Readable Proofs for the Horn Fragment

Lydia Kondylidou
l.kondylidou@ifi.lmu.de
LMU Munich
Munich, Germany

Jasmin Blanchette
jasmin.blanchette@ifi.lmu.de
LMU Munich
Munich, Germany

Nicholas Smallbone
nicsma@chalmers.se
Chalmers University of Technology
Gothenburg, Sweden

Uwe Waldmann
uwe@mpi-inf.mpg.de
Max Planck Institute for Informatics
Saarland Informatics Campus
Saarbrücken, Germany

Abstract

Proofs generated by resolution and superposition provers can be difficult for humans to read and understand, partly because the proofs are by contradiction, and partly because they rely on unification. We introduce a structured proof format for the Horn fragment of first-order logic with equality. The format expresses direct proofs. It combines equational rewriting chains and human-style "have/hence" derivations. It relies on matching but not full unification, increasing intelligibility. To interface with provers, we introduce an algorithm that translates resolution and superposition proofs to our format and have implemented it in a tool called Tälja. We have also connected Tälja to the Twee equational prover, which supports the Horn fragment via an encoding. We have evaluated our approach on TPTP benchmarks.

Keywords: First-order logic, Horn fragment, resolution, superposition, proofs

1 Introduction

Automatic theorem provers are useful tools, but the proofs they produce are often cryptic. A computer can check that a sequence of inference steps is correct, but a human trying to understand *why* a theorem holds gains little from a long list of clauses derived using low-level inference rules. This is unfortunate. Users of automatic tools often value not only the final QED but also any insights that a proof might convey. Moreover, when an automatic prover is used inside a proof assistant, the generated proofs are translated and embedded into a formal development [1, 10], so they ought to be readable.

Proofs found by resolution provers such as PyRes [33] and superposition provers such as E [32] and Vampire [24] can be particularly difficult to read and understand. The superposition calculus [6] generalizes the resolution calculus [31] with equality reasoning. For both calculi, two characteristics come together to make the proofs unintelligible. First, the proofs are *by contradiction*: The prover negates the goal to be proved, adds the negated goal to the set of axioms, and then derives falsity. To understand such a proof, we must keep track of those steps that depend on the negated goal and those that do not, a cognitive burden that grows with proof size. Second, to check a proof step, we must try to unify specific syntactic elements. Unification is subtle enough on its own, and in addition, most provers do not indicate the positions at which inferences occur, which must be reconstructed by trial and error. In the following example, it is not obvious whether the premises entail the conclusion:

$$\frac{\mathsf{P}(x, \mathsf{f}(y, y), y, \mathsf{f}(\mathsf{a}, \mathsf{a})) \qquad \neg\mathsf{P}(x, y, y, x) \lor \neg\mathsf{P}(x, x, y, y)}{\neg\mathsf{P}(\mathsf{f}(\mathsf{f}(\mathsf{a}, \mathsf{a}), \mathsf{f}(\mathsf{a}, \mathsf{a})), \mathsf{f}(\mathsf{a}, \mathsf{a}), \mathsf{f}(\mathsf{a}, \mathsf{a}), \mathsf{f}(\mathsf{f}(\mathsf{a}, \mathsf{a}), \mathsf{f}(\mathsf{a}, \mathsf{a})))}$$

We introduce a structured proof format for the *Horn fragment* of the formalism underlying superposition, first-order logic with equality. Horn clauses form a well-studied, well-behaved fragment; in particular, theorems within the fragment can be proved without leaving the fragment. Our format is intended as a stepping stone toward readable proofs for full first-order logic and higher-order logic [8, 9, 36].

The format aims for human readability in two ways. First, it expresses *direct proofs*, which proceed from the axioms toward the goal, rather than proofs by contradiction. We can follow the reasoning without mentally tracking which intermediate steps originate from the negated goal. Second, proof steps rely on *matching* instead of full unification. Matching is a one-sided form of unification that binds variables in the rewrite rule to subterms of the term being rewritten.

The format is a hybrid. First, it supports equality chains in the style of the DISCOUNT [5], Waldmeister [20], and Twee [35] equational provers. In the purely equational case, a proof consists of a sequence of lemmas, each justified by an equality chain, followed by the restatement of the goal, also justified by an equality chain. An equality chain is a sequence $t_1 \approx t_2 \approx \cdots \approx t_n$ in which each $\approx$ step rewrites a subterm of the preceding term using an axiom or previously proved lemma as a rewrite rule. The equality symbol is written $\approx$ to distinguish it from syntactic equality $=$.



This work is licensed under a Creative Commons Attribution 4.0 International License.

Example 1. The following proof establishes the goal $\mathsf{g}(\mathsf{g}(\mathsf{a})) \approx \mathsf{a}$ from two axioms via a lemma:

Axiom 1: $\mathsf{g}(x) \approx \mathsf{f}(x)$

Axiom 2: $\mathsf{f}(x) \approx x$

Lemma 3: $\mathsf{g}(x) \approx x$
Proof:
  $\mathsf{g}(x)$
$\approx$ { by axiom 1 }
  $\mathsf{f}(x)$
$\approx$ { by axiom 2 }
  x

Goal 1: $\mathsf{g}(\mathsf{g}(\mathsf{a})) \approx \mathsf{a}$
Proof:
  $\mathsf{g}(\mathsf{g}(\mathsf{a}))$
$\approx$ { by lemma 3 }
  $\mathsf{g}(\mathsf{a})$
$\approx$ { by lemma 3 }
  a ■

Second, our format supports hyperresolution [7] steps. From an implication $A_1 \wedge \cdots \wedge A_m \rightarrow B$ and proofs of instances of the premises, it derives a corresponding instance of the conclusion. This is expressed using a suggestive, Isar-inspired [39] "have/hence" block. A "have" step states an intermediate fact, "hence" derives the conclusion, and multiple premises are combined with "and." Equality chains and "have/hence" blocks can be combined in the same proof.

Example 2. The following hyperresolution-based proof establishes $\mathsf{R}(\mathsf{a}, \mathsf{b})$ from five axioms via a lemma:

Axiom 1: $\mathsf{U}(\mathsf{b})$

Axiom 2: $\mathsf{U}(x) \rightarrow \mathsf{Q}(x)$

Axiom 3: $\mathsf{S}(\mathsf{a})$

Axiom 4: $\mathsf{S}(x) \rightarrow \mathsf{P}(x)$

Axiom 5: $\mathsf{P}(x) \wedge \mathsf{Q}(y) \rightarrow \mathsf{R}(x, y)$

Lemma 6: $\mathsf{Q}(\mathsf{b})$
Proof:
  have $\mathsf{U}(\mathsf{b})$
    by axiom 1
  hence $\mathsf{Q}(\mathsf{b})$
    by axiom 2

Goal 1: $\mathsf{R}(\mathsf{a}, \mathsf{b})$
Proof:
  have $\mathsf{S}(\mathsf{a})$
    by axiom 3
  hence $\mathsf{P}(\mathsf{a})$
    by axiom 4
  and $\mathsf{Q}(\mathsf{b})$
    by lemma 6
  hence $\mathsf{R}(\mathsf{a}, \mathsf{b})$
    by axiom 5 ■

To interface with existing provers, we introduce a translation algorithm that converts resolution and superposition proofs to our format. The algorithm first expands the given proof, which corresponds to a directed acyclic graph (DAG), into a proof tree. It then processes relevant clauses in order, translating each into a hyperresolution step expressed as a "have/hence" block; equational rewriting steps are assembled into equality chains. We prove the algorithm correct.

We have implemented the algorithm in a tool called *Tälja*. The tool works with proofs generated by E and Vampire from Horn problems. We have also integrated Tälja with Twee, an equational prover that supports the Horn fragment via an encoding. This integration required extending Twee to output resolution and rewriting proofs. We have evaluated Tälja and the Twee integration on relevant categories of the de facto standard TPTP (Thousands of Problems for Theorem Provers) [38] benchmark suite to confirm our approach's usefulness.

2 Background

We briefly describe clausal first-order logic as well as the resolution and superposition calculi based on it. The focus is on the aspects that are relevant to proofs and proof generation.

2.1 Clausal First-Order Logic

We work in simply typed classical first-order logic. We assume a fixed set of *base types* and an infinite set X of typed variables w, x, y, z. A *signature* Σ consists of typed function symbols $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{f}, \mathsf{g}, \ldots$ and typed predicate symbols $\mathsf{P}, \mathsf{Q}, \mathsf{R}, \ldots$.

Terms s, t, u, v are defined inductively as being either a variable $x \in X$ or the application $\mathsf{f}(t_1, \ldots, t_n)$ of an n-ary function symbol f from Σ to n suitably typed terms $t_1, \ldots, t_n$. *Atoms* A, B are applications $\mathsf{P}(t_1, \ldots, t_n)$ of an n-ary predicate symbol P from Σ to n suitably typed terms. For both kinds of applications, we omit the parentheses if $n = 0$. *Literals* L are atoms A or negated atoms $\neg A$; we call the former *positive* literals and the latter *negative* literals. Given a literal L, we write $\overline{L}$ for its complement, with $\overline{A} = \neg A$ and $\overline{\neg A} = A$.

Clauses C, D, E are finite multisets $\{L_1, \ldots, L_n\}$ of literals, usually written as $L_1 \vee \cdots \vee L_n$; thus, $\mathsf{P}(x) \vee \mathsf{Q}(y)$ and $\mathsf{Q}(y) \vee \mathsf{P}(x)$ are considered syntactically equal. The case where $n = 0$ corresponds to the empty clause, written $\bot$. Variables in clauses are regarded as being implicitly universally quantified; thus, the clause $\mathsf{P}(x) \vee \mathsf{Q}(y)$ is equivalent to the formula $\forall x, y.\ \mathsf{P}(x) \vee \mathsf{Q}(y)$.

Horn clauses are clauses that contain at most one positive literal. It is customary to write the Horn clauses

$$\neg A_1 \vee \cdots \vee \neg A_m \vee B \quad \text{and} \quad \neg A_1 \vee \cdots \vee \neg A_m$$

as

$$A_1 \wedge \cdots \wedge A_m \rightarrow B \quad \text{and} \quad A_1 \wedge \cdots \wedge A_m \rightarrow \bot$$

respectively. A special case of the Horn fragment is *equational logic*, in which equality is the only predicate symbol, and all axioms and the goal are single positive literals.

Terms, atoms, literals, and clauses are given a formal semantics by means of an *interpretation* J, which maps types to nonempty sets, function symbols to functions, and predicate symbols to predicates. The equality symbol $\approx$ is always interpreted as meaning equality. The interpretation is complemented by a *variable assignment* ξ. A clause is *valid* if, for all interpretations and variable assignments, one of its literals evaluates to true.

2.2 Resolution Calculus

Resolution is a calculus for clausal first-order logic. It was initially developed by Robinson [31] and has been refined and extended over the years [7]. Our work's primary focus is on superposition, but resolution is a useful stepping stone.

Resolution provers work by saturation: From the axioms and the negated goal, they repeatedly apply inference rules that create conclusions, from which further inferences may apply, until—in case of success—the empty clause $\bot$ is derived. The resolution calculus consists of the inference rule

$$\frac{C \vee A \qquad D \vee \neg B}{(C \vee D)\sigma}\ \text{resolution}$$

The substitution σ is a most general unifier (MGU) of A and B. In particular, applying σ to A and B makes them syntactically equal. For example, an MGU of $\mathrm{P}(\mathrm{a}, y)$ and $\mathrm{P}(x, \mathrm{b})$ is $\{x \mapsto \mathrm{a}, y \mapsto \mathrm{b}\}$; applying it to both atoms yields $\mathrm{P}(\mathrm{a}, \mathrm{b})$.

The rule operates on clauses matching the premises and derives new clauses. Since clauses are multisets, a schematic premise such as $D \vee \neg B$ stands for an arbitrary clause that contains a negative literal $\neg B$ anywhere and zero or more literals forming the subclause D. Furthermore, although this is implicit above, variables in the two premises are renamed apart before the rule is applied. An important special case of resolution arises when C and D are $\bot$. Then the conclusion $(C \vee D)\sigma$ is also $\bot$.

The inference rule is *sound*: If the premises are true according to an interpretation J and every variable assignment, then the conclusion is also true according to J and every variable assignment. Resolution proofs show how to derive $\bot$ from the problem's axioms and negated goal, thereby establishing that the goal follows from the axioms. These proofs are conceptually DAGs but usually expressed in a sequential format.

