Semi-intelligible Isar Proofs from Machine-Generated Proofs

Jasmin Christian Blanchette, Sascha Böhme, Mathias Fleury, Steffen Juilf Smolka, Albert Steckermeier

J. Autom. Reasoning manuscript No.
(will be inserted by the editor)

Semi-intelligible Isar Proofs from Machine-Generated Proofs

Jasmin Christian Blanchette · **Sascha Böhme** ·
Mathias Fleury · **Steffen Juilf Smolka** ·
Albert Steckermeier

Received: date / Accepted: date

Abstract Sledgehammer is a component of the Isabelle/HOL proof assistant that integrates external automatic theorem provers (ATPs) to discharge interactive proof obligations. As a safeguard against bugs, the proofs found by the external provers are reconstructed in Isabelle. Reconstructing complex arguments involves translating them to Isabelle's Isar format, supplying suitable justifications for each step. Sledgehammer transforms the proofs by contradiction into direct proofs; it iteratively tests and compresses the output, resulting in simpler and faster proofs; and it supports a wide range of ATPs, including E, LEO-II, Satallax, SPASS, Vampire, veriT, Waldmeister, and Z3.

Keywords Automatic theorem provers · Proof assistants · Natural deduction

1 Introduction

Sledgehammer [12,62] is a proof tool that connects the Isabelle/HOL proof assistant [53,54] with external automatic theorem provers (ATPs), including first-order superposition provers, higher-order provers, and solvers based on satisfiability modulo theories (SMT). Given an interactive proof goal, it heuristically selects hundreds of facts (lemmas, definitions, and axioms) from Isabelle's vast libraries, translates them to the external provers' logics, and invokes the provers in parallel (Section 2).

J. C. Blanchette
Inria Nancy & LORIA, Équipe VeriDis, 615, rue du Jardin Botanique, 54602 Villers-lès-Nancy, France
Max-Planck-Institut für Informatik, RG1, Campus E1 4, 66123 Saarbrücken, Germany
E-mail: jasmin.blanchette@{inria.fr,mpi-inf.mpg.de}

S. Böhme · A. Steckermeier
Institut für Informatik, Technische Universität München, Boltzmannstraße 3, 85748 Garching, Germany
E-mail: {boehmes,steckerm}@in.tum.de

M. Fleury
École normale supérieure de Rennes, Campus de Ker Lann, Avenue Robert Schuman, 35170 Bruz, France
Max-Planck-Institut für Informatik, RG1, Campus E1 4, 66123 Saarbrücken, Germany
E-mail: mathias.fleury@{ens-rennes.fr,mpi-inf.mpg.de}

S. J. Smolka
Department of Computer Science, Cornell University, 411 Gates Hall, Ithaca, New York, USA
E-mail: smolka@cs.cornell.edu

Although Sledgehammer can be trusted as an oracle [13], most users are satisfied only once the reasoning has been reduced to Isabelle primitives. When Sledgehammer was originally conceived, the plan was to have it deliver detailed justifications in Isabelle's Isar (Intelligible Semi-Automated Reasoning) language [86], a textual format inspired by the pioneering Mizar system [46]. Paulson and Susanto [63] designed a prototype that performed inference-by-inference translation of ATP proofs into Isar proofs and justified each Isar inference using *metis*, a proof method based on Hurd's superposition prover Metis [36]. For example, given the conjecture

lemma $X = Y \cup Z \longleftrightarrow Y \subseteq X \wedge Z \subseteq X \wedge (\forall V.\ Y \subseteq V \wedge Z \subseteq V \longrightarrow X \subseteq V)$

their procedure generated the following Isar text:

proof *neg_clausify*
  fix v
  assume *0*: $Y \subseteq X \vee X = Y \cup Z$
  assume *1*: $Z \subseteq X \vee X = Y \cup Z$
  assume *2*: $(\neg\, Y \subseteq X \vee \neg\, Z \subseteq X \vee Y \subseteq v) \vee X \neq Y \cup Z$
  assume *3*: $(\neg\, Y \subseteq X \vee \neg\, Z \subseteq X \vee Z \subseteq v) \vee X \neq Y \cup Z$
  assume *4*: $(\neg\, Y \subseteq X \vee \neg\, Z \subseteq X \vee \neg\, X \subseteq v) \vee X \neq Y \cup Z$
  assume *5*: $\forall V.\ ((\neg\, Y \subseteq V \vee \neg\, Z \subseteq V) \vee X \subseteq V) \vee X = Y \cup Z$
  have *6*: $\mathsf{sup}\ Y\ Z \neq X \vee \neg\, X \subseteq v \vee \neg\, Y \subseteq X \vee \neg\, Z \subseteq X$ **by** (*metis 4*)
  have *7*: $Z \subseteq v \vee \mathsf{sup}\ Y\ Z \neq X \vee \neg\, Y \subseteq X$ **by** (*metis 1 3 Un_upper2*)
  have *8*: $Z \subseteq v \vee \mathsf{sup}\ Y\ Z \neq X$ **by** (*metis 0 7 Un_upper1*)
  have *9*: $\mathsf{sup}\ Y\ Z = X \vee \neg\, Z \subseteq X \vee \neg\, Y \subseteq X$
    by (*metis 5 equalityI Un_least Un_upper1 Un_upper2*)
  have *10*: $Y \subseteq v$ **by** (*metis 0 1 2 9 Un_upper1 Un_upper2*)
  have *11*: $X \subseteq v$ **by** (*metis 0 1 8 9 10 Un_least Un_upper1 Un_upper2*)
  show False **by** (*metis 0 1 6 9 11 Un_upper1 Un_upper2*)
qed

The *neg_clausify* tactic on the first line recasts the conjecture into negated clauses, so that it has the same shape as in the corresponding ATP problem. The negated conjecture clauses are repeated using the **assume** command; then inferences are performed using **have**, culminating with a contradiction. The last step of an Isar proof is announced by the keyword **show**. The names *equalityI*, *Un_least*, *Un_upper1*, and *Un_upper2* refer to lemmas about the set operators $\subseteq$ and $\cup$.

Paulson and Susanto's approach was temporarily abandoned for several reasons: The resulting proofs by contradiction were often syntactically incorrect due to technical issues, notably the lack of necessary type annotations in the printed formulas, leading to unprovable goals. The generated proofs were often unpalatable, so that users were disinclined to insert them in their formalization. Moreover, although *metis* cannot compete with the fastest provers, a single call with the list of needed lemmas usually suffices to re-find the proof in reasonable time. Indeed, the equivalence above can be discharged within milliseconds using the line

by (*metis equalityI Un_least Un_upper1 Un_upper2*)

However, proof reconstruction with a single *metis* call means that the proof must be re-found each time the formalization is processed. This sometimes fails for difficult proofs that *metis* cannot re-find within a reasonable time and is vulnerable to small changes in the formalization. It also provides no answer to users who would like to understand the proof,

whether it be novices who expect to learn from it, experts who must satisfy their curiosity, or merely skeptics. But perhaps more importantly, *metis* supports no theories beyond equality, which is becoming a bottleneck as automatic provers are being extended with procedures for theory reasoning. The *smt* proof method [15, 17], which is based on the SMT solver Z3 [51], is a powerful, arithmetic-capable alternative to *metis*, but it depends on the availability of Z3 for proof replay, which hinders its acceptance among users; moreover, due to its incomplete quantifier handling, it can fail to re-find a proof produced by a superposition prover.

The remedy to all these issues is well known: to generate detailed, structured Isar proofs based on the machine-generated proofs, as originally envisioned by Paulson and Susanto. But this requires addressing the issues that plagued their prototype, as well as generalizing their approach so that it works with the vast collection of automatic provers that are now supported by Sledgehammer.

This article presents a new module for translating ATP proofs into readable Isar proofs. This module features a number of enhancements that increase the intelligibility and robustness of the output (Section 3). The implementation naturally decomposes itself into a number of general-purpose procedures, described abstractly in separate sections of this article.

The first obstacle to readability is that the Isar proof, like the underlying ATP proof, is by contradiction. A procedure transforms proofs by contradiction into direct proofs—or *redirects* the proofs (Section 4). The output is a direct proof expressed in natural deduction extended with case analyses and nested subproofs.

The typical architecture of modern first-order ATPs combines a clausifier and a reasoning core that assumes quantifier-free clause normal form (CNF). It is the clausifier's duty to skolemize the problem and move the remaining (essentially universal) quantifiers to the front of the formulas, where they can be omitted. Sledgehammer historically performed clausification itself, using the *neg_clausify* tactic, which implemented a naive exponential application of distributive laws. This was changed to use the ATPs' native clausifiers, since they normally generate a polynomial number of clauses and include other optimizations [4]. However, skolemization transforms a formula into an equisatisfiable, but not equivalent, formula; as a result, it must be treated specially when reconstructing the proof (Section 5).

ATP proofs can involve dozens, hundreds, or even thousands of inferences. When translating them to Isar, it can be beneficial to compress straightforward chains of deduction and to try various proof methods as alternatives to *metis* (Section 6). This postprocessing can make the resulting proof faster to process. Moreover, many users prefer concise proofs, either because they want to avoid cluttering their formalizations or because they find the shorter proofs easier to understand. If several proofs have been concurrently found by different provers, the user can choose the fastest one.

Despite laudable efforts toward standardizing the output of automatic provers [9, 79], each system currently has its own set of inference rules and sometimes even its own output format. The following provers are supported:

- Unit-equality prover: Waldmeister [33];
- First-order superposition provers: E [70], SPASS [84], and Vampire [66];
- Higher-order provers: LEO-II [8] and Satallax [20];
- SMT solvers: veriT [18] and Z3 [51].

The best behaved provers in this list take problems in TPTP (Thousands of Problem for Theorem Provers) format [77], produce proofs in TSTP (Thousands of Solutions for Theorem Provers) format [79], explicitly record skolemization as an inference, and otherwise perform only inferences that correspond to logical implications, which can usually be re-

played using *metis*. But even for these provers, some adaptations are necessary to integrate them (Section 7).

The Isar proof construction module has been developed over several years, starting from Paulson and Susanto's prototype. A naive attempt at redirecting proofs by contradiction was sketched at the PAAR 2010 and IWIL 2010 workshops [61, 62]. However, that procedure could exhibit exponential behavior. The linear procedure described in this article was introduced in a paper presented at the PxTP 2013 workshop [11]. Proof postprocessing was discussed in a second PxTP 2013 paper [72]. The integration of LEO-II, Satallax, and veriT was described in an internship report [28]. The corresponding work for Waldmeister is described in a B.Sc. thesis [73]. The text of this article is largely based on the two PxTP 2013 papers, which themselves overlap with a Ph.D. thesis [10, Section 6.8] and a B.Sc. thesis [71]. Important additions include a section on the specific ATPs (Section 7), real-world examples (Section 8), and an experimental evaluation (Section 9).

The implementation is an integral part of Isabelle. Although the focus is on this specific proof assistant, many of the techniques presented here are applicable to proof construction for other Sledgehammer-like tools, such as HOL^{Y}Hammer for HOL Light [38, 39] and Miz$\mathbb{AR}$ for Mizar [1, 69].

Related Work. There is a considerable body of research about making ATP proofs intelligible. Early work focused on translating resolution proofs into natural deduction calculi [50, 64]. Although they are arguably more readable, these calculi operate at the logical level, whereas humans reason mostly at the "assertion level," invoking definitions and lemmas without providing the full logical details. A line of research focused on transforming natural deduction proofs into assertion-level proofs [3, 35], culminating with the systems TRAMP [47] and Otterfier [88]. More related work includes the identification of obvious inferences [26, 68], the successful transformation of the EQP-generated proof of the Robbins conjecture using ILF [25], and the use of TPTP-based tools to present Mizar articles [81].

It would have been interesting to try out TRAMP and Otterfier, but these are large pieces of unmaintained software that are hardly installable on modern machines and that only support older ATP systems. Regardless, the problem looks somewhat different in the context of Sledgehammer. Because the provers are given hundreds of lemmas as axioms, they tend to find short proofs with few lemmas. Moreover, Sledgehammer can coalesce consecutive inferences if short proofs are desired. Replaying an inference is usually a minor issue, thanks to proof methods such as *metis* and *linarith*. In this respect, the most similar work is the textual proof generation for Miz$\mathbb{AR}$ [1], but it replays skolemization by introducing axioms.

2 Preliminaries

This article is concerned with a number of systems: a portfolio of automatic theorem provers on one side (Section 2.1), the Isabelle/HOL proof assistant and its Isar language on the other side (Sections 2.2 and 2.3), and the Sledgehammer tool as a bridge in between (Section 2.4).

2.1 Automatic Theorem Provers

Despite important technological differences, the ATPs of interest roughly follow the same general principles. They take a self-contained problem as input, consisting of a list of axioms and a conjecture. In case of success, they produce a proof of $\bot$ (falsity) from a subset of the

axioms and the negated conjecture. The derivation is a list of inferences, each depending on previous formulas; it can be viewed as a directed acyclic graph.

The concrete syntax varies from prover to prover. In the automated reasoning community revolving around the International Conference on Automated Deduction (CADE) and the CADE ATP System Competition (CASC) [78], the TPTP and TSTP syntaxes have emerged as de facto standards. TPTP defines a hierarchy of languages, including FOF (first-order form), TFF0 (typed first-order form), and THF0 (typed higher-order form). Despite slight syntactic inconsistencies, the subset chain FOF $\subset$ TFF0 $\subset$ THF0 essentially holds. The SMT-LIB 2 input syntax [7], supported by most modern SMT solvers, is conceptually similar to TPTP TFF0. TSTP specifies the basic syntax of a proof, as a list of inferences, but does not mandate any proof system. On the SMT side, there is no uniform format for solutions.

2.2 Isabelle/HOL

The Isabelle/HOL proof assistant is based on polymorphic higher-order logic (HOL) [31] extended with axiomatic type classes [85]. The types and terms of HOL are that of the simply typed λ-calculus [24] augmented with type constructors, type variables, and term constants. The types are either type variables (e.g., α, β) or n-ary type constructors, usually written in postfix notation (e.g, α *list*). Nullary type constructors (e.g., *nat*) are also called type constants. The binary type constructor $\alpha \Rightarrow \beta$ is interpreted as the (total) function space from α to β. Type variables can carry type class constraints, which are essentially predicates on types. An example is the *linorder* class, which is true only for types τ equipped with a linear order (op $< :: \tau \Rightarrow \tau \Rightarrow$ *bool*).

Terms are either constants (e.g., 0, sin, op $<$), variables (e.g., x), function applications (e.g., $f\ x$), or λ-abstractions (e.g., $\lambda x.\ f\ x\ x$). Constants and variables can be functions. HOL formulas are simply terms of type *bool*. The familiar connectives ($\neg$, $\wedge$, $\vee$, $\longrightarrow$) and quantifiers ($\forall$, $\exists$) are predefined. Constants can be polymorphic; for example, $\mathsf{map} :: (\alpha \Rightarrow \beta) \Rightarrow \alpha\ \mathit{list} \Rightarrow \beta\ \mathit{list}$ applies a function elementwise to a list of α elements.

Isabelle is a generic theorem prover whose metalogic is an intuitionistic fragment of HOL. In the metalogic, propositions have type *prop*, universal quantification is written $\bigwedge$, implication is written $\Longrightarrow$, and equality is written $\equiv$. The object logic is embedded in the metalogic using a constant $\mathsf{Trueprop} :: \mathit{bool} \Rightarrow \mathit{prop}$, which is normally not printed. In the examples, we preserve the distinction between the two levels to avoid distracting the trained Isabelle eye, but readers unfamiliar with the system can safely consider the symbols $\bigwedge$, $\Longrightarrow$, and $\equiv$ as aliases for $\forall$, $\longrightarrow$, and $=$.

Types are inferred using Hindley–Milner inference. Type annotations $:: \tau$ give rise to additional constraints that further restrict the inferred types. A classic example where type annotations are needed is $x + y = y + x$. Without type annotations, the formula is parsed as $(x :: \alpha) + (y :: \alpha) = (y :: \alpha) + (x :: \alpha)$, where α belongs to the *plus* type class, which provides the $+$ operator but imposes no semantics on it. An annotation is necessary to make the formula provable—e.g., $(x :: \mathit{int}) + y = y + x$. A single annotation is sufficient here because of the constraints arising from the most general types of the involved operators: op $+ :: \alpha \Rightarrow \alpha \Rightarrow \alpha$ and op $= :: \alpha \Rightarrow \alpha \Rightarrow$ *bool*.

For both types and terms, Isabelle distinguishes two kinds of free variable: *schematic variables*, which can be instantiated, and *nonschematic variables*, which stand for fixed, unknown entities. When stating a conjecture and proving it, the type and term variables are

normally fixed, and once it is proved, they become schematic so that users of the lemma can instantiate them when applying the lemma.

2.3 Isar Proofs

At the textual level, Isabelle proofs can be expressed either as so-called **apply** scripts or as structured Isar proofs. An **apply** script is a sequence of tactic applications that transform the goal until it is discharged. In contrast, Isar proofs have a forward structure, stating intermediate properties and proving each of them with a single proof method introduced by the keyword **by**. This makes them more suitable for reconstructing ATP proofs.

Isar proofs are a linear representation of natural deduction proofs in the style of Jaśkowski [37]. Unlike Gentzen-style trees [30], they allow the sharing of common derivations. The Isar proof format is amply documented elsewhere [53, 86]; here, we attempt a brief description of the syntax needed for the rest of this article.

Isar proofs are surrounded by **proof** and **qed**. The **proof** keyword optionally takes a tactic as argument to transform the goal. This facility was exploited by Paulson and Susanto [63] to recast the conjecture into negated clauses, as we mentioned in the introduction. With our framework, this is no longer necessary, and hence our proofs will specify the minus symbol (−) as argument to **proof**, signifying no proof method.

The body of the proof mimics the corresponding goal. A goal of the form $\bigwedge x_1 \ldots x_k.\ A_1 \Longrightarrow \cdots \Longrightarrow A_m \Longrightarrow C$ is typically accompanied by a proof block of the form

proof −
  fix $x_1 \ldots x_k$
  assume a_1: A_1
    ⋮
  assume a_m: A_m
  have l_1: P_1 **by** ...
    ⋮
  have l_n: P_n **by** ...
  show C **by** ...
qed

where $l_1,\ldots,l_n$ and $P_1,\ldots,P_n$ are the optional labels (names) and statements of intermediate properties, respectively, and similarly for the assumptions. Existential properties can be stated using **obtain**. The command

obtain x **where** $P\ x$

is semantically identical to

have $\exists x.\ P\ x$

but in the first case x can be used in further **have** commands, like a Skolem constant.

