Exploiting instantiations from paramodulation proofs in Isabelle/HOL

Lukas Bartl, Jasmin Blanchette, Tobias Nipkow

Angaben zur Veröffentlichung / Publication details:

Bartl, Lukas, Jasmin Blanchette, and Tobias Nipkow. 2025. "Exploiting instantiations from paramodulation proofs in Isabelle/HOL." In *Automated Deduction –CADE 30: 30th International Conference on Automated Deduction, Stuttgart, Germany, July 28-31, 2025, proceedings*, edited by Clark Barrett and Uwe Waldmann, 573–93. Berlin: Springer. https://doi.org/10.1007/978-3-031-99984-0_30.

Exploiting Instantiations from Paramodulation Proofs in Isabelle/HOL

Lukas Bartl[1(✉)], Jasmin Blanchette[2], and Tobias Nipkow[3]

[1] Universität Augsburg, Augsburg, Germany
lukas.bartl@uni-a.de
[2] Ludwig-Maximilians-Universität München, Munich, Germany
jasmin.blanchette@ifi.lmu.de
[3] Technische Universität München, Munich, Germany
nipkow@in.tum.de

Abstract. Metis is an ordered paramodulation prover built into the Isabelle/HOL proof assistant. It attempts to close the current goal using a given list of lemmas. Typically these lemmas are found by Sledgehammer, a tool that integrates external automatic provers. We present a new tool that analyzes successful Metis proofs to derive variable instantiations. These increase Sledgehammer's success rate, improve the speed of Sledgehammer-generated proofs, and help users understand why a goal follows from the lemmas.

Keywords: instantiations · paramodulation · Isabelle/HOL · Sledgehammer

1 Introduction

Sledgehammer [31] is undoubtedly one of the interactive proof assistant Isabelle/HOL's [27] most popular components. This component gives Isabelle access to many external automatic theorem provers (ATPs), which can mechanize routine proofs [12,16]. When the user invokes Sledgehammer on a proposition to be proved, it performs the following steps:

1. It invokes a number of external ATPs with the goal and the background library of *facts* (definitions, lemmas, theorems, etc.); this involves translating the goal and facts from higher-order logic to the ATPs' logics.
2. For any proof that is found, the used facts are extracted from it and the internal prover *Metis* [20], based on ordered paramodulation, is invoked to reconstruct an Isabelle proof based on this list of facts only.

This paper is about obtaining and exploiting the instantiations of the facts used in the proofs. Our motivation is that using instantiated facts

C. Barrett and U. Waldmann (Eds.): CADE 2025, LNAI 15943, pp. 573–593, 2025.
https://doi.org/10.1007/978-3-031-99984-0_30

- can speed up proof reconstruction (and even turn timeouts into successes);
- can improve readability of proofs for humans; and
- can lead to simpler proofs.

Example 1. We start with a simple example where we believe human readability is improved. We work in a linear order with $\bot$ and $\top$. Sledgehammer proves that in the context of the facts

$$\top \not< x \ (1) \qquad x \neq \bot \leftrightarrow \bot < x \ (2)$$

the assumption $\bot \not< \top$ implies $\mathtt{a} = \mathtt{b}$, where $\mathtt{a}$ and $\mathtt{b}$ are arbitrary constants. Why is that? It turns out that the following instantiations are used:

$$\top \not< \mathtt{a} \ (1\mathtt{a}) \qquad \top \not< \mathtt{b} \ (1\mathtt{b})$$
$$\mathtt{a} \neq \bot \leftrightarrow \bot < \mathtt{a} \ (2\mathtt{a}) \qquad \mathtt{b} \neq \bot \leftrightarrow \bot < \mathtt{b} \ (2\mathtt{b}) \qquad \top \neq \bot \leftrightarrow \bot < \top \ (2\top)$$

Clearly $\bot \not< \top$ implies $\bot = \top$ via $(2\top)$. This implies $\bot \not< \mathtt{a}$ and $\bot \not< \mathtt{b}$ via $(1\mathtt{a})$ and $(1\mathtt{b})$. Now we obtain $\mathtt{a} = \bot$ and $\mathtt{b} = \bot$ via $(2\mathtt{a})$ and $(2\mathtt{b})$, and thus $\mathtt{a} = \mathtt{b}$. Although this deduction still needs some thought, we believe that it is far simpler than if we had to work out the derivation from (1) and (2) alone.

This paper is structured as follows. Section 2 presents the necessary background about Isabelle, Sledgehammer, and Metis. Section 3 presents further introductory examples. The core of the paper is in Sect. 4, which explains how to obtain the instantiations used during a Metis proof and translate them into instantiations of the corresponding Isabelle facts. Section 5 explains how Sledgehammer was extended to cooperate with the extended Metis prover. Section 6 presents our empirical evaluation of how instantiations improve the performance of Sledgehammer. We end with related work and the conclusion.

Our extensions are available as part of Isabelle starting with version 2025.[1] They are documented in the Sledgehammer user's manual [5]. The raw data for our evaluation is available online.[2]

2 Background

Isabelle. Isabelle/HOL [27,41] is a proof assistant for polymorphic higher-order logic enriched with type classes [34]. It is written primarily in Standard ML. It has an inference kernel through which all logical inferences must go to be deemed acceptable. At the user level, notations largely follow mathematical practice.

Sledgehammer. The Sledgehammer tool [31] consists of six main components:

1. The *relevance filter* (or "premise selector") heuristically selects a subset of the available facts as likely relevant to the current goal. Typically, hundreds of

[1] https://isabelle.in.tum.de/website-Isabelle2025/.
[2] https://nekoka-project.github.io/pubs/instantiations_data.zip.

facts can be chosen without overwhelming ATPs. Sledgehammer includes two relevance filters [8,23], which can be combined.

2. The *translation module* constructs an ATP problem from the selected facts and the current goal, translating Isabelle's polymorphic higher-order logic to the ATP's logic [7,22].
3. The ATP tries to prove the problem. Actually, multiple ATPs can be run in parallel. Commonly used ATPs include cvc5 [3], E [39], Leo-III [35], SPASS [10], Vampire [4], veriT [13], Z3 [25], and Zipperposition [38].
4. If one or more ATPs find a proof, the *proof minimization module* repeatedly invokes each ATP with subsets of the facts referenced in the respective proof, trying to reduce the number of dependencies and speed up the next steps.
5. The *proof reconstruction module* transforms each ATP proof into a textual Isabelle proof. Reconstruction means that Sledgehammer and the ATP need not be trusted. Typically, the structure of the ATP proof is discarded, and the Isabelle proof consists of a single proof method (often *metis*) invoked with the facts referenced in the ATP proof. Detailed Isabelle proofs, or Isar proofs [40], are available as an experimental feature [6].
6. For each ATP proof, the *preplay module* tries out various proof methods before they are presented to the user. If several methods succeed, the fastest one is chosen [6].

As an example, suppose Sledgehammer selects 512 facts $f_1, \ldots, f_{512}$ and passes them, along with the goal, to E. Then E finds a proof involving three facts, f_{10}, f_{73}, and f_{359}, and minimization reduces this list to two: f_{10} and f_{359}. By trial and error (and preplaying), Sledgehammer determines that the proof method *metis* with f_{10} and f_{359} as arguments solves the Isabelle goal in 23 ms, and no other proof method succeeds, so this *metis* call is suggested to the user.

The Metis ATP. Metis [20] is an ATP for untyped first-order logic with equality written in Standard ML. It is based on ordered paramodulation, a variant of superposition [2]. Although Metis was developed as a standalone program, it is also incorporated in Isabelle's source code, so that it is always available.