Example 3. A resolution proof from clauses 1–3 follows:

1.	$\mathrm{P}(z', z') \vee \neg\mathrm{Q}(z)$	axiom
2.	$\mathrm{P}(x', \mathrm{b}) \vee \mathrm{Q}(\mathrm{f}(x', x))$	axiom
3.	$\neg\mathrm{P}(\mathrm{a}, y)$	negated goal
4.	$\neg\mathrm{Q}(z)$	by resolution from 1 and 3
5.	$\mathrm{Q}(\mathrm{f}(\mathrm{a}, x))$	by resolution from 2 and 3
6.	$\bot$	by resolution from 5 and 4

■

Most resolution provers implement simplification rules to optimize proof search. Two relevant ones are

$$\frac{D \vee L \qquad C \vee (D \vee \overline{L})\sigma}{C \vee D\sigma}\ \text{subsumption resolution}$$

$$\frac{C \vee L \vee L}{C \vee L}\ \text{duplicate literal elimination}$$

The subsumption resolution rule can be seen as a combination of a resolution step with the instance $(D \vee L)\sigma$ of the first premise and zero or more duplicate literal elimination steps. The resolution rule has further side conditions that restrict the search space; these are unnecessary for soundness and are irrelevant to us. The calculus also includes a factoring rule for non-Horn clauses that plays no role in our work.

2.3 Superposition Calculus

With the resolution calculus, an equality predicate can be axiomatized, but the resulting explicit resolution inferences with the equality axioms are extremely detrimental for proof search. Robinson and Wos's paramodulation calculus [29] avoids explicit inferences with the equality axioms by integrating equality directly into the inference rules. Bachmair and Ganzinger's superposition calculus [6] improves paramodulation using ordering restrictions, selection functions, and a general redundancy concept.

In the following, equality is written in infix notation as $t \approx t'$. We write $t \not\approx t'$ for $\neg(t \approx t')$. Equality is implicitly commutative, meaning that the atoms $\mathrm{a} \approx \mathrm{b}$ and $\mathrm{b} \approx \mathrm{a}$ are considered syntactically equal.

Superposition extends resolution with two schematic inference rules. The first one is

$$\frac{C \vee s \approx s' \qquad D \vee L[u]}{(C \vee D \vee L[s'])\sigma}\ \text{superposition}$$

The notation $L[\,]$ represents a literal with a hole. The literals $L[u]$ and $L[s']$ are obtained by filling the hole in $L[\,]$ with the terms u and s', respectively. The substitution σ is an MGU of s and u. The second inference rule is

$$\frac{C \vee s \not\approx s'}{C\sigma}\ \text{equality resolution}$$

where σ is an MGU of s and s'.

The rules have further side conditions that restrict the search space. These are irrelevant to us. The calculus also includes an equality factoring rule for non-Horn clauses, which plays no role in our work.

Superposition proofs are by contradiction and usually expressed in a sequential format.

Example 4. A superposition proof from clauses 1–3 follows:

1. $x \approx \mathsf{a} \vee \mathsf{g}(\mathsf{b}, x) \approx \mathsf{f}(x)$ axiom
2. $\mathsf{c} \not\approx \mathsf{a}$ axiom
3. $\mathsf{h}(\mathsf{g}(\mathsf{b}, \mathsf{c})) \not\approx \mathsf{h}(\mathsf{f}(\mathsf{c}))$ negated goal
4. $\mathsf{c} \approx \mathsf{a} \vee \mathsf{h}(\mathsf{f}(\mathsf{c})) \not\approx \mathsf{h}(\mathsf{f}(\mathsf{c}))$ by superposition from 1 and 3
5. $\mathsf{c} \approx \mathsf{a}$ by equality resolution from 4
6. $\mathsf{a} \not\approx \mathsf{a}$ by superposition from 5 and 2
7. $\bot$ by equality resolution from 6 ■

Superposition provers use simplification rules to optimize proof search. They rely on rules such as subsumption resolution and duplicate literal elimination inherited from resolution as well as equality-specific rules such as

$$\frac{s \approx s' \qquad C \vee L[s\sigma]}{C \vee L[s'\sigma]}\ \text{demodulation}$$

where σ is an arbitrary substitution.

In practice, problems given to resolution and superposition provers are not always in clausal form and can combine arbitrary logical connectives and quantifiers in arbitrary ways. A preprocessor transforms free-form formulas to clauses. The preprocessing steps are recorded in the generated proofs, so that these match the original problems.

3 Proof Format

A proof in our format consists of a list of *axioms*, followed by zero or more *lemmas*, followed by one or more *goals*. The axioms restate the premises of the input problem, the lemmas represent intermediate facts or Horn clauses, and the goals are the statements to prove. The axioms, lemmas, and goals are numbered. Each lemma and goal is immediately followed by a subproof. Our format expresses direct proofs, in which reasoning proceeds from the axioms through the lemmas to the goals. All clauses are written using the customary Horn clause notation with implications instead of disjunctions.

The format permits both equality chains and "have/hence" blocks as subproofs. An *equality chain*, in the style of DISCOUNT and other equational provers, is a sequence $t_1 \approx t_2 \approx \cdots \approx t_n$ in which each step rewrites a subterm of the preceding term using an axiom or lemma as a left-to-right or right-to-left rewrite rule. A reflexive goal $t \approx t$ is proved by the chain t, which has no steps. A "have/hence" block represents a sequence of hyperresolution inferences. The keyword "have" introduces the first premise, and the keyword "and" introduces each additional premise. The keyword "hence" derives the conclusion from all accumulated premises by applying the specified axiom or lemma. The conclusion specified using "hence" can in turn serve as the first premise of a subsequent "hence", possibly in conjunction with "and". When the axioms alone are contradictory, a block derives $\bot$, from which every goal follows.

In the proof of a lemma of the form $A_1 \wedge \cdots \wedge A_m \to B$, where $m > 0$, a "have" or "hence" statement can be used to repeat one of the atoms A_i, or an instance therefore. In that case, the statement's proof is "by assumption". In addition, in a "hence" statement, the "rw" annotation indicates a rewrite step that uses the cited axiom or lemma as a rewrite rule on a subterm of the previous step's conclusion. By default, rewrite rules are applied from left to right; the "R→L" annotation indicates a right-to-left application.

Examples 1 and 2 are valid proofs in our format. The former is purely equational; the latter is based exclusively on hyperresolution. Example 5 combines both styles.

Example 5. The following proof establishes $\mathsf{P}(\mathsf{g}(\mathsf{a}))$ from four axioms via a lemma:

Axiom 1: $\mathsf{Q}(\mathsf{a})$

Axiom 2: $\mathsf{f}(x) \approx x$

Axiom 3: $\mathsf{g}(x) \approx \mathsf{f}(x)$

Axiom 4: $\mathsf{Q}(x) \wedge \mathsf{g}(x) \approx x \to \mathsf{P}(\mathsf{g}(\mathsf{a}))$

Lemma 5: $\mathsf{g}(x) \approx x$
Proof:
  $\mathsf{g}(x)$
$\approx$ { by axiom 3 }
  $\mathsf{f}(x)$
$\approx$ { by axiom 2 }
  x

Goal 1: $\mathsf{P}(\mathsf{g}(\mathsf{a}))$
Proof:
  have $\mathsf{Q}(\mathsf{a})$
    by axiom 1
  and $\mathsf{g}(\mathsf{a}) \approx \mathsf{a}$
    by lemma 5
  hence $\mathsf{P}(\mathsf{g}(\mathsf{a}))$
    by axiom 4 ■

Equality chains can be mechanically unfolded into sequences of "have/hence" steps if desired, but equality chains are generally more readable.

Example 6. The following proof is equivalent to the one in Example 5, with the equality chain written as "have/hence" steps instead. Our algorithm does not produce this form:

Axiom 1: $\mathsf{Q}(\mathsf{a})$

Axiom 2: $\mathsf{f}(x) \approx x$

Axiom 3: $\mathsf{g}(x) \approx \mathsf{f}(x)$

Axiom 4: $\mathsf{Q}(x) \wedge \mathsf{g}(x) \approx x \to \mathsf{P}(\mathsf{g}(\mathsf{a}))$

Goal 1: $\mathsf{P}(\mathsf{g}(\mathsf{a}))$
Proof:
  have $\mathsf{g}(\mathsf{a}) \approx \mathsf{f}(\mathsf{a})$
    by axiom 3
  hence $\mathsf{g}(\mathsf{a}) \approx \mathsf{a}$
    by rw axiom 2
  and $\mathsf{Q}(\mathsf{a})$
    by axiom 1

hence P(g(a))
by axiom 4 ∎

4 Translation Algorithm

We present a translation algorithm from resolution and superposition proofs to our format as pseudocode with an accompanying description, followed by three worked examples and a correctness proof.

4.1 Description

Provers produce refutational proofs presented as DAGs of derived clauses. The translation algorithm takes such a DAG as the input proof and expands it into a proof tree T.

Following Robinson [30], we categorize each clause of the input proof as an *electron* if it consists of a single atom and as a *nucleus* otherwise. In the Horn fragment, each nucleus takes the form $A_1 \wedge \cdots \wedge A_m \to B$, where $A_1, \ldots, A_m$ are its *body atoms* and the *head* B is either an atom or $\bot$. Among electrons we distinguish between *input electrons*, which are among the input axioms, and *derived electrons*, which arise as inner nodes of the DAG. Among nuclei we similarly distinguish between *axiom nuclei*, which are among the input axioms and have a positive head, and the *goal nucleus*, which is derived from the negated goal and has head $\bot$.

Unlike a hyperresolution step, each inference of T has one or two premises. Each node of the expanded proof tree T is assigned a *position*, a binary string recording its path from the root $\bot$. The root has position ε. Appending $d \in \{0, 1\}$ to position p gives the position of the premise numbered d of the node at p, and the only premise of a unary inference is at $p1$. For each binary inference, the premise supplying a positive atom precedes the one consuming it. For resolution, the premise where the resolved atom occurs unnegated is placed at position $p0$ and the other at position $p1$. For superposition and demodulation, the premise providing the rewriting equation comes first. Between two unit equations, either of which can be read as rewriting the other, the premise listed first in the input proof is taken as the one rewritten, so the other one comes first. When neither criterion applies, the premise order of the input proof is kept, except that between two positive units (i.e., single-literal clauses) the equational one is placed first. A clause shared between multiple derivations receives a separate position for each occurrence. The resulting positions are totally ordered lexicographically. This ordering is designed to reflect the order in which literals are derived in the direct proof.

The algorithm operates under a substitution θ derived from the input proof. Since variables are local to clauses, each variable is uniquely identified by its name and the clause's position. Applying θ to every clause of T must yield a ground (i.e., variable-free) proof tree, with fresh constants for the variables that no inference binds, so θ has to agree with the unifier of every inference. For example, suppose that the prover derives the conclusion $C_0\sigma$ from premises C_1 and C_2, where σ is the unifier of the resolved literals. Then the variables of C_1 and C_2 are instantiated by σ, and the variables of $C_0\sigma$ in turn by the unifier of the inference that consumes $C_0\sigma$. The grounding substitution θ is therefore obtained by composing the unifiers of all inferences, tracing the proof from the root to the leaves.

A variable that no inference binds—for example, a variable of a unit resolved against a variable of another clause—is mapped to a fresh constant. Such a constant occurs in no axiom, so a fact derived about it holds for every value of it. When the head of a nucleus is stored as a derived electron, the fresh constants it shares with the body are therefore replaced by variables again. The head of a nucleus may moreover contain variables that do not occur in its body. These variables stay free, so that the derived electron is as general as the proof allows. Given a substitution θ, we denote by θ' the substitution obtained by replacing the fresh constants introduced by θ with variables.

The translation algorithm maintains a set *Units* containing one entry for each electron. Each entry is represented as a quadruple $(\mathit{name}, E, \mathit{proof}, \mathit{pos})$, where E is the electron, and *name*, *proof*, and *pos* record its output name, stored proof, and position, respectively. In the Horn fragment, the proof tree must contain at least one positive unit leaf unless equality resolution closes a body atom; without one, there is nothing to resolve the negative body literals against, and $\bot$ can never be derived. Such clauses are precisely the input electrons, so *Units* is always nonempty from the start.

The main procedure `translate_proof` (Algorithm 1) begins with `collect_leaves`(T), which partitions the leaves into electrons and nuclei. It initializes *Units* with one entry per input electron, returns the nuclei as a list sorted by lexicographic position, and also returns a list of the goal atoms with their positions. The algorithm then processes the nuclei left to right, i.e., in order of position. Every nucleus that produced a premise of another nucleus at position p has a strictly smaller position than p; when a nucleus is reached, all electrons it depends on are therefore already in *Units*.