Formulas introduced by **have**, **obtain**, and **show** must be followed either by a single proof method (e.g., **by** *metis*) or by a nested **proof**–**qed** block. Facts can be added as assumptions to the goal via the **using** keyword. For many proof methods, this is the only way of making them use additional facts. Other methods can also take facts as argument; for example,

have $1+1=2$ **using** *one_add_one* **by** *metis*

is a synonym for

have $1+1=2$ **by** (*metis one_add_one*)

Consecutive steps can be chained together using **then**. For example,

have l: P **by** ...
have m: Q **using** l **by** ...

can be abbreviated to

have P **by** ...
then have m: Q **by** ...

The abbreviations **hence** = **then have** and **thus** = **then show** are frequently used.

A general form of chaining makes it possible to have an arbitrary number of intermediate facts flow into a proof step. Instead of

have l_1: P_1 **by** ...
have l_2: P_2 **by** ...
⋮
have l_n: P_n **by** ...
have m: Q **using** $l_1\ l_2 \ldots l_n$ **by** ...

we can write

have P_1 **by** ...
moreover have P_2 **by** ...
⋮
moreover have P_n **by** ...
ultimately have m: Q **by** ...

As we see from these examples, much of Isar syntax is concerned with shortening common idioms. As a final abbreviation, an intermediate fact with a nested proof such as

have $\bigwedge x_1 \ldots x_k.\ A_1 \Longrightarrow \cdots \Longrightarrow A_m \Longrightarrow C$
proof −
fix $x_1 \ldots x_k$
assume A_1
⋮
assume A_m
⟨*intermediate facts*⟩
show C **by** ...
qed

can be inlined into a block of the form

{ **fix** $x_1 \ldots x_k$
assume A_1
⋮
assume A_m
⟨*intermediate facts*⟩
have C **by** ... }

2.4 Sledgehammer

Sledgehammer integrates third-party automatic theorem provers to increase the level of automation in Isabelle/HOL. It consists of three main components:

1. The *relevance filter* [44,49] heuristically selects a few hundred facts from the thousands available in background theories.
2. The *translation module* [13,48] encodes polymorphic higher-order propositions in the target prover's logic (e.g., untyped or monomorphic first-order logic).
3. The *reconstruction module* (described in this article) produces an Isar proof that can be inserted in an Isabelle development.

Given that automatic provers are highly sensitive to the encoding of the problem, the translation module plays a crucial role. The translation involves two steps:

1. If the target system is first-order, higher-order features such as λ-abstractions and partial function applications must be encoded.
2. If the target system is ignorant of polymorphism, the polymorphic type information, including type classes, must be encoded, either by heuristically grounding the types in the problem (a process known as monomorphization) or by representing types as terms.

3 The Translation Pipeline

The translation from an ATP proof to an Isabelle Isar proof involves two main intermediate data structures. The ATP proof is first parsed and translated into a *proof by contradiction* of the same shape but with HOL formulas instead of first-order formulas (Section 3.1). This intermediate data structure is then transformed into a *direct proof* (Section 3.2), from which Isar proof text is synthesized. Various operations are implemented on these data structures to enhance the proof.

3.1 Proofs by Contradiction

The ATP proof is first translated into an Isabelle proof by contradiction. This step preserves the graph structure of the proof, but the nodes are labeled by HOL formulas. This translation corresponds largely to the work by Paulson and Susanto [63].

Some consolidation can already take place at this level. ATPs tend to record many more inferences than are interesting to Isabelle users. Trivial operations such as clausification and variable renaming produce chains of inference that can be collapsed.

Paulson and Susanto [63] describe how HOL terms, types, and type classes are reconstructed from their encoded form. Their code had to be adapted to cope with the variety of type encodings supported by modern versions of Sledgehammer [13], but nonetheless their description fairly accurately describes the current state of affairs.

Because we work in a classical logic, we can silently eliminate double negations. Automatic theorem provers perform this transformation in their clausifier. If the conjecture is a negation $\neg\,\phi$, we write ϕ for the negated conjecture, implicitly appealing to double negation elimination. In such cases, the proof by contradiction is more correctly called a "proof of negation." For uniformity, we also refer to such proofs as proofs by contradiction, the distinction being mostly relevant for intuitionistic logics.

3.2 Direct Proofs

The proof redirection algorithm presented in Section 4 takes a proof by contradiction as the input and produces a direct proof, expressed in a fragment of Isar. The abstract syntax of proofs and inferences is given by the production rules

$$\begin{aligned} \text{Proofs:} \quad & \pi ::= (\textsf{fix}\ x^*)?\ (\textsf{assume}\ l\text{:}\ \phi)^*\ \iota^* \\ \text{Inferences:} \quad & \iota ::= \textsf{then}?\ \textsf{have}\ (l\text{:})?\ \phi\ l^*\ \pi^* \\ & \quad\ \mid \textsf{then}?\ \textsf{obtain}\ x^*\ \textsf{where}\ (l\text{:})?\ \phi\ l^*\ \pi^* \end{aligned}$$

where x ranges over HOL variables (which may be of function types), l over Isar fact labels, and ϕ over HOL formulas. Question marks (?) denote optional material, and asterisks (*) denote repetition. Nested proof blocks are possible, as indicated by the syntax π^*.

A **fix** command fixes the specified variables in the local context, and **assume** enriches the context with an assumption. Standard inferences are performed using **have**. Its variant **obtain** establishes the existence of HOL variables for which a property holds and adds them to the context. The optional **then** keyword indicates that the previous fact is needed to prove the current fact.

Once the direct proof is constructed, it is iteratively tested and compressed. Finally, **then** is introduced to chain proof steps. The **then** keyword is only a convenience; the same effect can be achieved less elegantly using labels. At the end, useless labels are removed, and the remaining labels are changed to *f1*, *f2*, etc.

The final step of the translation pipeline produces a textual Isar proof. This step is straightforward, but some care is needed to generate strings that can be parsed back by Isabelle. This is especially an issue for formulas, where type annotations might be needed (Section 6.5).

3.3 Example

The following Isabelle theory fragment declares a two-valued *state* datatype, defines a flip function, and states a conjecture about it:

datatype *state* = On | Off

fun flip :: *state* ⇒ *state* **where**
flip On = Off |
flip Off = On

lemma flip $x \neq x$

Invoking Sledgehammer launches a collection of ATPs. The conjecture is easy, so they rapidly return. Given the problem in TPTP FOF, Vampire delivers the proof shown in Figure 1, expressed in a slightly idealized TSTP-like format. Each line gives a formula number, a role, and a formula. The formulas used from the original problem are listed first (formulas 51, 52, 55, 57, 58, and 774). Any problem formula that can be used to prove the conjecture is an axiom for the ATP, irrespective of its status in Isabelle (lemma, definition, or actual axiom). The rightmost columns indicate how the formulas was arrived at: Either it appeared in the original problem, in which case its identifier is given (e.g., *flip_simps_1*), or it was derived from one or more already proved formulas using a Vampire-specific proof rule.

51	axiom	$\mathsf{flip}(\mathsf{on}) = \mathsf{off}$	*flip_simps_1*	
52	axiom	$\mathsf{flip}(\mathsf{off}) = \mathsf{on}$	*flip_simps_2*	
55	axiom	$\neg\, \mathsf{off} = \mathsf{on}$	*state_distinct_1*	
57	axiom	$\forall X.\, (\neg\, \mathsf{state}(X) = \mathsf{on} \longrightarrow \mathsf{state}(X) = \mathsf{off})$	*state_exhaust*	
58	axiom	$\mathsf{state}(\mathsf{s}) = \mathsf{s}$	*type_of_s*	
774	conj	$\neg\, \mathsf{flip}(\mathsf{s}) = \mathsf{s}$	*goal*	
775	neg_conj	$\neg\, \neg\, \mathsf{flip}(\mathsf{s}) = \mathsf{s}$	774	negate
776	neg_conj	$\mathsf{flip}(\mathsf{s}) = \mathsf{s}$	775	flatten
781	plain	$\mathsf{off} \neq \mathsf{on}$	55	flatten
892	plain	$\forall X.\, (\neg\, \mathsf{state}(X) = \mathsf{on} \longrightarrow \mathsf{state}(X) = \mathsf{off})$	57	rectify
893	plain	$\forall X.\, (\mathsf{state}(X) \neq \mathsf{on} \longrightarrow \mathsf{state}(X) = \mathsf{off})$	892	flatten
1596	plain	$\forall X.\, (\mathsf{state}(X) = \mathsf{on} \vee \mathsf{state}(X) = \mathsf{off})$	893	ennf_trans
2238	neg_conj	$\mathsf{flip}(\mathsf{s}) = \mathsf{s}$	776	cnf_trans
2239	plain	$\mathsf{state}(\mathsf{s}) = \mathsf{s}$	58	cnf_trans
2287	plain	$\mathsf{flip}(\mathsf{on}) = \mathsf{off}$	51	cnf_trans
2288	plain	$\mathsf{flip}(\mathsf{off}) = \mathsf{on}$	52	cnf_trans
2375	plain	$\mathsf{off} \neq \mathsf{on}$	781	cnf_trans
2485	plain	$\forall X.\, (\mathsf{state}(X) = \mathsf{off} \vee \mathsf{state}(X) = \mathsf{on})$	1596	cnf_trans
3342	plain	$\mathsf{on} = \mathsf{s} \vee \mathsf{state}(\mathsf{s}) = \mathsf{off}$	2239, 2485	superpos
3362	plain	$\mathsf{on} = \mathsf{s} \vee \mathsf{off} = \mathsf{s}$	3342, 2239	fwd_demod
3402	neg_conj	$\mathsf{flip}(\mathsf{on}) = \mathsf{on} \vee \mathsf{off} = \mathsf{s}$	2238, 3362	superpos
3404	neg_conj	$\mathsf{off} = \mathsf{on} \vee \mathsf{off} = \mathsf{s}$	3402, 2287	fwd_demod
3405	neg_conj	$\mathsf{off} = \mathsf{s}$	3404, 2375	subsum_res
3407	neg_conj	$\mathsf{flip}(\mathsf{off}) = \mathsf{off}$	3405, 2238	bwd_demod
3408	neg_conj	$\mathsf{off} = \mathsf{on}$	3407, 2288	fwd_demod
3409	neg_conj	$\bot$	3408, 2375	subsum_res

Fig. 1 A proof produced by Vampire

If Sledgehammer's *isar_proofs* option is enabled, or if one-line proof reconstruction failed, textual Isar proof reconstruction is attempted. The Isabelle proof by contradiction for the ATP proof above is as follows:

775	$\mathsf{flip}\ s = s$	$\neg$ *goal*
3402	$\mathsf{flip}\ \mathsf{On} = \mathsf{On} \vee \mathsf{Off} = s$	775, *state.exhaust*
3404	$\mathsf{Off} = \mathsf{On} \vee \mathsf{Off} = s$	3402, *flip.simps*(1)
3405	$\mathsf{Off} = s$	3404, *state.distinct*(1)
3407	$\mathsf{flip}\ \mathsf{Off} = \mathsf{Off}$	775, 3405
3409	False	3407, *flip.simps*(2), *state.distinct*(1)

Since the goal is a disequality, by double negation elimination the negated goal happens to be a positive formula ($\mathsf{flip}\ s = s$). Linear inference chains are drastically compressed, and the lemmas

$$\begin{aligned} \textit{state.distinct}(1)&\colon\ \mathsf{Off} \neq \mathsf{On} \\ \textit{state.exhaust}&\colon\ (y = \mathsf{On} \Longrightarrow P) \Longrightarrow (y = \mathsf{Off} \Longrightarrow P) \Longrightarrow P \\ \textit{flip.simps}(1)&\colon\ \mathsf{flip}\ \mathsf{On} = \mathsf{Off} \\ \textit{flip.simps}(2)&\colon\ \mathsf{flip}\ \mathsf{Off} = \mathsf{On} \end{aligned}$$

are referenced by name rather than repeated. The passage from FOF to HOL also eliminates encoded type information, such as the state function and the auxiliary axiom *type_of_s*. After redirection, the proof becomes

have 3407: $\mathsf{flip}\ \mathsf{Off} \neq \mathsf{Off}$ [*flip.simps*(2), *state.distinct*(1)] []
have 3405: $\mathsf{flip}\ s \neq s \vee \mathsf{Off} \neq s$ [3407] []

have 3404: flip $s \neq s \vee \mathsf{Off} \neq s \wedge \mathsf{Off} \neq \mathsf{On}$ [3405, *state.distinct*(1)] []
have 3402: flip $s \neq s \vee$ flip $\mathsf{On} \neq \mathsf{On} \wedge \mathsf{Off} \neq s$ [3404, *flip.simps*(1)] []
have 775: flip $s \neq s$ [3402, *state.exhaust*] []

Compression and cleanup simplify the proof further:

have flip $\mathsf{Off} \neq \mathsf{Off}$ [*flip.simps*(2), *state.distinct*(1)] []
then have flip $s \neq s$ [*flip.simps*(1), *state.distinct*(1), *state.exhaust*] []

From this simplified direct proof, the Isar proof is easy to produce:

proof −
  have $\mathsf{Off} \neq$ flip Off **by** (*metis flip.simps*(2) *state.distinct*(1))
  thus flip $s \neq s$ **by** (*metis flip.simps*(1) *state.distinct*(1) *state.exhaust*)
qed

The example is somewhat odd in the way machine-generated proofs often are. A human prover would likely have clearly distinguished the On and Off cases, discharging them separately and combining the result. Here, the On case is inlined in the last step, which also applies the exhaustion rule to justify the case distinction.

4 Proof Redirection

Knuth, Larrabee, and Roberts call the unnecessary use of proof by contradiction a sin against mathematical exposition [42, Section 3]. What makes such proofs difficult to read is that they contain a mixture of theorems with respect to the specified axioms (forward steps) and of formulas whose derivation is tainted by the negated conjecture (backward steps). The resulting bidirectionality is often enough to confuse readers. It could be argued that proof by contradiction is the most natural way to prove a negative formula, such as the flip $s \neq s$ example above, but the issue of bidirectionality also arises in such cases.

The redirection algorithm presented below is not tied to a specific calculus or logic, but it does require contraposition and double negation elimination. In particular, it works on the Isar proofs generated by Sledgehammer or directly on first-order TSTP proofs [79]. The direct proofs are expressed in a simple Isar-like syntax, which can be regarded as natural deduction extended with case analyses and nested subproofs (Section 4.1). The algorithm is first demonstrated on a few examples (Section 4.2) before it is presented in more detail, both in prose and as Standard ML pseudocode (Section 4.3).

Excluding a linear number of additional inferences that justify case analyses, each inference in the proof by contradiction gives rise to one inference in the direct proof. The algorithm can easily process proofs with hundreds or thousands of inferences. The procedure is admittedly fairly straightforward; it would not be surprising if it were part of folklore or a special case of existing work.

4.1 Proof Notations

Proof Graphs. ATP proofs identify formulas by numbers. There may be several conjectures, in which case they are interpreted disjunctively. The negated conjectures and user-provided axioms are typically numbered $0, 1, 2, \ldots, n-1$, and the derivations performed during proof search (whether or not they participate in the final proof) are numbered sequentially from n.

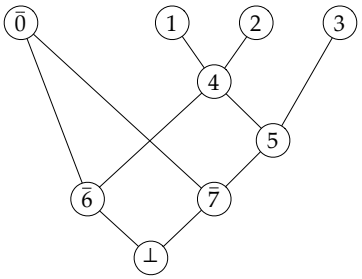


Fig. 2 A proof graph

proof *neg_clausify*
  assume $\bar{0}$
  have 4 **by** (*metis* 1 2)
  have 5 **by** (*metis* 3 4)
  have $\bar{6}$ **by** (*metis* $\bar{0}$ 4)
  have $\bar{7}$ **by** (*metis* $\bar{0}$ 5)
  show ⊥ **by** (*metis* $\bar{6}$ $\bar{7}$)
qed

proof −
  have 4 **by** (*metis* 1 2)
  have 5 **by** (*metis* 3 4)
  have $6 \vee 7$ **by** *metis*
  moreover
  { **assume** 6
    have 0 **by** (*metis* 4 6) }
  moreover
  { **assume** 7
    have 0 **by** (*metis* 5 7) }
  ultimately show 0 **by** *metis*
qed

Fig. 3 An Isar proof by contradiction (left) and the corresponding direct proof (right)

We abstract the ATP proofs by ignoring the formulas and keeping only the numbers. We call formulas *atoms* since we are not interested in their structure. The letters a, b denote atoms.

An atom is *tainted* if it is one of the negated conjectures or has been derived, directly or indirectly, from a negated conjecture. For convenience, we *relabel* the ATP proof's atoms so that tainted atoms are decorated with a bar, denoting negation. Thus, if atom 3, corresponding to the formula ϕ, is tainted, it is relabeled to $\bar{3}$, but it still stands for ϕ and is called an atom despite the negative bar. After the relabeling, removing the bar negates the formula; accordingly, 3 stands for $\neg\,\phi$.

A proof graph is a directed acyclic graph in which an edge $a \to a'$ indicates that atom a is used to derive atom a'. Proof graphs are required to have exactly one sink node, whose formula is $\bot$, and only one connected component. It is natural to write $\bot$ rather than a numeric label for the sink node in examples. We adopt the convention that derived nodes appear lower than their parent nodes in the graph and omit the arrowheads. Figure 2 gives an example.

Isar Proofs. Proof graphs cannot represent proofs by case analysis and only serve for the redirection algorithm's input. We need more powerful notations for the output (Section 3.2). Figure 3 shows a proof by contradiction and the corresponding direct proof.

Notice that the direct proof involves a two-way case analysis on a disjunction $(6 \vee 7)$. Generalized disjunctions of the form $a_1 \vee \cdots \vee a_m$ are called *clauses* and are denoted by the letters c, d, e. Clauses are considered equal modulo associativity, commutativity, and idempotence. Sets of clauses are denoted by Γ.

Proof redirection requires that inferences can be redirected using the contrapositive but otherwise makes no assumptions about the proof calculus. Inferences that introduce new

symbols can also be redirected; for example, skolemization becomes "unherbrandization" (Section 5.2).

Shorthand Proofs. The last proof format is an ad hoc shorthand notation for a subset of Isar. In their simplest form, these shorthand proofs are a list of derivations $c_1, \ldots, c_m \rhd c$ whose intuitive meaning is: "From the hypotheses c_1 and … and c_m, infer c." The clauses on the left-hand side are interpreted as a set Γ.