Thanks to its calculus, Metis is reasonably performant, although it cannot compete with state-of-the-art superposition provers such as Vampire [36]. Metis's main strengths are the readability of its source code and the simplicity and fine granularity of its proof format. Proofs are expressed using the following six inference rules:

$$\frac{}{C}\ \textsc{Axiom} \qquad \frac{}{A \vee \neg A}\ \textsc{Assume} \qquad \frac{C}{C\sigma}\ \textsc{Subst}$$

$$\frac{}{t = t}\ \textsc{Refl} \qquad \frac{}{s \neq t \vee \neg L[s]_p \vee L[t]_p}\ \textsc{Equality} \qquad \frac{C \vee A \quad \neg A \vee D}{C \vee D}\ \textsc{Resolve}$$

Notice that substitution is captured by the explicit SUBST rule instead of being part of RESOLVE. Moreover, all equality reasoning is reduced to REFL and EQUALITY. For proof search, Metis relies on a more efficient calculus that performs ordered paramodulation, but the proofs are then translated to the above fine-granular rules.

The *metis* Proof Method. Isabelle's *metis* (with a lowercase m) proof method [32] builds on the Metis (with an uppercase M) ATP to provide general-purpose proof automation. The *metis* proof method takes a list of facts as argument and translates them, together with the current goal, from polymorphic higher-order logic to untyped first-order logic. The resulting *axiom clauses* are then introduced in the proof attempt using the AXIOM rule. The translation uses the same techniques (and the same code) as Sledgehammer. Next, *metis* invokes the Metis ATP, and if Metis finds a proof, it is reconstructed step by step using Isabelle's inference kernel, so that the goal becomes an Isabelle theorem. A *metis* call is considered successful if the Metis ATP found a proof and Isabelle reconstructed it.

Since it may be hard for the user to determine which facts are necessary for a proof, in practice *metis* is almost always used in conjunction with Sledgehammer.

3 Examples

Before we study our *metis* and Sledgehammer extensions in detail, we take a look at a few examples that illustrate how instantiating facts can not only improve readability but also speed up proof reconstruction and lead to simpler proofs. The final example demonstrates how more complex terms are displayed.

Example 2. We present a goal from the *Archive of Formal Proofs* [9], where instantiating facts substantially speeds up the proof and turns a timeout into a success. This example stems from a formalization of Tarski's axioms for Euclidean geometry [15]. The lemma *cong_mid2__cong* states the following property, where `Cong` denotes the congruence relation and `Midpoint` states that a point lies exactly in the middle of two other points:

$$\texttt{Midpoint M A B} \longrightarrow \texttt{Midpoint M' A' B'} \longrightarrow \texttt{Cong A M A' M'} \longrightarrow \texttt{Cong A B A' B'}$$

When invoking Sledgehammer on a modern laptop for this goal, the external ATP cvc5 [3] finds a proof using the facts *cong_inner_transitivity*, *l2_l1_b*, *midpoint_bet*, and *midpoint_cong*. However, *metis* times out; i.e., it fails to find a proof derived from these facts within 1 s. After inferring the instantiations and instantiating the facts, though, *metis* successfully solves the goal in 50 ms. The instantiations are simple: Each free variable in the facts is replaced by one of $\texttt{M}, \texttt{A}, \texttt{B}, \texttt{M}', \texttt{A}', \texttt{B}'$. The facts *midpoint_bet* and *midpoint_cong* each have two different instantiations and thus appear twice in the resulting Isabelle proof, which may also improve readability.

Example 3. Instantiating facts not only speeds up *metis* but also leads to simpler and more readable proofs. Suppose that we want to prove the goal $0 < \mathtt{i} \longrightarrow \mathtt{x} \leq \mathtt{x}^{\mathtt{i}}$ for natural numbers $\mathtt{i}, \mathtt{x}$. When invoking Sledgehammer on a modern laptop, the external ATP veriT [13] finds a proof including the following facts, where `Suc` returns the successor of its argument:

$$\begin{aligned} 1 &= \mathtt{Suc}\ 0 && (\mathit{nat\_1}) \\ a \leq 0 &\longrightarrow a = 0 && (\mathit{bot\_nat\_0.extremum\_uniqueI}) \\ m \not\leq n &\longleftrightarrow \mathtt{Suc}\ n \leq m && (\mathit{not\_less\_eq\_eq}) \\ 1 \leq a \longrightarrow 0 < n &\longrightarrow a \leq a^n && (\mathit{self\_le\_power}) \\ 0 \leq a &\longrightarrow 0 \leq a^n && (\mathit{zero\_le\_power}) \end{aligned}$$

The *metis* method can solve the goal with these facts in 82 ms. However, after inferring the instantiations and instantiating the facts, *metis* needs only 41 ms to solve the goal, and the *auto* [29] proof method is even faster, needing only 7 ms. Additionally, *auto* requires only the following two instantiated facts:

$$\begin{aligned} \mathtt{x} \not\leq 0 &\longleftrightarrow \mathtt{Suc}\ 0 \leq \mathtt{x} && (\mathit{not\_less\_eq\_eq} \text{ with } \{m \mapsto \mathtt{x}, n \mapsto 0\}) \\ 1 \leq \mathtt{x} \longrightarrow 0 < \mathtt{i} &\longrightarrow \mathtt{x} \leq \mathtt{x}^{\mathtt{i}} && (\mathit{self\_le\_power} \text{ with } \{a \mapsto \mathtt{x}, n \mapsto \mathtt{i}\}) \end{aligned}$$

By contrast, *auto* fails to find a proof derived from these facts without instantiations. The Isabelle proof using *auto* is simpler and more readable than the *metis* proof, since trivial facts were eliminated. This helps the user focus on the relevant facts when trying to understand the proof.

Example 4. In the preceding examples, the instantiations were simple, with each free variable being replaced by a single symbol. There are also proofs where instantiations are more complex, possibly involving λ-abstractions and quantified variables. Consider the goal

$$\mathtt{surj}\ (\lambda n.\ \mathtt{g}\ (\mathtt{Suc}\ n)) \longrightarrow (\exists m.\ \mathtt{P}\ (\mathtt{Suc}\ (\mathtt{g}\ m))) \longrightarrow (\exists n.\ \mathtt{P}\ (\mathtt{g}\ (\mathtt{Suc}\ n)))$$

where `surj` states that a function is surjective. We have the fact $\mathtt{surj}\ f \longrightarrow (\exists x.\ f\ x = y)$ at our disposal. Our *metis* extension suggests instantiating f with $\lambda c.\ \mathtt{g}\ (\mathtt{Suc}\ c)$ and y with $\mathtt{Suc}\ (\mathtt{g}\ \_)$ in the fact. Notice that bound variables can be renamed in the inferred instantiations (here, n became c). Additionally, '_'—which corresponds to a fresh free variable—can appear as a placeholder for quantified variables (e.g., m), since there is no way to refer to them.

4 Instantiations from Metis Proofs

We extend the *metis* proof method so that it infers the instantiations of facts used in a proof and suggests them to the user. This extension is executed after a successful *metis* proof, provided that the *metis_instantiate* option is enabled.

Inference of Instantiations for Metis Proofs. To infer the instantiations of the Isabelle facts, we first infer the instantiations of the Metis clauses. These are the substitutions that were applied to the axiom clauses in the Metis proof.

Definition 1. A *Metis theorem* θ is recursively defined as a triple $(C, r, \bar{\theta})$ consisting of a Metis clause C, an inference rule r, and a list of Metis theorems $\bar{\theta}$, where C is derived directly from $\bar{\theta}$ using the rule r. A *Metis proof* is a Metis theorem for the empty clause, denoted by `False`.