For each nucleus $A_1 \wedge \cdots \wedge A_m \to B$, the algorithm processes the body atoms in turn. The set of candidate electrons *Elecs* is initialized to the entries in *Units* whose position precedes the nucleus, and grows as each body atom is matched. The algorithm tries to match each body atom A_i with the electron E_i of a candidate entry U_i in *Elecs*. For each body atom A_i, the algorithm applies the substitution θ to obtain the ground atom $A_i\theta$, then calls `find_elec` (Algorithm 2) to search for a candidate electron matching $A_i\theta$. It first attempts a direct match with each candidate, establishing $A_i\theta = E_i\sigma_i$, where σ_i maps the electron's variables to terms. Since $A_i\theta$ is ground, a match instantiates only the electron. In particular, a fresh constant in $A_i\theta$ is matched by a variable of the electron. If no candidate matches directly, it retries each candidate's *rewriting chain* (`try_chain`), the sequence of rewriting steps

Algorithm 1: `translate_proof`(T)

Input: T: resolution/superposition proof tree
Output: structured proof (emitted)

$(\mathit{Units},\ \mathit{Nuclei},\ [(G_1, p_{G_1}), \ldots, (G_q, p_{G_q})]) \leftarrow$ `collect_leaves`(T)
let θ be the substitution derived from the input proof
for each $(\mathit{pos},\ \mathit{clause}) \in \mathit{Nuclei}$ **do**
  $(A_1, \ldots, A_m \to B) \leftarrow \mathit{clause}$
  if $B \neq \bot$ **then**
    $\mathit{Elecs} \leftarrow \{\, U \in \mathit{Units} \mid U.\mathit{pos} \prec \mathit{pos} \,\}$
    $\mathit{block} \leftarrow$ ""
    for $i = 1$ ***to*** m **do**
      $(U_i,\ \sigma_i) \leftarrow$ `find_elec`$(A_i\theta,\ \mathit{Elecs},\ \mathit{Units},\ \mathit{pos})$
      $E_i \leftarrow U_i.E$
      $\mathit{Elecs} \leftarrow \mathit{Elecs} \cup \{\, U_i[E \mapsto E_i\sigma_i] \,\}$
      $\mathit{block}_i \leftarrow$ `make_block`$(E_i,\ \sigma_i,\ [\,],\ \mathit{Units})$
      if $i = 1$ **then**
        $\mathit{block} \leftarrow \mathit{block}_i$
      else
        if $A_i\theta \notin \mathit{Units}$ **then**
          $\mathit{Units} \leftarrow \mathit{Units} \cup \{(\mathit{nil},\ A_i\theta,\ \mathit{block}_i,\ \mathit{nil})\}$
        $\mathit{name}_i \leftarrow$ `name_of`$(A_i\theta,\ \mathit{Units})$
        $\mathit{block} \leftarrow \mathit{block} \cdot$ "and $A_i\theta$ by name_i"
  if $B = \bot$ **then**
    for $j = 1$ ***to*** q **do**
      if $\exists\ (\mathit{name},\ E,\ \mathit{proof},\ \mathit{pos}) \in \mathit{Units}, \rho_0, \mathit{rw}$ *s.t.* $E\,\rho_0 \to^*_{\mathit{rw}} G_j\theta$, *with rw empty or the rewriting chain of* E **then**
        $\mathit{block} \leftarrow$ `make_block`$(E,\ \rho_0,\ \mathit{rw},\ \mathit{Units})$
        emit "Goal j: $G_j\theta$"
        emit "Proof:" $\cdot$ block
      else
        $(l \approx r) \coloneqq G_j\theta$, read as $G_j\theta \approx$ true if G_j is no equation
        $\mathit{chain} \leftarrow$ `get_chain`$(l,\ r,\ p_{G_j},\ \mathit{Units})$
        emit "Goal j: $G_j\theta$"
        emit "Proof: l"
        for each $(\mathit{name},\ \mathit{dir},\ t) \in \mathit{chain}$ **do**
          emit "$\approx t$ by name" $\cdot$ ("R→L" if $\mathit{dir} = R{\to}L$)
    return
  else
    $\mathit{block} \leftarrow \mathit{block} \cdot$ "hence $B\theta'$ by `name`(clause)"
    $\mathit{Units} \leftarrow \mathit{Units} \cup \{(\mathit{nil},\ B\theta',\ \mathit{block},\ \mathit{pos})\}$

Algorithm 2: `find_elec`$(A_i\theta,\ \mathit{Elecs},\ \mathit{Units},\ \mathit{pos})$

Input: $A_i\theta$: ground body atom; *Elecs*: candidate electrons; *Units*: unit set; *pos*: clause position
Output: (U_i, σ_i)

for each $U \in \mathit{Elecs}$ **do**
  if ***let*** $\sigma_i \leftarrow$ `try_match`$(A_i\theta,\ U.E)$ **then**
    return $(U,\ \sigma_i)$
for each $U \in \mathit{Elecs}$ **do**
  if ***let*** $(U', \sigma_i) \leftarrow$ `try_chain`$(A_i\theta,\ U)$ **then**
    $\mathit{Units} \leftarrow \mathit{Units} \cup \{U'\}$
    return $(U',\ \sigma_i)$

Algorithm 3: `make_block`$(E,\ \rho,\ \mathit{rw},\ \mathit{Units})$

Input: E: atom; ρ: substitution; *rw*: list of rewrite steps; *Units*: unit set
Output: subproof establishing $E\,\rho$ via rewriting

if $(\mathit{nil},\ E,\ \mathit{proof},\ \mathit{pos}) \in \mathit{Units}$ ***and*** $\neg$ `is_eq_chain`(proof) **then**
  $\mathit{block} \leftarrow \mathit{proof}\ \rho$
else
  $\mathit{block} \leftarrow$ "have $E\,\rho$ by `name_of`$(E,\ \mathit{Units})$"
for each $(\mathit{name},\ \mathit{dir},\ t) \in \mathit{rw}$ **do**
  $\mathit{block} \leftarrow \mathit{block} \cdot$ "hence t by rw name" $\cdot$ ("R→L" if $\mathit{dir} = R{\to}L$)
return block

recorded when that electron was derived or simplified in the input proof. If the electron rewritten along this chain matches $A_i\theta$, it is added to *Units* as a new unit, proved by the chain. The input proof may also rewrite the body atom itself before resolving it, and then $A_i\theta$ is rewritten along these recorded steps and matched with an electron, and the steps are appended in reverse, so the block still ends with $A_i\theta$. One of the two attempts succeeds, since under θ the electron resolved against A_i in T matches $A_i\theta$, possibly after its recorded rewrites. After each match, the instantiated electron is added to *Elecs*, making it available as a candidate for subsequent body atoms within the same nucleus.

Next, for each body atom, `make_block` (Algorithm 3) constructs the corresponding subproof, which is either the instantiation by σ_i of a stored proof or a single "have $E_i\sigma_i$ by *name*" line referencing the matching axiom or lemma. As a readability heuristic, equality chains are always promoted to lemmas and referenced by name instead of inlined. If rewrite steps are present, each is appended as a "hence t by rw *name*" line, where *name* is the name of the axiom or lemma applied in the rewrite step, with an explicit direction if the application is right to left. For the first body atom $A_1\theta$, the subproof is inlined as the start of the nucleus proof. For each subsequent $A_i\theta$, a lemma name is assigned to $A_i\theta$, and the subproof becomes the proof of that lemma, which the "and" premise cites. A subproof that only cites an axiom or lemma gives that name instead. A final "hence $B\theta'$" closes the nucleus proof. The assembled subproof encodes one hyperresolution step. The set *Units* is then extended with a new derived electron for $B\theta'$, with the fresh constants introduced by θ replaced by variables, so the electron may be nonground. The new entry carries no lemma name initially and records the nucleus's position. A name is assigned when the electron is cited as an "and" premise or carries an equality chain proof.

Among the nuclei processed in the main loop, the negated goal nucleus has the form $G_1 \wedge \cdots \wedge G_q \to \bot$, i.e., $B = \bot$. When it is reached, for each G_j the algorithm searches *Units*

for an electron matching $G_j\theta$ under some substitution ρ_0, assembles the proof with `make_block`, and emits it as the goal's proof. The electron may first be rewritten along its rewriting chain *rw*; `make_block` appends the rewrite steps. For goals $l \approx r$ where no such electron is found, with an atom $G_j\theta$ read as $G_j\theta \approx \mathsf{true}$, `get_chain` returns a list of rewriting steps establishing $l \approx r$. Each step is a tuple $(name, dir, t)$ recording the applied unit, direction, and intermediate term. For an atom, the chain ends with a fact, and the proof is printed as that fact followed by the rewrite steps in reverse.

The algorithm yields unit lemmas, by naming equality chains and derived electrons cited as premises. The subproof above each shared clause is duplicated at every use, leading in the worst case to an exponential explosion in proof size. To avoid this, we extend the algorithm to perform *lemma introduction*. Whenever the input DAG contains a derived Horn clause C with a head that is used at least twice in the DAG, the algorithm makes it a lemma. It negates the clause, yielding the formula $\exists x_1, \ldots, x_k.\, \neg C$, where the variables x_i correspond to C's free variables, and replaces each x_i by a fresh Skolem constant c_i. For $C = A_1 \wedge \cdots \wedge A_m \to B$, with $m = 0$ for a unit, this yields the ground atoms $A_1\sigma, \ldots, A_m\sigma$ and the negated ground atom $\neg B\sigma$, where σ maps each x_i to c_i. The algorithm then extracts the derivation of C from the input proof as a refutation of these clauses from the problem's axioms and translates it recursively, citing each $A_i\sigma$ by assumption. The constants c_i are treated as arbitrary but fixed constants throughout the proof steps. The statement of the resulting lemma is C. The resulting proof, with each c_i replaced back by x_i, is emitted as a lemma, and C is thereafter treated as an axiom when translating the input proof.

Example 7. We start with the following resolution proof:

1.	$\mathsf{U}(\mathsf{b})$	axiom
2.	$\neg\mathsf{U}(x) \vee \mathsf{Q}(x)$	axiom
3.	$\mathsf{S}(\mathsf{a})$	axiom
4.	$\neg\mathsf{S}(x) \vee \mathsf{P}(x)$	axiom
5.	$\neg\mathsf{P}(x) \vee \neg\mathsf{Q}(y) \vee \mathsf{R}(x, y)$	axiom
6.	$\neg\mathsf{R}(\mathsf{a}, \mathsf{b})$	negated goal
7.	$\mathsf{P}(\mathsf{a})$	by resolution from 4 and 3
8.	$\mathsf{Q}(\mathsf{b})$	by resolution from 2 and 1
9.	$\neg\mathsf{P}(\mathsf{a}) \vee \neg\mathsf{Q}(\mathsf{b})$	by resolution from 5 and 6
10.	$\neg\mathsf{Q}(\mathsf{b})$	by resolution from 9 and 7
11.	$\bot$	by resolution from 10 and 8

The above proof is essentially a DAG. The corresponding proof tree is

$\bot$	(pos. ε)
$\quad\mathsf{Q}(\mathsf{b})$	(pos. `0`)
$\quad\quad\mathsf{U}(\mathsf{b})$	(axiom 1, pos. `00`)
$\quad\quad\neg\mathsf{U}(x) \vee \mathsf{Q}(x)$	(axiom 2, pos. `01`)
$\quad\neg\mathsf{Q}(\mathsf{b})$	(pos. 1)
$\quad\quad\mathsf{P}(\mathsf{a})$	(pos. `10`)
$\quad\quad\quad\mathsf{S}(\mathsf{a})$	(axiom 3, pos. `100`)
$\quad\quad\quad\neg\mathsf{S}(x) \vee \mathsf{P}(x)$	(axiom 4, pos. `101`)
$\quad\neg\mathsf{P}(\mathsf{a}) \vee \neg\mathsf{Q}(\mathsf{b})$	(pos. 11)
$\quad\quad\neg\mathsf{P}(x) \vee \neg\mathsf{Q}(y) \vee \mathsf{R}(x, y)$	(axiom 5, pos. `110`)
$\quad\quad\neg\mathsf{R}(\mathsf{a}, \mathsf{b})$	(goal, pos. `111`)

Axioms 1 and 3 are input electrons. Axioms 2, 4, and 5 and the negated goal are nuclei, with goal atom $\mathsf{R}(\mathsf{a}, \mathsf{b})$. *Units* is initialized with

$$\{(\text{“axiom 1”}, \mathsf{U}(\mathsf{b}), \mathit{nil}, \mathtt{00}),\ (\text{“axiom 3”}, \mathsf{S}(\mathsf{a}), \mathit{nil}, \mathtt{100})\}$$

Axioms 2, 4, and 5 each use a variable x. This does not mean that they refer to the same variable; the prover names variables independently in each clause of the input proof. We distinguish them by subscripting with the clause's position. The substitution θ derived from the input proof is then

$$\theta = \{x_{\mathtt{01}} \mapsto \mathsf{b},\ x_{\mathtt{101}} \mapsto \mathsf{a},\ x_{\mathtt{110}} \mapsto \mathsf{a},\ y_{\mathtt{110}} \mapsto \mathsf{b}\}$$

Algorithm 1 processes the nuclei from left to right in the proof tree. The first nucleus is $\neg\mathsf{U}(x) \vee \mathsf{Q}(x)$, at position `01`. For $\mathsf{U}(x)$, applying θ gives $\mathsf{U}(\mathsf{b})$. The algorithm searches *Units* for a matching electron at an earlier position and finds a direct match with axiom 1 at position `00`. It then constructs a subproof of the head instance $\mathsf{Q}(\mathsf{b})$, opening with “have $\mathsf{U}(\mathsf{b})$ by axiom 1” and closing with “hence $\mathsf{Q}(\mathsf{b})$ by axiom 2”. The derived electron $\mathsf{Q}(\mathsf{b})$ is added to *Units*.