If a hypothesis c_i is the previous derivation's conclusion, we can omit it and write $\blacktriangleright$ instead of $\rhd$. This notation mimics Isar, with $\rhd$ for **have** (or **show**) and $\blacktriangleright$ for **hence** (or **thus**). Depending on whether we use the abbreviated format, our running example becomes

$$\begin{array}{r} 1,2 \rhd 4 \\ 3,4 \rhd 5 \\ \overline{0},4 \rhd \overline{6} \\ \overline{0},5 \rhd \overline{7} \\ \overline{6},\overline{7} \rhd \bot \end{array} \quad \text{or} \quad \begin{array}{r} 1,2 \rhd 4 \\ 3 \blacktriangleright 5 \\ \overline{0},4 \rhd \overline{6} \\ \overline{0},5 \rhd \overline{7} \\ \overline{6} \blacktriangleright \bot \end{array}$$

Each derivation $\Gamma \rhd c$ is essentially a sequent with Γ as the antecedent and c as the succedent. For proofs by contradiction, the clauses in the antecedent are either the negated conjecture ($\overline{0}$), atoms that correspond to background facts (1, 2, and 3), or atoms that were proved in preceding sequents (4, 5, $\overline{6}$, and $\overline{7}$); the succedent of the last sequent is always $\bot$.

Direct proofs can be presented in the same way, but the negated conjecture $\overline{0}$ may not appear in any of the sequents' antecedents, and the last sequent must have the conjecture 0 as its succedent. In some of the direct proofs, it is useful to introduce case analyses. For example:

$$\begin{array}{r} 1,2 \rhd 4 \\ 3 \blacktriangleright 5 \\ \rhd 6 \vee 7 \\ \left[\begin{array}{r|r} [6] & [7] \\ 4 \blacktriangleright 0 & 5 \blacktriangleright 0 \end{array}\right] \end{array}$$

In general, case analysis blocks are of the form

$$\left[\begin{array}{r|c|r} [c_1] & \cdots & [c_m] \\ \Gamma_{11} \rhd d_{11} & \ldots & \Gamma_{m1} \rhd d_{m1} \\ \vdots & & \vdots \\ \Gamma_{1k_1} \rhd d_{1k_1} & \cdots & \Gamma_{mk_m} \rhd d_{mk_m} \end{array}\right]$$

with the requirement that a sequent with the succedent $c_1 \vee \cdots \vee c_m$ has been proved immediately above the case analysis. Each of the branches must also be a valid proof. The assumptions $[c_i]$ may be used to discharge hypotheses in the same branch, as if they had been sequents $\rhd c_i$. The case analysis will sometimes be regarded as a sequent

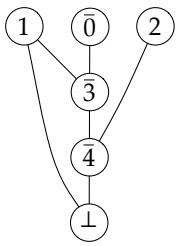
$$c_1 \vee \cdots \vee c_m,\ \left(\bigcup_{i,j} (\Gamma_{ij} - c_i - \bigcup_{j'<j} d_{ij'})\right) \rhd d_{1k_1} \vee \cdots \vee d_{mk_m}$$

by ignoring its internal structure.

4.2 Examples of Proof Redirection

Before reviewing the redirection algorithm, we consider four examples of proofs by contradiction and redirect them into a direct proof. The first example has a simple linear structure, the second and third examples involve a "lasso," and the last example has a complicated, spaghetti-like structure.

A Linear Proof. We start with a simple proof by contradiction expressed as a proof graph and in our shorthand notation:



$$\begin{aligned} \overline{0},1 &\rhd \overline{3}\\ 2,\overline{3} &\rhd \overline{4}\\ 1,\overline{4} &\rhd \bot \end{aligned}$$

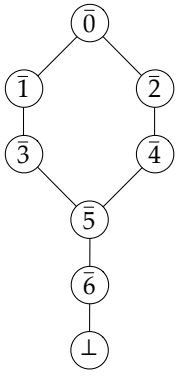
We redirect the sequents using sequent-level contraposition to eliminate all taints (represented as bars after the relabeling). This gives

$$\begin{aligned} 1,3 &\rhd 0\\ 2,4 &\rhd 3\\ 1 &\rhd 4 \end{aligned}$$

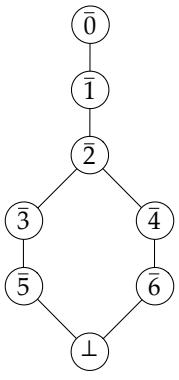
We then obtain the direct proof by reversing the order of the sequents, and introduce ▶ where it is possible without changing the order of the sequents:[1]

proof −	
have 4 **by** (*metis* 1)	$1 \rhd 4$
hence 3 **by** (*metis* 2)	$2 \blacktriangleright 3$
thus 0 **by** (*metis* 1)	$1 \blacktriangleright 0$
qed	

Lasso-Shaped Proofs. The next two examples look superficially like lassos but are of course acyclic, as required of all proof graphs. Recall that all edges are oriented downward by convention.



$$\begin{aligned} \overline{0} &\rhd \overline{1}\\ \overline{0} &\rhd \overline{2}\\ \overline{1} &\rhd \overline{3}\\ \overline{2} &\rhd \overline{4}\\ \overline{3},\overline{4} &\rhd \overline{5}\\ \overline{5} &\rhd \overline{6}\\ \overline{6} &\rhd \bot \end{aligned}$$



$$\begin{aligned} \overline{0} &\rhd \overline{1}\\ \overline{1} &\rhd \overline{2}\\ \overline{2} &\rhd \overline{3}\\ \overline{2} &\rhd \overline{4}\\ \overline{3} &\rhd \overline{5}\\ \overline{4} &\rhd \overline{6}\\ \overline{5},\overline{6} &\rhd \bot \end{aligned}$$

[1] An obvious improvement, with we have not implemented, would be to reorder statements to maximize the use of **then**. Karol Pąk has explored this problem in the context of Mizar [57].

We first consider the example on the left-hand side. Starting from $\bot$, it is easy to redirect the stem:

$$\begin{array}{r} \rhd\ 6 \\ 6 \rhd 5 \\ 5 \rhd 3 \vee 4 \end{array}$$

When applying the contrapositive to eliminate the negations in $\overline{3},\overline{4} \rhd \overline{5}$, we obtain a disjunction in the succedent: $5 \rhd 3 \vee 4$. To continue from there, we introduce a case analysis. In each branch, we can finish the proof:

$$\left[\begin{array}{r|r} [3] & [4] \\ 3 \rhd 1 & 4 \rhd 2 \\ 1 \rhd 0 & 2 \rhd 0 \end{array}\right]$$

In the second lasso example, the cycle occurs near the end of the contradiction proof. A disjunction already arises when we redirect the last derivation. Naively finishing each branch independently leads to a fair amount of duplication:

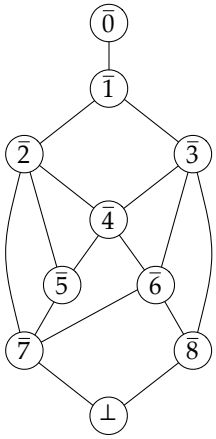
$$\begin{array}{c} \rhd\ 5 \vee 6 \\ \left[\begin{array}{r|r} [5] & [6] \\ 5 \rhd 3 & 6 \rhd 4 \\ 3 \rhd 2 & 4 \rhd 2 \\ 2 \rhd 1 & 2 \rhd 1 \\ 1 \rhd 0 & 1 \rhd 0 \end{array}\right] \end{array}$$

The key observation is that the two branches can share the last two inferences. This yields the following proof (without and with $\blacktriangleright$):

$$\begin{array}{c} \rhd\ 5 \vee 6 \\ \left[\begin{array}{r|r} [5] & [6] \\ 5 \rhd 3 & 6 \rhd 4 \\ 3 \rhd 2 & 4 \rhd 2 \end{array}\right] \\ 2 \rhd 1 \\ 1 \rhd 0 \end{array} \qquad \begin{array}{c} \rhd\ 5 \vee 6 \\ \left[\begin{array}{r|r} [5] & [6] \\ \blacktriangleright\ 3 & \blacktriangleright\ 4 \\ \blacktriangleright\ 2 & \blacktriangleright\ 2 \end{array}\right] \\ \blacktriangleright\ 1 \\ \blacktriangleright\ 0 \end{array}$$

Here we were fortunate that the branches were joinable on the atom 2. To avoid duplication, we must in general join on a disjunction $a_1 \vee \cdots \vee a_m$, as in the next example.

A Spaghetti-like Proof. The final example is more complicated:



$$\begin{array}{r} \overline{0} \rhd \overline{1} \\ \overline{1} \rhd \overline{2} \\ \overline{1} \rhd \overline{3} \\ \overline{2},\overline{3} \rhd \overline{4} \\ \overline{2},\overline{4} \rhd \overline{5} \\ \overline{3},\overline{4} \rhd \overline{6} \\ \overline{2},\overline{5},\overline{6} \rhd \overline{7} \\ \overline{3},\overline{6} \rhd \overline{8} \\ \overline{7},\overline{8} \rhd \bot \end{array}$$

We start with the contrapositive of the last sequent:

$$\rhd\ 7 \vee 8$$

We perform a case analysis on $7 \vee 8$. Since we want to avoid duplication in the two branches, we first determine which nodes are reachable in the refutation graph by navigating upward from either $\overline{7}$ or $\overline{8}$ but not from both. The only such nodes are $\overline{5}$, $\overline{7}$, and $\overline{8}$. In each branch, we can perform derivations of the form $\Gamma \rhd b$ where $\Gamma \cap \{5,7,8\} \neq \emptyset$, without fear of duplication. Following this rule, we can only perform one inference in the right branch before we must stop:

$$\begin{array}{c}[8]\\ 8\ \rhd\ 3 \vee 6\end{array}$$

Any further inferences would need to be repeated in the left branch, so it is indeed a good idea to stop. The left branch starts as follows:

$$\begin{array}{c}[7]\\ 7\ \rhd\ 2 \vee 5 \vee 6\end{array}$$

We would now like to perform the inference $5 \rhd 2 \vee 4$. This would certainly not lead to any duplication, because $\overline{5}$ is not reachable from $\overline{8}$ by navigating upward in the refutation graph. However, we cannot discharge the hypothesis 5, having established only the disjunction $2 \vee 5 \vee 6$. We need a case analysis on the disjunction to proceed:

$$\left[\begin{array}{c|c|c} [2] & \begin{array}{c}[5]\\ 5\ \rhd\ 2 \vee 4\end{array} & [6] \end{array}\right]$$

The 2 and 6 subbranches are left alone, because there is no node that is reachable only from $\overline{2}$ or $\overline{6}$ but not from the other two nodes in $\{\overline{2},\overline{5},\overline{6}\}$ by navigating upward in the refutation graph. Since only one branch is nontrivial, it is arguably more aesthetically pleasing to abbreviate the entire case analysis to

$$2 \vee 5 \vee 6\ \rhd\ 2 \vee 4 \vee 6$$

Putting this all together, the outer case analysis becomes

$$\left[\begin{array}{c|c} \begin{array}{c}[7]\\ \blacktriangleright\ 2 \vee 5 \vee 6\\ \blacktriangleright\ 2 \vee 4 \vee 6\end{array} & \begin{array}{c}[8]\\ \blacktriangleright\ 3 \vee 6\end{array} \end{array}\right]$$

The left branch proves $2 \vee 4 \vee 6$, the right branch proves $3 \vee 6$; hence, both branches together prove $2 \vee 3 \vee 4 \vee 6$. Next, we perform the inference $6 \rhd 3 \vee 4$. This requires a case analysis on $2 \vee 3 \vee 4 \vee 6$:

$$\left[\begin{array}{c|c|c|c} [2] & [3] & [4] & \begin{array}{c}[6]\\ 6\ \rhd\ 3 \vee 4\end{array} \end{array}\right]$$

This proves $2 \vee 3 \vee 4$. Since only one branch is nontrivial, we prefer to abbreviate the case analysis to

$$2 \vee 3 \vee 4 \vee 6\ \rhd\ 2 \vee 3 \vee 4$$

```
proof −
  have 7∨8 by metis
  moreover
  { assume 7
    hence 2∨5∨6 by metis
    hence 2∨4∨6 by metis }
  moreover
  { assume 8
    hence 3∨6 by metis }
  ultimately have 2∨3∨4∨6 by metis
  hence 2∨3∨4 by metis
  hence 2∨3 by metis
  moreover
  { assume 2
    hence 1 by metis }
  moreover
  { assume 3
    hence 1 by metis }
  ultimately have 1 by metis
  thus 0 by metis
qed
```

$$
\begin{array}{c}
\rhd\ 7\vee 8 \\
\left[\begin{array}{c|c} \begin{array}{c}[7]\\ \blacktriangleright\ 2\vee 5\vee 6\\ \blacktriangleright\ 2\vee 4\vee 6\end{array} & \begin{array}{c}[8]\\ \blacktriangleright\ 3\vee 6\end{array}\end{array}\right] \\
\blacktriangleright\ 2\vee 3\vee 4 \\
\blacktriangleright\ 2\vee 3 \\
\left[\begin{array}{c|c} \begin{array}{c}[2]\\ \blacktriangleright\ 1\end{array} & \begin{array}{c}[3]\\ \blacktriangleright\ 1\end{array}\end{array}\right] \\
\blacktriangleright\ 0
\end{array}
$$

Fig. 4 A complicated Isar proof (left) and the corresponding shorthand proof (right)

It may help to think of such abbreviated inferences as instances of rewriting modulo associativity, commutativity, and idempotence. Here, 6 is rewritten to $3\vee 4$ in $2\vee 3\vee 4\vee 6$, resulting in $2\vee 3\vee 4$. Similarly, the sequent $4\rhd 2\vee 3$ gives rise to the case analysis

$$\left[\begin{array}{c|c|c} [2] & [3] & \begin{array}{c}[4]\\ 4\rhd 2\vee 3\end{array}\end{array}\right]$$

which can be abbreviated as well. We are left with $2\vee 3$. The rest is analogous to the second lasso-shaped proof:

$$\begin{array}{c}\left[\begin{array}{c|c} \begin{array}{c}[2]\\ 2\rhd 1\end{array} & \begin{array}{c}[3]\\ 3\rhd 1\end{array}\end{array}\right]\\ 1\rhd 0\end{array}$$

Putting all of this together, we obtain the proof shown in Figure 4, expressed in Isar and in shorthand. The result is arguably quite respectable, considering the spaghetti-like graph we started with.

4.3 The Redirection Algorithm

The process we applied in the examples of Section 4.2 can be generalized into an algorithm. The algorithm takes an arbitrary proof by contradiction expressed as a set of sequents as input and produces a proof in our Isar-like shorthand notation, with sequents and case analysis blocks. The proof is constructed one inference at a time starting from $\top$ (the negation of $\bot$) until the conjecture (or the disjunction of the conjectures) is proved.

Basic Concepts. A fundamental operation is sequent-level contraposition. Let $a_1,\ldots,a_m$ be the untainted atoms and $\overline{b_1},\ldots,\overline{b_n}$ the tainted atoms of a proof by contradiction. The proof then consists of the following three kinds of sequent (with $n>0$):

$$a_1,\ldots,a_m,\overline{b_1},\ldots,\overline{b_n}\rhd\bot \qquad a_1,\ldots,a_m,\overline{b_1},\ldots,\overline{b_n}\rhd\overline{b} \qquad a_1,\ldots,a_m\rhd a$$

Their *contrapositives* are, respectively,

$$a_1,\ldots,a_m \rhd b_1 \vee \cdots \vee b_n \qquad a_1,\ldots,a_m,b \rhd b_1 \vee \cdots \vee b_n \qquad a_1,\ldots,a_m \rhd a$$

We call the contrapositives of the sequents in the proof by contradiction the *redirected sequents*. Based on the set of redirected sequents, we define the *atomic inference graph* (AIG) with, for each redirected sequent $\Gamma \rhd c$, an edge from each atom in Γ to each atom in c, and no additional edges. The AIG encodes the order in which the atoms can be inferred in a direct proof. Navigating forward (downward) in this graph along the unnegated tainted atoms b_j corresponds to navigating backward (upward) in the refutation graph along the $\overline{b_j}$'s.

Like the underlying refutation graph, the AIG is acyclic and connected. Potential cycles would involve either only untainted atoms a_i, only tainted atoms b_j's, or a mixture of both kinds. A cycle $a_{i_1} \to \cdots \to a_{i_k} \to a_{i_1}$ is impossible, because the contrapositive leaves these inferences unchanged and hence the cycle would need to occur in the refutation graph, which is acyclic by definition. A cycle $b_{j_1} \to \cdots \to b_{j_k} \to b_{j_1}$ is impossible, because the contrapositive turns all the edges around and hence the reverse cycle would need to occur in the refutation graph. Finally, mixed cycles necessarily involve an edge $b \to a$, which is impossible because redirected sequents with untainted atoms a can only have untainted atoms as predecessors.

Given a set of (tainted or untainted) atoms A, the *zone* of an atom $a \in A$ with respect to A is the set of possibly trivial descendants of a in the AIG that are not descendants of any of the other atoms in A. As a trivial descendant of itself, a will either belong to its own zone or to no zone all at, but this is not important for the algorithm. Zones identify inferences that can safely be performed inside a branch in a case analysis.

The Algorithm. The algorithm keeps track of the *last-proved clause* (initially $\top$), the set of *already proved atoms* (initially the set of facts taken as axioms), and the set of *remaining sequents* to use (initially all the redirected sequents provided as input). It performs the following steps:

1. If there are no remaining sequents, stop.
2. If the last-proved clause is $\top$ or a single atom:
   2.1. Choose a sequent $\Gamma \rhd c$ among the remaining sequents that can be proved using only already proved atoms, preferring sequents with a single atom in their succedent.
   2.2. Append $\Gamma \rhd c$ to the proof.
   2.3. Make c the last-proved clause, add c to the already proved atoms if it is an atom, and remove $\Gamma \rhd c$ from the remaining sequents.
   2.4. Go to step 1.
3. Otherwise, the last-proved succedent is of the form $a_1 \vee \cdots \vee a_m$. An m-way case analysis is called for:[2]
   3.1. Compute the zone of each atom a_i with respect to $\{a_1,\ldots,a_m\}$.
   3.2. For each a_i, compute the set $\mathcal{S}_i$ of sequents $\Gamma \rhd c$ such that Γ consists only of already proved atoms or atoms within a_i's zone.
   3.3. Recursively invoke the algorithm m times, once for each a_i, each time with a_i as the last-proved clause, a_i added to the already proved atoms, and $\mathcal{S}_i$ as the set of remaining sequents. This step yields m (possibly empty) subproofs $\pi_1, \ldots, \pi_m$.

[2] A straightforward generalization would be to perform a m'-way case analysis, with $m' < m$, by preserving some disjunctions. For example, we could perform a three-way case analysis with $a_1 \vee a_2$, a_3, and a_4 as the assumptions instead of breaking all the disjunctions in a four-way analysis. This could lead to a nicer output if the disjuncts are carefully chosen.

3.4. Append the following case analysis block to the proof:

$$\left[\begin{array}{c|c|c} [a_1] & \cdots & [a_m] \\ \pi_1 & \cdots & \pi_m \end{array}\right]$$

3.5. Make the succedent $b_1 \vee \cdots \vee b_n$ of the case analysis block (regarded as a sequent) the last-proved clause, add b_1 to the already proved atoms if $k = 1$, and remove all sequents belonging to any of the sets $\mathcal{S}_i$ from the remaining sequents.

