-level, but the binding does takes place, in the meta-meta-language, in the phrase that follows it. In Isabelle/HOL, free object variables are implicitly bound by '$\bigwedge$' at the meta-level.

Since we will use Isabelle-style notation to make meta-inference and meta-binding explicit, we will extend Isabelle's notation with a meta-existential binder, denoted by '$\bigvee$'. This will be used in consistency-preserving calculi (such as that of Leo-II) but not in validity-preserving calculi (such as that of Isabelle/HOL).

Following Isabelle's non-classical character, we give an intuitionistic semantics to '$\bigvee x.P$': it denotes a pair $(d, \pi^{P(d)})$ consisting of a witness d and a proof π that $P(d) = \mathsf{True}$.

However, this notation will be used only to make meta-level reasoning more precise, and will not be interpreted by Isabelle. We will never use '$\bigwedge$' and '$\bigvee$' in the same formula.

Definition 1 (closed, closed modulo, scoped constants, signature extension). *A formula is* closed *if all its object variables are object-level bound. A formula is* closed modulo *a* signature extension *if the object-level free variables are bound by $\bigwedge$ or $\bigvee$ at the meta-level. The set of $\bigwedge$ or $\bigvee$-bound symbols which are in scope in a subterm is called the* signature extension *relative to which that subterm is interpreted. The object-level variables which are bound at the meta-level are called* scoped constants.

Notation 2. *We write $F_1[F_2]$, where $F_{i\in\{1,2\}}$ range over (object-logic) formulas, to suggest that F_2 occurs in F_1. If F_2 is an object-level variable then it means that $F_1[F_2]$ is not closed.*

From here onward whenever we speak of a rule $\frac{A_1 \ldots A_n}{A_{n+1}}$ in a consistency-preserving calculus, we implicitly include the following variable-closure constraints:

$$\bigvee \boldsymbol{\kappa}\ \frac{A_1 \qquad \ldots \qquad A_n}{\bigvee \boldsymbol{\kappa'} A_{n+1}}$$

That is, $A_1, \ldots, A_{n+1}$ are in scope of $\bigvee \boldsymbol{\kappa}$, and A_{n+1} is also in scope of $\bigvee \boldsymbol{\kappa'}$.

2.3 Henkin semantics

We will rely on the semantical framework described by Henkin for higher-order logic, as described and developed by Benzmüller et al [12]. In particular, we will work relative to the model class $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ (extensional higher-order logic without Choice). We did not handle Choice yet since, in Leo-II's calculus at least, the Choice rule simply manages the instantiation of the axiom of Choice (in such a way that avoids simulating Cut [13] during proof search). Thus, the Choice rule *enables* further inferences to take place, but does not carry out inferences itself. This behaviour could be simulated in Isabelle/HOL by instantiating a scheme which formalises the axiom of Choice, and feeding these instantiations to the counterparts of the Leo-II inferences which rely on them to eventually derive the empty clause.

3 Rule transformations

In this section we discuss a general theoretical correspondence between Leo-II proofs and Isabelle/HOL theorems, and prove that. in principle, for each Leo-II-theorem F, a Leo-II-proof of F can be reconstructed in Isabelle/HOL to show that F is an Isabelle/HOL-theorem (Corollary 8). We then discuss an approach which allows for more efficient implementation of proof reconstruction, without sacrificing completeness. In what follows we will use Isabelle as a formal meta-language, and we describe its relevant features as we proceed.

3.1 Tableau embeddings

Since LEO-II's is a refutation calculus, it proves results of the form $\models \phi \longrightarrow \mathsf{False}$. Rules in such a calculus are consistency-preserving, but not generally validity-preserving. In the results that follow, the precise rules of a calculus don't always matter — we study properties of a calculus as a whole, by relying on its soundness and completeness to assure us that any inconsistent set of formulas can be refuted.

In what follows let $\mathfrak{C}$ represent an arbitrary sound-and-complete refutation calculus whose inference rules can be adequately formalised as Isabelle terms (that is, their side-conditions are limited to variable conditions).[3] Let $\mathfrak{V}$ represent a calculus whose rules preserve validity, and which can be formalised as Isabelle terms; Isabelle/HOL is such as calculus. We take soundness and completeness to be relative to extensional Henkin models, but several of the properties described below hold in a broad class of classical logics.

We start with a basic lemma which establishes a correspondence between two families of calculi.

Lemma 3 (Refutation correspondence). *For all sets of sentences Φ, $\Phi \vdash_{\mathfrak{C}} \Box$ iff $\vdash_{\mathfrak{V}} \Phi \longrightarrow \mathsf{False}$.*

Proof sketch: The semantic expansions of $\Phi \vdash_{\mathfrak{C}} \Box$ and $\vdash_{\mathfrak{V}} \Phi \longrightarrow \mathsf{False}$ are identical.
(Proof in §A.2) $\lhd$

To reconstruct LEO-II proofs we will only use the 'fi' direction of Lemma 3. But the lemma is not directly usable, since it does not show us how to transform a proof from $\vdash_{\mathfrak{C}}$ to $\vdash_{\mathfrak{V}}$. We now define a transformation for mapping inferences from $\vdash_{\mathfrak{C}}$ to $\vdash_{\mathfrak{V}}$. It will be used to map proofs between the two systems. Intuitively, the transformation formalises the consistency-preservation property of a $\vdash_{\mathfrak{C}}$-rule $\frac{A_1 \ldots A_n}{B}$ by producing another rule which says: if there is no model satisfying $A_1, \ldots, A_n, B$ then there is no model satisfying $A_1, \ldots, A_n$; moreover, this latter rule is admissible in $\vdash_{\mathfrak{V}}$ (i.e., it is validity-preserving). This is also the informal argument of Lemma 5, which validates this transformation.

Definition 4 (Contranegation). *Given a rule r: $\bigvee \boldsymbol{\kappa} \dfrac{A_1 \qquad \ldots \qquad A_n}{\bigvee \boldsymbol{\kappa}' B}$ where $\bigvee \boldsymbol{\kappa}, \boldsymbol{\kappa}' B$ is closed modulo, we transform it into $r^{\perp}$:*

[3] This constraint may exclude exotic formalisations, but it is weak enough to include LEO-II's calculus.

$$\bigwedge \boldsymbol{\kappa}\ \dfrac{A_1 \qquad \cdots \qquad A_n \qquad \bigwedge \boldsymbol{\kappa},\boldsymbol{\kappa'} \begin{bmatrix} A_1,\ldots,A_n,B \\ \vdots \\ \mathsf{False}\end{bmatrix}}{\mathsf{False}}$$

In Definition 4, the result $r^{\perp}$ is closed modulo. Also, since '$\bigwedge$' is a binder, nested bindings can always be renamed to avoid variable capture. Thus we can always ensure that $\Sigma \cap \kappa = \emptyset$, $\Sigma \cap \kappa' = \emptyset$, and $\kappa \cap \kappa' = \emptyset$.

Lemma 5. *Assuming $\vdash_{\mathfrak{C}}$ to be sound, and $\vdash_{\mathfrak{V}}$ to be complete, then for each inference $r \in \vdash_{\mathfrak{C}}$, r is consistency-preserving iff $r^{\perp}$ is validity-preserving.*

Proof sketch: We use the soundness of r to derive $r^{\perp}$ from r. Through semantic reasoning, the two rules can be shown to be equivalent within a refutation setting. (Proof in §A.3) $\lhd$

Notation 6. *The symbol π will range over proofs. We write $\pi \in \vdash_{\mathfrak{C}}$ if π is encoded in a consistency-preserving calculus. Proofs consist of chains of instances of inference rules $(r_1, \ldots, r_n)$ in that calculus.*

We write '$A \vdash_{\mathcal{C},\pi} B$' to mean that from A we can derive B using calculus $\vdash_{\mathcal{C}}$, and that π is such a derivation.

We can now prove a constructive version of Lemma 3. A proof π can be mapped into $\pi^{\perp}$ by mapping each rule r element-wise to $r^{\perp}$.

Lemma 7 (Refutation correspondence – constructive). *For every proof $\pi \in \vdash_{\mathfrak{C}}$, and for every Φ, $\Phi \vdash_{\mathfrak{C},\pi} \Box$ iff $\vdash_{\mathfrak{V},\pi^{\perp}} \Phi \longrightarrow \mathsf{False}$.*

Proof sketch: Induction on derivations π.
(Proof in §A.4) $\lhd$

In principle then we should be able to map any proof from LEO-II's calculus into that of Isabelle/HOL.

Corollary 8. *If* LEO-II *is sound and Isabelle/HOL is complete, then every theorem proved by* LEO-II *is also a theorem of Isabelle/HOL.*

Proof sketch: The proof follows from Lemma 7, the content of which describes how to map a LEO-II proof into an Isabelle/HOL proof. We also verified this by checking that, for each of the rules in LEO-II's calculus, the transformation yields Isabelle/HOL-admissible rules. (cf §A.5) $\lhd$

Remark 9. The converse of Corollary 8 *could* be true in principle, since in (this version of) HOL, *refutation completeness* means the same thing as *completeness*.[4] However, the converse is not believed to be true in practice because of the following:

[4] The negations we prove valid are fed to a contradiction rule, which allows us to show the theorem-hood of arbitrary formulas by proving their double-negation. Use of the contradiction rule is equivalent to assuming that the stability axiom [16, §2.3] $\neg\neg A \rightarrow A$ holds for all predicates A.