Next, the algorithm processes $\neg\mathsf{S}(x) \vee \mathsf{P}(x)$ at position `101`. For the body atom $\mathsf{S}(x)$, applying θ gives $\mathsf{S}(\mathsf{a})$, which matches axiom 3 in *Units*. A subproof of $\mathsf{P}(\mathsf{a})$ is constructed in the same way as before, and the derived electron $\mathsf{P}(\mathsf{a})$ is added to *Units*.

The algorithm then processes $\neg\mathsf{P}(x) \vee \neg\mathsf{Q}(y) \vee \mathsf{R}(x, y)$ at position `110`. After it applies θ, the two body atoms become $\mathsf{P}(\mathsf{a})$ and $\mathsf{Q}(\mathsf{b})$. $\mathsf{P}(\mathsf{a})$ matches the derived electron $\mathsf{P}(\mathsf{a})$. Its stored subproof is inlined. $\mathsf{Q}(\mathsf{b})$ matches the derived electron $\mathsf{Q}(\mathsf{b})$, but since it is the second body atom, it is promoted to a new lemma, lemma 6. Its proof is the subproof stored for $\mathsf{Q}(\mathsf{b})$. The resulting proof opens with “have $\mathsf{S}(\mathsf{a})$ by axiom 3”, continues with “hence $\mathsf{P}(\mathsf{a})$ by axiom 4”, adds “and $\mathsf{Q}(\mathsf{b})$ by lemma 6”, and closes with “hence $\mathsf{R}(\mathsf{a}, \mathsf{b})$ by axiom 5”. The derived electron $\mathsf{R}(\mathsf{a}, \mathsf{b})$ is added to *Units*.

Finally, the algorithm processes the negated goal $\neg\mathsf{R}(\mathsf{a}, \mathsf{b})$ at position 111. This clause has the form $\mathsf{R}(\mathsf{a}, \mathsf{b}) \to \bot$, so its body atom $\mathsf{R}(\mathsf{a}, \mathsf{b})$ is the goal atom G_1. Since $B = \bot$ for this nucleus, the algorithm matches $\mathsf{R}(\mathsf{a}, \mathsf{b})$ with the stored electron $\mathsf{R}(\mathsf{a}, \mathsf{b})$. Its subproof is emitted directly as goal 1. The result is the proof presented in Example 2. ■

Example 8. We start with the following superposition proof:

1.	$\mathsf{g}(x) \approx \mathsf{f}(x)$	axiom
2.	$\mathsf{f}(x) \approx x$	axiom
3.	$\mathsf{g}(\mathsf{g}(\mathsf{a})) \not\approx \mathsf{a}$	negated goal
4.	$\mathsf{g}(x) \approx x$	by superposition from 2 and 1
5.	$\mathsf{g}(\mathsf{a}) \not\approx \mathsf{a}$	by superposition from 3 and 4
6.	$\mathsf{a} \not\approx \mathsf{a}$	by superposition from 5 and 4
7.	$\bot$	by equality resolution from 6

Before expanding this proof into a tree, the algorithm performs lemma introduction. Clause 4, $\mathrm{g}(x) \approx x$, is not an axiom and is used twice in the proof, as a premise of clauses 5 and 6. The algorithm negates the clause and replaces x by a fresh Skolem constant c, yielding $\mathrm{g}(\mathrm{c}) \not\approx \mathrm{c}$. Applying the translation recursively yields the chain $\mathrm{g}(\mathrm{c}) \approx \mathrm{f}(\mathrm{c}) \approx \mathrm{c}$ from axioms 1 and 2. Replacing c back by x gives the chain $\mathrm{g}(x) \approx \mathrm{f}(x) \approx x$. The algorithm promotes this chain to a named lemma, lemma 3, which states $\mathrm{g}(x) \approx x$ and has the chain as its proof, and replaces clause 4 by lemma 3 before expanding the proof into a tree. As a result, the derivation of clause 4 is translated only once.

The corresponding proof tree is

$\bot$ (pos. ε)
- $\mathrm{a} \not\approx \mathrm{a}$ (pos. 1)
  - $\mathrm{g}(x) \approx x$ (lemma 3, pos. 10)
  - $\mathrm{g}(\mathrm{a}) \not\approx \mathrm{a}$ (pos. 11)
    - $\mathrm{g}(x) \approx x$ (lemma 3, pos. 110)
    - $\mathrm{g}(\mathrm{g}(\mathrm{a})) \not\approx \mathrm{a}$ (goal, pos. 111)

Axioms 1 and 2 occur only in the proof of lemma 3, which acts as an input electron. The negated goal is $\mathrm{g}(\mathrm{g}(\mathrm{a})) \not\approx \mathrm{a}$. *Units* is initialized with lemma 3 at its first position

$$\{(\text{“lemma 3”}, \mathrm{g}(x) \approx x, \mathrm{g}(x) \approx \mathrm{f}(x) \approx x, \mathtt{10})\}$$

The only nucleus is the negated goal at position 111, with body atom $\mathrm{g}(\mathrm{g}(\mathrm{a})) \approx \mathrm{a}$ and head $\bot$. Since this nucleus is ground, θ leaves it unchanged. No entry in *Units* can match this atom directly. The algorithm finds the equality chain $\mathrm{g}(\mathrm{g}(\mathrm{a})) \approx \mathrm{g}(\mathrm{a}) \approx \mathrm{a}$ using lemma 3 twice, with $x \mapsto \mathrm{g}(\mathrm{a})$ and $x \mapsto \mathrm{a}$ respectively, and emits it as the goal proof. The result is the proof presented in Example 1. ■

Example 9. We start with the following superposition proof:

1.	$\mathrm{Q}(\mathrm{a})$	axiom
2.	$\mathrm{f}(x) \approx x$	axiom
3.	$\mathrm{g}(x) \approx \mathrm{f}(x)$	axiom
4.	$\neg\mathrm{Q}(x) \vee \mathrm{g}(x) \not\approx x \vee \mathrm{P}(\mathrm{g}(\mathrm{a}))$	axiom
5.	$\neg\mathrm{P}(\mathrm{g}(\mathrm{a}))$	negated goal
6.	$\mathrm{g}(x) \approx x$	by superposition from 3 and 2
7.	$\neg\mathrm{Q}(x) \vee \mathrm{g}(x) \not\approx x$	by resolution from 4 and 5
8.	$\neg\mathrm{Q}(x)$	by resolution from 7 and 6
9.	$\bot$	by resolution from 1 and 8

The corresponding proof tree is

$\bot$ (pos. ε)
- $\mathrm{Q}(\mathrm{a})$ (axiom 1, pos. 0)
- $\neg\mathrm{Q}(x)$ (pos. 1)
  - $\mathrm{g}(x) \approx x$ (pos. 10)
    - $\mathrm{f}(x) \approx x$ (axiom 2, pos. 100)
    - $\mathrm{g}(x) \approx \mathrm{f}(x)$ (axiom 3, pos. 101)
  - $\neg\mathrm{Q}(x) \vee \mathrm{g}(x) \not\approx x$ (pos. 11)
    - $\neg\mathrm{Q}(x) \vee \mathrm{g}(x) \not\approx x \vee \mathrm{P}(\mathrm{g}(\mathrm{a}))$ (axiom 4, pos. 110)
    - $\neg\mathrm{P}(\mathrm{g}(\mathrm{a}))$ (goal, pos. 111)

Axioms 1, 2, and 3 are input electrons, whereas axiom 4 and the negated goal are nuclei, with goal atom $\mathrm{P}(\mathrm{g}(\mathrm{a}))$. *Units* is initialized with

$$\{(\text{“axiom 1”}, \mathrm{Q}(\mathrm{a}), \mathit{nil}, \mathtt{0}),\ (\text{“axiom 2”}, \mathrm{f}(x) \approx x, \mathit{nil}, \mathtt{100}),\ (\text{“axiom 3”}, \mathrm{g}(x) \approx \mathrm{f}(x), \mathit{nil}, \mathtt{101})\}$$

On the leaves, the substitution θ is

$$\theta = \{x_{\mathtt{100}} \mapsto \mathrm{a},\ x_{\mathtt{101}} \mapsto \mathrm{a},\ x_{\mathtt{110}} \mapsto \mathrm{a}\}$$

Algorithm 1 processes the nucleus $\neg\mathrm{Q}(x) \vee \mathrm{g}(x) \not\approx x \vee \mathrm{P}(\mathrm{g}(\mathrm{a}))$ at position 110, i.e., $\mathrm{Q}(x) \wedge \mathrm{g}(x) \approx x \rightarrow \mathrm{P}(\mathrm{g}(\mathrm{a}))$. Under θ, its body atoms become $\mathrm{Q}(\mathrm{a})$ and $\mathrm{g}(\mathrm{a}) \approx \mathrm{a}$. The first matches axiom 1 directly. The second matches neither equational electron directly. The input proof derives $\mathrm{g}(x) \approx x$ by rewriting axiom 3 using axiom 2, which replaces $\mathrm{f}(x)$ by x. The algorithm reads this step as the chain $\mathrm{g}(x) \approx \mathrm{f}(x) \approx x$. This chain's conclusion $\mathrm{g}(x) \approx x$ matches the body atom $\mathrm{g}(\mathrm{a}) \approx \mathrm{a}$ under the substitution $x \mapsto \mathrm{a}$. The algorithm promotes the chain to a named lemma, lemma 5, which states $\mathrm{g}(x) \approx x$ and has the chain as its proof. It then assembles a subproof for $\mathrm{P}(\mathrm{g}(\mathrm{a}))$, opening with "have $\mathrm{Q}(\mathrm{a})$ by axiom 1", continuing with "and $\mathrm{g}(\mathrm{a}) \approx \mathrm{a}$ by lemma 5" and closing with "hence $\mathrm{P}(\mathrm{g}(\mathrm{a}))$ by axiom 4", and adds $\mathrm{P}(\mathrm{g}(\mathrm{a}))$ to *Units* as a derived electron.

Finally, the negated goal $\neg\mathrm{P}(\mathrm{g}(\mathrm{a}))$ at position 111 is processed. Its body atom $\mathrm{P}(\mathrm{g}(\mathrm{a}))$ matches the derived electron in *Units*, and the corresponding proof is emitted as the goal proof. The result is the proof presented in Example 5. ■

4.2 Correctness

We have to prove that, given a resolution or superposition proof as input, the algorithm always produces a correct proof in our format as output.