Definition 2. A *Metis substitution* $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is a partial function from Metis variable names to Metis terms. The *composition* of two substitutions $\sigma_1 \circ \sigma_2$ first applies σ_2 and then σ_1, so that $C(\sigma_1 \circ \sigma_2) = (C\sigma_2)\sigma_1$.

Definition 3. The function $\text{infer}(\theta, \sigma)$ returns a list of pairs (C, σ'), where C is an axiom clause in the Metis theorem θ and σ' is the applied substitution. The argument σ is used as an accumulator and is initialized to $\varnothing$ for a Metis proof. The function is defined recursively as follows:

$$\text{infer}((C, r, []), \sigma) = \begin{cases} [(C, \sigma)] & \text{if } r = \textsc{Axiom} \\ [] & \text{otherwise} \end{cases}$$

$$\text{infer}((C, \textsc{Subst}, [\theta]), \sigma) = \text{infer}(\theta, \sigma \circ \sigma')$$

$$\text{infer}((C, \textsc{Resolve}, [\theta_1, \theta_2]), \sigma) = \text{infer}(\theta_1, \sigma) \mathbin{@} \text{infer}(\theta_2, \sigma)$$

where σ' in the second equation is the substitution carried by the Subst inference rule on the left, and @ in the third equation denotes list concatenation.

Example 5. Suppose that we want to prove the goal $1 <$ `Suc` (`Suc x`) on natural numbers from the following facts:

$$m < n \longrightarrow \texttt{Suc}\ m < \texttt{Suc}\ n \qquad \texttt{Suc}\ 0 = 1 \qquad 0 < \texttt{Suc}\ n$$

We translate the goal and the facts into first-order logic by using the predicate symbol `less` for $<$. Metis finds the following proof:

(1) Axiom: $\neg\texttt{less}(1, \texttt{Suc}(\texttt{Suc}(\texttt{x})))$

(2) Axiom: $\neg\texttt{less}(m, n) \vee \texttt{less}(\texttt{Suc}(m), \texttt{Suc}(n))$

(3) Subst from (2) using $\sigma = \{m \mapsto 0, n \mapsto y\}$:
$\neg\texttt{less}(0, y) \vee \texttt{less}(\texttt{Suc}(0), \texttt{Suc}(y))$

(4) Axiom: $\texttt{Suc}(0) = 1$

(5) Equality: $\texttt{Suc}(0) \neq 1 \vee \neg\texttt{less}(\texttt{Suc}(0), \texttt{Suc}(y)) \vee \texttt{less}(1, \texttt{Suc}(y))$

(6) Resolve from (4) and (5): $\neg\texttt{less}(\texttt{Suc}(0), \texttt{Suc}(y)) \vee \texttt{less}(1, \texttt{Suc}(y))$

(7) Resolve from (3) and (6): $\neg\texttt{less}(0, y) \vee \texttt{less}(1, \texttt{Suc}(y))$

(8) Subst from (7) using $\sigma = \{y \mapsto \texttt{Suc}(\texttt{x})\}$:
$\neg\texttt{less}(0, \texttt{Suc}(\texttt{x})) \vee \texttt{less}(1, \texttt{Suc}(\texttt{Suc}(\texttt{x})))$

(9) Resolve from (1) and (8): $\neg\texttt{less}(0, \texttt{Suc}(\texttt{x}))$

(10) AXIOM: $\texttt{less}(0, \texttt{Suc}(n))$
(11) SUBST from (10) using $\sigma = \{n \mapsto \texttt{x}\}$: $\texttt{less}(0, \texttt{Suc}(\texttt{x}))$
(12) RESOLVE from (9) and (11): $\texttt{False}$

We can now infer the substitutions that were applied to the axiom clauses:

$$\begin{aligned}
\text{infer}(12, \varnothing) &= \text{infer}(1, \varnothing) \mathbin{@} \text{infer}(2, \{y \mapsto \texttt{Suc}(\texttt{x})\} \circ \{m \mapsto 0, n \mapsto y\}) \\
&\quad \mathbin{@} \text{infer}(4, \{y \mapsto \texttt{Suc}(\texttt{x})\}) \mathbin{@} \text{infer}(5, \{y \mapsto \texttt{Suc}(\texttt{x})\}) \\
&\quad \mathbin{@} \text{infer}(10, \{n \mapsto \texttt{x}\}) \\
&= \text{infer}(1, \varnothing) \mathbin{@} \text{infer}(2, \{y \mapsto \texttt{Suc}(\texttt{x}), m \mapsto 0, n \mapsto \texttt{Suc}(\texttt{x})\}), \\
&\quad \mathbin{@} \text{infer}(4, \{y \mapsto \texttt{Suc}(\texttt{x})\}) \mathbin{@} [] \mathbin{@} \text{infer}(10, \{n \mapsto \texttt{x}\}) \\
&= [(\neg\texttt{less}(1, \texttt{Suc}(\texttt{Suc}(\texttt{x}))), \varnothing), \\
&\quad (\neg\texttt{less}(m, n) \vee \texttt{less}(\texttt{Suc}(m), \texttt{Suc}(n)), \\
&\quad \{y \mapsto \texttt{Suc}(\texttt{x}), m \mapsto 0, n \mapsto \texttt{Suc}(\texttt{x})\}), \\
&\quad (\texttt{Suc}(0) = 1, \{y \mapsto \texttt{Suc}(\texttt{x})\}), (\texttt{less}(0, \texttt{Suc}(n)), \{n \mapsto \texttt{x}\})]
\end{aligned}$$

Given a Metis proof θ, the call $\text{infer}(\theta, \varnothing)$ yields a list of pairs (C, σ), where C is an axiom clause and σ is the applied substitution. If the proof is retried, the *instantiated axiom clauses*—i.e., the list of all clauses $C\sigma$ corresponding to the pairs (C, σ)—can be used instead of the original axiom clauses to restrict the search space and speed up Metis. After instantiation, a Metis proof is still possible:

Theorem 1. *A Metis proof θ can be transformed into a new Metis proof derived from the instantiated axiom clauses (defined above). The new Metis proof does not involve the* SUBST *inference rule and uses at most as many proof steps as θ.*

The proof is by induction on the structure of the Metis proof θ. It proceeds by instantiating the clauses in θ with infer's current accumulator value and removing SUBST inference steps. Because clauses are sets and not multisets of literals, instantiation can shorten clauses by unifying literals. This can result in the elimination of resolution steps and thereby entire subproofs.

Proof. We annotate each Metis theorem in θ with infer's current accumulator value using the call $\text{annotate}(\theta, \varnothing)$, defined as follows:

$$\begin{aligned}
\text{annotate}((C, r, []), \sigma) &= (C, r, [], \sigma) \\
\text{annotate}((C, \textsc{Subst}, [\theta']), \sigma) &= (C, \textsc{Subst}, [\text{annotate}(\theta', \sigma \circ \sigma')], \sigma) \\
\text{annotate}((C, \textsc{Resolve}, [\theta_1, \theta_2]), \sigma) &= (C, \textsc{Resolve}, \\
&\qquad [\text{annotate}(\theta_1, \sigma), \text{annotate}(\theta_2, \sigma)], \sigma)
\end{aligned}$$

where σ' is again the substitution carried by the SUBST inference rule.

It suffices to show that each annotated Metis theorem $(C, r, \bar{\theta}, \sigma)$ contained in $\text{annotate}(\theta, \varnothing)$ can be transformed into a new Metis theorem $(C', r', \bar{\theta}')$ such

that $C' \subseteq C\sigma$ holds (with $C' = C\sigma$ for axiom clauses), SUBST inference steps are removed, and no new proof step is introduced. Since $C' \subseteq \texttt{False}\,\varnothing$ implies $C' = \texttt{False}$, the whole Metis proof θ is transformed into a new Metis proof. By the definitions of the infer and annotate functions, $C' = C\sigma$ for axiom clauses implies that the new Metis proof is derived from the instantiated axiom clauses.