1. Isabelle's calculus is polymorphic, while Leo-II's is monomorphic, and we don't know of a complete encoding from the former into the latter. We could restrict the translation to monomorphic Isabelle.
2. It is not known if Isabelle/HOL's theory is stronger than the facts which are Henkin valid. To deal with this, we could make sure to restrict the translation to Isabelle/HOL theorems which are purely derived in HOL (by filtering away those Isabelle/HOL theorems which rely on non-HOL axioms). More generally, we could encode the definitions from arbitrary Isabelle/HOL theories into Leo-II, which should make the two systems equistrong.
3. Leo-II's calculus (never mind the implementation) has not been proved to be Henkin-complete. For that matter, Isabelle/HOL has not been proved complete either, but the HOL calculus on which Isabelle/HOL is based [17] has been proved sound and complete [18].

Structurally, Definition 4 maps $\vdash_{\mathfrak{C}}$-style systems into left-handed free-variable tableau systems, the inferences of which can be admitted in $\vdash_{\mathfrak{W}}$-style systems. In our setting, the tableau system is shallowly embedded in Isabelle/HOL. The reconstructed fragment of the search space forms a tableau branch; this branches further when disjunctions are processed, with each disjunct forming a new branch. The preceding elements of the branch are shared by all subsequent branches.

Remark 10. When we reconstruct a $\vdash_{\mathfrak{C}}$-refutation into a $\vdash_{\mathfrak{W}}$-proof via Definition 4 and Lemma 7, we are being less efficient than when we would use a tableau calculus directly, because the latter deletes redundant formulas from the branches. Specifically, when an α-rule is applied the original conjunction is deleted. (Note that we only delete a formula when a logically equivalent set of formulas is added to the branch; anything weaker would generally introduce incompleteness.) It should be possible to modify Definition 4, and subsequently Lemma 7, to allow this deletion. This is a tiny optimisation which makes the reconstruction behave like a tableaux proof, but we do not pursue this here.

3.2 Isabelle encoding of tableau embedding

We now look at some rules from Leo-II's calculus, and the formalisation of their contranegation encoding in Isabelle/HOL as meta-theorems. Meta-theorems are essentially admissible rules. We have checked that the contranegation $r^{\perp}$ of each Leo-II rule r is valid in Isabelle/HOL. This is described in Lemma 15 and the Isabelle/HOL script is included in §E.2. Furthermore, in §D.1 we provide an example a proof which is translated using this method. We also provide a detailed description of the role of logical variables in this calculus in §B.

Let $\mathbf{C}, \mathbf{D}$ be meta-variables ranging over Leo-II clauses, and A, B be meta-variables ranging over formulas. Then Leo-II's resolution rule

$$\frac{\mathbf{C} \vee [A]^{\mathsf{True}} \qquad \mathbf{D} \vee [B]^{\mathsf{False}}}{\mathbf{C} \vee \mathbf{D} \vee [A = B]^{\mathsf{False}}}$$

whose transformed (Definition 4) form is

$$\dfrac{\mathbf{C} \vee [A]^{\mathsf{True}} \qquad \mathbf{D} \vee [B]^{\mathsf{False}} \qquad \begin{array}{c} \mathbf{C} \vee [A]^{\mathsf{True}} \quad \mathbf{D} \vee [B]^{\mathsf{False}} \quad \mathbf{C} \vee \mathbf{D} \vee [A = B]^{\mathsf{False}} \\ \vdots \\ \mathsf{False} \end{array}}{\mathsf{False}}$$

can be encoded in Isabelle/HOL as

$$\left[\begin{array}{l} \mathbf{C} \vee [A]^{\mathsf{True}}; \\ \mathbf{D} \vee [B]^{\mathsf{False}}; \\ \left[\begin{array}{l} \mathbf{C} \vee [A]^{\mathsf{True}}; \\ \mathbf{D} \vee [B]^{\mathsf{False}}; \\ \mathbf{C} \vee \mathbf{D} \vee [A = B]^{\mathsf{False}} \end{array}\right] \Longrightarrow \mathsf{False} \end{array}\right] \Longrightarrow \mathsf{False}$$

where LEO-II literals $[A]^p$ (where $p \in \{\mathsf{True}, \mathsf{False}\}$) are encoded in Isabelle/HOL as $A = p$. LEO-II clauses are encoded as disjunctive HOL formulas. In Isabelle notation, ';' acts as a conjunction between hypothesis: $[A; B] \Longrightarrow C$ abbreviates $A \Longrightarrow B \Longrightarrow C$.

The LEO-II Skolemisation rule

$$\frac{\mathbf{C} \vee [\forall P]^{\mathsf{False}}}{\mathbf{C} \vee [P\mathsf{sk}]^{\mathsf{False}}} \ \mathsf{sk} \text{ fresh constant}$$

and which can be rewritten to encode the side-condition in the meta-language

$$\frac{\mathbf{C} \vee [\forall P]^{\mathsf{False}}}{\bigvee \mathsf{sk}.\ \mathbf{C} \vee [P\mathsf{sk}]^{\mathsf{False}}}$$

and whose transformed (Definition 4) form is

$$\dfrac{\mathbf{C} \vee [\forall P]^{\mathsf{False}} \qquad \bigwedge \mathsf{sk} \left[\begin{array}{c} \mathbf{C} \vee [\forall P]^{\mathsf{False}} \quad \mathbf{C} \vee [P\mathsf{sk}]^{\mathsf{False}} \\ \vdots \\ \mathsf{False} \end{array}\right]}{\mathsf{False}}$$

can be encoded as

$$\left[\begin{array}{l} \mathbf{C} \vee [\forall P]^{\mathsf{False}}; \\ \bigwedge \mathsf{sk} \left[\begin{array}{l} \mathbf{C} \vee [\forall P]^{\mathsf{False}}; \\ \mathbf{C} \vee [P\mathsf{sk}]^{\mathsf{False}} \end{array}\right] \Longrightarrow \mathsf{False} \end{array}\right] \Longrightarrow \mathsf{False}$$

Note that the $\bigvee$-bound sk in the $\vdash_{\mathfrak{C}}$-style calculus becomes $\bigwedge$-bound when mapped into the $\vdash_{\mathfrak{Y}}$ calculus. This is detailed in the proof of Lemma 5, but intuitively, the $\bigvee$-bound sk can denote any element of the domain which can satisfy the formula, whereas the $\bigwedge$-bound sk is used in a setting where *any* denotation of sk will ultimately result in a refutation.

And finally, the LEO-II generalisation rule

$$\frac{\mathbf{C} \vee [\forall P]^{\mathsf{True}}}{\mathbf{C} \vee [PX]^{\mathsf{True}}}\ X \text{ fresh variable}$$

and whose transformed (Definition 4) form is

$$\dfrac{\mathbf{C} \vee [\forall P]^{\mathsf{True}} \qquad \begin{array}{c}\mathbf{C} \vee [P\ X]^{\mathsf{True}} \\ \vdots \\ \mathsf{False}\end{array}}{\mathsf{False}}$$

can be encoded as

$$\left[\begin{array}{l}\mathbf{C} \vee [\forall P]^{\mathsf{True}}; \\ \left[\begin{array}{l}\mathbf{C} \vee [\forall P]^{\mathsf{True}}; \\ \mathbf{C} \vee [PX]^{\mathsf{True}}\end{array}\right] \Longrightarrow \mathsf{False}\end{array}\right] \Longrightarrow \mathsf{False}$$

3.3 Implicative embeddings

The technique described above suffers in practice because it does not delete subsumed hypotheses. In addition to space complexity, there is some implementational complexity incurred when implementing the previous technique in Isabelle. Normally, automatic provers use data structures allowing them to address specific clauses; but Isabelle natively only allows the addressing of subgoals — not hypotheses. This means that picking which clauses to apply an inference to is not a simple matter: we might need to rotate the clauses making up the hypotheses, or else backtrack through a space, since we cannot 'point' to the relevant clauses. Having fewer items in the set of hypotheses makes it easier to manage that set.

Using the embedding we have studied so far would produce rather bulky reconstructions. While it would be convenient to apply the transformation in Definition 4 uniformly to obtain a reconstructed proof, this wouldn't work very well in practice.

Definition 4 is particularly useful when most of the rules $r \in \vdash_{\mathfrak{C}}$ are not validity-preserving. However, inspecting LEO-II's calculus reveals that only two rules aren't validity preserving: Skolemisation and splitting.

If a $r \in \vdash_{\mathfrak{C}}$ is validity preserving, then it should be admissible in $\vdash_{\mathfrak{Y}}$. We should be able to encode every such rule directly in Isabelle/HOL, and prove it to be a meta-theorem. We call such an embedding *implicative*.

Compared to the tableau embedding (§3.1), using such an embedding offers appealing features:

– We can formalise each r as a scheme, validate it in Isabelle/HOL (proving it to be a meta-theorem), then instantiate it to obtain the precise instances of r used by LEO-II. Clauses in LEO-II are ground, therefore we don't have free universal variables (but we may have Skolem constants) at the level of inferences.