Let us start with some theoretical considerations. Let T be an input proof tree for a Horn problem using any of the rules (subsumption) resolution, duplicate literal elimination, superposition, demodulation, and equality resolution. To transform T, we must keep track of the ancestry of literals occurring in the tree. To this end, we assign sets of label sets to literal occurrences in T. For sets of sets $\mathcal{A}$ and $\mathcal{B}$, define $\mathcal{A} \sqcup \mathcal{B} = \{\, x \cup y \mid x \in \mathcal{A},\ y \in \mathcal{B} \,\}$. We now proceed in the following way: If L is a literal in a leaf clause in T, we assign the set $\{\{\ell\}\}$ to it, where ℓ is an arbitrary unique identifier. (If the same clause occurs at several leaf positions in T, or if one literal occurs multiple times in a clause, the different occurrences of literals get different identifiers.) If a literal L in the conclusion of a superposition or demodulation inference is obtained by combining two literals with sets of label sets S_1 and S_2, we assign the set $S_1 \sqcup S_2$ to L. If a literal L in the conclusion of a duplicate literal elimination is obtained by combining two literals with sets of label sets S_1 and S_2,

we assign the set $S_1 \cup S_2$ to L. (Note that in the Horn case, a duplicated literal is necessarily negative.) Otherwise, that is, if a literal L in the conclusion of an inference results from a literal with a set of label sets S in a premise by mere copying and possibly instantiation, we assign the set S to L. Subsumption resolution inferences are treated like combinations of resolution and duplicate literal elimination inferences.

The function *leaflit* maps a label ℓ to the unique literal in a leaf clause to which $\{\{\ell\}\}$ is assigned. The function *leafpos* maps a label ℓ to the unique position p in T such that a literal L in a leaf clause at position p is assigned the set $\{\{\ell\}\}$. We call a label ℓ positive or negative, respectively, if $\mathit{leaflit}(\ell)$ is a positive or negative literal. We define an ordering $\succ$ on labels in such a way that $\ell \succ \ell'$ if $\mathit{leafpos}(\ell)$ is lexicographically larger than $\mathit{leafpos}(\ell')$.

Lemmas 1 and 2 can be proved by induction over the proof structure and case analysis over the possible inferences:

Lemma 1. *If L is a positive literal in a clause in T, then it is assigned a singleton set $\{M\}$ and every label in M is positive.*

Lemma 2. *If L is a negative literal in a clause in T, then it is assigned a set S such that every $M \in S$ contains exactly one negative label ℓ and $\ell \succ \ell'$ for all positive labels ℓ' in M.*

We say that a negative label ℓ *disappears* in T using a set of positive labels M if there is an equality resolution inference

$$\frac{C \vee s \not\approx s'}{C\sigma}$$

in T where $s \not\approx s'$ is assigned a set $\{\{\ell\} \cup M\} \cup S$, or if there is a resolution inference

$$\frac{C_1 \vee A_1 \qquad C_2 \vee \neg A_2}{C_1\sigma \vee C_2\sigma}$$

in T where A_1 is assigned a set $\{M_1\}$, $\neg A_2$ is assigned a set $\{\{\ell\} \cup M_2\} \cup S$, and $M = M_1 \cup M_2$.

Since the input proof tree has the empty clause at the root, every negative label must eventually disappear. Using Lemma 2, we obtain the following lemma:

Lemma 3. *If a negative literal in a leaf clause in T is assigned a set $\{\{\ell\}\}$, then ℓ disappears in T in an equality resolution inference or resolution inference using a finite set of positive labels M and $\ell \succ \ell'$ for all labels ℓ' in M.*

If T is a ground input proof tree, then every literal occurring anywhere in T is a logical consequence of its ancestor literals. The following lemma formalizes this fact. It is again easily proved by induction over the proof structure and case analysis over the possible inferences:

Lemma 4. *Assume that T is a ground resolution or superposition proof tree. If a literal L_0 in a clause in T is assigned a set $\{\{\ell_1, \ldots, \ell_n\}\} \cup S$, then $\{\mathit{leaflit}(\ell_1), \ldots, \mathit{leaflit}(\ell_n)\} \models L_0$.*

Analogously, if a literal $s \not\approx s$ is eliminated in a ground equality resolution inference, then the ancestor literals of $s \not\approx s$ are inconsistent, and if literals A and $\neg A$ are eliminated in a resolution inference, then the ancestor literals of A and $\neg A$ are inconsistent:

Lemma 5. *Assume that T is a ground resolution or superposition proof tree. If a negative label ℓ_0 disappears in T using a finite set of positive labels $\{\ell_1, \ldots, \ell_n\}$, then $\{\mathit{leaflit}(\ell_0), \ldots, \mathit{leaflit}(\ell_n)\} \models \bot$.*

Theorem 1. *Assume that T is a resolution or superposition proof tree and that $T\theta$ is a ground resolution or superposition proof tree obtained from T by applying some grounding substitution θ to all clauses in T. Let $(C_k)_{1 \le k \le n}$ be the sequence of all leaf clauses in T that are not positive unit clauses, in ascending order with respect to the position order. For $1 \le k \le n$ let $C_k = D_k \vee B_k$, where $D_k = \neg A_{k1} \vee \cdots \vee \neg A_{km_k}$ is a disjunction of negative literals and B_k is either a positive literal or $\bot$. Let the substitution θ'_k be defined like θ, except that θ'_k maps all variables that occur in B_k but not in D_k to themselves. Let p_k be the position of C_k in T and let E_k be the set of all positive unit leaf clauses that occur in T at positions smaller than p_k. For $1 \le k \le n$ and $0 \le i \le m_k$ let*

$$U_{ki} = E_k \cup \bigcup_{\substack{1 \le j < k,\\ 1 \le l \le m_j}} \{A_{jl}\theta\} \cup \bigcup_{1 \le l < i} \{A_{kl}\theta\} \cup \bigcup_{1 \le j < k} \{B_j\theta'_j\}.$$

Then the following properties hold:

(i) For every $1 \le k \le n$ and $1 \le i \le m_k$, there exists a rewrite proof of $A_{ki}\theta$ using equations from U_{ki}.

(ii) For every $1 \le k \le n$, $\bigcup_{1 \le l \le m_k} \{A_{kl}\theta\} \cup \{C_k\} \models B_k\theta'_k$.

Proof. (i) The literal $\neg A_{ki}\theta$ is a negative literal occurring in a leaf clause in $T\theta$ at position p_k. By Lemma 3, its label ℓ must disappear in $T\theta$ using positive labels that are all smaller than ℓ. By Lemma 5, this implies that there are positive literals $A'_1\theta, \ldots, A'_h\theta$ occurring in $T\theta$ at leaf positions smaller than p_k such that $\{\neg A_{ki}\theta, A'_1\theta, \ldots, A'_h\theta\} \models \bot$ or, in other words, $\{A'_1\theta, \ldots, A'_h\theta\} \models A_{ki}\theta$. Now for each A'_l there are two possibilities: If A'_l is a positive unit leaf clause in T, then we have $A'_l \in E_k \subseteq U_{ki}$ and the unit clause A'_l entails its ground instance $A'_l\theta$, therefore $U_{ki} \models A'_l\theta$. Otherwise, A'_l is a positive literal in a nonunit leaf clause in T, that is, a positive literal B_j in a leaf clause C_j with $j < k$. Then $A'_l\theta = B_j\theta$ is a ground instance of the (nonleaf) unit clause $B_j\theta'_j$ and therefore entailed by it. Since $B_j\theta'_j \in U_{ki}$, we again get $U_{ki} \models A'_l\theta$. In a Horn refutation tree, an inference with a negative premise has a negative conclusion, and every inference with a negative conclusion has exactly one negative premise, which comes second, since the premise supplying a positive literal precedes the one consuming it. Hence C_n is the only leaf clause with head $\bot$, and U_{ki} contains only positive unit clauses. Reading each nonequational atom A as the equation $A \approx \mathrm{true}$, U_{ki} is a set of unit equations that

entails $A_{ki}\theta$, so there is a rewrite proof of $A_{ki}\theta$ using U_{ki} by Birkhoff's theorem.

(ii) This follows immediately by hyperresolution. □

Putting together the rewrite proofs obtained for part (i) and the hyperresolution steps obtained for part (ii), we obtain a proof with the desired shape. Note that the last leaf clause that is not a positive unit, that is, C_n, must necessarily have the head $\bot$, so that the "if $B = \bot$" part is ultimately reached. When C_n is the negated goal, its body atoms are the goal atoms, so each goal $G_j\theta$, where $1 \le j \le m_n$, is proved by the rewrite proof that part (i) gives for it.

In the algorithm, the set *Nuclei* contains $\{ C_k \mid 1 \le k \le n \}$. *Units* is initially the set of positive unit leaf clauses in T and is extended as each nucleus is processed. When the body atom A_{ki} is processed, the set *Elecs* contains E_k and every $B_j\theta'_j$ with $j < k$, which is all that part (i) uses.

The rewrite proofs are obtained from the equations in *Units*. Part (i) ensures that when C_k is processed, a rewrite proof of each body atom $A_{ki}\theta$ always exists. The algorithm finds it by matching $A_{ki}\theta$ directly with an electron in *Units* or by following *rewriting chains* extracted from the input proof. Note that the proof of Theorem 1 would also work if $\bigcup_{1\le j<k,\,1\le l\le m_j}\{A_{jl}\theta\} \cup \bigcup_{1\le l<i}\{A_{kl}\theta\}$ were omitted from the definition of U_{ki}. In the implementation, however, it is convenient to include these atoms because they sometimes lead to shorter proofs.

5 Implementation

Our approach is implemented in a tool called Tälja, developed in Haskell. The overall process is as follows: The E superposition prover, the Twee equational prover, or the Vampire superposition prover is invoked on a Horn problem. The output is a resolution or superposition proof in the CNF, FOF, or TFF subformat of TSTP (Thousands of Solutions from Theorem Provers) [38]. Tälja takes this output as its input and generates a proof in our proposed format. It can also output the proof as a TSTP derivation or as a Lean file. For a FOF or TFF conjecture of the form $A_1 \wedge \cdots \wedge A_m \to B$, the hypotheses $A_1, \ldots, A_m$ are listed with the axioms and B is the goal.

TSTP proofs generally do not record unifiers, so Tälja reconstructs θ by solving all inferences of the proof tree at once. Each inference is read in every way the prover may have performed it, that is, which literals were resolved, which subterm was rewritten, and in which orientation an equation was used. This includes an equality resolution the prover folds into the step, a rewrite at a position that is a variable in the premise, and two body atoms it merges before resolving. In a Twee step that rewrites atoms with an equation between atoms, each atom P is read as the equation $P \approx \text{true}$. An inference that no reading explains is left out, and variables that no inference binds become fresh constants.

The implementation follows Algorithm 1, with a few refinements. For each body atom, Tälja first tries the instances matched for earlier body atoms of the same nucleus, then the electron that the input proof itself used at this inference, then derived electrons, and finally axioms and lemmas. Moreover, matching is performed with backtracking. A body atom may match several electrons, and the step assembled from an early choice for A_i may be rejected by the validation described below. In that case, the algorithm retries with a different electron for A_i. In addition, each assembled hyperresolution step is validated before it is emitted. The applied rule, instantiated so that its body atoms coincide with the matched premises, must derive the emitted conclusion. When this check fails, the nucleus is skipped and its head is not stored. The translation fails only when a goal is then left without a proof. This does not contradict the correctness theorem, which assumes that θ is given, since the implementation has to reconstruct θ from the input proof. Finally, unreferenced lemmas are pruned.

The implementations of `try_chain` (Algorithm 2) and `get_chain` (Algorithm 1, line 28) both read the rewriting information present in the input proof. Resolution and superposition provers often simplify clauses by applying unit equalities before an inference step. These steps appear in the TSTP proof as forward or backward demodulation, rewriting, definition unfolding, or superposition inferences from a unit equation. For each leaf in the proof tree, Tälja extracts the sequence of such steps along the path to that leaf, yielding a *rewriting chain*. Each step is read from the conclusion of its inference in the input proof, which determines the applied equality, its instance, the direction, and the rewritten subterm. In `find_elec`, when no direct match succeeds, the candidate electron is rewritten along its chain and the match is retried. In `get_chain`, for equational goals $l \approx r$, the SplitChain construction of Denzinger and Schulz [15] splits the rewriting chain apart at the goal's proof-tree position into steps reducing the left-hand side l and steps reducing the right-hand side r, meeting at a common intermediate term t and yielding $l \to^* t \leftarrow^* r$.

6 Integration with Twee

Twee natively supports only equational problems, but it can also process Horn problems via an encoding [11], which we summarize here. Given a Horn problem, Twee first transforms it to eliminate predicate literals and clauses with more than one negative literal. Specifically, it transforms the predicate literal $\mathsf{P}(t_1, \ldots, t_n)$ into the equation $\mathsf{P}(t_1, \ldots, t_n) \approx \text{true}$, where true is a fresh constant, and transforms the clause $t_1 \approx u_1 \wedge \ldots \wedge t_n \approx u_n \to v \approx w$ into $\mathsf{tuple}_n(t_1, \ldots, t_n) \approx \mathsf{tuple}_n(u_1, \ldots, u_n) \to v \approx w$, where tuple_n is a fresh function symbol.