Assume that we have an annotated Metis theorem $(C, r, \bar{\theta}, \sigma)$ contained in annotate$(\theta, \varnothing)$. We prove the proposition by induction on the inference rule r. In the base case, r has no premises, so $\bar{\theta} = []$ and r is one of AXIOM, ASSUME, REFL, and EQUALITY. We construct the new Metis theorem $(C', r', \bar{\theta}') = (C\sigma, r, [])$ so that $C' = C\sigma \subseteq C\sigma$ holds.

If $r =$ SUBST, we have $\bar{\theta} = [\theta_1]$, where $\theta_1 = (C_1, r_1, \bar{\theta}_1)$, and $C = C_1\sigma'$, where σ' is the substitution carried by the SUBST inference rule. Inductively, we can transform annotate$(\theta_1, \sigma \circ \sigma')$ into a new Metis theorem $\theta' = (C', r', \bar{\theta}')$ with $C' \subseteq C_1(\sigma \circ \sigma')$. Using $C_1(\sigma \circ \sigma') = (C_1\sigma')\sigma = C\sigma$, we can deduce $C' \subseteq C\sigma$, so we use θ' as the new Metis theorem and remove the substitution step.

If $r =$ RESOLVE, then $\bar{\theta} = [\theta_1, \theta_2]$, where $\theta_1 = (C_1, r_1, \bar{\theta}_1)$, $\theta_2 = (C_2, r_2, \bar{\theta}_2)$, $C_1 = D \vee A$, $C_2 = \neg A \vee E$, and $C = D \vee E$. Inductively, we can transform annotate(θ_1, σ) into a new Metis theorem $\theta'_1 = (C'_1, r'_1, \bar{\theta}'_1)$ with $C'_1 \subseteq C_1\sigma$ and annotate(θ_2, σ) into $\theta'_2 = (C'_2, r'_2, \bar{\theta}'_2)$ with $C'_2 \subseteq C_2\sigma$. Since instantiation can shorten clauses, we distinguish between two cases: In the first case, $A\sigma$ is contained in the clauses C'_1 and C'_2, so we obtain D' and E', where $C'_1 = D' \vee A\sigma$ and $C'_2 = \neg\, A\sigma \vee E'$. Then we can construct the new Metis theorem $(C', r', \bar{\theta}') = (D' \vee E', \text{RESOLVE}, [\theta'_1, \theta'_2])$ and $C' \subseteq C\sigma$ holds since $D' \vee E' \subseteq D\sigma \vee E\sigma = C\sigma$. In the second case, we find $\theta' = (C', r', \bar{\theta}') \in \{\theta'_1, \theta'_2\}$ such that C' does not contain $A\sigma$. Then $C' \subseteq C\sigma$ holds, and we can use θ' as the new Metis theorem, thereby removing the resolution step as well as the other subproof.

Translation of Metis Terms to Isabelle Terms. Once the instantiations of the Metis clauses have been inferred, a *translation procedure* converts the contained Metis terms into Isabelle terms that can be presented to the user.

The list obtained from the infer function is filtered to include only those pairs (C, σ) where the Metis clause C is derived from an Isabelle fact φ, and not from the negated goal. The procedure then continues to translate only Metis terms t of substitution elements $(x \mapsto t) \in \sigma$ where the Metis variable x occurs in C and corresponds to a free variable in φ. There may be other Metis variables, including those corresponding to type variables or quantified variables. The direct instantiation of type variables is seldom useful, since types can be inferred from the instantiated terms. Metis variables that correspond to quantified Isabelle variables may emerge during clausification; however, since these are bound in Isabelle, there is no way to refer to them.

The translation procedure must decode constructs that are introduced by the translation from polymorphic higher-order logic to untyped first-order logic in *metis*. This decoding uses the same code as the proof reconstruction in *metis* [32], where Metis terms emerge during reconstruction of the SUBST inference rule. The introduced constructs include the encoding of Isabelle symbols using Metis

constants, the encoding of free variables using Metis variables, and the encoding of partial application using a distinguished binary symbol `app` [22]. For example, `map f` and `map f` xs might be translated to $\texttt{app}(\texttt{map}, \texttt{f})$ and $\texttt{app}(\texttt{app}(\texttt{map}, \texttt{f}), xs)$, where `map` and `f` are Metis constants and xs is a Metis variable. While decoding these constructs is straightforward, decoding the type information (for which there are several encodings [7]) was considered too complicated for proof reconstruction in *metis*, so type inference is used instead [32].

The resulting Isabelle terms may still contain *Skolem terms* and encodings of λ*-abstractions*. Such *metis*-internal constructs may not appear in terms suggested to the user. Thus our translation procedure must eliminate them.

Skolem terms are eliminated by replacing them with a *wildcard*, which is displayed as '_' and corresponds to a fresh free variable (as demonstrated in Example 4). Consequently, the instantiated facts may still contain free variables, for which Metis must substitute the corresponding Skolem terms. It does not suffice to simply replace the Skolem symbols with '_'; their arguments must also be removed. This is necessary because the arity of Skolem symbols can change due to the instantiation of facts. For example, the Skolem symbol introduced for $\exists y.\ x < y$ depends on x and therefore requires an argument, whereas the Skolem symbol for the instance $\exists y.\ 0 < y$ does not require any argument.

The *metis* proof method encodes λ-abstractions using either `SKBCI` combinators [37] or λ-lifted supercombinators [19]. To eliminate these combinators, the translation procedure replaces them with their definition, followed by a $\beta\eta$-reduction of the resulting terms. For example, the term $\lambda x.\ 0$ is encoded as `K` 0, where `K` $a\ b = a$. When the translation procedure detects the term `K` 0 1, it replaces `K` with $\lambda a.\ \lambda b.\ a$ and produces the term 0 after β-reduction. If λ-abstractions remain after $\beta\eta$-reduction, the bound variables may have different names than in the original λ-abstractions (as demonstrated in Examples 4 and 7), but this is not a problem since Isabelle equates terms up to α-equivalence.

In the final step of the translation procedure, all remaining free variables (i.e., all free variables occurring in the Isabelle terms and all uninstantiated variables in the clauses) are instantiated with the Isabelle polymorphic constant `undefined`, which provides a witness for every type's inhabitedness. These variables emerge when the proof was possible without concrete terms. For example, $\texttt{a} = \texttt{b}$ can be derived from $x + \texttt{a} = x + \texttt{b}$ without choosing a value for x. Instantiation of such variables is sensible, since it restricts the search space, but users can prevent this behavior by disabling the *metis_instantiate_undefined* option, in which case the variables are replaced by '_'.

Instantiation of Isabelle Facts. The translation procedure yields a list of Isabelle facts and instantiations of their free variables. A single fact may possess multiple instantiations if it is used multiple times in the proof. To reduce the number of instantiations and to avoid duplicates, we merge instantiations whenever possible:

Definition 4. A (*variable*) *instantiation* $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ for an Isabelle fact φ is a partial function from variable names to Isabelle terms in which all variable names $x_1, \ldots, x_n$ occur in φ as free variables. Two instantiations ι_1, ι_2 for the same fact φ can be *merged* into $\iota_1 \cup \iota_2$ if $\iota_1(x)$ equals $\iota_2(x)$ up to α-equivalence for all shared variable names $x \in \mathrm{dom}(\iota_1) \cap \mathrm{dom}(\iota_2)$.