3.6. Go to step 1.

Whenever a redirected sequent is generated, it is removed from the set of remaining sequents. In step 3, the recursive calls operate on pairwise disjoint subsets $\mathcal{S}_i$ of the remaining sequents. Consequently, each redirected sequent appears at most once in the generated proof, and the resulting direct proof contains the same number of inferences as the initial proof by contradiction. In Isar, each case analysis is additionally justified by a proof method, such as *metis*.

In the degenerate case where no atoms are tainted (i.e., the prover exploited an inconsistency in the axiom set), the generated proof is simply a linearization of the refutation graph, and the last inference proves $\bot$ (which is, unusually, untainted). To produce a syntactically valid Isar proof, a final inference must be added to derive the conjecture from $\bot$.

Pseudocode. To make the above description more concrete, the algorithm is presented in Standard ML pseudocode below. The pseudocode is fairly faithful to the description above. Atoms are represented by integers and literals by sets (lists) of integers. Go-to statements are implemented by recursion, and the state is threaded through recursive calls as three arguments (*last*, *earlier*, and *seqs*).

One notable difference with the informal description, justified by a desire to avoid code duplication, is that the set of already proved atoms, called *earlier*, excludes the last-proved clause *last*. Hence, we take *last* ∪ *earlier* to obtain the already proved atoms, where *last* is either the empty list (representing $\top$) or a singleton list (representing a single atom).

Shorthand proofs are represented as lists of *inference*s:

```
datatype inference =
  Have of int list × int list
| Cases of (int × inference list) list
```

The main function implementing the algorithm follows:

```
fun redirect last earlier seqs =
  if null seqs then
    []
  else if length last ≤ 1 then
    let val provable = filter (fn (Γ, _) ⇒ Γ ⊆ last ∪ earlier) seqs
        val horn_provable = filter (fn (_, [_]) ⇒ true | _ ⇒ false) provable
        val (Γ, c) = hd (horn_provable @ provable)
    in Have (Γ, c) :: redirect c (last ∪ earlier) (seqs − {(Γ, c)}) end
  else
    let val zs = zones_of (length last) (map (descendants_of seqs) last)
        val S = map (fn z ⇒ filter (fn (Γ, _) ⇒ Γ ⊆ earlier ∪ z) seqs) zs
        val cases = map (fn (a, ss) ⇒ (a, redirect [a] earlier ss)) (zip last S)
    in Cases cases :: redirect (succedent_of_cases cases) earlier (seqs − ⋃S) end
```

The code uses familiar ML functions, such as :: ("cons"), hd ("head," i.e., first element), @ ("append"), map, filter, and zip. Thus, hd (*horn_provable* @ *provable*), corresponding to step 2.1, returns the first sequent among the remaining sequents that can be proved using only already proved atoms, preferring sequents with a single atom in their succedent ("Horn sequents"). The pseudocode also relies on a descendants_of function that returns the descendants of the specified node in the AIG associated with *seqs*; its definition is omitted. Finally, the code depends on the following straightforward functions:

fun zones_of 0 _ = []
  | zones_of n (B :: Bs) = ($B - \bigcup Bs$) :: zones_of ($n-1$) (Bs @ [B])

fun succedent_of_inf (Have (_, c)) = c
  | succedent_of_inf (Cases *cases*) = succedent_of_cases *cases*
and succedent_of_case (a, []) = [a]
  | succedent_of_case (_, *infs*) = succedent_of_inf (last *infs*)
and succedent_of_cases *cases* = $\bigcup$(map succedent_of_case *cases*)

Correctness. It is not hard to convince ourselves that the proof output by redirect is correct by inspecting the code. A Have (Γ, c) sequent is appended only if all the atoms in Γ have been proved (or assumed) already, and a case analysis on $a_1 \vee \cdots \vee a_m$ always follows a sequent with the succedent $a_1 \vee \cdots \vee a_m$. Whenever a sequent is output, it is removed from *seqs*. The function returns only if *seqs* is empty, at which point the conjecture must have been proved (except in the degenerate case where the negated conjecture does not participate in the refutation).

Termination is not quite as obvious. The recursion is well-founded, because the pair (length *seqs*, length *last*) becomes strictly smaller with respect to the lexicographic extension of $<$ on natural numbers for each of the three recursive calls in the function's body.

- For the first recursive call, the list $seqs - \{(\Gamma, c)\}$ is strictly shorter than *seqs* since $(\Gamma, c) \in seqs$.
- The second call is performed for each branch of a case analysis; the *ss* argument is a (not necessarily strict) subset of the caller's *seqs*, and the list $[a]$ is strictly shorter than *last*, which has length 2 or more.
- For the third call, the key property is that at least one of the zones is nonempty, from which we obtain $seqs - \bigcup \mathcal{S} \subset seqs$. If all the zones were empty, each atom a_i would be the descendant of at least one atom $a_{i'}$ in the AIG (with $i' \neq i$), which is impossible because the AIG is acyclic.

As for run-time exceptions, the only worrisome construct is the hd call in redirect's second branch. We must convince ourselves that there exists at least one sequent $(\Gamma, c) \in seqs$ such that $\Gamma \subseteq last \cup earlier$. Intuitively, this is unsurprising because *seqs* is initialized from a well-formed refutation graph: The nonexistence of such a sequent would indicate a gap or a cycle in the refutation graph. More precisely, if there exist untainted atoms $\notin last \cup earlier$, these can always be processed first; indeed, the preference for sequents with a single atom in their succedent ensures that they are processed before the first case analysis. Otherwise:

- If *last* is [] (representing $\top$) or an untainted atom, the contrapositive $a_1, \ldots, a_m \rhd b_1 \vee \cdots \vee b_n$ of the very last inference is applicable since it only depends on untainted atoms, all of which have already been proved.

– Otherwise, *last* is a tainted atom b. The refutation graph must contain an inference $a_1,\ldots,a_m,\overline{b_1},\ldots,\overline{b_n} \rhd \overline{b}$, whose redirected inference is $a_1,\ldots,a_m,b \rhd b_1 \vee \cdots \vee b_n$. Since it only depends on b and untainted atoms, it is applicable.

Inlining. As a postprocessing step, we can optionally abbreviate case analyses in which only one branch is nontrivial, transforming

$$\left[\begin{array}{c|c|c|c|c|c|c} & & & [c_i] & & & \\ & & & d_{11},\ldots,d_{1k_1} \rhd e_1 & & & \\ & & & \vdots & & & \\ [c_1] & \cdots & [c_{i-1}] & d_{n1},\ldots,d_{nk_n} \rhd e_n & [c_{i+1}] & \cdots & [c_m] \end{array}\right] \quad \text{into} \quad \begin{array}{r} \widetilde{d_{11}},\ldots,\widetilde{d_{1k_1}} \rhd \widetilde{e_1} \\ \vdots \\ \widetilde{d_{n1}},\ldots,\widetilde{d_{nk_n}} \rhd \widetilde{e_n} \end{array}$$

where the function $\widetilde{\ }$ is the identity except for the assumption c_i and the conclusions $e_1,\ldots,e_n$:

$$\widetilde{c_i} = c_1 \vee \cdots \vee c_m \qquad\qquad \widetilde{e_j} = c_1 \vee \cdots \vee c_{i-1} \vee e_j \vee c_{i+1} \vee \cdots \vee c_m$$

It is debatable whether such inlining is a good idea. The resulting proof has a simpler structure, with fewer nested proof blocks. However, these nested blocks can help make complex proof more intelligible. Moreover, the n-fold repetition of the disjuncts $c_1,\ldots,c_{i-1},c_{i+1},\ldots,c_m$ clutters the proof and can slow it down.

The inlining procedure can be generalized to arbitrary case analysis blocks. We are grateful to Konstantin Korovin [43] for crucial insights. An m-way case analysis

$$\left[\begin{array}{c|c|c} [c_1] & \cdots & [c_m] \\ \Gamma_{11} \rhd e_{11} & \cdots & \Gamma_{m1} \rhd e_{m1} \\ \vdots & & \vdots \\ \Gamma_{1k_1} \rhd e_{1k_1} & \cdots & \Gamma_{mk_m} \rhd e_{mk_m} \end{array}\right]$$

can be rewritten into a sequence of m case analyses, each of the form

$$\left[\begin{array}{c|c|c|c|c|c} & & & [c_i] & & \\ & & & \Gamma_{i1} \rhd e_{i1} & & \\ & & & \vdots & & \\ [e_{1k_1}] & \cdots & [e_{(i-1)k_{i-1}}] & \Gamma_{ik_i} \rhd e_{ik_i} & [c_{i+1}] & \cdots & [c_m] \end{array}\right]$$

Each case analysis has only one nontrivial branch and can be inlined, yielding a branch-free proof. For the spaghetti-like proof of the previous section, this process yields

$$\begin{array}{l} \rhd\ 7 \vee 8 \\ \blacktriangleright\ 2 \vee 5 \vee 6 \vee 8 \\ \blacktriangleright\ 2 \vee 4 \vee 6 \vee 8 \\ \blacktriangleright\ 2 \vee 3 \vee 4 \vee 6 \\ \blacktriangleright\ 2 \vee 3 \vee 4 \\ \blacktriangleright\ 2 \vee 3 \\ \blacktriangleright\ 1 \vee 3 \\ \blacktriangleright\ 1 \\ \blacktriangleright\ 0 \end{array}$$

The example shows clearly that we rapidly obtain large disjunctions. In practice, each of the disjuncts would be an arbitrarily complex formula. Local definitions could be used to avoid repeating the formulas, but the loss of modularity is deplorable. Indeed, similar concerns

about Hoare-style proof outlines for separation logic have lead to the development of ribbon proofs [87], whose parallel "ribbons" evoke the branches of a case analysis.

If branch-free proofs are nonetheless desired, they can be generated more directly by iteratively "rewriting" the atoms, following a suggestion by Korovin. For example, starting from the sequent $\rhd 7 \vee 8$, rewriting 7 would involve resolving $\rhd 7 \vee 8$ with $7 \rhd 2 \vee 5 \vee 6$, resulting in $2 \vee 5 \vee 6 \vee 8$. In general, rewriting a tainted atom b_j within a sequent $\Gamma \rhd b_1 \vee \cdots \vee b_n$ involves resolving that sequent with the redirected sequent that has b_j in its assumptions. To guarantee that the procedure is linear, it suffices to rewrite atoms only if all their ancestors in the AIG have already been rewritten, thereby ensuring that atoms are rewritten only once.

5 Skolemization

Skolemization is a special worry when translating ATP proofs in textual Isar proofs. Conjecture and axioms are treated differently because of their different polarities. By convention, the axioms are positive and the conjecture is negative.[3] In the positive case, skolemization eliminates the essentially existential quantifiers (i.e., the positive occurrences of $\exists$ and the negative occurrences of $\forall$). In the negative case, it eliminates the essentially universal quantifiers. Negative skolemization is usually called dual skolemization or herbrandization [32].

5.1 The Positive Case

We start with the easier, positive case. Consider the following concrete but archetypal extract from an E or Vampire proof:

11 axiom $\forall X.\, \exists Y.\, \mathsf{p}(X, Y)$ *exists_P*
53 plain $\forall X.\, \mathsf{p}(X, \mathsf{y}(X))$ 11 skolem

In Isar, a similar effect is achieved using the **obtain** command:

obtain y **where** $\forall x.\, \mathsf{P}\, x\, (y\, x)$ **by** (*metis exists_P*)

In the abstract Isar-like data structure that stores direct proofs, the inference is represented as

obtain $[y]$ **where** 53: $\forall x.\, \mathsf{P}\, x\, (y\, x)$ [*exists_P*] []

The approach works for arbitrary quantifier prefixes. All essentially existential variables can be eliminated simultaneously. For example, the ATP proof fragment

18 axiom $\forall V.\, \exists W.\, \forall X.\, \exists Y.\, \forall Z.\, \mathsf{q}(V, W, X, Y, Z)$ *exists_Q*
90 plain $\forall V.\, \forall X.\, \forall Z.\, \mathsf{q}(V, \mathsf{w}(V), X, \mathsf{y}(V, X), Z)$ 18 skolem

is translated to

obtain $[w, y]$ **where** 90: $\forall v\, x\, z.\, \mathsf{Q}\, v\, (w\, v)\, x\, (y\, v\, x)\, z$ [*exists_Q*] []

Reconstruction crucially depends not only on *metis*'s clausifier but also on its support for mildly higher-order problems, because of the implicit existential quantification over the Skolem function symbols in **obtain**. Indeed, *metis* is powerful enough to prove a weak form of the HOL axiom of choice:

[3] This choice is justifiable from the point of view of an automatic prover that attempts to derive $\bot$ from a set of axioms and a negated conjecture, because all the premises it starts from and the formulas it derives are then considered positive.

lemma $(\forall x.\ \exists y.\ P\ x\ y) \Longrightarrow \exists f.\ \forall x.\ P\ x\ (f\ x)$
  by *metis*

Of course, nothing is derived ex nihilo: *metis* can prove the formula only because its clausifier depends on the axiom of choice in the first place. Furthermore, *metis* will succeed only if its clausifier puts the arguments to the Skolem functions in the same order as in the proof text. This is not difficult to ensure in practice: Both E and *metis* respect the order in which the universal variables are bound, whereas SPASS and Vampire use the opposite order, which is easy to reverse.

Positive skolemization suffers from a technical limitation connected to polymorphism: Lemmas containing polymorphic skolemizable variables cannot be reconstructed, because the variables introduced by **obtain** must have a ground type.[4] An easy workaround would be to relaunch Sledgehammer with a monomorphizing type encoding [13, Section 3] to obtain a more suitable ATP proof, in which all types are ground. A more challenging alternative would involve detecting which monomorphic instances of the problematic lemmas are needed and re-engineer the proof accordingly.

5.2 The Negative Case

In the ATPs, negative skolemization of the conjecture is simply reduced to positive skolemization of the negated conjecture. For example:

25	conj	$\forall V.\ \exists W.\ \forall X.\ \exists Y.\ \forall Z.\ \mathsf{q}(V, W, X, Y, Z)$	*goal*	
41	neg_conj	$\neg\ \forall V.\ \exists W.\ \forall X.\ \exists Y.\ \forall Z.\ \mathsf{q}(V, W, X, Y, Z)$	25	negate
43	neg_conj	$\neg\ \exists W.\ \exists Y.\ \mathsf{q}(\mathsf{v}, W, \mathsf{x}(W), Y, \mathsf{z}(W, Y))$	41	skolem

However, once the proof has been turned around in Sledgehammer, the last two lines are unnegated and exchanged: First, a proof of the (unnegated) conjecture is found for specific fixed variables (cf. formula 43 above); then these are generalized into quantified variables (cf. formula 41). A natural name for this process is *unherbrandization*. In Isar, the **fix** command achieves a similar effect, as in the example below:

lemma $\bigwedge x.\ \mathsf{R}\ x$
proof $-$
  fix x
  ⟨*core of the argument*⟩
  show $\mathsf{R}\ x$ **by** (*metis* ...)
qed

However, this works only for the outermost universal quantifiers. Since we cannot expect users to always state their conjectures in this format, we must generally use a nested proof block, enclosed in curly braces. Thus, the ATP proof fragment presented above is translated to

lemma $\forall v.\ \exists w.\ \forall x.\ \exists y.\ \forall z.\ \mathsf{Q}\ v\ w\ x\ y\ z$
proof $-$
  { **fix** $v\ x\ z$
    ⟨*core of the argument*⟩
    have $\exists w\, y.\ \mathsf{Q}\ v\ w\ (x\ w)\ y\ (z\ w\ y)$ **by** (*metis* ...) }

[4] In Isabelle, only constants can be polymorphic, and constants can only be introduced at the top level of the theory text, typically via a definition.

thus $\forall v.\ \exists w.\ \forall x.\ \exists y.\ \forall z.\ \mathsf{Q}\ v\ w\ x\ y\ z$ **by** *metis*
qed

Seen from outside, the nested block proves the formula $\bigwedge v\, x\, z.\ \exists w\, y.\ \mathsf{Q}\ v\ w\ (x\ w)\ y\ (z\ w\ y)$. From there, *metis* derives the desired formula $\forall v.\ \exists w.\ \forall x.\ \exists y.\ \forall z.\ \mathsf{Q}\ v\ w\ x\ y\ z$, in which the quantifiers alternate. In the data structure that stores direct Isar-like proofs, the proof would be represented as

have 41: $\forall v.\ \exists w.\ \forall x.\ \exists y.\ \forall z.\ \mathsf{Q}\ v\ w\ x\ y\ z$ []
[**fix** $[v, x, z]$
⟨*core of the argument*⟩
have 43: $\exists w\, y.\ \mathsf{Q}\ v\ w\ (x\ w)\ y\ (z\ w\ y)$ [...] []]

An easy optimization, which is not yet implemented, would be to omit the nested proof block for conjectures of the form $\bigwedge x_1 \ldots x_n.\ \phi$, where ϕ contains no essentially universal quantifiers. It should also be possible to move the inferences that do not depend on the herbrandized symbols outside the nested block.

Given a HOL problem, the *metis* method clausifies it and translates it to first-order logic, invokes the Metis superposition prover, and replays the Metis inferences using suitable Isabelle tactics. Skolemization is simulated using Hilbert's choice operator ε [63]; for example, $\forall x.\ \exists y.\ \mathsf{P}\ x\ y$ is skolemized into $\forall x.\ \mathsf{P}\ x\ (\varepsilon y.\ \mathsf{P}\ x\ y)$. A newer experimental skolemizer exploits Isabelle's schematic variables to eliminates the dependency on Hilbert's choice [10, Section 6.6.7], only requiring the axiom of choice to move the existentials to the front. Whichever approach is used, Sledgehammer's textual proof construction exploits *metis*'s machinery instead of replicating it textually.

6 Postprocessing

After an ATP proof has been redirected and transformed into a direct Isar proof, a number of postprocessing steps take place to improve its legibility, efficiency, and in some cases correctness:

1. Sledgehammer users waste precious time on proofs that fail or take too long. *Proof preplay* addresses this by testing the generated proofs for a few seconds before displaying them (Section 6.1). Preplaying is performed in conjunction with most of the other steps to validate them in a pragmatic way.
2. Although *metis* is the proof method that resembles the external ATPs the most, it is often advantageous to try out *alternative proof methods* (Section 6.2).
3. The generated proofs can be arbitrarily detailed depending on which ATP is used. This *proof compression* collapses consecutive ATP inferences into single Isar inferences (Section 6.3).
4. ATPs frequently use many more facts than are necessary, making it harder for proof methods to re-find the proof. In addition, most proof methods are aware of background libraries and might not need to be given all the facts that an ATP needs. *Proof minimization* attempts to eliminate needless dependencies (Section 6.4).
5. Isabelle can *serialize* logical formulas as text strings, but it does not always understand its own output. Terms are often read back with overly general polymorphic types, resulting in failures. Annotating each subterm with type constraints impedes readability. Instead, Sledgehammer now employs an algorithm that introduces a locally minimal, complete set of type annotations (Section 6.5).