We are left with a Horn problem where all clauses either are unit equations or have the form $t \approx u \to v \approx w$. To

encode the remaining clauses, Twee introduces the function $\mathsf{ifeq}(x, y, z, w)$, which in the intended model expresses "if $x \approx y$ then z else w", and the following axiom:

$$\mathsf{ifeq}(x, x, y, z) \approx y \qquad \text{(IFEQ-ELIM)}$$

It then encodes $t \approx u \to v \approx w$ as the equation

$$\mathsf{ifeq}(t, u, v, w) \approx w \qquad \text{(IFEQ-INTRO)}$$

Notice that if we can prove $t \approx u$, then we can deduce $v \approx w$ as follows: $v \approx \mathsf{ifeq}(t, t, v, w) \approx \mathsf{ifeq}(t, u, v, w) \approx w$.

Example 10. Twee encodes the problem presented in Example 2 to the following equations and finds a proof similar to the following one. (Twee's proof is slightly different in that it does not introduce lemma 6.)

Axiom 0: $\mathsf{ifeq}(x, x, y, z) \approx y$

Axiom 1: $\mathsf{U}(\mathsf{b}) \approx \mathsf{true}$

Axiom 2: $\mathsf{ifeq}(\mathsf{U}(x), \mathsf{true}, \mathsf{Q}(x), \mathsf{true}) \approx \mathsf{true}$

Axiom 3: $\mathsf{S}(\mathsf{a}) \approx \mathsf{true}$

Axiom 4: $\mathsf{ifeq}(\mathsf{S}(x), \mathsf{true}, \mathsf{P}(x), \mathsf{true}) \approx \mathsf{true}$

Axiom 5: $\mathsf{ifeq}(\mathsf{tuple}(\mathsf{P}(x), \mathsf{Q}(y)), \mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{R}(x, y), \mathsf{true}) \approx \mathsf{true}$

Lemma 6: $\mathsf{Q}(\mathsf{b}) \approx \mathsf{true}$
Proof:
$\mathsf{Q}(\mathsf{b})$
$\approx$ { by axiom 0 }
$\mathsf{ifeq}(\mathsf{true}, \mathsf{true}, \mathsf{Q}(\mathsf{b}), \mathsf{true})$
$\approx$ { by axiom 1 }
$\mathsf{ifeq}(\mathsf{U}(\mathsf{b}), \mathsf{true}, \mathsf{Q}(\mathsf{b}), \mathsf{true})$
$\approx$ { by axiom 2 }
true

Goal 1: $\mathsf{R}(\mathsf{a}, \mathsf{b}) \approx \mathsf{true}$
Proof:
$\mathsf{R}(\mathsf{a}, \mathsf{b})$
$\approx$ { by axiom 0 }
$\mathsf{ifeq}(\mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{R}(\mathsf{a}, \mathsf{b}), \mathsf{true})$
$\approx$ { by axiom 4 }
$\mathsf{ifeq}(\mathsf{tuple}(\mathsf{ifeq}(\mathsf{S}(\mathsf{a}), \mathsf{true}, \mathsf{P}(\mathsf{a}), \mathsf{true}), \mathsf{true}), \mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{R}(\mathsf{a}, \mathsf{b}), \mathsf{true})$
$\approx$ { by axiom 3 }
$\mathsf{ifeq}(\mathsf{tuple}(\mathsf{ifeq}(\mathsf{true}, \mathsf{true}, \mathsf{P}(\mathsf{a}), \mathsf{true}), \mathsf{true}), \mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{R}(\mathsf{a}, \mathsf{b}), \mathsf{true})$
$\approx$ { by axiom 0 }
$\mathsf{ifeq}(\mathsf{tuple}(\mathsf{P}(\mathsf{a}), \mathsf{true}), \mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{R}(\mathsf{a}, \mathsf{b}), \mathsf{true})$
$\approx$ { by lemma 6 }
$\mathsf{ifeq}(\mathsf{tuple}(\mathsf{P}(\mathsf{a}), \mathsf{Q}(\mathsf{b})), \mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{R}(\mathsf{a}, \mathsf{b}), \mathsf{true})$
$\approx$ { by axiom 5 }
true ■

This proof is not entirely satisfactory since it reasons about the encoded equational problem rather than the original Horn problem. We would much prefer a proof similar to Example 2. Therefore, we have implemented a proof transformation that *decodes* a proof of an encoded problem back into a proof of the original problem. The decoded proof is based entirely on rewriting and resolution, so we can then use the algorithm of Section 4 to turn it into a human-readable proof.

We discuss completeness of our decoding algorithm in Section 6.3; we believe that it is incomplete but that it can be refined to be complete. In practice, it fails only rarely.

6.1 A Naive Approach

To motivate the decoding algorithm, we first show why a naive approach fails. Suppose that we define a function D that decodes an equation involving ifeq to its meaning as a Horn clause. We might then define a function that decodes a proof of $t \approx u$ to a proof of $D(t \approx u)$ by recursion on the proof structure. For example, suppose that the encoded proof ends in a transitivity step

$$\frac{t \approx u \qquad u \approx v}{t \approx v}$$

By recursion, we get proofs of $D(t \approx u)$ and $D(u \approx v)$, and we need to construct a proof of $D(t \approx v)$. That is, the following inference rule must be admissible:

$$\frac{D(t \approx u) \qquad D(u \approx v)}{D(t \approx v)}$$

If every inference rule is preserved by D in the sense above, this yields a decoding algorithm that is sound and complete.

Unfortunately, there seems to be no definition of D that makes this scheme work. To take a simple example, what should $\mathsf{ifeq}(s, t, u, v) \approx w$ decode to? Looking at equations (IFEQ-ELIM) and (IFEQ-INTRO), we should choose $s \approx t \to u \approx w$, since then (IFEQ-ELIM) decodes to the tautology $x \approx x \to y \approx y$, and (IFEQ-INTRO) correctly decodes to $t \approx u \to v \approx w$. But now take the following inference step from the proof of lemma 6, where $\mathbf{t}$ abbreviates the vector of terms $(\mathsf{U}(\mathsf{b}), \mathsf{true}, \mathsf{Q}(\mathsf{b}), \mathsf{true})$:

$$\frac{\mathsf{Q}(\mathsf{b}) \approx \mathsf{ifeq}(\mathbf{t}) \qquad \mathsf{ifeq}(\mathbf{t}) \approx \mathsf{true}}{\mathsf{Q}(\mathsf{b}) \approx \mathsf{true}}$$

Decoding this according to D gives

$$\frac{\mathsf{U}(\mathsf{b}) \approx \mathsf{true} \to \mathsf{Q}(\mathsf{b}) \approx \mathsf{Q}(\mathsf{b}) \quad \mathsf{U}(\mathsf{b}) \approx \mathsf{true} \to \mathsf{Q}(\mathsf{b}) \approx \mathsf{true}}{\mathsf{Q}(\mathsf{b}) \approx \mathsf{true}}$$

which is not a valid inference. Notice that the proof of the first premise uses the step $\mathsf{U}(\mathsf{b}) \approx \mathsf{true}$, which is exactly what we need to prove the conclusion by resolution. Unfortunately, D has transformed the first premise into a tautology, so this crucial fact has been lost.

6.2 A Decoding via Labeled Proofs

The problem is how to design D so that it preserves all information from the encoded equation. We first show how to do this for an equation that contains a single occurrence of ifeq. Call the equation $t \approx u[\mathsf{ifeq}(v_1, v_2, v_3, v_4)]$ and its decoding ϕ. The decoding is different depending on *which axiom* introduced the ifeq symbol:

- If it was (IFEQ-ELIM), then to have used it we must have $v_1 \approx v_2$, and it follows that $\mathsf{ifeq}(v_1, v_2, v_3, v_4) \approx v_3$. Hence we define ϕ as $v_1 \approx v_2 \wedge t \approx u[v_3]$.
- If it was an encoded Horn clause, then we must have $v_1 \approx v_2 \to v_3 \approx v_4$. By inspection of (IFEQ-INTRO), we also have $\mathsf{ifeq}(v_1, v_2, v_3, v_4) \approx v_4$. Hence we define ϕ as $(v_1 \approx v_2 \to v_3 \approx v_4) \wedge t \approx u[v_4]$.

Now the first premise of our example above decodes to $\mathsf{U(b)} \approx \mathsf{true} \wedge \mathsf{Q(b)} \approx \mathsf{Q(b)}$, the second decodes to $(\mathsf{U(b)} \approx \mathsf{true} \to \mathsf{Q(b)} \approx \mathsf{true}) \wedge \mathsf{true} \approx \mathsf{true}$, and the conclusion follows by resolution. This decoding works for any transitivity step $t \approx u[\mathsf{ifeq}(v_1, v_2, v_3, v_4)] \approx w$ involving a single ifeq:

- If both premises used (IFEQ-ELIM) to introduce the ifeq, then we have $t \approx u[v_3] \approx w$. Similarly, if they both used an encoded Horn clause, we have $t \approx u[v_4] \approx w$.
- Otherwise, we have (for example) $v_1 \approx v_2 \wedge t \approx u[v_3]$ from the first premise and $(v_1 \approx v_2 \to v_3 \approx v_4) \wedge u[v_4] \approx w$ from the second premise. We resolve to get $v_3 \approx v_4$ and rewrite to get $t \approx u[v_3] \approx u[v_4] \approx w$.

Now we show how to generalize this approach to proofs with an arbitrary number of ifeqs. The idea is to *label* each occurrence of ifeq with what sort of axiom was used to introduce it, and use that labeling to drive the decoding. Our algorithm starts by labeling the axioms: In axiom (IFEQ-ELIM), the ifeq is labeled with ELIM, and in encoded Horn clauses it is labeled with INTRO. For Example 10, this results in

Axiom 0: $\mathsf{ifeq}_{\text{ELIM}}(x, x, y, z) \approx y$
Axiom 2: $\mathsf{ifeq}_{\text{INTRO}}(\mathsf{U}(x), \mathsf{true}, \mathsf{Q}(x), \mathsf{true}) \approx \mathsf{true}$
Axiom 4: $\mathsf{ifeq}_{\text{INTRO}}(\mathsf{S}(x), \mathsf{true}, \mathsf{P}(x), \mathsf{true}) \approx \mathsf{true}$
Axiom 5: $\mathsf{ifeq}_{\text{INTRO}}(\mathsf{tuple}(\mathsf{P}(x), \mathsf{Q}(y)), \mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{R}(x, y), \mathsf{true}) \approx \mathsf{true}$

The algorithm then adjusts the proofs of the lemmas and goal so that they use the labeled versions of the axioms, which means that these proofs in turn become labeled. Sometimes this creates a *label clash*: a proof $t \approx u[\mathsf{ifeq}(\ldots)] \approx v$ where the $t \approx u$ step and $u \approx v$ step require different labels for the ifeq. We permit such label clashes.