Merging can be used to avoid duplicates, since every instantiation can be merged with itself. Conversely, if two instantiations stem from the same clause C and can be merged, they are equal up to α-equivalence. This is because the translation procedure considers only variables occurring in C, and all uninstantiated variables of C are instantiated with `undefined` or '_'. As a result, the domains of the instantiations are exactly the variable names occurring in C that correspond to a free variable in the fact.

By the above argument, if different instantiations can be merged, they must stem from different clauses. In practice, *metis* will still succeed with the merged instantiation, since the clausifier will split the instantiated fact into the instantiated clauses again. Consequently, merging does not lose information.

Example 6. Consider the following Isabelle fact φ:

$$\text{even } n \longrightarrow 0 \leq x^n \wedge (-y)^n = y^n$$

The clausifier splits it into two clauses. Suppose that both clauses are used in a *metis* proof and we inferred the instantiations $\iota_1 = \{x \mapsto \mathtt{a},\ n \mapsto 2\}$ and $\iota_2 = \{y \mapsto \mathtt{b}, n \mapsto 2\}$. Since the variable name n is mapped to the same term in both instantiations, these can be merged to $\iota_1 \cup \iota_2 = \{x \mapsto \mathtt{a},\ y \mapsto \mathtt{b},\ n \mapsto 2\}$. By instantiating φ with $\iota_1 \cup \iota_2$, we obtain the following new fact:

$$\text{even } 2 \longrightarrow 0 \leq \mathtt{a}^2 \wedge (-\mathtt{b})^2 = \mathtt{b}^2$$

If *metis* is invoked again using this fact instead of the original fact, the clausifier will again split it into the two clauses, instantiated with ι_1 and ι_2, respectively.

Since we use type inference instead of reconstructing type information, the translation procedure may produce Isabelle terms with overly generic types. Given that types can be displayed in Isabelle (e.g., using the *show_types* option) and overly generic types can prevent the folding of abbreviations, we attempt to concretize the types of the terms of a variable instantiation ι for a fact φ. We achieve this through type unification [28]: The types of the terms of ι are unified with the types of the corresponding variables in φ, and the unifier is then applied to all the types contained in the terms of ι. It is crucial to find a single unifier for the entire instantiation ι, since the same type variable may appear in multiple types of the free variables in φ, and they must be synchronized.

Finally, if instantiations have been found, *metis* suggests that the user replaces the current *metis* call with a new call using the instantiated facts. Each instantiated fact is displayed by its name together with an annotation that specifies to instantiate it with the terms of the corresponding instantiation. If there are multiple instantiations of a single fact, the fact is displayed multiple times. By replacing the call, the user may get a more readable and faster proof.

5 Sledgehammer Extension

Since *metis* calls are usually generated by Sledgehammer, we also extend Sledgehammer's proof reconstruction module to enable direct generation of proofs with instantiated facts. By using Sledgehammer's preplay module, this extension can not only generate faster and more readable proofs but also transform timeouts into successes as well as produce simpler proofs using other proof methods than *metis*.

Preplay with Instantiations. Sledgehammer's proof reconstruction is started after an external ATP has found and minimized a proof. The facts referenced in the ATP proof are then used to generate an Isabelle proof. The *metis* extension we presented in Sect. 4 can now be invoked to infer the instantiations of these facts. To do so, *metis* must first succeed—both the Metis proof (for inferring the instantiations) and the reconstruction in Isabelle (for ensuring that the instantiations are type-correct, if an unsound type encoding was used) are essential.

After instantiating the facts, Sledgehammer's preplay tries out the proof with the instantiated facts. Even though *metis* has already found a proof, this is beneficial for a number of reasons:

1. The preplay module includes an additional minimization tool that repeatedly invokes a successful proof method with subsets of the facts. Its purpose is to reduce the number of facts and thereby simplify the generated Isabelle proof. Instantiated facts might be redundant and, consequently, can be eliminated by this minimization. Consider the goal $\mathtt{a} < -\mathtt{b} \longleftrightarrow \mathtt{b} < -\mathtt{a}$ and the fact $x < -y \longleftrightarrow y < -x$. This fact is translated to two Metis clauses, corresponding to the two implications $x < -y \longrightarrow y < -x$ and $y < -x \longrightarrow x < -y$. Since both clauses are equal up to the naming of variables, Metis uses only the first one to prove both directions of the goal. Accordingly, the instantiations $\{x \mapsto \mathtt{a}, y \mapsto \mathtt{b}\}$ and $\{x \mapsto \mathtt{b}, y \mapsto \mathtt{a}\}$ are inferred, resulting in two instantiated facts. Since only one of these is necessary, the minimization tool removes one of them.
2. Via a mechanism called *try0*, preplay tries a variety of proof methods [6], including multiple *metis* calls with different options (describing the encoding of types and λ-abstractions) and other standard proof methods, such as *simp* [26], *blast* [30], and *auto* [29]. This can result in even faster Isabelle proofs, since other proof methods are frequently faster than *metis* but sometimes require instantiated facts to find any proof at all (as demonstrated in Example 3). Additionally, Isabelle proofs resulting from methods such as *simp*, *blast*, and *auto* are often simpler and more readable, since they do not require the user to provide all the necessary facts for the proof. Minimization removes the facts that are irrelevant for these proof methods.
3. According to Theorem 1, a new Metis proof derived from the instantiated Metis clauses is possible. However, this does not necessarily extend to Isabelle proofs using *metis*. A new *metis* proof using the instantiated Isabelle facts may

not be possible, since their encoding is not guaranteed to produce the same Metis clauses again. This is mostly due to $\beta\eta$-conversion (as demonstrated in Example 7). This rarely happens in practice but is nonetheless one reason to preplay the new *metis* proof.

During preplay, each proof method is executed for a predetermined duration (called the *preplay time limit*, which defaults to 1 s). The instantiation of facts often speeds up proofs, possibly turning timeouts into successes. However, in order to instantiate the facts, *metis* must first succeed. Consequently, the *metis* call used to infer the instantiations is given more time than the proof methods during preplay. More precisely, it is executed for five times the preplay time limit (i.e., 5 s by default).

An alternative would be to increase the preplay time limit in the first place. However, this would work less well in practice, because users typically insert the generated proofs directly into the Isabelle proof text. Given that these are re-executed each time the text is processed, multiple slow proofs could substantially increase the overall processing time.

Our Sledgehammer extension can be controlled through the three-valued Sledgehammer option *instantiate*. If this option is set to *false*, the extension is disabled. If the option is set to *true*, *metis* is started with the *metis_instantiate* option enabled directly after an ATP has found and minimized a proof, at the beginning of Sledgehammer's proof reconstruction module. In most cases, this leads to Isabelle proofs with instantiated facts; otherwise, a proof without instantiations is displayed. The latter case may occur if *metis* is unable to find a proof (but, e.g., *auto* is able to), the proof does not use any facts, or there are no instantiations (e.g., if the facts contain no free variables).

Finally, if the option is set to *smart*, the instantiation of facts is started only if preplay failed (i.e., no proof method was successful within the preplay time limit). If the instantiation was successful, preplay is invoked once more with the instantiated facts, as usual. Therefore, Isabelle proofs with instantiated facts are displayed only if *metis* takes more time than the preplay time limit to prove the goal from the original facts and instantiation can speed up the proof so that it takes less time than the preplay time limit (as demonstrated in Example 2).

Although setting the option to *true* leads to faster, simpler, and more readable proofs, it is too disruptive. Most users, most of the time, are satisfied with uninstantiated facts, which are less verbose. They are pleased if they occasionally get a proof with instantiated facts that would have failed otherwise. Therefore, we make *smart* the default.