6.1 Proof Preplay

Isar proofs generated from ATP proofs sometimes fail. We already mentioned that skolemization is not supported for polymorphic variables (Section 5.2). The TSTP parser occasionally goes wrong if it encounters unexpected, undocumented syntax. The ATP proof can also contain inferences that are ill-typed from an Isabelle point of view—despite the use of globally sound encodings, individual inferences can violate the type discipline. Moreover, the proof reconstruction code is not bug-free. And even in the absence of errors, the Isar proofs can fail because *metis* or another proof method fails to discharge a proof obligation within a reasonable amount of time.

Therefore, it is advantageous to try out, or *preplay*, the proofs before presenting them to users. The proofs are then printed together with timing information, including warnings about timeouts or failures. Since Sledgehammer invokes multiple ATPs in parallel, users must often choose between several proofs. Based on the preplay information, they can make an informed decision while being spared the tedium of having to test them manually. Even when the Isar proof is not entirely correct, it is usually possible to repair it—sometimes by invoking Sledgehammer again on any failing inference steps.

The following example, based on an Isabelle formalization of the Robbins conjecture [82], demonstrates preplaying. When invoked on the goal $x \sqcup -x = -x \sqcup --x$ and configured to use Waldmeister, Sledgehammer produces the following output:

Try this: **by** (*metis huntington sup_assoc sup_comm*) (> 3 s, timed out).

Isar proof (54 steps, 1.33 s):

proof −
  have *f1*: $\bigwedge x_1\, x_2.\ -(-x_1 \sqcup x_2) \sqcup -(-x_1 \sqcup -x_2) = x_1$
    by (*metis huntington sup_comm*)
  have *f2*: $\bigwedge x_1\, x_2\, x_3.\ x_1 \sqcup (x_2 \sqcup x_3) = x_3 \sqcup (x_1 \sqcup x_2)$
    by (*metis sup_assoc sup_comm*)
  have $\bigwedge x_1\, x_2\, x_3.\ x_1 \sqcup (x_2 \sqcup x_3) = x_2 \sqcup x_1 \sqcup x_3$
    by (*metis sup_assoc sup_comm*)
  hence *f3*: $\bigwedge x_1\, x_2\, x_3.\ x_1 \sqcup (x_2 \sqcup x_3) = x_2 \sqcup (x_1 \sqcup x_3)$
    by (*metis sup_assoc*)
  have *f4*: $\bigwedge x_1\, x_2\, x_3.\ x_1 \sqcup (x_2 \sqcup x_3) = x_3 \sqcup (x_2 \sqcup x_1)$
    using *f2* **by** (*metis sup_comm*)
  have *f5*: $\bigwedge x_1\, x_2.\ x_1 = -(x_2 \sqcup -x_1) \sqcup -(-x_1 \sqcup -x_2)$
    using *f1* **by** (*metis sup_comm*)
  hence *f6*: $\bigwedge x_1\, x_2.\ x_1 = -(x_2 \sqcup -x_1) \sqcup -(-x_2 \sqcup -x_1)$
    by (*metis sup_comm*)
    ⋮
  hence $x \sqcup ---x = x \sqcup (-(-x \sqcup x) \sqcup -(--x \sqcup --x))$
    using *f10* **by** *metis*
  hence $x \sqcup ---x = x \sqcup -x$
    using *f12* **by** *metis*
  hence $--x = -(x \sqcup -x) \sqcup -(-x \sqcup ---x)$
    using *f6* **by** *metis*
  hence $--x = -(x \sqcup -x) \sqcup -(-x \sqcup -x)$
    using *f22* **by** *metis*
  hence $--x = x$

using *f5* **by** *metis*
thus $x \sqcup -x = -x \sqcup --x$
by (*metis sup_comm*)
qed

Waldmeister found a difficult proof involving the same three lemmas over and over (*huntington*, *sup_assoc*, and *sup_comm*). However, *metis* fails to re-find the proof within 3 seconds, as indicated by the mention "> 3 s, timed out" on the first line. (Indeed, *metis* or any other Isabelle proof method stands no chance even if given several minutes.) In contrast, the above (abridged) 54-step Isar proof was replayed in 1.33 seconds. Users can click it to insert it in their proof text and move on to the next conjecture.

Behind the scenes, the Isar proof preplay procedure starts by enriching the context with all the local facts introduced in the proof (*f1*, *f2*, etc.). For each inference $\Gamma \vdash \phi$, it measures the time *metis* takes to deduce ϕ from Γ and stores it in a data structure. The total is printed at the end, with a '>' prefix if any of the *metis* calls timed out. In the rare event that a *metis* call failed prematurely, Sledgehammer displays the mention "failed" in the banner.

An alternative approach would have been to have Isabelle parse the Isar proof using its usual interfaces, thereby covering more potential sources of error. For example, with our approach the Isabelle terms are not printed and re-parsed; because of Isabelle's flexible syntax, parsing is problematic despite our best efforts (Section 6.5). On the other hand, the better coverage would come at the price of additional overhead, and it is not clear how to achieve it technically. More importantly, the alternative approach offers no way to collect timing information on a per-step basis. This information is essential for proof compression (Section 6.3); recomputing it would waste the user's time.

6.2 Alternative Proof Methods

With proof preplay in place, it is easy to try out other proof methods than *metis*. This is especially useful to reconstruct proofs with theory-specific or higher-order reasoning, for which *metis* is likely to fail. For each inference in an Isar proof, a selection of the following methods is tried:

metis	the superposition prover Metis [36, 63]
meson	a model elimination procedure [45, 59]
satx	a simple SAT solver [83]
blast	an untyped tableau prover [60]
simp	conditional equational reasoning [52]
auto	combination of equational and tableau reasoning [58]
fastforce	more exhaustive version of *auto*
force	even more exhaustive version of *auto*
moura	a custom method for reconstructing Z3 skolemization (Section 7.5)
linarith	a decision procedure for linear arithmetic
presburger	another decision procedure for linear arithmetic [23]
algebra	a normalization method for rings

6.3 Proof Compression

It is often beneficial to compress Isar proofs by eliminating intermediate steps. Compressed proofs can be faster to recheck. When the Robbins example from Section 6.1 is compressed

from 54 to 29 steps, Isabelle also takes nearly half a second less to process it. Moreover, many users prefer concise Isar proofs, either because they want to avoid cluttering their theory files or because they find the shorter proofs simpler to understand. Of course, compression can also be harmful: A *metis* one-line proof is nothing but an Isar proof compressed to the extreme, and it can be both very slow and very cryptic.

Whereas intelligibility is in the eye of the beholder, speed can be measured precisely via preplay. Our compression procedure considers candidate pairs of inferences and performs the merger if the resulting inference is fast enough—no more than 50% slower than the two original inferences taken together. This 50% tolerance factor embodies a trade-off between processing speed and conciseness. Given the inferences $\Gamma_1 \vdash \phi_1$ and $\Gamma_2 \uplus \{\phi_1\} \vdash \phi_2$, where ϕ_1 is not referenced elsewhere in the proof (in an antecedent), the merged inference is $\Gamma_1 \cup \Gamma_2 \vdash \phi_2$.

The algorithm consists of the following steps:

1. Initialize the worklist with all inferences $\Gamma \vdash \phi$ such that ϕ is referenced only once in the rest of the proof.
2. If the worklist is empty, stop; otherwise, take an inference $\Gamma_1 \vdash \phi_1$ from the worklist.
3. Let $\Gamma_2 \uplus \{\phi_1\} \vdash \phi_2$ be the unique inference that references ϕ_1. Try to merge the two inferences as described above. If this succeeds, add any emerging singly referenced facts belonging to $\Gamma_1 \cap \Gamma_2$ to the worklist.
4. Go to step 2.

Step 2 nondeterministically picks an inference. Our implementation prefers inferences with long formulas, because these clutter the proof more. In step 3, merging the two inferences may give rise to new singly referenced facts ϕ that were referenced by both ϕ_1 and ϕ_2 (i.e., $\phi \in \Gamma_1 \cap \Gamma_2$) but not by any other inferences.

The process is guided by the performance of preplaying. Users who want to understand the proof may find that too many details have been optimized away. For them, an option controls the compression factor, which bounds the number of mergers before the algorithm stops in relation to the length of the uncompressed proof.

6.4 Proof Minimization

Sledgehammer's minimization tool takes a set of facts appearing in an inference and repeatedly calls the prover with subsets of the facts to find a locally minimal set. Depending on the number of initial facts, it relies on either of these two algorithms:

1. The naive linear algorithm attempts to remove one fact at a time. This can require as many prover invocations as there are facts in the initial set.
2. The binary algorithm, due to Bradley and Manna [19, Section 4.3], recursively bisects the facts. It performs best when a small fraction of the facts are actually required [16, Section 7].

Given an n-fact proof, the linear algorithm always needs n calls to the external prover, whereas the binary algorithm requires anywhere between $\log_2 n$ and $2n$ calls, depending on how many facts are actually needed [16, Section 7.1]. Sledgehammer selects the binary algorithm if $n > 20$. The binary algorithm is used for ATPs that do not produce proofs or unsatisfiable cores, in which case n could be in the hundreds. For minimizing individual inferences in an Isar proof, the linear algorithm is generally preferable.

Because of the multiple prover invocations (often with unprovable problems), minimization often consumes more time than the proof search itself. An obvious improvement to the textbook minimization algorithms is to inspect the ATP proofs and eliminate any fact that is not referenced in it. Another improvement is to use the time required by the last successful proof as the timeout for the next one, instead of a fixed, necessarily liberal timeout. Both improvements are implemented in Sledgehammer and are described in more detail elsewhere [10, Section 6.6.5].

6.5 Serialization

To ensure that types are inferred correctly when the generated HOL formulas are parsed again by Isabelle, it is necessary to introduce type annotations. However, redundant annotations should be avoided: If we insisted on annotating each subterm, the simple equation $xs = ys$, where xs and ys range over lists of integers, would be rendered as

$$((\mathrm{op} = :: \mathit{int\ list} \Rightarrow \mathit{int\ list} \Rightarrow \mathit{bool})\ (\mathit{xs} :: \mathit{int\ list}) :: \mathit{int\ list} \Rightarrow \mathit{bool})\ (\mathit{ys} :: \mathit{int\ list}) :: \mathit{bool}$$

The goal is not to make the Hindley–Milner inference redundant but rather to guide it.

Paulson and Susanto's prototype generates no type annotations at all. Isabelle provides alternative print modes (e.g., one mode annotates all bound variables at the binding site) but none of them is complete. This may seem surprising to users familiar with other proof assistants, but Isabelle's extremely flexible syntax, combined with type classes, means that some terms cannot be parsed back.

We implemented a custom "print mode" for Sledgehammer, which might become an official Isabelle mode in a future release. The underlying algorithm computes a locally minimal set of type annotations for a formula and inserts the annotations. In Isabelle, type annotations are represented by a polymorphic constant ann_α of type $\alpha \Rightarrow \alpha$ that can be thought of as the identity function. The term $\mathsf{ann}_\tau\ t$ is printed as $t :: \tau$. In the presentation below, the notation t^τ indicates that term t has type τ.

Given a well-typed formula ϕ to annotate, the algorithm starts by replacing all the types in ϕ by the special placeholder _. It then infers the most general types for ϕ using Hindley–Milner inference, resulting in a formula $\phi^\star$ in which the placeholders are instantiated. Let α_1, $\ldots$, α_m be the type variables occurring in $\phi^\star$. Next, the algorithm computes the substitution $\rho = \{\alpha_1 \mapsto \tau_1, \ldots, \alpha_m \mapsto \tau_m\}$ such that $\phi^\star\rho = \phi$, which must exists if ϕ is well-typed and the inferred types in $\phi^\star$ are the most general. Finally, the algorithm inserts type annotations of the form $:: \tau$ that *cover* all the type variables α_i in ρ's domain—i.e., such that each type variable α_i occurs in at least one type annotation.

The last step is where the complexity arises. The algorithm assigns a cost to each candidate site t^τ in ϕ where a type annotation can be inserted. The cost is given as a triple of numbers:

$$\text{cost of } t^\tau = (\text{size of } \tau,\ \text{size of } t,\ \text{preorder index of } t \text{ in } \phi)$$

Triples are compared lexicographically. The first two components encode a preference for smaller annotations and smaller annotated terms. The third component resolves ties by preferring annotations occurring closer to the beginning of the printed formula. All subterms of ϕ are potential candidates to carry type annotations. (It would be desirable to consider the binding sites of variables in quantifiers and λ-abstractions as candidates as well, but unfortunately these are simply name–type pairs and not terms in Isabelle.) Each site t^τ is also associated with the set of type variables α_i it covers.

The goal is to compute a locally minimal set of sites that completely covers all type variables. The resulting cost need not be a global minimum, though; computing the minimum amounts to solving the weighted set cover problem, which is NP-hard [40]. One could probably use a SAT solver to solve the problem efficiently, but we prefer a more direct greedy approach, which is polynomial and produces satisfactory results in practice.

Starting with the set of all possible sites, the algorithm iteratively removes the most expensive redundant site until the set is minimal in the sense that removing any site from it would make it incomplete. This reverse greedy approach ensures that a minimal set will be reached eventually. In contrast, the standard greedy approach could yield a too large set: For the term $\mathsf{h}^{nat\Rightarrow real}\ \mathsf{c}^{nat}$ generalized to $\mathsf{h}^{\alpha\Rightarrow\beta}\ \mathsf{c}^{\alpha}$, it would first pick c to cover α, only to find out that h must be annotated as well to cover β, making the first site redundant.

The names of the variables α_i introduced in $\phi^\star$ are irrelevant as long as they are fresh. In a postprocessing step, variables that occur only once anywhere inside $\tau_1,\ldots,\tau_m$ are replaced by _, and annotations $:: \tau$ that cover only variables converted to _ are omitted. Thus, the formula $\mathsf{length}\ ([] :: \alpha\ list) = 0$ is printed as $\mathsf{length}\ [] = 0$ without undesirable gain of generality.

7 System-Specific Technicalities

Although most of the Isar proof construction module is generic, some work is necessary to integrate specific ATPs. We have so far integrated eight provers, focusing on those that are highly performant or that show promise and whose proof output is detailed enough. They are reviewed in turn below, starting with the Musterkind E and concluding with Satallax and its nonstandard proof output.

7.1 E

E is perhaps the prover that best implements the TPTP and TSTP standards. It is also one of the most performant systems, as judged by its consistent second-place rankings at CASC [78]. Regrettably, it does not support the typed syntax TFF0 yet, meaning that all type information must be encoded.

E's proof output is remarkably free of hard-to-translate constructs. Skolemization is recorded in the proof as applications of the 'skolemize' rule, in the style demonstrated in Sections 3 and 5.

7.2 Vampire

Vampire also implements TPTP and TSTP, including TFF0. It is the strongest system at CASC. Its output is similar to E's. Skolemization is recorded as applications of the 'skolemisation' rule. As noted earlier, Skolem arguments appear in the reverse order of that expected by *metis*, but they can easily be reversed.

Vampire's preprocessor implements some optimizations that introduce symbols. Another difficulty is that splitting yields proofs that are beyond what our framework can handle: Even if the framework's output format supports case analyses, its input does not. Some versions of the prover even output huge binary decision diagrams, without attempting to integrate them with the TSTP syntax. These features are disabled when producing an Isar

proof, following a hint we found at a tutorial [34, p. 20]. They can still be used for proof search proper, for reducing the hundreds of facts given to the prover to the (usually) much smaller set actually needed for the proof.

Although Vampire supports TFF0 in its input, its proofs contain no type information. Types can normally be inferred from the function and predicate symbols occurring in the problem (whose types are encoded in the names [13, Section 3]), with the exception of formulas of the form $\forall X\, Y.\ X = Y$, for which reconstruction will fail.

7.3 SPASS

SPASS generates proofs in its custom DFG (Deutsche Forschungsgemeinschaft) format only, even though it can parse TPTP FOF. Fortunately, DFG is based on similar concepts and can be represented using the same data structure as TSTP.

Splitting must be disabled to obtain intelligible proofs. (SPASS records splitting in an obfuscated way that makes it very difficult to analyze the proof afterward.) The main difficulty in integrating SPASS is that clausification, including skolemization, is not recorded in the proofs. The proof is expressed in terms of the clausified problem, as output by the SPASS's preprocessor, FLOTTER. This violates what we call the *Russian doll principle*—the notion that a metaprover B can encapsulate a prover A by reducing B-problems to A-problems and translating back A-solutions (proofs or models) to B-solutions.

But since SPASS is such a powerful prover, especially in the context of Isabelle and Sledgehammer [14], it is worthwhile to provide the missing link, namely, a translation of a proof of the clausified problem to a proof of the original problem.

Our solution is to unskolemize the problem clauses that appear in the proof and use that information to enrich the proof with skolemization inferences. Given a CNF problem, expressed as a single formula with disjunctions inside conjunctions, the unskolemization algorithm performs the following steps:

1. If there are no remaining Skolem symbols, return the universal closure of the formula.
2. If there exists a Skolem constant c, take the closure $\exists X.\ \phi[X/\mathsf{c}]$ of the formula ϕ and continue recursively with the body of $\exists X.\ \phi[X/\mathsf{c}]$.
3. Otherwise, let $\mathsf{f}(X, \bar{t})$ be an application of a Skolem function:
   3.1. Let χ, ψ be a partition of the conjuncts of ϕ such that each clause in χ contains an occurrence of a Skolem function $\mathsf{g}(Y, \bar{u})$ applied to a variable Y distinct from X, and each ψ does not.
   3.2. Take the closure $\forall X.\ \psi[\mathsf{f}_1(\overline{t_1})/\mathsf{f}_1(X, \overline{t_1}), \ldots, \mathsf{f}_n(\overline{t_n})/\mathsf{f}_n(X, \overline{t_n})]$, where each n-ary Skolem function f_j that takes X as its first argument is transformed into an $(n-1)$-ary function that does not.
   3.3. Proceed recursively with χ and the body of $\forall X.\ \psi[\mathsf{f}_1(\overline{t_1})/\mathsf{f}_1(X, \overline{t_1}), \ldots, \mathsf{f}_n(\overline{t_n})/\mathsf{f}_n(X, \overline{t_n})]$, and take the conjunction of the two results.