- Since we don't have free universal variables, we can avoid using logical variables when chaining reconstructed inferences. This is desirable since handling such variables requires us to rely on unification, which is not easy to control or tune. Moreover, the scope of such variables spans across branches, creating dependencies.
- All inference rules can be reconstructed separately in isolation, then backchained together. Indeed, the reconstruction of separate inferences can be parallelised, since (as mentioned above) they are independent from one another. Since we are reconstructing each inference separately, we get a local reasoning scope for each rule, and can write specialised code to target each rule, without fearing interference from ongoing reconstruction in other parts of the proof.
- This backchaining-based approach lends itself well for implementation over Isabelle's core inference engine, which works through resolution. In principle Isabelle's calculus is a constructive sequent calculus, but its reasoning engine, for efficiency reasons, mainly relies on a function implementing a derived resolution rule, rather than on the primitive rules of the Isabelle calculus [14].

This approach is clearly desirable to the tableau embedding, but it is also clearly not immediately and universally applicable to all of LEO-II's rules. We will now describe how to handle Skolemisation in order to implicatively embed such inferences. We describe the handling of splitting in the next section.

Implicative Skolemisation. Instead of assigning a fresh constant to a witness, Hilbert's rule describes the witness using the information at hand (i.e., that such a witness, if it exists, must satisfy a specific predicate):

$$\frac{\exists P}{P(\varepsilon P)}$$

Hilbert's rule can be used to simulate Skolemisation by validating the following scheme for a faux Skolem constant c (that is, c is simply a theory-scoped constant):

$$\frac{\exists P \qquad \mathsf{c} = \varepsilon P}{P\mathsf{c}}$$

To use this, however, we would need to have Skolem equations (such as $\mathsf{c} = \varepsilon P$ above). LEO-II doesn't provide these definitions, but we can discover them during a pre-processing phase.

Since we are emulating Skolemisation, the constants must be kept in the context of clause-level universally-quantified variables in order to ensure soundness. Indeed, simulating LEO-II's inference requires this.

Remark 11. Skolem constants are not adequately distinguished in higher-order logic proofs encoded in TPTP. In such encodings, a Skolem constant c appears as a normal constant. Such a constant's scope extends throughout the theory. This is unnecessary: a Skolem constant should be proof-scoped.

3.4 Combining embeddings for Leo-II proof reconstruction

In the previous section we have seen how we can directly interpret all but one of LEO-II's inferences in Isabelle/HOL, allowing us to use an implicative embedding (§3.3) for those inferences.

The remaining rule is splitting; we use a tableau (§3.1) embedding for splitting, and obtain a hybrid embedding technique which allows arbitrary LEO-II proofs to be mapped to Isabelle/HOL theorems. In fact, the simplified counterpart to splitting turns out to be the $\vee E$ (Disjunction Elimination) rule, which is encoded in Isabelle/HOL as

$$\begin{bmatrix} \mathbf{C} \vee \mathbf{D}; \\ [\mathbf{C}] \Longrightarrow \mathsf{False}; \\ [\mathbf{D}] \Longrightarrow \mathsf{False} \end{bmatrix} \Longrightarrow \mathsf{False}$$

By using the validity-preserving counterparts to the LEO-II rules, we can backward-chain through the inferences. This is a style of reasoning which Isabelle can handle very well.

Adequacy. Soundness is assured since (i) the rules which are implicatively embedded are validity-preserving, and (ii) the $\vee E$ rule used for splitting is sound. Completeness is assured since every LEO-II inference — be it validity-preserving or not — can be interpreted as an Isabelle/HOL meta-theorem, via the embeddings described above. Thus we are still assured that every LEO-II theorem is a theorem of Isabelle/HOL.

4 Implementation

We used the Isabelle-integrated TPTP parser and interpreter developed during previous work [1]. It is convenient that both TPTP-encoded problems *and* proofs share the same syntax; in TPTP the two kinds of formal objects differ mainly in the annotations given to formulas. These annotations contain information related to the inference made by a theorem-prover.

When a LEO-II proof is loaded, it goes through four proof transformations:

1. Eliminating redundant parts of the proof — occasionally LEO-II includes redundant chains of inferences which do not contribute to the refutation.
2. Extracting subproofs related to splitting. Each subproof yields a refutation. The set of nodes generated during a subproof are disjoint from the sets from other subproofs. Each subproof is used to construct a lemma, which is fed into the $\vee E$ rule.
3. Separating instantiation from other inferences. Unfortunately LEO-II sometimes overloads inferences with instantiation. This transformation separates the two into consecutive inferences, allowing us to handle them separately.
4. Discovering Skolemisation equations (described in §3.3). This involves analysing the syntax of Skolemisation steps to produce Skolemisation equations, and adding these as axioms to the theory.

At this point we have a proof graph which has been filtered from redundant information, refined (in case inferences were overloaded with instantiation information), and segmented (to identify subproofs related to splitting). The proof graph is one of the two key pieces of information we use during reconstruction.

The other piece of information consists of a dictionary which maps vertices in the proof graph to LEO-II inferences. These inferences are interpreted as Isabelle/HOL meta-formulas, and reconstructed to give meta-theorems. We then walk the graph, applying the reconstructed inferences, to reconstruct the entire LEO-II proof into an Isabelle/HOL theorem. All of our inferences are checked by Isabelle's trusted kernel; this acts as the final arbiter of validity, and protects against any bugs in our implementation. Our code can be downloaded from `http://www.cl.cam.ac.uk/~ns441/files/lpar_src.tgz`.

update

Our prototype has the following limitations:

1. It cannot handle compound inferences — it only handles single inferences. LEO-II can batch together inferences to shorten proofs, but this makes reconstruction much harder. For instance, compound rules would make it difficult to discover Skolemisation equations. We do not know if this limitation is surmountable if we are to have feasible reconstruction complexity.
2. Currently we do not handle the rules in LEO-II's calculus which deal with the Axiom of Choice. AC-support is a new feature in LEO-II [19], but it only involves instantiating AC. Reconstructing AC-related LEO-II-inferences in Isabelle/HOL is a natural extension to our work.
3. Currently we do not handle the contributions which E makes to finding a refutation. Recall that LEO-II collaborates with E to find a refutation. Interpretation of E proofs in Isabelle/HOL is still an open problem. Some of the techniques in this article might apply, but tackling E remains future work.

5 Evaluation

We obtained a set of test proofs by running LEO-II 1.6 for 30s on all THF problems in the TPTP 5.4.0 problem set. LEO-II produced 1537 proofs. The reconstructor was run with a timeout of 10s and reconstructed 1442 (93.8%) proofs completely. The failures fell into three categories:

Syntactic. There were two such cases: one relates to an exotic node-naming feature of TPTP (which only appears in a syntax-testing problem), and the other relates to a bug in LEO-II's proof output.

Partial reconstruction. There were 12 cases of this. Half of these cases appear to be problems in our interpretation of inferences. We found that increasing the unification depth improves completeness but hurts performance. The other half of these cases appear to arise from alignment problems when a proof is translated to an intermediate language which we use to represent proofs. We feed these proofs to an implementation of an abstract proof-checking machine (which in turn relies on Isabelle's kernel). On occasion, one of the instructions is too strong, and reconstructs more than one step of the proof; this causes the subsequent reconstruction steps to fail.

Timeouts. There were 81 cases of this. The most likely cause of a timeout is problem size. All of these problems are fairly big: ordered by file-size, the first timeout occurs at the 1088th problem, and the second at the 1432nd (out of 1537).

Comparison with existing reconstruction. We compared our approach with re-finding the proofs using Metis [2] and Z3 [3]—these are the means currently used by Sledgehammer to reconstruct proofs found by externals provers. We checked if any axioms/definitions not used in a Leo-II proof could be filtered away, and ensured that Metis and Z3 did not use any additional facts from the Isabelle library of lemmas. Essentially, this test involves trying to re-find the proof in 10s (which Leo-II might have taken 30s to find), since the proof re-finding approach doesn't benefit from the structure of the Leo-II proof. As a result, Metis and Z3 reconstructed 57.3% and 68.9% of the proofs respectively.

Our results and scripts can be downloaded from `http://www.cl.cam.ac.uk/~ns441/files/lpar_results.tgz`. We used a repository version of Isabelle2013, and the experiments were done on a 1.6GHz Intel Core2 Linux box with 2GB RAM.

update

6 Related work

The methods described in §3 elaborate the discussion of Troelstra and Schwichtenberg [16], who describe translations between various calculi whose inference rules are validity-preserving, and resolution calculi. Troelstra and Schwichtenberg restrict themselves to propositional and first-order systems, and do not explore uniform tableau embeddings or their combination with other embeddings.

Techniques to encode clauses and inferences usually play to the strengths of the target system. For instance, it is preferable to avoid manipulations related to associativity and commutativity in clauses during proof reconstruction; this is achieved when using Isabelle by encoding clauses using Isabelle's sequents, thus leaving Isabelle to handle the commutativity and associativity of connectives. This technique was used by Paulson [20] to reconstruct proofs from a tableau prover that he had extended Isabelle with. A similar technique was described by Weber and Amjad [21] for encoding clauses. They carefully tune their technique to obtain good performance on Isabelle's resolution engine. Semantically, representation is logically insignificant, but in practice is matters a great deal. Weber and Amjad focus on reconstructing resolution proofs from SMT solvers, and they essentially use propositional logic, where complications related to binding and variables do not arise. These techniques seem to be related to the tableau embedding (§3.1), but they focus on encoding clauses and specific inferences, not on mapping a calculus. We are not aware of this embedding being described previously. We haven't tried to tune the tableau embedding by using the clause-encoding techniques described earlier.