Extensionality for Metis. Extensionality states that two functions are equal if they yield the same results for the same arguments. In Isabelle, extensionality is a basic axiom called *ext*:

$$(\forall x.\ f\ x = g\ x) \longrightarrow f = g$$

Since Metis targets first-order logic, it is unaware of this principle. Instead, *ext* must be passed to *metis* (which uses a first-order encoding) when extensionality is required. Since this enlarges the search space, this is not done by default.

However, the need for extensionality can change as facts are instantiated. In other words, *ext* may not be necessary for a proof derived from the original facts, but it may be necessary for a proof derived from the instantiated facts, and vice versa. This phenomenon is mostly due to Isabelle's application of $\beta\eta$-conversion.

Example 7. Recall the fact $\texttt{surj}\ f \longrightarrow (\exists x.\ f\ x = y)$, and suppose that we want to prove the following goal:

$$\texttt{surj}\ (\lambda x.\ \lambda y.\ \texttt{g}\ y\ x) \longrightarrow (\forall x.\ \texttt{P}\ (\lambda y.\ \texttt{g}\ y\ x)) \longrightarrow \texttt{P}\ \texttt{h}$$

To encode the λ-abstractions, we use λ-lifting [19] and thus introduce a new supercombinator $\texttt{A}$ with the definition $\texttt{A}\ a\ b = \texttt{g}\ b\ a$, leading to a new goal:

$$\texttt{surj}\ \texttt{A} \longrightarrow (\forall x.\ \texttt{P}\ (\texttt{A}\ x)) \longrightarrow \texttt{P}\ \texttt{h}$$

We invoke *metis*, which infers that f should be instantiated with $\texttt{A}$ and y should be instantiated with $\texttt{h}$. As described in Sect. 4, $\texttt{A}$ is replaced by $\lambda a.\ \lambda b.\ \texttt{g}\ a\ b$ again, resulting in the following instantiated fact:

$$\texttt{surj}\ (\lambda a.\ \lambda b.\ \texttt{g}\ b\ a) \longrightarrow (\exists x.\ (\lambda b.\ \texttt{g}\ b\ x) = \texttt{h})$$

Notice that Isabelle applied β-reduction, resulting in the modified λ-abstraction $\lambda b.\ \texttt{g}\ b\ x$. If we attempt a proof with *metis* and λ-lifting again, this λ-abstraction will be encoded as another supercombinator $\texttt{B}$ with the definition $\texttt{B}\ a = \texttt{g}\ a\ \texttt{sk}$, where $\texttt{sk}$ is the Skolem constant introduced for the existentially quantified variable x. Using $\texttt{A}$'s and $\texttt{B}$'s definitions, Metis can prove that $\texttt{A}\ \texttt{sk}\ z = \texttt{B}\ z$ for every z, but extensionality is needed to conclude that $\texttt{A}\ \texttt{sk} = \texttt{B}$ and thus to exchange the two terms and prove the goal. Therefore, *metis* can prove the goal from the instantiated fact only if we add the fact *ext*.

Thus, *ext* may be necessary for a proof derived from the instantiated facts. Conversely, if $\beta\eta$-conversion eliminates λ-abstractions, *ext* might be unnecessary for a proof derived from the instantiated facts even though it is needed for a proof derived from the original facts. Therefore, we also extend the preplay module so that a few *metis* calls with the additional fact *ext* are tried after other proof methods have failed, including multiple *metis* calls with different options. These new *metis* calls use λ-lifting as the encoding of λ-abstractions, since λ-lifting benefits more from *ext* than the `SKBCI` combinators [22].

The *metis* calls with the fact *ext* are also useful regardless of instantiations, since some ATPs, such as Zipperposition [38], are based on higher-order logic and already include extensionality. As a result, their proofs do not need *ext*, and proof reconstruction might be unsuccessful without the new *metis* calls.

6 Evaluation

Our empirical evaluation is based on the repository revision d3c0734059ee (October 25, 2024) of Isabelle and 4082096ade5a (October 25, 2024) of the *Archive of Formal Proofs* [9].

Setup. We performed the evaluation using the Slurm batch system [21] on the computer resources of the Institute for Informatics at Ludwig-Maximilians-Universität München. We requested 32 GiB of RAM and 8 logical processors (CPU threads). Given that not all Sledgehammer steps and ATPs are deterministic and time limits are used, the results are not entirely reproducible. To increase reproducibility, we used a fresh Isabelle installation and reset the state of the learning-based relevance filter MaSh [8] before each evaluation run.

We used the testing and evaluation tool Mirabelle [16], which is included in Isabelle. Mirabelle applies a selected *action* to each selected *goal*. We used the same 50 *Archive* entries as Desharnais et al. [16], from which we randomly selected 100 goals per entry, resulting in a total of 5 000 goals. As the action, we invoked Sledgehammer with several options:

- The *try0* option was disabled so that Sledgehammer's preplay exclusively tried *metis*. This was done to test *metis* extensively, since we extended this proof method, and to keep the evaluation feasible in reasonable time, since other proof methods frequently caused the Isabelle process to abort for technical reasons.
- The *provers* option for specifying the external ATPs was set to use the high-performance [16] superposition provers E, Vampire, and Zipperposition. We excluded satisfiability-modulo-theories (SMT) solvers since these support theories, such as linear arithmetic, that are not built into *metis*.
- The *strict* option was enabled to force the use of sound type encodings for the external ATPs to produce type-correct proofs that should be reconstructable.

We performed two evaluation runs, with Sledgehammer's *instantiate* option set to *smart* in the first run and to *true* in the second run. From the Mirabelle output of each run, we extracted the number of successful Isabelle proofs with and without instantiations, the average execution times of Sledgehammer and *metis*, and the kinds of *metis* calls used in the generated Isabelle proofs. With this information, we try to answer the following research questions:

1. Is it possible to reconstruct additional proofs by instantiating the facts, and in how many cases is a proof derived from the instantiated facts impossible?
2. What are the advantages of enabling Sledgehammer's *instantiate* option? That is, how often are Isabelle proofs with instantiated facts actually presented to users, how much longer do Sledgehammer executions take, and how much faster are Isabelle proofs?
3. What do Sledgehammer-generated *metis* calls look like after instantiating the facts, and what are the benefits of adding the fact *ext*?

Results. We consider only Sledgehammer invocations where an external ATP has found a proof. Sledgehammer's proof reconstruction module initially attempts to generate a *one-line proof* consisting of a single proof method invoked with facts, which can now have instantiations thanks to our extensions. If this

attempt is unsuccessful, Sledgehammer tries to transform the ATP proof into an Isar proof [6]. If this also fails, the proof reconstruction is considered to have *failed*. The number of occurrences of each of these outcomes in each evaluation run is shown in Table 1.

Table 1. Proof reconstruction for Sledgehammer invocations yielding ATP proofs

Evaluation run	One-line proofs		Isar proofs	Failed	Total
	W/o inst.	With inst.			
1 (*instantiate* = *smart*)	3 128	44	37	164	3 373
2 (*instantiate* = *true*)	805	2 349	45	159	3 358

In the first evaluation run, there were a total of 3 373 Sledgehammer invocations yielding ATP proofs. Whereas 3 128 ATP proofs (92.7%) were reconstructed directly as one-line proofs without instantiations, 44 additional ATP proofs (1.3%) could be reconstructed by instantiating the facts and thereby speeding up the *metis* proofs. This number may seem low, but this corresponds to 18.0% of cases where there was no one-line proof before. The remaining 6.0% of ATP proofs could not be reconstructed as one-line proofs. This number may seem alarmingly high, but detailed Isar proofs were available in 1.1% of the cases, and in practice the *try0* option is usually enabled, which leads to further proof methods being tried in addition to *metis*. We note that our approach brings similar benefits to Sledgehammer's success rate as Isar proof reconstruction.