The algorithm will be illustrated on an example. Let $\mathsf{f}, \mathsf{g}, \mathsf{h}, \mathsf{k}$ be Skolem functions. Let

$$\mathsf{r}(A, D, E) \wedge \mathsf{p}(A, B, \mathsf{f}(A, B), C, \mathsf{g}(A, B, C)) \wedge \mathsf{q}(A, D, \mathsf{h}(A, D), E, \mathsf{k}(A, D, E))$$

be the conjunction of all the problem clauses that appear in the proof. The unskolemization algorithm produces the formula

$$\forall A.\ \big(\forall D.\ \exists H.\ \forall E.\ \exists K.\ \mathsf{r}(A, D, E) \wedge \mathsf{q}(A, D, H, E, K)\big) \wedge \big(\forall B.\ \exists F.\ \forall C.\ \exists G.\ \mathsf{p}(A, B, F, C, G)\big)$$

The steps of the algorithm are reproduced below:

$$\underbrace{\mathsf{r}(A,D,E) \wedge \mathsf{p}(A,B,\mathsf{f}(A,B),C,\mathsf{g}(A,B,C)) \wedge \mathsf{q}(A,D,\mathsf{h}(A,D),E,\mathsf{k}(A,D,E))}$$
$$\forall A.\ \underbrace{\mathsf{r}(A,D,E) \wedge \mathsf{p}(A,B,\mathsf{f}(B),C,\mathsf{g}(B,C)) \wedge \mathsf{q}(A,D,\mathsf{h}(D),E,\mathsf{k}(D,E))}$$
$$\big(\forall D.\ \underbrace{\mathsf{r}(A,D,E) \wedge \mathsf{q}(A,D,\mathsf{h},E,\mathsf{k}(E))}\big) \wedge \underbrace{\mathsf{p}(A,B,\mathsf{f}(B),C,\mathsf{g}(B,C))}$$
$$\exists H.\ \underbrace{\mathsf{r}(A,D,E) \wedge \mathsf{q}(A,D,H,E,\mathsf{k}(E))} \quad \forall B.\ \underbrace{\mathsf{p}(A,B,\mathsf{f},C,\mathsf{g}(C))}$$
$$\forall E.\ \underbrace{\mathsf{r}(A,D,E) \wedge \mathsf{q}(A,D,H,E,\mathsf{k})} \quad \exists F.\ \underbrace{\mathsf{p}(A,B,F,C,\mathsf{g}(C))}$$
$$\exists K.\ \mathsf{r}(A,D,E) \wedge \mathsf{q}(A,D,H,E,K) \quad \forall C.\ \underbrace{\mathsf{p}(A,B,F,C,\mathsf{g})}$$
$$\exists G.\ \mathsf{p}(A,B,F,C,G)$$

The unskolemization algorithm is not perfect. We believe it is complete for what one could call naive skolemization, but the algorithm implemented by FLOTTER is anything but naive [4]. Furthermore, it is possible to construct examples where clauses are derived from both the negated conjecture and from other facts, leading to confusion in the redirection algorithm. Nonetheless, our approach appears to work fairly well in practice, especially after disabling a number of FLOTTER optimizations by passing appropriate options.

7.4 Waldmeister

Regrettably, the official version of Waldmeister cannot parse TPTP. The version available remotely via the SystemOnTPTP service [76] includes a translator from the untyped TPTP CNF UEQ format, which targets unit equality provers, to Waldmeister's native format. A unit equality problems consists of n axioms of the form $t = u$ and one negated conjecture of the form $t \neq u$, where t and u are first-order terms.

Paradoxically, Waldmeister can output TSTP proofs. However, these contain a number of oddities: The original fact names are not preserved; to restore this information, we must compare the formulas in the proofs with those in the original problems, modulo variable renaming and symmetry of equality. Worse, the endgame of any Waldmeister proof is highly abnormal. It consists of four inferences of the following form:

1.0.0.0	conj	$s = t$		*goal*
1.0.0.1	plain	$u = t$	1.0.0.0, 0.*a.b.c*	reduction
1.0.0.2	plain	$u = u$	1.0.0.1, 0.*d.e.f*	reduction
1.0.0.3	plain	true	1.0.0.2	trivial

(The numbers 0.*a.b.c* and 0.*d.e.f* refer to formulas $s = u$ and $t = u$ derived earlier in the proof. The last four steps are always numbered 1.0.0.0 to 1.0.0.3.) The last inference would appear to derive true, which is not a particularly impressive achievement. The endgame makes more sense if we negate the four formulas and adjust the formula roles accordingly:

1.0.0.0	neg_conj	$s \neq t$		*negated goal*
1.0.0.1	neg_conj	$u \neq t$	1.0.0.0, 0.*a.b.c*	reduction
1.0.0.2	neg_conj	$u \neq u$	1.0.0.1, 0.*d.e.f*	reduction
1.0.0.3	neg_conj	false	1.0.0.2	trivial

The redirection algorithm takes over from there to produce a direct proof, as with the superposition provers:

have 1.0.0.1: $u = t$ [0.*d*.*e*.*f*] []
have 1.0.0.0: $s = t$ [1.0.0.1, 0.*a*.*b*.*c*] []

This can be shortened even further to

have 1.0.0.0: $s = t$ [0.*a*.*b*.*c*, 0.*d*.*e*.*f*] []

Another peculiarity of the Waldmeister integration is related to its restrictive logic. Logical connectives such as $\wedge$ and $\vee$ must be encoded as terms and axiomatized. Because the logic is so weak, it is not even possible to express the requirement that all Boolean-valued terms are equal to either true or false.

Unlike for the other provers, skolemization takes place in Sledgehammer. The skolemization steps are recorded so that they can be retrofitted to the Waldmeister proof, without requiring unskolemization (cf. SPASS, Section 7.3).

Essentially existential variables in the goal are translated to universal variables in the negated conjecture; thus, $\exists x.\ \mathsf{f}\ x = \mathsf{g}\ x$ is translated to $\mathsf{f}(X) \neq \mathsf{g}(X)$. This triggers the use of narrowing in Waldmeister, which results in even odder proofs than usual. Isar proof reconstruction currently fails in this case.

7.5 Z3

Z3 is integrated in Sledgehammer via the SMT-LIB format for problems. The original integration [15, 17] relied on version 1 of the format. Isabelle now uses version 2.

Z3 proofs are expressed in a custom format inspired by SMT-LIB. A proof is simply a (large) SMT-LIB term, where proof rules are represented by function symbols and formulas and terms appear unencoded as arguments to proof rules. Inferences can be reused thanks to a 'let' construct. Parsing a Z3 proof results in a directed acyclic graph whose sink node is $\bot$, much in the style of those obtained from TSTP-based provers.

For a number of years, proof reconstruction for Z3 was performed by a dedicated Isabelle proof method, called *smt*, that parses a Z3 proof and replays the inferences using standard Isabelle methods (such as *simp* and *arith*). This requires Z3 to be installed on the user's machine for replaying. Unfortunately, in some cases a specific version of the solver is needed to re-find a proof. As a matter of policy, any theory files included in the Isabelle distribution or the *Archive of Formal Proofs* [41] may not depend on *smt*.

A partial solution to these issues is to cache Z3 proofs in a file that accompanies the Isabelle theory. Whenever the theory is reprocessed, the cache is consulted before actually calling Z3. However, the cache is fragile: Even trivial changes such as renaming a constant will cause a lookup failure.

Isar proof reconstruction appears to be the superior approach: By representing Z3 proofs as structured Isabelle proofs, we finally get Z3 out of the replay loop. Some of the code from the *smt* method, such as the proof parser, can be reused.[5]

The first difficulty in translating Z3 proofs into Isar proofs is that Z3, like Waldmeister, does not identify the axioms by name. And like Waldmeister, it silently normalizes the formulas; for example, a formula of the form $P \longrightarrow Q \longrightarrow R$ in the problem may appear as $P \wedge Q \longrightarrow R$ in the proof. Such minor alterations are enough to cause reconstruction failures. Without closely inspecting the solver's code, it is impossible to tell whether we have identified all such cases.

[5] Newer versions of Z3 also support TPTP TFF0 as input and TSTP as output format. We ran into some basic parsing issues, due to unbalanced parentheses in the TSTP output, and since we already had a tried and tested parser for native proofs, we did not investigate this further.

One of the main strengths of SMT solvers is their support for linear arithmetic. When trying alternative proof methods (Section 6.2), Sledgehammer will use different subsets of methods for different Z3 inferences. In particular, it will first try *linarith*, *presburger*, and *algebra* to replay an arithmetic inference, before falling back on other methods.

Z3 proofs include definitions, which are axioms of the form $\mathsf{f}(X_1,\ldots,X_n) = \ldots$ where f is a fresh symbol that does not occur on the right-hand side. These definitions are simply inlined. It should not be difficult to extend the Isar proof reconstruction module to support these: Isar provides **let** and **def** constructs that could be used to mimic the Z3 proof.

Similarly, Z3 supports nested subproofs. At any point in the proof graph, a 'hypothesis' rule introduces a local assumption, which is eventually discharged by a 'lemma' rule. When they are cleanly nested, the 'hypothesis' and 'lemma' rules can be seen as delimiters for a nested proof block. It should be possible to extend the module's data structures described in Section 3 to support nested proof blocks. Isar provides the syntax $\{\ldots\}$ that could be used for this. For the moment, each hypothesis is simply added explicitly to all formulas that appear in the nested block, after which the block structure can be ignored.

Z3 has a peculiar notion of skolemization. A typical prover, such as E or Vampire, can go from the original axiom to its skolemized version in one inference:

11	axiom	$\forall X.\, \exists Y.\, \mathsf{p}(X, Y)$	*exists_P*
53	plain	$\forall X.\, \mathsf{p}(X, \mathsf{f}(X))$	11 skolem

In contrast, Z3 introduces a skolemization axiom, introduced by the 'sk' rule. The axiom can be used to perform skolemization:

11	axiom	$\forall X.\, \exists Y.\, \mathsf{p}(X, Y)$	*exists_P*
12	axiom	$\forall X.\, (\exists Y.\, \mathsf{p}(X, Y)) = \mathsf{p}(X, \mathsf{f}(X))$	sk
53	plain	$\forall X.\, \mathsf{p}(X, \mathsf{f}(X))$	11, 12

At the Isar level, the skolemization axiom (step 12) corresponds to an **obtain**, which itself embodies an existential $(\exists p.\ \ldots)$.[6] The proof obligation is a tautology, but one that is too difficult for *metis*, *blast*, *meson*, or any of the other standard methods.

Our initial approach was to rewrite the skolemization axiom into a syntactically weaker version, by moving the outermost universal quantifiers under the equivalence:

12	axiom	$(\forall X.\, \exists Y.\, \mathsf{p}(X, Y)) = (\forall X.\, \mathsf{p}(X, \mathsf{f}(X)))$	sk

One direction of the equivalence amounts to skolemization as performed by typical provers, whereas the other direction is trivial. This worked in some cases (such as the simple example above) but failed in other cases, where the stronger formula was necessary.

Instead, we developed a simple proof method, called *moura* (after Leonardo de Moura, who implemented Z3's skolemizer), that can prove such formulas in Isabelle from the axiom of choice: $(\forall x.\, \exists y.\, q\ x\ y) \Longrightarrow (\exists f.\, \forall x.\, q\ x\ (f\ x))$. The *moura* method consists of a call to *auto* augmented with the axiom of choice as a so-called safe introduction rule, meaning that the axiom will be aggressively resolved against the goal. The *auto* invocation is optionally followed by *metis* or *blast* to finish the work if necessary.

As an example, consider the skolemization axiom $\forall X.\, (\exists Y.\, \mathsf{p}(X, Y)) = \mathsf{p}(X, \mathsf{f}(X))$ from above. The corresponding Isabelle proof obligation is $\exists f.\, \forall x.\, (\exists y.\, \mathsf{p}\ x\ y) = \mathsf{p}\ x\ (f\ x)$. Applying the axiom of choice as an introduction rule yields the goal $\forall x.\, \exists y.\, (\exists y.\, \mathsf{p}\ x\ y) = \mathsf{p}\ x\ y$, a tautology that can easily be discharged by *auto*.

[6] Strictly speaking, the proof obligation associated with an **obtain** has the form $(\bigwedge p.\ \ldots \Longrightarrow Q) \Longrightarrow Q$. However, this is logically equivalent to $\exists p.\ \ldots$, and proof methods such as *metis* can cope with both forms.

In general, the proof obligation associated with a skolemization axiom can start with n existential quantifiers, if n Skolem functions are introduced simultaneously. This is handled by augmenting *auto* with generalized variants of the axiom of choice, of the form $(\forall x.\, \exists y_1 \ldots y_n.\, q\ x\ y_1 \ldots y_n) \Longrightarrow (\exists f_1 \ldots f_n.\, \forall x.\, q\ x\ (f_1\ x) \ldots (f_n\ x))$, all of which are consequences of the standard axiom of choice (the $n = 1$ case).

7.6 veriT

The SMT solver veriT was designed to produce highly detailed proofs, to facilitate its integration with proof assistants (notably Coq [2]), while offering reasonable performance. Its proof format was carefully designed to possibly serve as a standard [9]. The format is conceptually similar to TSTP, with one line per inference.

Skolemization is captured by a 'tmp_skolemize' rule, which is similar to the corresponding rules found in typical provers. A connected rule is called 'tmp_ite_elim': From an antecedent of the form $(\text{if } s \text{ then } t \text{ else } u) = v$, it derives $\mathsf{c} = v \wedge (\text{if } s \text{ then } \mathsf{c} = t \text{ else } \mathsf{c} = u)$. The constant c can be thought of as the Skolem constant emerging from $\exists c.\ c = v \wedge (\text{if } s \text{ then } c = t \text{ else } c = u)$. Strangely enough, veriT will sometimes introduce the same constant c multiple times, but always specified by the same formula. These duplicates must be coalesced, to avoid introducing several **obtain** variables with the same names, each shadowing the previous one, resulting in reconstruction failures.

The solver's proof output also features nested proof blocks with local assumptions. These are eliminated in the same way as for Z3.

7.7 LEO-II

LEO-II produces its proofs in the TSTP format. Unusually for an ATP, its proof format is thoroughly documented [75]. LEO-II proofs typically consist of a number of generally higher-order inferences followed by a single first-order inference, found by the underlying first-order ATP (by default, E), that derives $\bot$.

The higher-order steps performed by LEO-II itself are reconstructed by trying various Isabelle proof methods, in the hope that one of them will succeed. Even *metis*, which is primarily first-order, will sometimes succeed at solving higher-order problems, thanks to its encoding of λ-abstractions as combinators. (Recall that *metis* is strong enough to justify skolemization steps in Isar proofs, which are beyond first-order logic.) Extensionality, as embodied by the rule 'extcnf_equal_neg', is handled by passing Isabelle's *ext* axiom, $(\forall x.\, f\ x = g\ x) \Longrightarrow f = g$, to the proof method.

The last step can in principle be replayed by *metis*, but this may take too long (or fail for technical reasons). In particular, LEO-II sometimes fails to extract the necessary lemmas from an E proof and will then return a ridiculously large set of spurious dependencies. LEO-II also provides an option (`-proofoutput 2`) that translates the individual E inferences into LEO-II inferences, to embed them into the larger LEO-II proof, but this mode of operation is extremely slow and failed on all examples we considered.

LEO-II proofs contain a number of so-called logistic rules, whose succedent does not logically follow from the antecedent. Most of these are disabled by specifying the appropriate options (`-notReplLeibnizEQ -notReplAndrewsEQ -notUseExtCnfCmbd`).

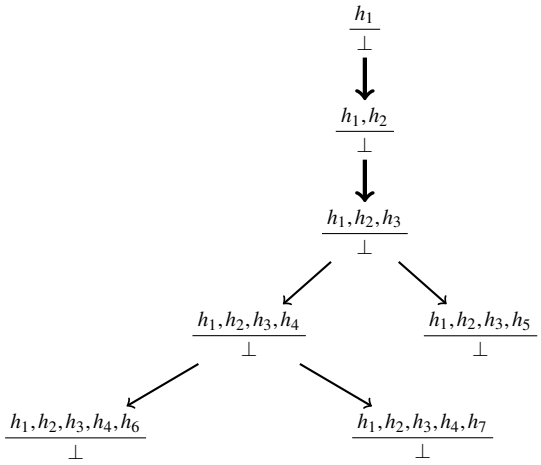


Fig. 5 Graph for a proof produced by Satallax

7.8 Satallax

Satallax supports three output formats: a TSTP-like format, Coq's Ltac tactic language, and Coq proof terms [21]. The TSTP-like format was added at our request and appears to be the most appropriate for reconstruction in Isabelle, but it deviates from standard TSTP in important, undocumented ways. Often, we were able to make sense of it only by inspecting the corresponding Coq proofs. In the Isabelle integration, a preprocessor transforms the TSTP-like proof into a standard proof by contradiction before the rest of the translation pipeline can take over.

A TSTP-like proof produced by Satallax can be seen as a tree, as depicted in Figure 5. The root corresponds to the negated conjecture. The edges leaving from one node denote an inference that gives rise to one or two new goals. Compared with standard proofs by contradiction, which derive formulas from formulas, this goal-directed approach is backward. It is reminiscent of the interaction in tactic-based systems such as Coq and Isabelle.

A node is a set of hypotheses and the goal $\bot$. We write $\frac{h_1,\ldots,h_n}{\bot}$ to indicate that $\bot$ follows from the hypotheses $h_1,\ldots,h_n$. The leaf nodes are annotated by a proof rule that detects contradictory hypotheses. It is convenient to distinguish between the initial linear fragment of the tree (with the thicker edges) and the branching part. The linear part can be directly translated to a standard TSTP proof, as follows.

- h_0 is always the conjecture:
  0 conj h_0
- h_1 is always the negated conjecture:
  1 neg_conj $h_1 \implies \bot$ 0
- For each step in the linear part, if $\frac{h_1,\ldots,h_n,h_{n+1}}{\bot}$ is derived from $\frac{h_1,\ldots,h_n}{\bot}$, then we can produce the following inference:
  n plain h_n $1,2,\ldots,n-1$
  $n+1$ plain h_{n+1} $1,2,\ldots,n$

The translation works recursively for the branching part, as follows.

- A leaf $\frac{h_1,\ldots,h_n}{\bot}$ is translated to

13 plain $h_1 \Longrightarrow \cdots \Longrightarrow h_n \Longrightarrow \bot$

– Given a node $\dfrac{h_1,\ldots,h_n}{\bot}$ that gives rise to $\dfrac{h_1,\ldots,h_n,h_{n+1}}{\bot}$ and $\dfrac{h_1,\ldots,h_n,h_{n+2}}{\bot}$, two goals that were translated to

12 plain $h_1 \Longrightarrow \cdots \Longrightarrow h_n \Longrightarrow h_{n+1} \Longrightarrow \bot$ ...