The implicative approach (§3.3) has been used for reconstructing proofs too [22]. This approach is appealing because of its simplicity. It too can benefit from use of clause-encoding techniques. To our knowledge, the combination of tableau-embedding and implicative-embedding had not been explored yet.

7 Discussion

Herbrand-Skolem connection. One of the most surprising observations which arose during this work was that Skolemisation is mapped to Gentzens's Existential Elimination rule ($\exists E$) by the tableau embedding.

However there is a lingering peculiarity: the latter relies on Skolem constants rather than full Skolem terms. In the interest of soundness and completeness [23], we expect Skolemisation in higher-order logic to generate Skolem terms headed by a fresh Skolem constant having a fixed arity. In this setting, this seems to transfer over to the "universal" variables: logical variables. Here we observe the duality seen between the approaches of Skolem and Herbrand: the latter relies on the the fixation of existential variables into place-holders for arbitrary-but-fixed elements (which can be formalised as constants), and treats universal variables as witness candidates. In Isabelle, these candidates are ranged over by logical variables, which carry with them information about which existential constants are in scope.

Granularity of inference rules. We observed that proof-finding during proof-reconstruction can be very expensive. For instance, LEO-II can hide a chain of normalisation inferences (including Skolemisation) behind a single, compound inference called `extcnf_combined`. We discovered that attempting to validate this inference using the approach described in §3.3 can take time in $\mathcal{O}(n!)$ where n is the number of existentially-bound variables (cf. §C.1). Thus, having proofs composed of fine-grained inferences can be highly desirable; this relates to limitation 1 described in §4.

Currently, we treat E subproofs as compound meta-theorems proved by an oracle. In future work we would like to find ways of reconstructing (in Isabelle/HOL) proofs generated by E, and by the combination LEO-II+E.

TPTP proof format. The TPTP languages are a very useful medium for communicating problems and proofs, and a lot of care has gone into managing the growth of their specification. While working on this project we noticed that the following features might facilitate the reconstruction of TPTP proofs:

Annotating constants with their scope. Theories in TPTP are implicit, based on type and constant declarations, and axioms stated. Problems pertain to the same theory if they make, or import, the same declarations. Currently it is assumed that once a constant is declared, its scope extends throughout the theory; and proofs are seen as chains of reasoning occurring within a theory. That is, there is no encapsulation of inferences within proofs. It would be useful to limit the scope of Skolem constants to specific proofs (cf.

Remark 11), rather than not distinguish them from normal constants in a theory.

Distinguishing certain nodes in a proof. In particular, the conclusion node. There are two reasons for this:
- It provides a starting point for certain proof analyses; currently we heuristically assume that the conclusion node is the last node in a proof script, but this is not generally valid.
- This could serve as the end-of-scope marker for scoped constants (cf. Remark 11).

Currently it's possible to make prover-specific annotations in TPTP, but the structure of proofs, and the scope of symbols, seem like universal-enough concepts to deserve representation in the shared standard.

8 Conclusion

There is an increasing trend for logic tools to produce proofs which can be inspected or used by other tools, and for languages to be standardised across systems. However, there remain many obstacles to actually using proofs produced by other systems, because proof reconstruction is difficult in practice; it usually involves defining an ad hoc embedding of the source calculus into the target. The tableau embedding (§3.1) facilitates reconstruction by relying on a transformation applied uniformly to all inferences in the source calculus, to obtain admissible rules in the target calculus. In order to work, it relies on metatheoretical properties of the source and target calculi: that they are sound and complete.

We also discussed how this technique, despite the elegance of its uniformity, does not scale as well as an ad hoc embedding. We implemented and evaluated such an embedding for Leo-II in Isabelle/HOL. However, we believe that the tableau embedding can be useful for several reasons. It is much easier for building proof-reconstruction prototypes compared to using an ad hoc embedding. More importantly, calculi which make more use of consistency-preserving rules than Leo-II might be easier or computationally cheaper to model using a tableau embedding. For instance, carrying out a validity-preserving clausification directly in Isabelle/HOL would be very expensive, and E's clausification calculus makes much heavier use on consistency-preserving steps than Leo-II's, because the former relies on Tseitin-style clausification. This can be accommodated by the tableau embedding. Further work could study optimisations which could make this sort of embedding more appealing in practice as well as in theory.

References

1. Sultana, N., Blanchette, J., Paulson, L.: LEO-II and Satallax on the Sledgehammer test bench. Journal of Applied Logic **11**(1) (2013) 91–102

2. Hurd, J.: First-order proof tactics in higher-order logic theorem provers. In Archer, M., Vito, B.D., Muñoz, C., eds.: Proceedings of STRATA 2003. Number NASA/CP-2003-212448 in NASA Technical Reports (2003) 56–68
3. de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Proceedings of TACAS 2008. Volume 4963 of LNCS. Springer (2008) 337–340
4. Benzmüller, C., Theiss, F., Paulson, L., Fietzke, A.: LEO-II - a cooperative automatic theorem prover for higher-order logic. In: Proceedings of IJCAR 2008. Volume 5195 of LNCS., Springer (2008) 162–170
5. Schulz, S.: E – A Brainiac Theorem Prover. AI Commun. **15**(2/3) (2002) 111–126
6. Sutcliffe, G.: The TPTP problem library and associated infrastructure. Journal of Automated Reasoning **43**(4) (2009) 337–362
7. Sultana, N., Benzmüller, C.: Understanding LEO-II's proofs. In Ternovska, E., Korovin, K., Schulz, S., eds.: Proceedings of IWIL 2012. (2012)
8. Paulson, L.: Isabelle - A Generic Theorem Prover (with a contribution by T. Nipkow). Volume 828 of LNCS. Springer (1994)
9. Nipkow, T., Paulson, L., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic. Volume 2283 of LNCS. Springer (2002)
10. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic **5**(2) (1940) 56–68
11. Berghofer, S.: Proofs, Programs and Executable Specifications in Higher Order Logic. PhD thesis, Technical University Munich (2003)
12. Benzmüller, C., Brown, C., Kohlhase, M.: Higher-order semantics and extensionality. Journal of Symbolic Logic **69**(4) (2004) 1027–1088
13. Benzmüller, C., Brown, C., Kohlhase, M.: Cut-simulation and impredicativity. Logical Methods in Computer Science **5**(1:6) (2009) 1–21
14. Paulson, L.C.: The foundation of a generic theorem prover. Journal of Automated Reasoning **5**(3) (1989) 363–397
15. Miller, D.: Unification under a mixed prefix. Journal of Symbolic Computation **14**(4) (1992) 321–358
16. Troelstra, A.S., Schwichtenberg, H.: Basic proof theory. Number 43. Cambridge University Press (2000)
17. Gordon, M.: HOL : A machine oriented formulation of higher order logic. Technical Report UCAM-CL-TR-68, University of Cambridge, Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, UK (1985)
18. Gordon, M.J., Melham, T.F.: Introduction to HOL: a theorem proving environment for higher order logic. Cambridge University Press (1993)
19. Benzmüller, C., Sultana, N.: LEO-II version 1.5. In: Proceedings of PxTP 2013. Volume 14 of EPiC Series., EasyChair (2013) 2–10
20. Paulson, L.C.: Generic automatic proof tools. In Veroff, R., ed.: Automated Reasoning and its Applications: Essays in Honor of Larry Wos, MIT Press (1997)
21. Weber, T., Amjad, H.: Efficiently checking propositional refutations in HOL theorem provers. Journal of Applied Logic **7**(1) (2009) 26 – 40
22. Böhme, S.: Proving Theorems of Higher-Order Logic with SMT Solvers. PhD thesis, Technical University Munich (2012)
23. Miller, D.: Proofs in Higher-Order Logic. PhD thesis, Carnegie Mellon University (1983)

A Proofs

A.1 Lemma 12

In the proof of Lemma 5, given in §A.3, we will rely on the following lemma about signature extension.

Lemma 12. *For any Σ-model $\mathcal{M} = (\mathcal{D}, @, \mathcal{E}, v)$ and Σ'-model $\mathcal{M}' = (\mathcal{D}, @, \mathcal{E}', v)$ where $\Sigma \subseteq \Sigma'$, and where $\mathcal{E}_\varphi(\mathsf{c}) = \mathcal{E}'_\varphi(\mathsf{c})$ for all $\mathsf{c} \in \Sigma$, the following property holds: For any formula F with constants only in Σ, $\mathcal{M} \models F$ iff $\mathcal{M}' \models F$.*

Proof. We are to show that $v(\mathcal{E}_\varphi(F)) \equiv v(\mathcal{E}'_\varphi(F))$ for all φ, if $F \in \mathrm{wff}(\Sigma)$. We prove this by showing that $\mathcal{E}_\varphi(F) = \mathcal{E}'_\varphi(F)$ for all φ, if $F \in \mathrm{wff}(\Sigma)$.