Example 11. The labeled versions of lemma 6 and goal 1 are as follows:

Lemma 6: $\mathsf{Q(b)} \approx \mathsf{true}$
Proof:
$\mathsf{Q(b)}$
$\approx$ { by axiom 0 }
$\mathsf{ifeq}_{\text{ELIM}}(\mathsf{true}, \mathsf{true}, \mathsf{Q(b)}, \mathsf{true})$
$\approx$ { by axiom 1 }
$\mathsf{ifeq}_{\text{ELIM}}(\mathsf{U(b)}, \mathsf{true}, \mathsf{Q(b)}, \mathsf{true})$
{ label clash }
$\mathsf{ifeq}_{\text{INTRO}}(\mathsf{U(b)}, \mathsf{true}, \mathsf{Q(b)}, \mathsf{true})$
$\approx$ { by axiom 2 }
true

Goal 1: $\mathsf{R(a, b)} \approx \mathsf{true}$
Proof:
$\mathsf{R(a, b)}$
$\approx$ { by axiom 0 }
$\mathsf{ifeq}_{\text{ELIM}}(\mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{R(a, b)}, \mathsf{true})$
$\approx$ { by axiom 4 }
$\mathsf{ifeq}_{\text{ELIM}}(\mathsf{tuple}(\mathsf{ifeq}_{\text{INTRO}}(\mathsf{S(a)}, \mathsf{true}, \mathsf{P(a)}, \mathsf{true}), \mathsf{true}), \mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{R(a, b)}, \mathsf{true})$
$\approx$ { by axiom 3 }
$\mathsf{ifeq}_{\text{ELIM}}(\mathsf{tuple}(\mathsf{ifeq}_{\text{INTRO}}(\mathsf{true}, \mathsf{true}, \mathsf{P(a)}, \mathsf{true}), \mathsf{true}), \mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{R(a, b)}, \mathsf{true})$
{ label clash }
$\mathsf{ifeq}_{\text{ELIM}}(\mathsf{tuple}(\mathsf{ifeq}_{\text{ELIM}}(\mathsf{true}, \mathsf{true}, \mathsf{P(a)}, \mathsf{true}), \mathsf{true}), \mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{R(a, b)}, \mathsf{true})$
$\approx$ { by axiom 0 }
$\mathsf{ifeq}_{\text{ELIM}}(\mathsf{tuple}(\mathsf{P(a)}, \mathsf{true}), \mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{R(a, b)}, \mathsf{true})$
$\approx$ { by lemma 6 }
$\mathsf{ifeq}_{\text{ELIM}}(\mathsf{tuple}(\mathsf{P(a)}, \mathsf{Q(b)}), \mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{R(a, b)}, \mathsf{true})$
{ label clash }
$\mathsf{ifeq}_{\text{INTRO}}(\mathsf{tuple}(\mathsf{P(a)}, \mathsf{Q(b)}), \mathsf{tuple}(\mathsf{true}, \mathsf{true}), \mathsf{R(a, b)}, \mathsf{true})$
$\approx$ { by axiom 5 }
true ■

Formally, we model label clashes by admitting the following inference rule, as well as its symmetric variant, writing $\mathbf{v}$ to abbreviate the term vector (v_1, v_2, v_3, v_4):

$$\frac{t \approx u[\mathsf{ifeq}_{\text{ELIM}}(\mathbf{v})] \quad u[\mathsf{ifeq}_{\text{INTRO}}(\mathbf{v})] \approx w}{t \approx w}\ \text{label clash}$$

We claim that labeled proofs (with some caveats described in Section 6.3) can be decoded to first-order proofs based on rewriting and resolution. The main idea is that *label clashes correspond to resolution steps.* Every label clash becomes a resolution step, and all other inferences are passed through essentially unchanged. We start by defining the ifeq-erasure $E(t)$ of a term t as t with the following replacements made:

- Replace every subterm $\mathsf{ifeq}_{\text{ELIM}}(v_1, v_2, w_1, w_2)$ with w_1.
- Replace every subterm $\mathsf{ifeq}_{\text{INTRO}}(v_1, v_2, w_1, w_2)$ with w_2.

We lift E to formulas and to proofs. Notice that the ifeq-erasure of a proof is a valid equational proof *except* where a label clash occurs: the ifeq erases to a different argument on the two sides of the clash.

Next we define a decoding function D that takes a labeled equation and produces a set of Horn clauses; the meaning of an equation is the conjunction of this set. $D(t \approx u)$ consists of the following clauses:

- $E(t) \approx E(u)$;

- for every subterm of the form $\mathsf{ifeq}_{\text{ELIM}}(v_1, v_2, w_1, w_2)$, the clause $E(v_1) \approx E(v_2)$;
- for every subterm of the form $\mathsf{ifeq}_{\text{INTRO}}(v_1, v_2, w_1, w_2)$, the clause $E(v_1) \approx E(v_2) \to E(w_1) \approx E(w_2)$.

For example, for the label clash in lemma 6, we get

- $D(\mathsf{Q(b)} \approx \mathsf{ifeq}_{\text{ELIM}}(\mathsf{U(b)}, \mathsf{true}, \mathsf{Q(b)}, \mathsf{true})) = \{\mathsf{Q(b)} \approx \mathsf{Q(b)}, \mathsf{U(b)} \approx \mathsf{true}\}$ and
- $D(\mathsf{ifeq}_{\text{INTRO}}(\mathsf{U(b)}, \mathsf{true}, \mathsf{Q(b)}, \mathsf{true}) \approx \mathsf{true}) = \{\mathsf{true} \approx \mathsf{true}, \mathsf{U(b)} \approx \mathsf{true} \to \mathsf{Q(b)} \approx \mathsf{true}\}$

from which $\mathsf{Q(b)} \approx \mathsf{true}$ follows by resolution, as desired.

What remains is to decode the proofs. We start with the label clash rule. We must show that the following inference rule is admissible (abbreviating $\mathbf{v} = (v_1, v_2, v_3, v_4)$):

$$\frac{D(t \approx u[\mathsf{ifeq}_{\text{ELIM}}(\mathbf{v})]) \quad D(u[\mathsf{ifeq}_{\text{INTRO}}(\mathbf{v})] \approx w)}{D(t \approx w)}$$

The premises indeed entail $D(t \approx w)$, as the following argument shows. It suffices to prove $E(t) \approx E(w)$, since all other clauses of $D(t \approx w)$ are found in one of the premises: those involving t in the first premise, and those involving w in the second premise. $E(t) \approx E(w)$ can be proved as follows:

- From the first premise, we get $E(v_1) \approx E(v_2)$, and from the second, we get $E(v_1) \approx E(v_2) \to E(v_3) \approx E(v_4)$. Resolution gives $E(v_3) \approx E(v_4)$.
- From the first premise, we get $E(t) \approx E(u[v_3])$, and from the second, we get $E(u[v_4]) \approx E(w)$. Rewriting gives $E(t) \approx E(u[v_3]) \approx E(u[v_4]) \approx E(w)$.

To decode a label clash step, we simply construct this proof.

6.3 A Complication with Congruence

The remaining inference rules are those for equational logic. None of these induce resolution steps and, for the most part, an equational rewrite in the encoded proof turns into a rewrite or in some cases a set of parallel rewrites in the decoded proof. The main bookkeeping required is to keep track of which positions in $t \approx u$ correspond to which positions in $D(t \approx u)$, in order to apply rewrites that occur under the first two arguments of an ifeq.

There is one serious complication, and that has to do with the congruence rule. Suppose that we have the Horn clauses $t \approx t'$ and $t' \approx u \to v \approx w$, and consider the following labeled proof: $\mathsf{ifeq}_{\text{INTRO}}(t, u, v, w) \approx \mathsf{ifeq}_{\text{INTRO}}(t', u, v, w) \approx w$. This should decode to a proof of $t \approx u \to v \approx w$. The first step of the proof uses the congruence rule

$$\frac{t \approx t'}{\mathsf{ifeq}_{\text{INTRO}}(t, u, v, w) \approx \mathsf{ifeq}_{\text{INTRO}}(t', u, v, w)}$$

But this step cannot be decoded: The conclusion decodes to $w \approx w \wedge (t \approx u \to v \approx w) \wedge (t' \approx u \to v \approx w)$, but we have no justification for the two implications.

We believe that this can be solved by a more refined labeling scheme, where an ifeq that occurs only under a context would be labeled as inactive and decoded differently. For now, Twee instead uses an incomplete heuristic approach. Twee proves 907 non-purely-equational Horn TPTP problems (Section 7), 195 of them FOF, and this approach succeeds on all but two. Twee's approach has two parts. First, it handles congruence of ifeq lazily, only applying it in conjunction with transitivity or the label clash rule. In the example above, when decoding the first step, it will produce a proof of $t \approx t'$ and ignore the $\mathsf{ifeq}_{\text{INTRO}}$ context. Second, it uses context from neighboring steps to help with constructing proofs. In the example, once Twee gets to the second step of the proof, it applies the congruence rule to the first step to get $\mathsf{ifeq}_{\text{INTRO}}(t, u, v, w) \approx \mathsf{ifeq}_{\text{INTRO}}(t', u, v, w)$. To do this, it needs to prove the two implications above, which it does by using the proof of $t' \approx u \to v \approx w$ from the second step.

7 Evaluation

We evaluated Tälja on three categories of the TPTP v9.2.1 benchmark suite [38]: HNE (Horn, no equality), HEQ (Horn, with equality), and UEQ (unit equality). These categories contain 821, 452, and 1140 problems respectively, all in CNF. We also evaluated Tälja on the 408 FOF and 7 arithmetic-free TFF theorems of the suite whose axioms and conjecture are Horn clauses as stated. For each problem, we ran E, Twee, and Vampire with a 60-second timeout. E was run with its strategy schedule on one core; Vampire was run with AVATAR, its clause-splitting architecture, disabled; Twee was run with its default settings apart from the tuple encoding of Section 6 (option `--multi`). It does not support TFF. When a prover found a proof, the TSTP output was passed to Tälja, which was run with a 300-second timeout. Tälja, the benchmarking scripts, and the evaluation data are available as an artifact accompanying this paper.

The results are shown in Table 1. "Proved" is the number of problems each prover solved within the time limit. "Translated" is the number of proved problems for which Tälja produced a complete proof within the time limit.

In total, of the three proofs that Tälja did not translate, one exceeded the 300-second translation timeout. It is a Vampire refutation of about a million resolution steps. In the remaining two cases, Twee's proof output component failed.

To validate the proofs, we checked the Lean [13] files output by Tälja. Each step is an explicit proof term, such as an axiom applied to its premises or a congruence for a rewrite. All translated proofs were accepted by the Lean kernel. Tälja can also output a proof as a TSTP derivation, which we checked with GDV [37] for all translated FOF proofs. GDV verified all of them.

8 Related Work

The main inspiration for our proof format and translation algorithm is Denzinger and Schulz's [15] work on the DISCOUNT prover. Its output format is based on lemmas and

Table 1. Evaluation of Tälja on TPTP benchmarks

Set	Prover	Proved	Translated
HNE	E	757	757
	Twee	490	490
	Vampire	722	721
HEQ	E	386	386
	Twee	222	221
	Vampire	360	360
UEQ	E	831	831
	Twee	832	832
	Vampire	736	736
FOF	E	290	290
	Twee	195	194
	Vampire	244	244
TFF	E	6	6
	Vampire	6	6
Total	E	2270	2270
	Twee	1739	1737
	Vampire	2068	2067

equational chains and was adopted by Waldmeister and Twee. Denzinger and Schulz also designed sophisticated heuristics for determining which proof steps should be turned into lemmas. A simple criterion, employed in Twee, is to turn steps that are referenced more than once into lemmas and only those.

In the context of first-order logic without equality, the question of generating readable proofs has been answered multiple times, often with natural deduction as the target. Natural deduction is widely recognized as human-readable, although derivations tend to be very detailed. Miller and Felty [27] gave an early account of such a translation, representing natural deduction proofs as typed λ-terms whose types come from the corresponding resolution refutations. Lingenfelder [25] designed a translation from refutation graph proofs, which can be obtained from resolution proofs, to natural deduction. Huang [21, 22] introduced the notion of assertions to compress natural deduction proof patterns that humans would consider trivial into single steps. Derivations in this system mostly consist of the application of definitions or lemmas. Huang presented two translations, one from standard natural deduction proofs to assertion-level proofs [21], and another from resolution proofs to assertion-level proofs [22]. This line of research culminated with Meier's TRAMP proof translator [26], which implements a variant of Huang's second translation.

For propositional logic, Amjad's [2] translation converted SAT solver resolution-based refutations into natural deduction proofs in linear time, handling refutations with millions of inference steps and producing derivations that could be checked faster than the original refutations.

Orthogonally, De Nivelle [14] designed a translation that transforms resolution proofs containing Skolem symbols to natural deduction proofs without Skolem symbols. This approach is useful to reconstruct proofs in proof assistants that do not include choice axioms; it can also increase the proofs' readability.

In higher-order logic, Andrews [3] designed a translation from refutation matings, which are related to the connection calculus and hence tableaux, to natural deduction proofs. The TPS prover [4] applies this translation to the matings it finds to provide more readable justifications to its users.

Relatedly, the GAPT system [18] was designed to reconstruct a readable proof from an ATP proof using its *expansion proof*, the record of the ground instances used for each quantifier, via a tableau-style search. From there, Ebner and Schlaipfer [19] turned such proofs into natural deduction, often without needing excluded middle. The same ground instances were used by Ebner et al. [17] to find quantified lemmas. Finally, Ebner [16] adapted the search to intuitionistic logic and proved that it always succeeds on Horn problems.

Šinkarovs and Rawson [34] connect the Agda proof assistant to Vampire. On a logical fragment shared by both, Agda goals are passed to Vampire, and the classical proofs it finds are turned into constructive proof terms that Agda type-checks.