13 plain $h_1 \Longrightarrow \cdots \Longrightarrow h_n \Longrightarrow h_{n+2} \Longrightarrow \bot$ ...

the node of interest is translated to

14 plain $h_1 \Longrightarrow \cdots \Longrightarrow h_n \Longrightarrow \bot$ 12, 13

Observe that the dependencies are reversed: Where Satallax reduced one goal to two new subgoals, the repaired TSTP proof derives one formula (14) from a pair of formulas (12 and 13).

The hypotheses h_i can heavily burden the formulas and hence the resulting Isar proofs. Fortunately, we can often simplify them. The new hypotheses in the linear part (h_2 and h_3 in our example) are unconditionally true, because they occur before the first case distinction. Thus we can remove them from the chain of implications and add them to the dependencies. For example, instead of

7 plain $h_1 \Longrightarrow h_2 \Longrightarrow h_3 \Longrightarrow h_4 \Longrightarrow h_7 \Longrightarrow \bot$

the translation can produce

7 plain $h_4 \Longrightarrow h_7 \Longrightarrow \bot$ h_1,h_2,h_3

Another, more minor issue with reconstructing Satallax proofs is that invocations of extensionality are implicit in the proof, whereas Isabelle tactics do not apply it by default. As a workaround, the translation adds Isabelle's *ext* axiom as a dependency to each inference, relying on proof minimization (Section 6.4) to eliminate it where it is not needed.

8 Examples

The Isar proof construction module has been part of Isabelle ever since Paulson and Susanto implemented their prototype. However, it took several more years before we found it robust enough to have Sledgehammer run it whenever *metis* fails. Since then, Sledgehammer-generated Isar proofs have started appearing in user formalizations, usually in the face of a *metis* failure. We also hear from users who activated the feature to better understand a proof, confirming our hypothesis that machine-generated textual proofs can help experts who must satisfy their curiosity. As one user remarked, "Reading a proof that nobody wrote [is] a very nice sensation" [22].

To give a flavor of the Isar proofs that arise in practice, we present some specimens that we found in the *Archive of Formal Proofs* [41], a collection of user-contributed Isabelle formalizations (Section 8.1). These examples are complemented by a few more that arose as we worked on our own formalization (Section 8.2). The examples are reproduced almost exactly as we found them, except for some minor reformatting and renaming. We have deliberately chosen specimens at both ends of the readability gradients, to demonstrate both the strengths and the weaknesses of our approach in practice.

8.1 *Archive of Formal Proofs*

The first example originates from a formalization of the Babylonian method for computing *n*th roots [80]. Judging from the style, it appears not to have been tampered with:

have *f1*: $\forall n.$ rat_of_int $\lfloor$rat_of_nat $n\rfloor =$ rat_of_nat n
  using *of_int_of_nat_eq* **by** *simp*
have *f2*: $\forall n.$ real_of_int $\lfloor$rat_of_nat $n\rfloor =$ real n
  using *of_int_of_nat_eq real_eq_of_nat* **by** *auto*
have *f3*: $\forall i\ ia.\ \lfloor$rat_of_int i / rat_of_int $ia\rfloor = \lfloor$real_of_int i / real_of_int $ia\rfloor$
  using *div_is_floor_divide_rat div_is_floor_divide_real* **by** *simp*
have *f4*: $0 <$ rat_of_nat $p\rfloor$
  using p **by** *simp*
have $\lfloor S\rfloor \leq s$
  using *less floor_le_iff* **by** *auto*
hence $\lfloor$rat_of_int $\lfloor S\rfloor$ / rat_of_nat $p\rfloor \leq \lfloor$rat_of_int s / rat_of_nat $p\rfloor$
  using *f1 f3 f4* **by** (*metis div_is_floor_divide_real zdiv_mono1*)
hence $\lfloor S$ / real $p\rfloor \leq \lfloor$rat_of_int s / rat_of_nat $p\rfloor$
  using *f1 f2 f3 f4* **by** (*metis div_is_floor_divide_real floor_div_pos_int*)
thus S / real $p \leq$ real_of_int $(s$ div int $p) + 1$
  using *f1 f3*
  by (*metis div_is_floor_divide_real floor_le_iff floor_of_nat less_eq_real_def*)

The argument is fairly readable by the standards of machine-generated proofs. Each step is an unconditional equality or inequality.

The next example is extracted from a theory for verifying network security policies [27]. All the steps are discharged by *metis*, probably because an older version of Sledgehammer was used, which did not try alternative proof methods:

have *f1*: $\forall a\ as.\ -$ insert $(a :: \alpha)$ (coset as) = set $as - \{a\}$
  by (*metis compl_coset insert_code*(2) *set_removeAll*)
hence *f2*: $\forall f\ a\ l.\ f$ ‘ $(-$ insert $(a :: \alpha)$ (coset (succ_tran (undir l) a))) =
    set (map f (removeAll a (succ_tran (undir l) a)) :: *node_config list*)
  by (*metis undir_reach_def sinvar_eq_help1 set_removeAll*)
have *f3*: $\forall a\ l.\ -$ insert $(a :: \alpha)$ (coset (succ_tran (undir l) a)) =
    SNI.undir_reach (lg2g l) a
  using *f1* **by** (*metis SNI.undir_reach_def succ_tran_correct*
      undir_correct)
have $\neg\ nP$ ‘ $(-$ insert x (coset (succ_tran (undir G) x))) $\subseteq \{$Unrel$\} \vee$
    nP ‘ $(-$ insert x (coset (succ_tran (undir G) x))) $\subseteq \{$Unrel$\}$
  by *metis*
moreover
{ **assume** nP ‘ $(-$ insert x (coset (succ_tran (undir G) x))) $\subseteq \{$Unrel$\}$
  hence $(nP\ x =$ Inter $\longrightarrow nP$ ‘ SNI.undir_reach (lg2g G) $x \subseteq \{$Unrel$\}) =$
      $(nP\ x =$ Inter $\longrightarrow nP$ ‘ set (undir_reach $G\ x$) $\subseteq \{$Unrel$\})$
    using *f2 f3* **by** (*metis undir_reach_def image_set*) }
moreover
{ **assume** $\neg\ nP$ ‘ $(-$ insert x (coset (succ_tran (undir G) x))) $\subseteq \{$Unrel$\}$
  hence $(nP\ x =$ Inter $\longrightarrow nP$ ‘ SNI.undir_reach (lg2g G) $x \subseteq \{$Unrel$\}) =$
      $(nP\ x =$ Inter $\longrightarrow nP$ ‘ set (undir_reach $G\ x$) $\subseteq \{$Unrel$\})$
    using *f2 f3* **by** (*metis undir_reach_def image_set*) }
ultimately show $(nP\ x =$ Inter $\longrightarrow nP$ ‘ SNI.undir_reach (lg2g G) $x \subseteq \{$Unrel$\}) =$
      $(nP\ x =$ Inter $\longrightarrow nP$ ‘ set (undir_reach $G\ x$) $\subseteq \{$Unrel$\})$
  by *metis*

This example is hard to read, but it did allow the user to move on with his formalization. The two nested proof blocks, leading to the **ultimately show** statement, are produced by the proof redirection algorithm. Type annotations appear here and there to ensure that the right types are inferred.

Incidentally, we discovered that a modern version of Sledgehammer yields a much shorter proof, using one of the alternative proof methods described in Section 6.2:

by (*simp add*: *SNI.undir_reach_def succ_tran_correct undir_correct undir_reach_def*)

This is now the proof that appears in the *Archive of Formal Proofs*.

The third and last example from the archive is about regular algebras [29]. The skeleton of the proof, including the induction step, was written manually. The proofs of the base case and of the induction step were ostensibly filled in by Sledgehammer:

lemma *powsum_ub:* $i \leq n \Longrightarrow x^i \leq x_0^n$
proof (*induct n*)
  case 0 **show** *case*
    by (*metis* (*hide_lams*, *mono_tags*) *0.prems eq_iff le_0_eq power_0 powsum_00*)
next
  case (Suc *n*) **show** *case*
  proof −
    { **assume** *aa1*: $\mathsf{Suc}\, n \neq i$
      have *ff1*: $x_0^{\mathsf{Suc}\, n} \leq x_0^{\mathsf{Suc}\, n} \wedge \mathsf{Suc}\, n \neq i$
        using *aa1* **by** *fastforce*
      have *ff2*: $\exists a.\ x_0^n + a \leq x_0^{\mathsf{Suc}\, n} \wedge \mathsf{Suc}\, n \neq i$
        using *ff1 powsum2* **by** *auto*
      have $x^i \leq x_0^{\mathsf{Suc}\, n}$
        using *ff2* **by** (*metis Suc.hyps Suc.prems add_lub le_SucE less_eq_def*) }
    thus $x^i \leq x_0^{\mathsf{Suc}\, n}$
      using *less_eq_def powsum_split_var2* **by** *auto*
  qed
qed

The proof of the induction step features an interesting proof pattern: a nested proof block arising from proof redirection. A mathematician could have written, "We may assume without loss of generality that $n+1 \neq i$." In the formal proof, the non-loss of generality is justified by the very last step, which takes place outside the scope of the assumption $\mathsf{Suc}\, n \neq i$.

8.2 Metatheory of Resolution

Resolution was introduced in 1965 by Robinson [67] as a simple, elegant, and efficient calculus for propositional and first-order logic. Despite the rise of SAT solving, it is at the heart of E, LEO-II, SPASS, Vampire, and many other ATP systems. Its metatheory is elaborated in a chapter by Bachmair and Ganzinger [5].

One of the authors of this article, Blanchette, is involved in an effort to develop infrastructure for formalizing inference systems; together with Dmitriy Traytel, he has formalized parts of Bachmair and Ganzinger's chapter. Possibly due to the omnipresence of multisets, which are not as well supported as lists and sets by *auto*, Sledgehammer turned out to be invaluable. Often, it was possible to simply follow the paper proof and let the ATPs fill in the gaps. One example is the following Sledgehammer-generated fragment, which arose in the proof of compactness:

obtain *mm* :: α *clause set* $\Rightarrow$ α *clause set* $\Rightarrow$ α *clause* **where**
f1: $\bigwedge M\ Ma.\ (M \subseteq Ma \vee mm\ M\ Ma \in M) \wedge (mm\ M\ Ma \notin Ma \vee M \subseteq Ma)$
by (*metis subsetI*)
hence *f2*: $\bigwedge M\ Ma\ Mb.\ mm\ M\ (Ma \cup Mb) \notin Ma \vee M \subseteq Ma \cup Mb$
by (*meson Un_iff*)
hence $D \in \mathsf{saturate}\ (\mathcal{D} \cup \mathcal{E})$
using *f1* **by** (*metis in_sat_ee saturate_mono sup_commute*)
thus $E \in \mathsf{saturate}\ (\mathcal{D} \cup \mathcal{E})$
using *f2 f1* **by** (*meson in_sat_d inference_system.saturated_saturate*
saturate_mono saturatedD step.hyps(1) *subsetCE*)

The proof is difficult to relate to until one pays attention to what the Skolem function *mm* stands for: Given two sets of clauses $\mathcal{C}$ and $\mathcal{D}$, *mm* $\mathcal{C}$ $\mathcal{D}$ returns an element of $\mathcal{C} - \mathcal{D}$ if such an element exists.

The next and last example arose while enriching the theory of multisets:

obtain *j* **where** *j_len*: $j < \mathsf{length}\ xs'$ **and** *nth_j*: $xs' ! j = x$
proof −
assume $\bigwedge j.\ j < \mathsf{length}\ xs' \Longrightarrow xs' ! j = x \Longrightarrow thesis$
moreover have $\bigwedge k\ m\ n.\ (m :: nat) + n < m + k \vee \neg\ n < k$
by *linarith*
moreover have $\bigwedge n\ a\ as.\ n - n < \mathsf{length}\ (\mathsf{Cons}\ a\ as) \vee n < n$
by (*metis Nat.add_diff_inverse diff_add_inverse2 impossible_Cons le_add1*
less_diff_conv not_add_less2)
moreover have $\neg\ \mathsf{length}\ xs' < \mathsf{length}\ xs'$
by *blast*
ultimately show *thesis*
by (*metis* (*no_types*) *Cons.prems Nat.add_diff_inverse diff_add_inverse2*
length_append less_diff_conv list.set_intros(1) *multiset_of_eq_setD*
nth_append_length split_list)
qed

As we have come to expect from machine-generated proofs, some of the steps are odd. It is hard to imagine a human stating $n - n < \mathsf{length}\ (\mathsf{Cons}\ a\ as) \vee n < n$ as an intermediate property. Despite this, Blanchette inserted the proof unchanged in the theory file, only to discover a much simpler alternative, also due to Sledgehammer, one month later:

by (*metis Cons.prems in_set_conv_nth list.set_intros*(1) *multiset_of_eq_setD*)

This example hints at the variety of proofs that are possible for the same problem. Different ATPs find radically different proofs; sometimes the same ATP, invoked on a different day, produces different results. One reason for this is the use of machine learning in the relevance filter [44]. More variation is possible by changing the Isar proof compression factor or setting other Sledgehammer options.

9 Evaluation

Enhancements to Sledgehammer can be evaluated systematically by applying the tool to each goal arising in existing Isabelle theory files and measuring how many goals can be discharged automatically. Since the main motivation behind Isar proof construction is to increase the success rate, we compare the success rate of Sledgehammer without and with Isar proofs for each ATP (excluding Waldmeister, which cannot be run locally, cf. Section 7.4).

	Judgment Day			Arithmetic			Resolution		
	One-line	Isar	Oracle	One-line	Isar	Oracle	One-line	Isar	Oracle
E	613	+10	+ 0	239	+ 3	+ 0	364	+ 4	+ 0
SPASS	667	+ 5	+ 5	249	+ 3	+ 0	385	+ 4	+ 2
Vampire	692	+ 9	+ 6	243	+ 3	+ 0	395	+ 5	+ 0
veriT	535	+ 5	+ 3	237	+14	+ 6	336	+ 1	+ 0
Z3	615	+17	+11	243	+12	+ 3	384	+ 5	+ 3
LEO-II	413	+ 0	+ 5	122	+ 0	+ 0	210	+ 0	+ 5
Satallax	438	+ 4	+ 3	120	+ 0	+ 0	234	+ 0	+ 3

Fig. 6 Number of successful Sledgehammer invocations per ATP with one-line proof reconstruction, with Isar proof reconstruction, and with ATPs trusted as oracles

The benchmarks are partitioned into three suites, for a total of 2461 goals:

- *Judgment Day* (1230 goals) consists of seven theories from the Isabelle distribution and the *Archive of Formal Proofs* that have been used continuously since 2010 for evaluating Sledgehammer [16]. The theories were chosen to be representative of various Isabelle applications.
- *Arithmetic* (622 goals) consists of three theories involving linear and nonlinear arithmetic that were selected to evaluate SMT solvers [12].
- *Resolution* (609 goals) consists of nine theories belonging to the formalization of the metatheory of resolution (Section 8.2).

Be aware that benchmark suites such as Judgment Day keep on evolving together with the proof assistant. Also, the hardware is not always the same from evaluation to evaluation. Hence, success rate values should not be compared uncritically across papers.

The current experiments were carried out on Linux servers equipped with Intel Core2 Duo CPUs at 2.40 GHz. Each prover was given 30 seconds to solve each goal, but the 30-second slot was split into several slices, each corresponding to different problems and options to the prover. The results are summarized in Figure 6.

It is important to bear in mind that the evaluation is not a competition between the provers. Different provers are invoked with different problems and options, and although we have tried to optimize the setup for each, we might have missed an important configuration option. Each number must be seen as a lower bound on the potential of the prover.

When a proof is found, one-line proof reconstruction is attempted, using a portfolio of methods (*metis*, *meson*, *blast*, *simp*, *auto*, *fastforce*, *force*, *linarith*, and *presburger*). Reconstruction is a success if at least one of the method succeeds within 2 seconds. The number of successful one-line proofs is given in the "One-line" column of Figure 6. Unlike in some other Sledgehammer evaluations [12], the *smt* method is not included as a reconstructor, because of the limitations mentioned in Section 7.5. For arithmetic goals, this means that one-line proof reconstruction will often fail for SMT solvers.

If one-line reconstruction fails, Sledgehammer attempts to generate an Isar proof. The goal is considered solved if the Isar proof is successfully generated and replayed. This is reflected in the "Isar" column of Figure 6.

In case both reconstruction approaches fail, the user could in principle trust the external prover as an oracle, since the problem encoding is sound [13].[7] These reconstruction failures are recorded in the "Oracle" column of Figure 6.

[7] In practice, most Isabelle users in this case would work on a manual proofs, often getting some inspiration from the unreconstructable ATP-generated proof.

The data shows that Isar reconstruction makes a serious dent in Sledgehammer reconstruction failures: Most proofs that could not be replayed before by one-line proofs can now be reconstructed as Isar. Out of 7893 ATP successes overall, 104 (1.3%) can be reconstructed only thanks to the new Isar module, and 55 (0.7%) still require trusting the external prover. Overall, textual reconstruction via one-line or multi-line proofs succeeds in 7838 of 7893 cases (99.3%). The remaining reconstruction failures can be explained in various ways, notably:

- The symbols introduced by **obtain** cannot be polymorphic, which is sometimes an issue (cf. Section 5.1).
- Communication with E, but also in some cases with other provers, takes place through type encodings. Although the encodings are globally sound [13], individual inferences can be ill-typed, leading to Isar failures.
- Not all proof rules are supported for all provers. Notably, Z3's model-based quantifier instantiation strategy introduces odd symbols in the arithmetic inferences, and LEO-II's conjecture splitting rule, a logistic rule that cannot be disabled, would require a more general format for representing ATP proofs (or some other trick).
- Sometimes, the culprit is a mere failure to perform by Isabelle's proof methods. This can be an issue for the higher-order ATPs LEO-II and Satallax.

A 1.3% increase of the success rate might not sound like much, but as one of us remarked in a previous paper [14, Section 7]:

> When analyzing enhancements to automatic provers, it is important to remember what difference a modest-looking gain of a few percentage points can make to users. The benchmarks were chosen to be representative of typical Isabelle goals and include many that are either too easy or too hard to effectively evaluate automatic provers. Indeed, some of the most essential tools in Isabelle, such the arithmetic decision procedures, score well below 10% when applied indiscriminately to the entire Judgment Day suite.

In short, every percentage point counts.

10 Conclusion

Sledgehammer employs a variety of techniques to improve the readability and efficiency of the generated Isar proofs. Whenever one-line proof reconstruction fails or times out, users are offered detailed, direct Isar proofs that discharge the goal, sometimes after a small amount of manual tuning. While the output is designed for replaying proofs, it also has a pedagogical value: Unlike Isabelle's automatic tactics, which are black boxes, the proofs delivered by Sledgehammer can be inspected and understood. The direct proofs also form a good basis for manual tuning. Users who are interested in inspecting the proofs can force their generation by passing an option. Related options control preplay and compression.