Proceed by induction on F:

- Case F is a variable x: Using definition of evaluation functions [12, Remark 3.18], we have that $\mathcal{E}_\varphi(x) = \varphi(x) = \mathcal{E}'_\varphi(x)$.
- Case F is a constant c: Since $\mathsf{c} \in \mathrm{wff}(\Sigma)$, then it's necessary that $\mathsf{c} \in \Sigma$. By assumption we have $\mathcal{E}_\varphi(\mathsf{c}) = \mathcal{E}'_\varphi(\mathsf{c})$ for all $\mathsf{c} \in \Sigma$.
- Case F is an application $t_1 t_2$: $\mathcal{E}_\varphi(t_1 t_2) = \mathcal{E}_\varphi(t_1) @ \mathcal{E}_\varphi(t_2) = \mathcal{E}'_\varphi(t_1) @ \mathcal{E}'_\varphi(t_2) = \mathcal{E}'_\varphi(t_1 t_2)$ using IH.
- Case F is an abstraction $\lambda x.\ t$: IH gives us that $\mathcal{E}_{\varphi,[\mathsf{a}/x]}(t) = \mathcal{E}'_{\varphi,[\mathsf{a}/x]}(t)$ for each $\mathsf{a} \in \mathcal{D}$. Using the IH, for arbitrary a we have $\mathcal{E}_\varphi(\lambda x.\ t) @ \mathsf{a} = \mathcal{E}_{\varphi,[\mathsf{a}/x]}(t) = \mathcal{E}'_{\varphi,[\mathsf{a}/x]}(t) = \mathcal{E}'_\varphi(\lambda x.\ t) @ \mathsf{a}$. Therefore $\mathcal{E}_\varphi(\lambda x.\ t) = \mathcal{E}'_\varphi(\lambda x.\ t)$ by extensionality. □

A.2 Lemma 3

Proof. $\Phi \vdash_{\mathfrak{C}} \Box$ iff there is no Henkin model $\mathcal{M}$ such that $\mathcal{M} \models \Phi$. Recall that $\mathcal{M}$ would consist of a tuple $(\mathcal{D}, @, \mathcal{E}, v)$, and that $\mathcal{M} \models \Phi$ iff for all $\phi \in \Phi$, $v(\mathcal{E}(\phi)) \equiv \top$. (Since each $\phi \in \Phi$ is closed, we need not worry about variable assignments.)

Equivalently, for all Henkin models, there is a $\phi \in \Phi$ such that $v(\mathcal{E}(\phi)) \equiv \bot$. This is equivalent to saying that, for all Henkin models, $v(\mathcal{E}(\phi_1 \wedge \ldots \wedge \phi_n)) \equiv \bot$, where $\{\phi_1, \ldots, \phi_n\} = \Phi$, and where we use the usual interpretation of '$\wedge$' [12, Figure 2]. We will continue to use the symbol Φ to abbreviate its conjunctive form $\phi_1 \wedge \ldots \wedge \phi_n$.

This is equivalent to saying that, for all Henkin models, $v(\mathcal{E}(\Phi \longrightarrow \mathsf{False}) \equiv \top$ (by using the usual interpretation of '$\longrightarrow$' and 'False'). That is, $\models \Phi \longrightarrow \mathsf{False}$

From the assumption that $\vdash_{\mathfrak{Y}}$ is complete, we have $\vdash_{\mathfrak{Y}} \Phi \longrightarrow \mathsf{False}$. □

A.3 Lemma 5

Proof. From the soundness proof of $\vdash_{\mathfrak{C}}$ we have that for each rule $r \in \vdash_{\mathfrak{C}}$, where r is the tuple of (possibly several) hypotheses and a (single) conclusion, (H, C), if there exists a model $\mathcal{M}$ such that $\mathcal{M} \models H$ then there exists a model $\mathcal{M}'$ such that $\mathcal{M}' \models H$ and $\mathcal{M}' \models C$.

Note that the interpretation of free top-level variables (occurring in H and C) by $\mathcal{M}'$ is unconstrained; therefore $\mathcal{M}'$ must satisfy H and C for at least one possible instantiation of their free variables. For clarity, let us make free top-level variables explicit through Notation 2.[5] Then the statement of r's soundness becomes: if there exists a model $\mathcal{M}$ such that $\mathcal{M} \models H[\boldsymbol{x}]$ for some $\boldsymbol{x}$, then there exists a model $\mathcal{M}'$ such that $\mathcal{M}' \models H[\boldsymbol{x}]$ and $\mathcal{M}' \models C[\boldsymbol{y}]$ for some $\boldsymbol{x}, \boldsymbol{y}$.

Taking the contrapositive, this is equivalent to the statement: if there doesn't exist a model $\mathcal{M}'$ such that $\mathcal{M}' \models H[\boldsymbol{x}]$ and $\mathcal{M}' \models C[\boldsymbol{y}]$ for some $\boldsymbol{x}, \boldsymbol{y}$, then there doesn't exist a model $\mathcal{M}$ such that $\mathcal{M} \models H[\boldsymbol{x}]$ for some $\boldsymbol{x}$.

Recall that, for all ϕ, we have that $\models \neg\phi$ iff there doesn't exist a model $\mathcal{M}$ such that $\mathcal{M} \models \phi$. Also, for all ϕ_1, ϕ_2, we have that $\mathcal{M} \models \phi_1 \wedge \phi_2$ iff $\mathcal{M} \models \phi_1$ and $\mathcal{M} \models \phi_2$. By using these equivalences, and pushing in the negation, we can rewrite the previous statement as: if $\models \neg(H[\boldsymbol{x}] \wedge C[\boldsymbol{y}])$ then $\models \neg H[\boldsymbol{x}]$. This should hold for arbitrary $\boldsymbol{x}$ and $\boldsymbol{x}, \boldsymbol{y}$, because of pushing-in the negation. Note that the antecedent and consequent occurrences of $\boldsymbol{x}$ occur in different scopes; we make this explicit by using the $\bigwedge$ binder when we switch to using Isabelle notation. Using $\bigwedge$ also makes explicit that the formulas are expected to hold for *all* $\boldsymbol{x}$ and $\boldsymbol{x}, \boldsymbol{y}$ respectively.

Since $\vdash_{\mathfrak{Y}}$ is complete, we should be able to admit this in rule form:[6]

$$\frac{\vdash_{\mathfrak{Y}} \neg(H[\boldsymbol{x}] \wedge C[\boldsymbol{y}])}{\vdash_{\mathfrak{Y}} \neg H[\boldsymbol{x}]}$$

which in Isabelle notation is written:

$$\left(\bigwedge \boldsymbol{x}, \boldsymbol{y}.\ \neg(H[\boldsymbol{x}] \wedge C[\boldsymbol{y}])\right) \Longrightarrow \bigwedge \boldsymbol{x}.\ \neg H[\boldsymbol{x}]$$

Using the equivalence $(\neg A) = (A \longrightarrow \mathsf{False})$, followed by implication-introduction and simplification we get the equivalent rule

$$\left(\bigwedge \boldsymbol{x}, \boldsymbol{y}.\ \neg(H[\boldsymbol{x}] \wedge C[\boldsymbol{y}])\right) \Longrightarrow \bigwedge \boldsymbol{x}.\ H[\boldsymbol{x}] \Longrightarrow \mathsf{False}$$

By re-using the previous equivalence, together with the equivalence $(\longrightarrow) \equiv (\Longrightarrow)$, we obtain

$$\left(\bigwedge \boldsymbol{x}, \boldsymbol{y}.\ \begin{bmatrix} H[\boldsymbol{x}] \wedge C[\boldsymbol{y}] \\ \vdots \\ \mathsf{False} \end{bmatrix}\right) \Longrightarrow \bigwedge \boldsymbol{x}.\ H[\boldsymbol{x}] \Longrightarrow \mathsf{False}$$

which, when rewritten using more conventional notation, and by eliminating the top-level iterated conjunction in H (to give $H_1, \ldots, H_n$), is equivalent to

[5] Alternatively, we could have used the $\bigvee$-notation, but it is simpler to use object level unless we're using Isabelle notation. Also alternatively, we could have existentially-closed the object formula, but this would involve additional syntactic manipulation; it's best to keep things as they are as long as the semantics are understood and preserved.

[6] Even better, this rule is derivable.

$$\bigwedge \boldsymbol{x}\ \dfrac{\bigwedge \boldsymbol{x},\boldsymbol{y}\begin{bmatrix} H_1[\boldsymbol{x}],\ldots,H_n[\boldsymbol{x}],C[\boldsymbol{y}] \\ \vdots \\ \mathsf{False}\end{bmatrix} \qquad H_1[\boldsymbol{x}] \qquad \cdots \qquad H_n[\boldsymbol{x}]}{\mathsf{False}}$$

which is equivalent to $r^{\perp}$. Starting with the assumption that r is consistency-preserving we have arrived, via a series of iff-steps, at a validity-preserving rule equivalent to $r^{\perp}$. $\square$

A.4 Lemma 7

Proof. We argue by induction on derivations π.

Case $\pi = ()$**.** Note that $\pi^{\perp} = \pi = ()$. A proof π is empty if it is immediate that $\Phi \vdash_{\mathfrak{C},\pi} \square$. If a proof is empty, then it must be necessary that $\mathsf{False} \in \Phi$. The argument is by contradiction: assume that $\Phi \vdash_{\mathfrak{C}} \square$ can be shown in 0 steps, and assume, for contradiction, that $\mathsf{False} \notin \Phi$. Then it could not be true that $\Phi \vdash_{\mathfrak{C}} \square$ can be shown in 0 steps. The argument for $\vdash_{\mathfrak{V}} \Phi \longrightarrow \mathsf{False}$ is similar. From the fact that $\mathsf{False} \in \Phi$, it is straightforward to prove the base case.