Logicians have also studied proof translations for a different reason, asking how they affect proof size rather than readability. Cook and Reckhow's [12] notion of polynomial simulation lets one propositional proof system be judged more powerful than another based on how compactly it can express the same proofs. Nordström's [28] survey traces the consequences of this idea for SAT solving, connecting size lower bounds for resolution proofs to hard limits on how fast resolution-based solvers can run.

9 Conclusion

We introduced a format for capturing resolution and superposition proofs of problems in the Horn fragment of first-order logic. A proof in this format consists of a list of axioms and lemmas ending with the goals. Each lemma or goal is proved either by a chain of equalities or by a "have/hence" block performing hyperresolution steps. Such proofs are arguably more readable than the cryptic proofs by contradiction generated by resolution and superposition provers.

In addition, we designed a translation algorithm from resolution and superposition proofs to our format. We implemented it as a new tool, Tälja, connected it to E and Vampire, and extended Twee's output to interface with it. An evaluation on a variety of TPTP benchmarks showed that our tool translates nearly every proof the three provers find.

Our approach's main limitation is its restriction to Horn clauses. As future work, we want to extend the approach to non-Horn first-order problems and to higher-order logic. We would also like to generate calculation chains for binary predicates other than equality, such as inequalities, as is customary in pen-and-paper mathematics. We also plan to combine Tälja with Krympa [23], a tool for minimizing equational proofs, to produce shorter equality chains.

Acknowledgments

We thank Julius Bernert, Gabriel Ebner, Jan Johannsen, Stephan Schulz, Helmut Schwichtenberg, and Geoff Sutcliffe for suggesting several of the references mentioned in Section 8. We thank Michael Rawson for helpful discussions. We thank Moritz Roos for testing the artifact.

This research was cofunded by the European Union (ERC, Nekoka, 101083038), by the Deutsche Forschungsgemeinschaft (DFG, ConVeY, 378803395), and by the Swedish Research Council (VR, Semantically-guided theorem proving for mathematics, 2025-06153). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Jesse Alama. 2012. Escape to Mizar for ATPs. *CoRR* abs/1204.6615 (2012). arXiv:1204.6615 doi:10.48550/arXiv.1204.6615

[2] Hasan Amjad. 2007. A Compressing Translation from Propositional Resolution to Natural Deduction. In *FroCoS 2007 (LNCS, Vol. 4720)*, Boris Konev and Frank Wolter (Eds.). Springer, 88–102. doi:10.1007/978-3-540-74621-8_6

[3] Peter B. Andrews. 1980. Transforming Matings into Natural Deduction Proofs. In *CADE-5 (LNCS, Vol. 87)*, Wolfgang Bibel and Robert A. Kowalski (Eds.). Springer, 281–292. doi:10.1007/3-540-10009-1_22

[4] Peter B. Andrews and Chad E. Brown. 2006. TPS: A Hybrid Automatic-Interactive System for Developing Proofs. *J. Appl. Log.* 4, 4 (2006), 367–395. doi:10.1016/J.JAL.2005.10.002

[5] Jürgen Avenhaus, Jörg Denzinger, and Matthias Fuchs. 1995. DISCOUNT: A System for Distributed Equational Deduction. In *RTA-95 (LNCS, Vol. 914)*, Jieh Hsiang (Ed.). Springer, 397–402. doi:10.1007/3-540-59200-8_72

[6] Leo Bachmair and Harald Ganzinger. 1994. Rewrite-Based Equational Theorem Proving with Selection and Simplification. *J. Log. Comput.* 4, 3 (1994), 217–247. doi:10.1093/LOGCOM/4.3.217

[7] Leo Bachmair and Harald Ganzinger. 2001. Resolution Theorem Proving. In *Handbook of Automated Reasoning*, John Alan Robinson and Andrei Voronkov (Eds.). Vol. I. Elsevier and MIT Press, 19–99. doi:10.1016/B978-044450813-3/50004-7

[8] Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, and Petar Vukmirovic. 2023. Superposition for Higher-Order Logic. *J. Autom. Reason.* 67, 1 (2023), article number 10. doi:10.1007/S10817-022-09649-9

[9] Ahmed Bhayat and Martin Suda. 2024. A Higher-Order Vampire (Short Paper). In *IJCAR 2024 (LNCS, Vol. 14739)*, Christoph Benzmüller, Marijn J. H. Heule, and Renate A. Schmidt (Eds.). Springer, 75–85. doi:10.1007/978-3-031-63498-7_5

[10] Jasmin Christian Blanchette, Sascha Böhme, Mathias Fleury, Steffen Juilf Smolka, and Albert Steckermeier. 2016. Semi-intelligible Isar Proofs from Machine-Generated Proofs. *J. Autom. Reason.* 56, 2 (2016), 155–200. doi:10.1007/S10817-015-9335-3

[11] Koen Claessen and Nicholas Smallbone. 2018. Efficient Encodings of First-Order Horn Formulas in Equational Logic. In *IJCAR 2018 (LNCS, Vol. 10900)*, Didier Galmiche, Stephan Schulz, and Roberto Sebastiani (Eds.). Springer, 388–404. doi:10.1007/978-3-319-94205-6_26

[12] Stephen A. Cook and Robert A. Reckhow. 1979. The Relative Efficiency of Propositional Proof Systems. *J. Symb. Log.* 44, 1 (1979), 36–50. doi:10.2307/2273702

[13] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. 2015. The Lean Theorem Prover (System Description). In *CADE-25 (LNCS, Vol. 9195)*, Amy P. Felty and Aart Middeldorp (Eds.). Springer, 378–388. doi:10.1007/978-3-319-21401-6_26

[14] Hans de Nivelle. 2005. Translation of Resolution Proofs into Short First-Order Proofs without Choice Axioms. *Inf. Comput.* 199, 1–2 (2005), 24–54. doi:10.1016/J.IC.2004.10.011

[15] Jörg Denzinger and Stephan Schulz. 1994. Recording, Analyzing and Presenting Distributed Deduction Processes. In *PASCO 1994*, Hoon Hong (Ed.). World Scientific, 114–123. doi:10.1142/9789814533584_0012

[16] Gabriel Ebner. 2019. Herbrand Constructivization for Automated Intuitionistic Theorem Proving. In *TABLEAUX 2019 (LNCS, Vol. 11714)*, Serenella Cerrito and Andrei Popescu (Eds.). Springer, 355–373. doi:10.1007/978-3-030-29026-9_20

[17] Gabriel Ebner, Stefan Hetzl, Alexander Leitsch, Giselle Reis, and Daniel Weller. 2019. On the Generation of Quantified Lemmas. *J. Autom. Reason.* 63, 1 (2019), 95–126. doi:10.1007/S10817-018-9462-8

[18] Gabriel Ebner, Stefan Hetzl, Giselle Reis, Martin Riener, Simon Wolfsteiner, and Sebastian Zivota. 2016. System Description: GAPT 2.0. In *IJCAR 2016 (LNCS, Vol. 9706)*, Nicola Olivetti and Ashish Tiwari (Eds.). Springer, 293–301. doi:10.1007/978-3-319-40229-1_20

[19] Gabriel Ebner and Matthias Schlaipfer. 2018. Efficient Translation of Sequent Calculus Proofs Into Natural Deduction Proofs. In *PAAR 2018 (CEUR Workshop Proceedings, Vol. 2162)*, Boris Konev, Josef Urban, and Philipp Rümmer (Eds.). CEUR-WS.org, 17–33.

[20] Thomas Hillenbrand, Arnim Buch, Roland Vogt, and Bernd Löchner. 1997. WALDMEISTER - High-Performance Equational Deduction. *J. Autom. Reason.* 18, 2 (1997), 265–270. doi:10.1023/A:1005872405899

[21] Xiaorong Huang. 1994. Reconstructing Proofs at the Assertion Level. In *CADE-12 (LNCS, Vol. 814)*, Alan Bundy (Ed.). Springer, 738–752. doi:10.1007/3-540-58156-1_53

[22] Xiaorong Huang. 1996. Translating Machine-Generated Resolution Proofs into ND-Proofs at the Assertion Level. In *PRICAI '96 (LNCS, Vol. 1114)*, Norman Y. Foo and Randy Goebel (Eds.). Springer, 399–410. doi:10.1007/3-540-61532-6_34

[23] Lydia Kondylidou, Jasmin Blanchette, and Marijn J. H. Heule. 2026. Tao's Equational Proof Challenge Accepted. In *IJCAR 2026 (LNCS, Vol. 16688)*, Armin Biere, Carsten Lutz, and Sara Negri (Eds.). Springer, 3–21. doi:10.1007/978-3-032-32589-1_1

[24] Laura Kovács and Andrei Voronkov. 2013. First-Order Theorem Proving and Vampire. In *CAV 2013 (LNCS, Vol. 8044)*, Natasha Sharygina and Helmut Veith (Eds.). Springer, 1–35. doi:10.1007/978-3-642-39799-8_1

[25] Christoph Lingenfelder. 1989. Structuring Computer Generated Proofs. In *IJCAI 1989*, N. S. Sridharan (Ed.). Morgan Kaufmann, 378–383. http://ijcai.org/Proceedings/89-1/Papers/060.pdf

[26] Andreas Meier. 2000. System Description: TRAMP: Transformation of Machine-Found Proofs into ND-Proofs at the Assertion Level. In *CADE-17 (LNCS, Vol. 1831)*, David A. McAllester (Ed.). Springer, 460–464. doi:10.1007/10721959_37

[27] Dale Miller and Amy P. Felty. 1986. An Integration of Resolution and Natural Deduction Theorem Proving. In *AAAI 1986*, Tom Kehler (Ed.). Morgan Kaufmann, 198–202. http://www.aaai.org/Library/AAAI/1986/aaai86-032.php

[28] Jakob Nordström. 2015. On the interplay between proof complexity and SAT solving. *ACM SIGLOG News* 2, 3 (2015), 19–44. doi:10.1145/2815493.2815497

[29] George A. Robinson and Lawrence Wos. 1983. Paramodulation and Theorem-Proving in First-Order Theories with Equality. In *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, Jörg H. Siekmann and Graham Wrightson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 298–313. doi:10.1007/978-3-642-81955-1_19

[30] John Alan Robinson. 1965. Automatic Deduction with Hyper-Resolution. *International Journal of Computer Mathematics* 1, 3 (1965), 227–234. doi:10.1080/00207166508803260

[31] John Alan Robinson. 1965. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM* 12, 1 (1965), 23–41. doi:10.1145/321250.321253

[32] Stephan Schulz. 2002. E—a Brainiac Theorem Prover. *AI Commun.* 15, 2-3 (2002), 111–126. http://content.iospress.com/articles/ai-communications/aic260

[33] Stephan Schulz and Adam Pease. 2020. Teaching Automated Theorem Proving by Example: PyRes 1.2 (System Description). In *IJCAR 2020 (LNCS, Vol. 12167)*, Nicolas Peltier and Viorica Sofronie-Stokkermans (Eds.). Springer, 158–166. doi:10.1007/978-3-030-51054-1_9

[34] Artjoms Šinkarovs and Michael Rawson. 2026. When Agda met Vampire. *CoRR* abs/2602.18844 (2026). arXiv:2602.18844 doi:10.48550/ARXIV.2602.18844

[35] Nicholas Smallbone. 2021. Twee: An Equational Theorem Prover. In *CADE-28 (LNCS, Vol. 12699)*, André Platzer and Geoff Sutcliffe (Eds.). Springer, 602–613. doi:10.1007/978-3-030-79876-5_35

[36] Alexander Steen and Christoph Benzmüller. 2021. Extensional Higher-Order Paramodulation in Leo-III. *J. Autom. Reason.* 65, 6 (2021), 775–807. doi:10.1007/S10817-021-09588-X

[37] Geoff Sutcliffe. 2006. Semantic Derivation Verification: Techniques and Implementation. *Int. J. Artif. Intell. Tools* 15, 6 (2006), 1053–1070. doi:10.1142/S0218213006003119

[38] Geoff Sutcliffe. 2024. Stepping Stones in the TPTP World. In *IJCAR 2024 (LNCS, Vol. 14739)*, Christoph Benzmüller, Marijn J. H. Heule, and Renate A. Schmidt (Eds.). Springer, 30–50. doi:10.1007/978-3-031-63498-7_3

[39] Markus Wenzel. 1999. Isar - A Generic Interpretative Approach to Readable Formal Proof Documents. In *TPHOLs 1999 (LNCS, Vol. 1690)*, Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin-Mohring, and Laurent Théry (Eds.). Springer, 167–183. doi:10.1007/3-540-48256-3_12