This work is still in progress. Many aspects could be improved further; we mentioned a few in the previous sections. Our next priority is to identify and rectify any remaining failure cases: Preplaying insulates users from failures, but ideally valid ATP proofs should always lead to valid Isar proofs. We also want to integrate the SMT solver CVC4 [6], which performs extremely well on Isabelle problems [65, Section V] but whose LFSC (Logical Framework with Side Conditions) proofs [74] would need to be parsed and understood.

A possible further step would be to implement proof manipulation algorithms to simplify the proofs further before presenting them to users. For example, users normally prefer

sequential chains of deduction to the spaghetti-like structure of some machine-generated proofs; using appropriate algorithms, it should be possible to minimize the number of jumps or introduce block structure to separate independent subproofs. Similar work has been carried out for human-written proofs [55, 56], but we expect machine proofs to offer more opportunities for refactoring. Automatic discovery of concepts and lemmas would also be useful for larger proofs.

Today, most automatic provers have some proof output, but perhaps due to the low number of consumers this output is typically crude and poorly documented. Reconstruction as done in Sledgehammer reveals these weaknesses. Clearly, more could be done on the ATP side to increase correctness, readability, and interoperability of the generated proofs.

Acknowledgments Tobias Nipkow, Lawrence Paulson, Geoff Sutcliffe, Josef Urban, and Makarius Wenzel encouraged us to pursue this research. Stefan Berghofer and Cezary Kaliszyk shared ideas with us on how to address the problem of inserting type annotations. Konstantin Korovin graciously accepted that we include his unpublished alternative account of proof redirection. ATP developers Christoph Benzmüller, Nikolaj Bjørner, Chad Brown, David Desharbe, Pascal Fontaine, Thomas Hillenbrand, Kryštof Hoder, Leonardo de Moura, Stephan Schulz, Andrei Voronkov, Daniel Wand, and Christoph Weidenbach provided fixes and advice. Cezary Kaliszyk, Lawrence Paulson, Andrei Popescu, Mark Summerfield, and several anonymous reviewers provided helpful feedback on this article and the workshop papers leading to it. Blanchette's research was financed by the Deutsche Forschungsgemeinschaft (DFG) project Hardening the Hammer (grant NI 491/14-1).

References

1. Alama, J.: Escape to Mizar from ATPs. In: P. Fontaine, R. Schmidt, S. Schulz (eds.) PAAR-2012, *EPiC*, vol. 21, pp. 3–11. EasyChair (2013)
2. Armand, M., Faure, G., Grégoire, B., Keller, C., Théry, L., Werner, B.: A modular integration of SAT/SMT solvers to Coq through proof witnesses. In: J.P. Jouannaud, Z. Shao (eds.) CPP 2011, *LNCS*, vol. 7086, pp. 135–150. Springer (2011)
3. Autexier, S., Benzmüller, C., Fiedler, A., Horacek, H., Vo, Q.B.: Assertion-level proof representation with under-specification. Electr. Notes Theor. Comput. Sci. **93**, 5–23 (2004)
4. Azmy, N., Weidenbach, C.: Computing tiny clause normal forms. In: M.P. Bonacina (ed.) CADE-24, *LNCS*, vol. 7898, pp. 109–125. Springer (2013)
5. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: A. Robinson, A. Voronkov (eds.) Handbook of Automated Reasoning, vol. I, pp. 19–99. Elsevier (2001)
6. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: G. Gopalakrishnan, S. Qadeer (eds.) CAV 2011, *LNCS*, vol. 6806, pp. 171–177. Springer (2011)
7. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB standard—Version 2.0. In: A. Gupta, D. Kroening (eds.) SMT 2010 (2010)
8. Benzmüller, C., Paulson, L.C., Theiss, F., Fietzke, A.: LEO-II—A cooperative automatic theorem prover for higher-order logic. In: A. Armando, P. Baumgartner, G. Dowek (eds.) IJCAR 2008, *LNCS*, vol. 5195, pp. 162–170. Springer (2008)
9. Besson, F., Fontaine, P., Théry, L.: A flexible proof format for SMT: A proposal. In: P. Fontaine, A. Stump (eds.) PxTP 2011, pp. 15–26 (2011)
10. Blanchette, J.C.: Automatic proofs and refutations for higher-order logic. Ph.D. thesis, Dept. of Informatics, Technische Universität München (2012)
11. Blanchette, J.C.: Redirecting proofs by contradiction. In: J.C. Blanchette, J. Urban (eds.) PxTP 2013, *EPiC*, vol. 14, pp. 11–26. EasyChair (2013)
12. Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending Sledgehammer with SMT solvers. J. Autom. Reasoning **51**(1), 109–128 (2013)
13. Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. In: N. Piterman, S. Smolka (eds.) TACAS 2013, *LNCS*, vol. 7795, pp. 493–507. Springer (2013)
14. Blanchette, J.C., Popescu, A., Wand, D., Weidenbach, C.: More SPASS with Isabelle—Superposition with hard sorts and configurable simplification. In: L. Beringer, A. Felty (eds.) ITP 2012, *LNCS*, vol. 7406, pp. 345–360. Springer (2012)

15. Böhme, S.: Proving theorems of higher-order logic with SMT solvers. Ph.D. thesis, Dept. of Informatics, Technische Universität München (2012)
16. Böhme, S., Nipkow, T.: Sledgehammer: Judgement Day. In: J. Giesl, R. Hähnle (eds.) IJCAR 2010, *LNCS*, vol. 6173, pp. 107–121. Springer (2010)
17. Böhme, S., Weber, T.: Fast LCF-style proof reconstruction for Z3. In: M. Kaufmann, L. Paulson (eds.) ITP 2010, *LNCS*, vol. 6172, pp. 179–194. Springer (2010)
18. Bouton, T., de Oliveira, D.C.B., Déharbe, D., Fontaine, P.: veriT: An open, trustable and efficient SMT-solver. In: R.A. Schmidt (ed.) CADE-22, *LNCS*, vol. 5663, pp. 151–156. Springer (2009)
19. Bradley, A.R., Manna, Z.: Property-directed incremental invariant generation. Formal Asp. Comput. **20**, 379–405 (2008)
20. Brown, C.E.: Satallax: An automatic higher-order prover. In: B. Gramlich, D. Miller, U. Sattler (eds.) IJCAR 2012, *LNCS*, vol. 7364, pp. 111–117. Springer (2012)
21. Brown, C.E.: Using Satallax to generate proof terms for conjectures in Coq. Presentation at AIPA 2012 (2012)
22. Caminati, M.: Re: [isabelle] sledgehammer, smt and metis. `https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2014-November/msg00128.html` (2014)
23. Chaieb, A., Nipkow, T.: Verifying and reflecting quantifier elimination for Presburger arithmetic. In: G. Sutcliffe, A. Voronkov (eds.) LPAR 2005, *LNCS*, vol. 3835, pp. 367–380. Springer (2005)
24. Church, A.: A formulation of the simple theory of types. J. Symb. Log. **5**(2), 56–68 (1940)
25. Dahn, B.I.: Robbins algebras are Boolean: A revision of McCune's computer-generated solution of Robbins problem. J. Algebra **208**(2), 526–532 (1998)
26. Davis, M.: Obvious logical inferences. In: P.J. Hayes (ed.) IJCAI '81, pp. 530–531. William Kaufmann (1981)
27. Diekmann, C.: Network security policy verification. In: G. Klein, T. Nipkow, L. Paulson (eds.) Archive of Formal Proofs. `http://afp.sf.net/entries/Network_Security_Policy_Verification.shtml` (2014)
28. Fleury, M.: Translation of proofs provided by external provers. Internship report, Technische Universität München, `http://perso.eleves.ens-rennes.fr/~mfleur01/documents/Fleury_internship2014.pdf` (2014)
29. Foster, S., Struth, G.: Regular algebras. In: G. Klein, T. Nipkow, L. Paulson (eds.) Archive of Formal Proofs. `http://afp.sf.net/entries/Network_Security_Policy_Verification.shtml` (2014)
30. Gentzen, G.: Untersuchungen über das logische Schließen. Math. Z. **39**, 176–210 (1935)
31. Gordon, M.J.C., Melham, T.F. (eds.): Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. Cambridge University Press (1993)
32. Herbrand, J.: Thèses présentées à la faculté des sciences de paris pour obtenir le grade de docteur ès sciences mathématiques. Ph.D. thesis, Science Faculty, Université de Paris (1930)
33. Hillenbrand, T., Buch, A., Vogt, R., Löchner, B.: WALDMEISTER—High-performance equational deduction. J. Autom. Reasoning **18**(2), 265–270 (1997)
34. Hoder, K., Kovacs, L., Voronkov, A.: Vampire usage and demo. Presentation at the Vampire tutorial at CADE-23, `http://www.complang.tuwien.ac.at/lkovacs/Cade23_Tutorial_Slides/Session2_Slides.pdf` (2011)
35. Huang, X.: Translating machine-generated resolution proofs into ND-proofs at the assertion level. In: N.Y. Foo, R. Goebel (eds.) PRICAI '96, *LNCS*, vol. 1114, pp. 399–410. Springer (1996)
36. Hurd, J.: First-order proof tactics in higher-order logic theorem provers. In: M. Archer, B. Di Vito, C. Muñoz (eds.) Design and Application of Strategies/Tactics in Higher Order Logics, no. CP-2003-212448 in NASA Technical Reports, pp. 56–68 (2003)
37. Jaśkowski, S.: On the rules of suppositions in formal logic. Studia Logica **1**, 5–32 (1934)
38. Kaliszyk, C., Urban, J.: PRocH: Proof reconstruction for HOL Light. In: M.P. Bonacina (ed.) CADE-24, *LNCS*, vol. 7898, pp. 267–273. Springer (2013)
39. Kaliszyk, C., Urban, J.: Learning-assisted automated reasoning with Flyspeck. J. Autom. Reasoning **53**(2), 173–213 (2014)
40. Karp, R.M.: Reducibility among combinatorial problems. In: R.E. Miller, J.W. Thatcher (eds.) Complexity of Computer Computations, IBM Research Symposia Series, pp. 85–103. Plenum Press (1972)
41. Klein, G., Nipkow, T., Paulson, L. (eds.): Archive of Formal Proofs. `http://afp.sf.net/`
42. Knuth, D.E., Larrabee, T.L., Roberts, P.M.: Mathematical Writing. Mathematical Association of America (1989)
43. Korovin, K.: Private communication (2013)
44. Kühlwein, D., Blanchette, J.C., Kaliszyk, C., Urban, J.: MaSh: Machine learning for Sledgehammer. In: S. Blazy, C. Paulin-Mohring, D. Pichardie (eds.) ITP 2013, *LNCS*, vol. 7998, pp. 35–50. Springer (2013)
45. Loveland, D.W.: Automated Theorem Proving: A Logical Basis. North-Holland (1978)

46. Matuszewski, R., Rudnicki, P.: Mizar: The first 30 years. Mechanized Mathematics and Its Applications **4**(1), 3–24 (2005)
47. Meier, A.: TRAMP: Transformation of machine-found proofs into natural deduction proofs at the assertion level (system description). In: D. McAllester (ed.) CADE-17, *LNCS*, vol. 1831, pp. 460–464. Springer (2000)
48. Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. J. Autom. Reasoning **40**(1), 35–60 (2008)
49. Meng, J., Paulson, L.C.: Lightweight relevance filtering for machine-generated resolution problems. J. Applied Logic **7**(1), 41–57 (2009)
50. Miller, D., Felty, A.: An integration of resolution and natural deduction theorem proving. In: AAAI-86, vol. I: Science, pp. 198–202. Morgan Kaufmann (1986)
51. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: C.R. Ramakrishnan, J. Rehof (eds.) TACAS 2008, *LNCS*, vol. 4963, pp. 337–340. Springer (2008)
52. Nipkow, T.: Equational reasoning in Isabelle. Sci. Comput. Program **12**(2), 123–149 (1989)
53. Nipkow, T., Klein, G.: Concrete Semantics—With Isabelle/HOL. Springer (2014)
54. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, *LNCS*, vol. 2283. Springer (2002)
55. Pąk, K.: The methods of improving and reorganizing natural deduction proofs. In: MathUI10 (2010)
56. Pąk, K.: Methods of lemma extraction in natural deduction proofs. J. Autom. Reasoning **50**(2), 217–228 (2013)
57. Pąk, K.: Improving legibility of natural deduction proofs is not trivial. Log. Meth. Comput. Sci. **10**(3) (2014)
58. Paulson, L.C.: Isabelle: A Generic Theorem Prover, *LNCS*, vol. 828. Springer (1994)
59. Paulson, L.C.: Strategic principles in the design of Isabelle. In: CADE-15 Workshop on Strategies in Automated Deduction, pp. 11–17 (1998)
60. Paulson, L.C.: A generic tableau prover and its integration with Isabelle. J. Univ. Comp. Sci. **5**, 73–87 (1999)
61. Paulson, L.C.: Three years of experience with Sledgehammer, a practical link between automated and interactive theorem provers. In: B. Konev, R. Schmidt, S. Schulz (eds.) PAAR-2010, pp. 1–10 (2010)
62. Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: G. Sutcliffe, S. Schulz, E. Ternovska (eds.) IWIL-2010, *EPiC*, vol. 2, pp. 1–11. EasyChair (2012)
63. Paulson, L.C., Susanto, K.W.: Source-level proof reconstruction for interactive theorem proving. In: K. Schneider, J. Brandt (eds.) TPHOLs 2007, *LNCS*, vol. 4732, pp. 232–245. Springer (2007)
64. Pfenning, F.: Analytic and non-analytic proofs. In: R.E. Shostak (ed.) CADE-7, *LNCS*, vol. 170, pp. 393–413. Springer (1984)
65. Reynolds, A., Tinelli, C., de Moura, L.: Finding conflicting instances of quantified formulas in SMT. In: K. Claessen, V. Kuncak (eds.) FMCAD 2014, pp. 195–202. FMCAD Inc. (2014)
66. Riazanov, A., Voronkov, A.: The design and implementation of Vampire. AI Comm. **15**(2-3), 91–110 (2002)
67. Robinson, A.J.: A machine-oriented logic based on the resolution principle. J. ACM **12**(1), 23–41 (1965)
68. Rudnicki, P.: Obvious inferences. J. Autom. Reasoning **3**(4), 383–393 (1987)
69. Rudnicki, P., Urban, J.: Escape to ATP for Mizar. In: P. Fontaine, A. Stump (eds.) PxTP 2011, pp. 46–59 (2011)
70. Schulz, S.: System description: E 1.8. In: K. McMillan, A. Middeldorp, A. Voronkov (eds.) LPAR-19, *LNCS*, vol. 8312, pp. 735–743. Springer (2013)
71. Smolka, S.J.: Robust, semi-intelligible Isabelle proofs from ATP proofs. B.Sc. thesis, Dept. of Informatics, Technische Universität München (2013)
72. Smolka, S.J., Blanchette, J.C.: Robust, semi-intelligible Isabelle proofs from ATP proofs. In: J.C. Blanchette, J. Urban (eds.) PxTP 2013, *EPiC*, vol. 14, pp. 117–132. EasyChair (2013)
73. Steckermeier, A.: Extending Isabelle/HOL with the equality prover Waldmeister. B.Sc. thesis, Dept. of Informatics, Technische Universität München (2014)
74. Stump, A., Oe, D., Reynolds, A., Hadarean, L., Tinelli, C.: SMT proof checking using a logical framework. J. Formal Meth. Sys. Design **42**(1), 91–118 (2013)
75. Sultana, N., Benzmüller, C.: Understanding LEO-II's proofs. In: K. Korovin, S. Schulz, E. Ternovska (eds.) IWIL 2012, *EPiC*, vol. 22, pp. 33–52. EasyChair (2013)
76. Sutcliffe, G.: System description: SystemOnTPTP. In: D. McAllester (ed.) CADE-17, *LNCS*, vol. 1831, pp. 406–410. Springer (2000)
77. Sutcliffe, G.: The TPTP problem library and associated infrastructure—The FOF and CNF parts, v3.5.0. J. Autom. Reasoning **43**(4), 337–362 (2009)

78. Sutcliffe, G.: The CADE-24 automated theorem proving system competition—CASC-24. AI Commun. **27**(4), 405–416 (2014)
79. Sutcliffe, G., Zimmer, J., Schulz, S.: TSTP data-exchange formats for automated theorem proving tools. In: W. Zhang, V. Sorge (eds.) Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems, *Frontiers in Artificial Intelligence and Applications*, vol. 112, pp. 201–215. IOS Press (2004)
80. Thiemann, R.: Computing N-th roots using the Babylonian method. In: G. Klein, T. Nipkow, L. Paulson (eds.) Archive of Formal Proofs. `http://afp.sf.net/entries/Sqrt_Babylonian.shtml` (2013)
81. Urban, J., Sutcliffe, G., Trac, S., Puzis, Y.: Combining Mizar and TPTP semantic presentation and verification tools. In: A. Grabowski, A. Naumowicz (eds.) Computer Reconstruction of the Body of Mathematics, *Studies in Logic, Grammar and Rhetoric*, vol. 18(31), pp. 121–136. University of Białystok (2009)
82. Wampler-Doty, M.: A complete proof of the Robbins conjecture. In: G. Klein, T. Nipkow, L. Paulson (eds.) Archive of Formal Proofs. `http://afp.sf.net/entries/Robbins-Conjecture.shtml` (2010)
83. Weber, T.: Sat-based finite model generation for higher-order logic. Ph.D. thesis, Dept. of Informatics, Technische Universität München (2008)
84. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS version 3.5. In: R.A. Schmidt (ed.) CADE-22, *LNCS*, vol. 5663, pp. 140–145. Springer (2009)
85. Wenzel, M.: Type classes and overloading in higher-order logic. In: E.L. Gunter, A. Felty (eds.) TPHOLs 1997, *LNCS*, vol. 1275, pp. 307–322. Springer (1997)
86. Wenzel, M.: Isabelle/Isar—A generic framework for human-readable proof documents. In: R. Matuszewski, A. Zalewska (eds.) From Insight to Proof—Festschrift in Honour of Andrzej Trybulec, *Studies in Logic, Grammar and Rhetoric*, vol. 10(23). University of Białystok (2007)
87. Wickerson, J., Dodds, M., Parkinson, M.J.: Ribbon proofs for separation logic. In: M. Felleisen, P. Gardner (eds.) ESOP 2013, *LNCS*, vol. 7792, pp. 189–208. Springer (2013)
88. Zimmer, J., Meier, A., Sutcliffe, G., Zhan, Y.: Integrated proof transformation services. In: C. Benzmüller, W. Windsteiger (eds.) IJCAR WS 7 (2004)