Case $\pi = (\pi', r)$**.** W.l.o.g let r be the rule $\frac{A_1 \ldots A_n}{B}$. Working within an extended signature $\boldsymbol{\kappa'}$, the induction hypothesis allows us to assume that this property holds for π' and $\pi'^{\perp}$:

$$\Phi \cup \{A_1,\ldots,A_m,B\} \vdash_{\mathfrak{C},\pi'} \square \quad \text{iff} \quad \vdash_{\mathfrak{V},\pi'^{\perp}} \forall \boldsymbol{\kappa'}.\ \Phi \wedge A_1 \wedge \ldots \wedge A_m \wedge B \longrightarrow \mathsf{False}$$

We are to show that:

$$\Phi \cup \{A_1,\ldots,A_m\} \vdash_{\mathfrak{C},\pi} \square \quad \text{iff} \quad \vdash_{\mathfrak{V},\pi^{\perp}} \forall \boldsymbol{\kappa}.\ \Phi \wedge A_1 \wedge \ldots \wedge A_m \longrightarrow \mathsf{False}$$

where $\boldsymbol{\kappa} \subseteq \boldsymbol{\kappa'}$.

We start with the "fi" direction. Assume $\Phi \cup \{A_1,\ldots,A_m\} \vdash_{\mathfrak{C},\pi} \square$ to show $\vdash_{\mathfrak{V},\pi^{\perp}} \forall \boldsymbol{\kappa}.\ \Phi \wedge A_1 \wedge \ldots \wedge A_m \longrightarrow \mathsf{False}$.

Note that π consists of π' followed by r, and that if $\Phi \cup \{A_1,\ldots,A_m\} \vdash_{\mathfrak{C},\pi} \square$ then $\Phi \cup \{A_1,\ldots,A_m,B\} \vdash_{\mathfrak{C},\pi'} \square$, by the soundness of r.

We can use this to infer, via the IH, that $\vdash_{\mathfrak{V},\pi'^{\perp}} \forall \boldsymbol{\kappa'}.\ \Phi \wedge A_1 \wedge \ldots \wedge A_m \wedge B \longrightarrow \mathsf{False}$. We use this in $r^{\perp}$ (which we obtain via Lemma 5, together with its admissibility in $\vdash_{\mathfrak{V}}$), and introduce connectives to derive $\vdash_{\mathfrak{V},\pi^{\perp}} \forall \boldsymbol{\kappa}.\ \Phi \wedge A_1 \wedge \ldots \wedge A_m \longrightarrow \mathsf{False}$.

For the "if" direction, assume $\vdash_{\mathfrak{V},\pi^{\perp}} \forall \boldsymbol{\kappa}.\ \Phi \wedge A_1 \wedge \ldots \wedge A_m \longrightarrow \mathsf{False}$ to show $\Phi \cup \{A_1,\ldots,A_m\} \vdash_{\mathfrak{C},\pi} \square$. As before, the proof π relies on a subproof π' which proves $\vdash_{\mathfrak{V},\pi'^{\perp}} \forall \boldsymbol{\kappa'}.\ \Phi \wedge A_1 \wedge \ldots \wedge A_m \wedge B \longrightarrow \mathsf{False}$. Using the IH, we derive $\Phi \cup \{A_1,\ldots,A_m,B\} \vdash_{\mathfrak{C},\pi'} \square$. By the soundness of r, if $S = \Phi \cup \{A_1,\ldots,A_m,B\}$ is inconsistent then so is S/B. Therefore we conclude that $\Phi \cup \{A_1,\ldots,A_m\} \vdash_{\mathfrak{C},\pi} \square$.

$\square$

A.5 Corollary 8

This corollary follows from Lemma 7, but we can improve our confidence in the argument by checking the contranegation transformation on each LEO-II rule, to ensure that the result is derivable in Isabelle/HOL.

Then this enables us to prove the corollary's statement in another way: we can then show how chaining together the contranegated LEO-II rules in Isabelle/HOL yields a valid Isabelle/HOL proof. In a sense, this instantiates the argument made in Lemma 5 (in one direction) then unfolds it in Lemma 7.

For our proofs, we will use the LEO-II calculus described in earlier work [7]. We ignore the rules which are logistic, focussing on those which relate to normalisation, extensionality, unification, and resolution.

Definition 13. *The* core rules *of* LEO-II*'s calculus are those which belong to one of the following families:* {normalisation, extensionality, unification, resolution}

Lemma 14. *For each derivation π in* LEO-II *for Φ, there exists a derivation π' in* LEO-II *for Φ' such that π' consists of inferences made up solely of core rules, and that Φ is satisfiable iff Φ' is satisfiable.*

Proof. We need to show that each problem can be replaced with an equisatisfiable problem, and that arbitrary derivations from the former can be matched by core-rule-only derivations from the latter.

A derivation isn't necessarily a refutation—it doesn't need to end in an empty clause. Therefore the final rule r of a derivation could be any rule. We proceed by cases on the terminal rules of a derivation, and assume that the property holds for the preceding steps of the derivation.

If r is a core rule then the proof is immediate. We sketch the argument for the remaining rules.

- r is `negate_conjecture`: It cannot be the case that this occurs in arbitrary points in a derivation, since `negate_conjecture` occurs only at the very start of a refutation.
- r is `polarity_switch`: Replace the original literal with its double-negation. (Do this all the way up the derivation.) This doesn't change the formula's semantics, or the problem's satisfiability. Then double each application of negation-related normalisation rules to that literal. At this point, applying a single negation-related normalisation rule allows us to get the same effect as using the `polarity_switch`.
- r is `unfold_def`: Form Φ' by unfolding the definitions in Φ. This makes Φ' and Φ equisatisfiable. Form π' by taking π and deleting the `unfold_def`-inferences in π (connecting the parent of the `unfold_def` inference node directly with the node's child.)
- r is `rename`: Simply carry out the renaming at the source node.
- r is `copy`: Delete the inference, and have its child point to the copied clause.
- r is `fo_atp_e`: We do not handle E subproofs; so in this case we either treat this as an oracle step, or exclude derivations in which E played a part.

- r is `sim`: Add the simplification formulas to Φ', and make explicit any inference related to them in π'.
- r is `extcnf_combined` or `standard_cnf`: This shouldn't happen, since we work on expanded proofs. (That is, all normalisation steps are explicit. They are core rules.)
- r is `extcnf_forall_special_pos`: Replace with `extcnf_forall_pos`, and include the resulting derivation from `extcnf_forall_pos`.
- r is `split_conjecture`: Instead of splitting a clause, combine the instantiation information from the split branches, to find the refutation for the full clause.
- r is `solved_all_splits`: This cannot happen, since occurrences of this inference will be removed when handling `split_conjecture`.

□

Lemma 14 effectively describes a proof transformation which purifies LEO-II proofs to use the core calculus. This might also involve modifying the source clause set Φ to produce an equisatisfiable set Φ'.

Lemma 15. *For each rule r in* LEO-II*, if r is a core rule then we have that $r^{\perp}$ is derivable in Isabelle/HOL.*

Proof. Proceed by cases on the core rules. These were formalised and checked in Isabelle/HOL—the script is included in §E.2. Note that:
- The decomposition rule is an n-ary rule. In our implementation (§4) we implemented this as a function which accepts a parameter $1 < n < \omega$ and produces (and validates) an n-ary decomposition rule.
- The substitution and primitive substitution rules could not be formalised since they include a meta-logical operation (substitution) which is not explicit in Isabelle. In our description we factor-out instantiation steps from the proof, and apply the instantiations using a custom tactic.

□

Lemma 16. *Every theorem proved by* LEO-II *is also a theorem of Isabelle/HOL.*

Proof. Lemma 14 gives us the license to restrict our attention to the core rules of LEO-II. From Lemma 15 we have that every core rule of LEO-II corresponds to an admissible rule in Isabelle/HOL. Then we need to map how proofs (as sequences of applications of core rules) formalised in LEO-II's calculus map into a calculus which can be embedded in Isabelle/HOL. This is described in the "fi" direction of Lemma 7. □

B Role of logical variables

In the Isabelle encoding of generalisation given in §3.2, how do we know that X is fresh? Unlike with Skolem constants, we have not encoded this in the meta-language yet. More importantly, what is the purpose of X? X is intended to be a recipient for unification; LEO-II might eventually derive a contradiction by coming up with a suitable instantiation for X. So the freshness of X is only an artefact; we actually don't need X to be a variable: X stands for any suitable term. The meaning of "any suitable term" is: any λ-term, possibly including any constants from signature-extensions which have occurred up to this point in the proof (but not any subsequent ones, as will be discussed below.)

We can do away with this by relying on *logical variables* instead. Logical variables are extra-logical devices which act as candidates for instantiation (via unification), so they fulfill the role splendidly. Isabelle will manage these variables for us. Assuming that the signature is currently extended to include c, then if we rewrite the generalisation rule from §3.2 to make explicit this signature extension, we get

$$\bigvee \mathsf{c} \; \frac{\mathbf{C} \vee [\forall P]^{\mathsf{True}}}{\mathbf{C} \vee [P \; \hat{x}_{(\mathsf{c})}]^{\mathsf{True}}}$$

and whose simplified form of the transformation (Definition 4) is

$$\bigwedge \mathsf{c} \; \frac{\mathbf{C} \vee [\forall P]^{\mathsf{True}} \qquad \begin{array}{c} \mathbf{C} \vee [P \; \hat{x}_{(\mathsf{c})}]^{\mathsf{True}} \\ \vdots \\ \mathsf{False} \end{array}}{\mathsf{False}}$$

Remark 17. "Freshness" is not a concept that applies to $\hat{x}_{(\mathsf{c})}$, since it ranges over a class of terms, and its name isn't meaningful in the object-language (since it is an extralogical symbol).