In the second evaluation run, there were a total of 3 358 Sledgehammer invocations with ATP proofs. In 2 349 cases (70.0%), Sledgehammer suggested a one-line proof with instantiated facts. In 805 cases (24.0%), Sledgehammer suggested a one-line proof without instantiated facts. The raw evaluation data reveals that in most of these cases, the proof did not use any facts or there were no instantiations.

However, sometimes the instantiations were successfully inferred, but preplay failed to generate a *metis* proof derived from the instantiated facts. According to the raw data, there were 5 such cases (0.1%) in the first evaluation run and 18 (0.5%) in the second. There are several reasons for this, including the following:

- The instantiations did not speed up the *metis* proof enough to take less time than the preplay time limit. The instantiations could have even prolonged the proof, since they modify the problem and could affect Metis's heuristics.
- Isabelle's extremely flexible syntax allows ambiguities and does not guarantee that all terms can be parsed back [6].

Table 2 shows the average execution times of Sledgehammer invocations that successfully generated a one-line *metis* proof. The average execution time of Sledgehammer increased by 312 ms (+0.9%) from the first to the second evaluation run, which can be explained by the additional time required for instantiating

Table 2. Average execution times for a one-line proof

Evaluation run	Sledgehammer	Generated *metis* proof
1 (*instantiate* = *smart*)	34 185 ms	221 ms
2 (*instantiate* = *true*)	34 497 ms	137 ms

facts. However, this did not have a substantial impact on the overall duration of an average Sledgehammer execution. By contrast, the average execution time of *metis* decreased by 84 ms (−38.0%), indicating that the instantiation of facts markedly speeds up proofs. Notice that in about 25% of cases (805 and 44 in Table 1), the *metis* calls generated in the two evaluation runs were both without or both with instantiations. Consequently, for the remaining 75% of cases, it can be deduced that the average execution time decreases by more than 38%. If those proofs are inserted into an Isabelle proof text, this improvement applies every time the text is processed. In particular, every one of the frequent regression tests of the *Archive of Formal Proofs* with its more than 90 000 *metis* calls could benefit.

In Sledgehammer's preplay, a series of *metis* calls are tried, each involving different options that describe the encoding of types and λ-abstractions. Initially, the default options (type encoding `a` [7] and `SKBCI` combinators [37]) are tested, resulting in a standard *metis* call. If this fails, Sledgehammer tries multiple alternative options, selecting the fastest *metis* call with options. If this also fails, Sledgehammer tries the new *metis* calls with the additional fact *ext*. The number of occurrences of each of these kinds of *metis* calls in one-line proofs in each evaluation run is shown in Table 3.

Table 3. Kinds of *metis* calls in one-line proofs

Evaluation run	Standard *metis*	With options	With *ext*	Total
1 (*instantiate* = *smart*)	2 801	318	53	3 172
2 (*instantiate* = *true*)	3 008	110	36	3 154

The number of standard *metis* calls increased from the first evaluation run (88.3%) to the second (95.4%). This observation suggests that Metis's proof search becomes more efficient after the instantiation of facts, since special encodings are needed less often. Moreover, adding *ext* was effective, since it was used in 1.7% and 1.1% of the proofs, respectively, which would have failed otherwise.

Regarding our research question 1, the results show that Sledgehammer can reconstruct additional one-line proofs by instantiating the facts, and that a proof derived from the instantiated facts is almost always possible. Regarding question 2, we observe that enabling Sledgehammer's *instantiate* option leads to proofs with instantiations in 70% of the cases, which may help users understand the

generated proofs. Sledgehammer executions also take a bit longer, but the execution time of the generated Isabelle proofs is substantially reduced, which is much more important. Regarding question 3, we see that there are more standard *metis* calls without options after instantiating the facts and that adding the fact *ext* brings similar benefits to Sledgehammer's success rate as instantiating facts.

7 Related Work

The Metis ATP [20], on which our work is based, was developed by Hurd with the goal of integrating it into the HOL proof assistant [18]. Metis was integrated in Isabelle by Paulson and Susanto [32]. Metis's fine-grained proofs were particularly useful to us. Beyond the HOL and Isabelle integrations, there is also a Metis proof checker in Agda developed by Prieto-Cubides and Sicard-Ramírez [33].

Our tool joins the ranks of various proof analysis tools. The GAPT (General Architecture for Proof Theory) framework by Ebner et al. [17] consists of data structures, algorithms, parsers, and more, with the aim of supporting proof theory applications and ATPs. Other proof-manipulating frameworks are ProofCert by Miller and colleagues [14,24] and Dedukti by Dowek and colleagues [1,11].

Our work is also loosely related to the experimental reconstruction of ATP proofs as detailed Isabelle proofs, or Isar proofs [40], by Blanchette et al. [6].

8 Conclusion

We extended Isabelle's *metis* proof method and the Sledgehammer tool to infer variable instantiations from proofs and present them to users. This speeds up Sledgehammer's proof reconstruction and increases its success rate. It also helps users to understand the proof without inspecting all of its details.

We see three main directions for future work. First, we could provide type annotations when parsing of the instantiations is ambiguous. Second, we could provide instantiations not only for one-line proofs but also for individual steps in detailed Isar proofs. Third, we could extend the approach to more ATPs in order to obtain variable instantiations even when *metis* without instantiations fails. The superposition provers E, SPASS, Vampire, and Zipperposition are difficult to integrate, because they fail to preserve the connection between the variables before and after clausification. The *metis* proof method's clausifier, by contrast, preserves variable names. As for SMT solvers, some of them, including cvc5, can be asked to output the instantiations they needed to show unsatisfiability. Then there is no need to analyze the proofs to infer the instantiations.

Acknowledgments. We thank Xavier Généreux, Elisabeth Lempa, Jannis Limperg, Kirstin Peters, Mark Summerfield, and the anonymous reviewers for suggesting various textual improvements. Blanchette's research was cofunded by the European Union (ERC, Nekoka, 101083038). 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.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this paper.

References

1. Assaf, A., et al.: Dedukti: a logical framework based on the $\lambda\Pi$-calculus modulo theory. CoRR abs/2311.07185 (2023). https://doi.org/10.48550/ARXIV.2311.07185
2. Bachmair, L., Ganzinger, H.: On restrictions of ordered paramodulation with simplification. In: Stickel, M.E. (ed.) CADE 1990. LNCS, vol. 449, pp. 427–441. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-52885-7_105
3. Barbosa, H., et al.: cvc5: a versatile and industrial-strength SMT solver. In: TACAS 2022, Part I. LNCS, vol. 13243, pp. 415–442. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99524-9_24
4. Bhayat, A., Suda, M.: A higher-order Vampire (short paper). In: Benzmüller, C., Heule, M.J.H., Schmidt, R.A. (eds.) IJCAR 2024, Part I. LNCS, vol. 14739, pp. 75–85. Springer, Heidelberg (2024). https://doi.org/10.1007/978-3-031-63498-7_5
5. Blanchette, J.: Hammering away: a user's guide to Sledgehammer for Isabelle/HOL (2025). https://isabelle.in.tum.de/website-Isabelle2025/dist/Isabelle2025/doc/sledgehammer.pdf
6. Blanchette, J.C., Böhme, S., Fleury, M., Smolka, S.J., Steckermeier, A.: Semi-intelligible Isar proofs from machine-generated proofs. J. Autom. Reason. **56**(2), 155–200 (2016). https://doi.org/10.1007/S10817-015-9335-3
7. Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. Log. Meth. Comput. Sci. **12**(4) (2016). https://doi.org/10.2168/LMCS-12(4:13)2016
8. Blanchette, J.C., Greenaway, D., Kaliszyk, C., Kühlwein, D., Urban, J.: A learning-based fact selector for Isabelle/HOL. J. Autom. Reason. **57**(3), 219–244 (2016). https://doi.org/10.1007/s10817-016-9362-8
9. Blanchette, J.C., Haslbeck, M., Matichuk, D., Nipkow, T.: Mining the archive of formal proofs. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (eds.) CICM 2015. LNCS (LNAI), vol. 9150, pp. 3–17. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20615-8_1
10. Blanchette, J.C., Popescu, A., Wand, D., Weidenbach, C.: More SPASS with Isabelle: superposition with hard sorts and configurable simplification. In: Beringer, L., Felty, A. (eds.) ITP 2012. LNCS, vol. 7406, pp. 345–360. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32347-8_24
11. Blot, V., Dowek, G., Traversié, T., Winterhalter, T.: From rewrite rules to axioms in the λΠ-calculus modulo theory. In: Kobayashi, N., Worrell, J. (eds.) FoSSaCS 2024, Part II. LNCS, vol. 14575, pp. 3–23. Springer, Heidelberg (2024). https://doi.org/10.1007/978-3-031-57231-9_1