B.1 Impact on soundness and completeness

In this section we give two examples showing how the correct handling of logical variables is essential for both soundness and completeness.

Soundness. The formula $\forall x P(x, x) \longrightarrow \exists x \forall y \exists z P(x, y \; z)$ is not valid. For a counter-model, consider a model where $\mathcal{D}_\iota = \{\mathsf{a}, \mathsf{b}\}$ and $\mathcal{E}(P) = \{(\mathsf{a}, \mathsf{a}), (\mathsf{b}, \mathsf{b})\}$. Note that $\mathcal{D}_{\iota \to \iota}$ contains two constant functions, call them $\mathsf{f_a}$ and $\mathsf{f_b}$:

$$\begin{array}{c} f_a : \{(\mathsf{a}, \mathsf{a}), (\mathsf{b}, \mathsf{a})\} \\ f_b : \{(\mathsf{a}, \mathsf{b}), (\mathsf{b}, \mathsf{b})\} \end{array}$$

Clearly, '$\forall x P(x, x)$' is true in this model, but '$\exists x \forall y \exists z P(x, y \; z)$' is falsified as follows: if you pick a for x, then picking $\mathsf{f_b}$ for y excludes the possibility of finding a z which can satisfy the formula (that is, s.t. $\mathsf{f_b} \; y = \mathsf{a}$); alternatively, if you pick b for x, then picking $\mathsf{f_a}$ for y leads to the same problem.

However, this formula becomes provable if we ignore the context of logical variables. Writing '...' for unimportant parts of the search space we've already seen, and underlining the clauses which are acted upon in that step, the search space grows as follows:

1. $\left[\underline{\neg(\forall x P(x,x) \longrightarrow \exists x \forall y \exists z P(x, y\ z))}\right] \Longrightarrow \mathsf{False}$
2. $\left[\ldots; \underline{\forall x P(x,x)}; \neg(\exists x \forall y \exists z P(x, y\ z))\right] \Longrightarrow \mathsf{False}$
3. $\left[\ldots; P(\hat{x}_{()}, \hat{x}_{()}); \underline{\neg(\exists x \forall y \exists z P(x, y\ z))}\right] \Longrightarrow \mathsf{False}$
4. $\left[\ldots; P(\hat{x}_{()}, \hat{x}_{()}); \underline{\neg(\forall y \exists z P(\hat{x_2}_{()}, y\ z))}\right] \Longrightarrow \mathsf{False}$
5. $\left[\ldots; P(\hat{x}_{()}, \hat{x}_{()}); \underline{\neg(\exists z P(\hat{x_2}_{()}, \mathsf{c}\ z))}\right] \Longrightarrow \mathsf{False}$
6. $\left[\ldots; P(\hat{x}_{()}, \hat{x}_{()}); \underline{\neg(P(\hat{x_2}_{()}, \mathsf{c}\ \hat{z}_{(\mathsf{c})}))}\right] \Longrightarrow \mathsf{False}$

Note that $\hat{x}_{()}$ and $\hat{x_2}_{()}$ are distinct logical variables. We close the "proof" by resolving the two clauses, deriving an empty clause, via the following instantiations:

1. $\hat{x_2}_{()} \mapsto \mathsf{c}\ \hat{z}_{(\mathsf{c})}$
2. $\hat{x}_{()} \mapsto \mathsf{c}\ \hat{z}_{(\mathsf{c})}$

Of course, the unification should have been impossible: both $\hat{x_2}_{()}$ and $\hat{x}_{()}$ cannot be instantiated into terms which interpret the constant c.

Completeness. The formula $\forall xy \exists f. fx = fy$ is valid, but it is not provable unless logical variables carry context around with them. The "derivation" proceeds thus:

1. $\left[\underline{\neg(\forall xy \exists f. f\ x = f\ y)}\right] \Longrightarrow \mathsf{False}$
2. $\left[\ldots; \underline{\neg(\forall y \exists f. f\ \mathsf{x} = f\ y)}\right] \Longrightarrow \mathsf{False}$
3. $\left[\ldots; \underline{\neg(\exists f. f\ \mathsf{x} = f\ \mathsf{y})}\right] \Longrightarrow \mathsf{False}$
4. $\left[\ldots; \neg(\hat{f}_{()}\ \mathsf{x} = \hat{f}_{()}\ \mathsf{y})\right] \Longrightarrow \mathsf{False}$

Now we are stuck: we are constrained to instantiating $\hat{f}_{()}$ with $\lambda x x$, but that won't move the proof forward. Remember that we cannot instantiate $\hat{f}_{()}$ to x or y, since both of the latter are only contained in the extended, not original, signature.

The mistake occurred at step 4; the correct step is:

$$\left[\ldots; \neg(\hat{f}_{(\mathsf{x},\mathsf{y})}\ \mathsf{x} = \hat{f}_{(\mathsf{x},\mathsf{y})}\ \mathsf{y})\right] \Longrightarrow \mathsf{False}$$

Now, in addition to $\lambda x x$, we can instantiate $\hat{f}_{(\mathsf{x},\mathsf{y})}$ to $\lambda\_.\mathsf{x}$ or $\lambda\_.\mathsf{y}$, either of which can help us get a refutation.

Remark 18. The non-emptiness of domains still holds: we are not forbidden from unifying logical variables having identical contexts, such as $\hat{x}_{()}$ and $\hat{y}_{()}$ for example.

Remark 19. It is acceptable to deliberately lose information about the context — for instance, by mapping $\hat{x}_{(\mathsf{a})} \mapsto \hat{y}_{()}$. This acknowledges that any term we can use to instantiate $\hat{y}_{()}$ can also be used to instantiate $\hat{x}_{(\mathsf{a})}$. The opposite is not acceptable — e.g., by mapping $\hat{x}_{()} \mapsto \hat{y}_{(\mathsf{a})}$ — as it would be unsound.

C Calculations

C.1 Factorial complexity of Leo-II's extcnf_combined rule

Consider an `extcnf_combined` inference which has the following form:

$$\frac{[\forall x_1.\ P_1]^{\mathsf{False}} \vee \ldots \vee [\forall x_n.\ P_n]^{\mathsf{False}}}{\mathbf{D}}$$

where each x_i has been Skolemised in clause $\mathbf{D}$. W.l.o.g assume that there are no clause-level variables (that is, we will have Skolem constants in the conclusion, rather than (more general) Skolem terms). This means that $\mathbf{D}$ is identical to $[Q_1\ \mathsf{c}_1]^{\mathsf{False}} \vee \ldots \vee [Q_n\ \mathsf{c}_n]^{\mathsf{False}}$, where a Q_j is a normalised version of some P_i. Our task is to validate this inference:

$$\frac{[\forall x_1.\ P_1]^{\mathsf{False}} \vee \ldots \vee [\forall x_n.\ P_n]^{\mathsf{False}}}{[Q_1\ \mathsf{c}_1]^{\mathsf{False}} \vee \ldots \vee [Q_n\ \mathsf{c}_n]^{\mathsf{False}}}$$

We cannot assume that the literals have not been re-ordered. When attempting to expand this step into native inferences (e.g., Skolemisation), how do we match each c_i to the x_j for which it is a Skolem constant? In the best case, the types of each variable x_j are distinct: then we can simply perform the matching on the basis of types, and it takes time $\mathcal{O}(n)$. In the worst case, each x_j has the same type. In this case each matching is equally likely. We exclude approaches which involve looking ahead, using structural information of each P_i and each Q_j to try to infer a match, as these can be highly complex. Then we are left to try each matching, of which there are $n!$.

D Examples

D.1 $\exists A. \forall P. \exists B. (PAB \vee \neg PBA)$

We prove $\exists A. \forall P. \exists B. (PAB \vee \neg PBA)$ in a $\vdash_{\mathfrak{C}}$-style system based on Leo-II's calculus, then reconstruct it in a $\vdash_{\mathfrak{Y}}$-style system embedded in Isabelle/HOL.

1. $\{\neg(\exists A. \forall P. \exists B. (PAB \vee \neg PBA))\} \vdash_{\mathfrak{C}} \neg(\forall P. \exists B. (PX_AB \vee \neg PBX_A))$
2. $\{\ldots, \neg(\forall P. \exists B. (PX_AB \vee \neg PBX_A))\} \vdash_{\mathfrak{C}} \neg(\exists B. ((\mathsf{sk_P} X_A)X_AB \vee \neg(\mathsf{sk_P} X_A)BX_A))$
3. $\{\ldots, \neg(\exists B. ((\mathsf{sk_P} X_A)X_AB \vee \neg(\mathsf{sk_P} X_A)BX_A))\} \vdash_{\mathfrak{C}} \neg((\mathsf{sk_P} X_A)X_AX_B \vee \neg(\mathsf{sk_P} X_A)X_BX_A)$
4. $\{\ldots, \neg((\mathsf{sk_P} X_A)X_AX_B \vee \neg(\mathsf{sk_P} X_A)X_BX_A)\} \vdash_{\mathfrak{C}} \neg((\mathsf{sk_P} X_A)X_AX_B)$
5. $\{\ldots, \neg((\mathsf{sk_P} X_A)X_AX_B \vee \neg(\mathsf{sk_P} X_A)X_BX_A)\} \vdash_{\mathfrak{C}} \neg(\neg(\mathsf{sk_P} X_A)X_BX_A)$
6. $\{\ldots, \neg((\mathsf{sk_P} X_A)X_AX_B), \neg(\neg(\mathsf{sk_P} X_A)X_BX_A)\} \vdash_{\mathfrak{C}} (\mathsf{sk_P} X_A)X_BX_A$
7. $\{\ldots, \neg((\mathsf{sk_P} X_A)X_AX_B), \overline{(\mathsf{sk_P} X_A)X_BX_A\}} \vdash_{\mathfrak{C}} ((\mathsf{sk_P} X_A)X_AX_B) \neq ((\mathsf{sk_P} X_A)X_BX_A)$
8. $\{\ldots, ((\mathsf{sk_P} X_A)X_AX_B) \neq ((\mathsf{sk_P} X_A)X_BX_A)\} \vdash_{\mathfrak{C}} \Box$

In Isabelle/HOL we start the proof off by applying the proof-by-contradiction rule:

$$[[\neg P] \Longrightarrow \mathsf{False}] \Longrightarrow P$$

then proceed as follows:

1. $[\neg(\exists A.\forall P.\exists B.(PAB \vee \neg PBA))] \Longrightarrow \mathsf{False}$
2. $\Big[\ldots;\neg(\forall P.\exists B.(P\hat{A}_{()}B \vee \neg PB\hat{A}_{()}))\Big] \Longrightarrow \mathsf{False}$
3. $\Big[\ldots;\neg(\exists B.(\mathsf{P}\hat{A}_{()}B \vee \neg \mathsf{P}B\hat{A}_{()}))\Big] \Longrightarrow \mathsf{False}$
4. $\Big[\ldots;\neg(\mathsf{P}\hat{A}_{()}\hat{B}_{(\mathsf{P})} \vee \neg \mathsf{P}\hat{B}_{(\mathsf{P})}\hat{A}_{()})\Big] \Longrightarrow \mathsf{False}$
5. $\Big[\ldots;\neg(\mathsf{P}\hat{A}_{()}\hat{B}_{(\mathsf{P})});\underline{\neg(\neg \mathsf{P}\hat{B}_{(\mathsf{P})}\hat{A}_{()})}\Big] \Longrightarrow \mathsf{False}$
6. $\Big[\ldots;\neg(\mathsf{P}\hat{A}_{()}\hat{B}_{(\mathsf{P})});\mathsf{P}\hat{B}_{(\mathsf{P})}\hat{A}_{()}\Big] \Longrightarrow \mathsf{False}$
7. $\Big[\ldots;(\mathsf{P}\hat{A}_{()}\hat{B}_{(\mathsf{P})}) \neq (\mathsf{P}\hat{B}_{(\mathsf{P})})\hat{A}_{()}\Big] \Longrightarrow \mathsf{False}$
8. $[\ldots;\mathsf{False}] \Longrightarrow \mathsf{False}$

E Isabelle/HOL examples

E.1 Header

```
theory Example
imports HOL
begin
```

E.2 Calculus embedding

```
subsection "Calculus"

lemma extcnf_or_pos:
"[|C | ((A | B) = True);
   [|C | ((A | B) = True);
     C | (A = True) | (B = True)|] ==> False|] ==> False"
by auto

lemma extcnf_or_neg1:
"[|C | ((A | B) = False);
   [|C | ((A | B) = False);
     C | (A = False)|] ==> False|] ==> False"
by auto
lemma extcnf_or_neg2:
"[|C | ((A | B) = False);
   [|C | ((A | B) = False);
     C | (B = False)|] ==> False|] ==> False"
by auto

lemma extcnf_not_pos:
"[|C | ((~ A) = True);
   [|C | ((~ A) = True);
     C | (A = False)|] ==> False|] ==> False"
```

```
by auto

lemma extcnf_not_neg:
"[|C | ((~ A) = False);
   [|C | ((~ A) = False);
     C | (A = True)|] ==> False|] ==> False"
by auto

lemma extcnf_forall_pos:
"[|C | ((All A) = True);
   [|C | ((All A) = True);
     C | ((A X) = True)|] ==> False|] ==> False"
"[|C | ((Ex A) = False);
   [|C | ((Ex A) = False);
     C | ((A X) = False)|] ==> False|] ==> False"
by auto

lemma extcnf_forall_neg:
"[|C | ((All A) = False);
   !! sk. [|C | ((All A) = False);
            C | ((A sk) = False)|] ==> False|] ==> False"
by auto

lemma func_pos:
"[|C | ((M = N) = True);
   [|C | ((M = N) = True);
     C | (((M X) = (N X)) = True)|] ==> False|] ==> False"
by auto

lemma bool_pos:
"[|C | ((M = N) = True);
   [|C | ((M = N) = True);
     C | (((M --> N) & (N --> M)) = True)|] ==> False|] ==> False"
by auto

lemma func_neg:
"[|C | ((M = N) = False);
   !! sk. [|C | ((M = N) = False);
            C | (((M sk) = (N sk)) = False)|] ==> False|] ==> False"
by auto

lemma bool_neg:
"[|C | ((M = N) = False);
   [|C | ((M = N) = False);
     C | (((M --> N) & (N --> M)) = False)|] ==> False|] ==> False"
```

```
by auto

lemma triv:
"[|C | ((A = A) = False);
   [|C | ((A = A) = False);
     C|] ==> False|] ==> False"
by auto

lemma dec1:
"[|C | ((h U = h V) = False);
   [|C | ((h U = h V) = False);
     C | ((U = V) = False) |] ==> False|] ==> False"
by auto

lemma flexrigid:
"[|C | ((F U = h V) = False);
   [|C | ((F U = h V) = False);
     C | (F = G) | ((F U = h V) = False)|] ==> False|] ==> False"
by auto

lemma res:
"[|C | (A = True);
   D | (B = False);
   [|C | (A = True);
     D | (B = False);
     C | D | ((A = B) = False)|] ==> False|] ==> False"
by auto

lemma fac_restr:
"[|(A = p) | (B = p);
   [|(A = p) | (B = p);
     (A = p) | ((A = B) = False)|] ==> False|] ==> False"
by auto
```

E.3 Helper code and lemmas

```
subsection "Helpers"

lemma unfold_def:
  "(A --> B) = (~ A | B)"
  "(A & B) = (~ (~ A | ~ B))"
by auto

ML {*
  fun resolve_unify thm = HEADGOAL (rtac thm THEN' atac)
*}
```

```
lemma collapse_false:
  "(False | (False | P)) = (False | P)"
  "((False | False) | P) = (False | P)"
by auto
```

E.4 Isabelle/HOL-reconstructed examples

```
subsection "Examples"

lemma "False | (! P. P | ~ P) = False ==> False"
apply (tactic {*resolve_unify @{thm extcnf_forall_neg}*})
apply (tactic {*resolve_unify @{thm extcnf_or_neg1}*})
apply (tactic {*resolve_unify @{thm extcnf_or_neg2}*})
apply (tactic {*resolve_unify @{thm extcnf_not_neg}*})
apply (thin_tac "False | (! P. P | ~ P) = False")
apply (thin_tac "False | (! P. P | ~ P) = False")
apply (thin_tac "False | (sk | ~ sk) = False")
apply (thin_tac "False | (sk | ~ sk) = False")
apply (thin_tac "False | (sk | ~ sk) = False")
apply (thin_tac "False | (~ sk) = False")
apply (thin_tac "False | (~ sk) = False")
apply (rule res[of "False" _ "False"])
apply assumption
apply assumption
apply (simp only: collapse_false)
apply (rule triv)
apply assumption
apply assumption
done

lemma "False | (! A . ? X . ! B. (A X & B X) --> (B X & A X)) = False ==> False"
apply (tactic {*resolve_unify @{thm extcnf_forall_neg}*})
apply (thin_tac "False | (ALL A. EX X. ALL B. A X & B X --> B X & A X) = False")
apply (thin_tac "False | (ALL A. EX X. ALL B. A X & B X --> B X & A X) = False")
apply (erule extcnf_forall_pos)
apply (tactic {*resolve_unify @{thm extcnf_forall_neg}*})
apply (thin_tac "False | (EX X. ALL B. sk X & B X --> B X & sk X) = False")
apply (thin_tac "False | (ALL B. sk (?X5 sk) & B (?X5 sk) -->
  B (?X5 sk) & sk (?X5 sk)) = False")
apply (thin_tac "False | (ALL B. sk (?X5 sk) & B (?X5 sk) -->
  B (?X5 sk) & sk (?X5 sk)) = False")
apply (simp only: unfold_def)
apply (erule extcnf_or_neg1)
apply (erule extcnf_or_neg2)
apply (thin_tac "False | (~ ~ (~ sk (?X15 sk sk) | ~ ska (?X15 sk sk)) |
  ~ (~ ska (?X15 sk sk) | ~ sk (?X15 sk sk))) = False")
apply (erule extcnf_not_neg)
apply (rotate_tac 2)
apply (erule extcnf_not_neg)
apply (rotate_tac -1)
apply (erule extcnf_or_pos)

apply (erule extcnf_not_pos)
apply (rotate_tac -1)
apply (erule extcnf_or_neg1)
apply (rotate_tac -2)
apply (erule extcnf_or_neg2)

(*very unwieldy to control reconstruction manually from here*)
apply auto
done
```