12. Böhme, S., Nipkow, T.: Sledgehammer: judgement day. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. LNCS, vol. 6173, pp. 107–121. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14203-1_9
13. Bouton, T., Caminha B. de Oliveira, D., Déharbe, D., Fontaine, P.: veriT: an open, trustable and efficient SMT-solver. In: Schmidt, R.A. (ed.) CADE 2009. LNCS (LNAI), vol. 5663, pp. 151–156. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02959-2_12
14. Chihani, Z., Miller, D., Renaud, F.: A semantic framework for proof evidence. J. Autom. Reason. **59**(3), 287–330 (2017). https://doi.org/10.1007/S10817-016-9380-6
15. Coghetto, R.: Tarski's Parallel Postulate implies the 5th Postulate of Euclid, the Postulate of Playfair and the original Parallel Postulate of Euclid. Archive of Formal Proofs (2021). https://isa-afp.org/entries/IsaGeoCoq.html
16. Desharnais, M., Vukmirović, P., Blanchette, J., Wenzel, M.: Seventeen provers under the hammer. In: Andronick, J., de Moura, L. (eds.) ITP 2022, pp. 8:1–8:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.ITP.2022.8
17. Ebner, G., Hetzl, S., Reis, G., Riener, M., Wolfsteiner, S., Zivota, S.: System description: GAPT 2.0. In: Olivetti, N., Tiwari, A. (eds.) IJCAR 2016. LNCS (LNAI), vol. 9706, pp. 293–301. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40229-1_20
18. Gordon, M.J.C.: Introduction to the HOL system. In: Archer, M., Joyce, J.J., Levitt, K.N., Windley, P.J. (eds.) TPHOLs 1991, pp. 2–3. IEEE Computer Society (1991). https://ieeexplore.ieee.org/document/596265
19. Hughes, R.J.M.: Super-combinators: a new implementation method for applicative languages. In: LFP 1982, pp. 1–10. ACM Press (1982). https://doi.org/10.1145/800068.802129
20. Hurd, J.: First-order proof tactics in higher-order logic theorem provers. In: Archer, M., Vito, B.D., Muñoz, C. (eds.) Design and Application of Strategies/Tactics in Higher Order Logics, pp. 56–68. No. NASA/CP-2003-212448 in NASA Technical Reports, September 2003. https://www.gilith.com/papers/metis.pdf
21. Jette, M.A., Wickberg, T.: Architecture of the Slurm workload manager. In: Klusáček, D., Corbalán, J., Rodrigo, G.P. (eds.) JSSPP 2023, pp. 3–23. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-43943-8_1
22. Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. J. Autom. Reason. **40**(1), 35–60 (2007). https://doi.org/10.1007/s10817-007-9085-y
23. Meng, J., Paulson, L.C.: Lightweight relevance filtering for machine-generated resolution problems. J. App. Log. **7**(1), 41–57 (2009). https://doi.org/10.1016/J.JAL.2007.07.004
24. Miller, D.: Foundational proof certificates: making proof universal and permanent. In: Momigliano, A., Pientka, B., Pollack, R. (eds.) LFMTP 2013, pp. 1–2. ACM (2013). https://doi.org/10.1145/2503887.2503894
25. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
26. Nipkow, T.: Equational reasoning in Isabelle. Sci. Comput. Prog. **12**(2), 123–149 (1989). https://doi.org/10.1016/0167-6423(89)90038-5
27. Nipkow, T., Wenzel, M., Paulson, L.C. (eds.): Isabelle/HOL. LNCS, vol. 2283. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45949-9
28. Nipkow, T., Prehofer, C.: Type reconstruction for type classes. J. Funct. Prog. **5**(2), 201–224 (1995). https://doi.org/10.1017/S0956796800001325

29. Paulson, L.C. (ed.): Isabelle. LNCS, vol. 828. Springer, Heidelberg (1994). https://doi.org/10.1007/BFb0030541
30. Paulson, L.C.: A generic tableau prover and its integration with Isabelle. J. Univ. Comput. Sci. **5**(3), 73–87 (1999). https://doi.org/10.3217/jucs-005-03-0073
31. Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Schulz, S., Ternovska, E. (eds.) IWIL 2010. EPiC Series in Computing, vol. 2, pp. 1–11. EasyChair (2012). https://doi.org/10.29007/36dt
32. Paulson, L.C., Susanto, K.W.: Source-level proof reconstruction for interactive theorem proving. In: Schneider, K., Brandt, J. (eds.) TPHOLs 2007. LNCS, vol. 4732, pp. 232–245. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74591-4_18
33. Prieto-Cubides, J., Sicard-Ramírez, A.: Proof-reconstruction in type theory for propositional logic. https://raw.githubusercontent.com/jonaprieto/athena/master/pubs/paper/paper.pdf
34. Roßkopf, S., Nipkow, T.: A formalization and proof checker for Isabelle's metalogic. J. Autom. Reason. **67**(1), 1 (2023). https://doi.org/10.1007/s10817-022-09648-w
35. Steen, A., Benzmüller, C.: The higher-order prover Leo-III. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS (LNAI), vol. 10900, pp. 108–116. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94205-6_8
36. Sutcliffe, G.: The CADE-23 automated theorem proving system competition–CASC-23. AI Commun. **25**(1), 49–63 (2012). https://doi.org/10.3233/AIC-2012-0512
37. Turner, D.A.: A new implementation technique for applicative languages. Softw. Pract. Exper. **9**(1), 31–49 (1979). https://doi.org/10.1002/spe.4380090105
38. Vukmirović, P., Bentkamp, A., Blanchette, J., Cruanes, S., Nummelin, V., Tourret, S.: Making higher-order superposition work. In: Platzer, A., Sutcliffe, G. (eds.) CADE-28. LNCS, vol. 12699, pp. 415–432. Springer, Heidelberg (2021). https://doi.org/10.1007/S10817-021-09613-Z
39. Vukmirović, P., Blanchette, J., Schulz, S.: Extending a high-performance prover to higher-order logic. In: Sankaranarayanan, S., Sharygina, N. (eds.) TACAS 2023, Part II. LNCS, vol. 13994, pp. 111–129. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-30820-8_10
40. Wenzel, M.: Isar—a generic interpretative approach to readable formal proof documents. In: Bertot, Y., Dowek, G., Théry, L., Hirschowitz, A., Paulin, C. (eds.) TPHOLs 1999. LNCS, vol. 1690, pp. 167–183. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48256-3_12
41. Wenzel, M.: Isabelle/Isar—A Versatile Environment for Human-Readable Formal Proof Documents. Ph.D. thesis, Institut für Informatik, Technische Universität München (2002). http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2002/wenzel.pdf