FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor Thesis in Computer Science

Synthesis of Robust, Semi-Intelligble Isar Proofs from ATP Proofs

Steffen Juilf Smolka

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor Thesis in Computer Science

Synthesis of Robust, Semi-Intelligble Isar Proofs from ATP Proofs

Synthese robuster, semi-verständlicher Isar-Beweise aus ATP-Beweisen

Author:	Steffen Juilf Smolka
Supervisor:	Prof. Tobias Nipkow, Ph.D.
Advisor:	Dr. Jasmin Christian Blanchette
Submission Date:	August 1, 2013

Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I assure the single handed composition of this bachelor thesis only supported by declared resources.

München, den 20. Oktober 2013 Steffen Juilf Smolka

Acknowledgments

I want to express my deep gratitude to my advisor Jasmin Blanchette, who could not have been a better mentor to me. He motivated and supported me in writing my first paper and accompanied me to the first conference I ever attended. Throughout the time I worked with him, he was always there for me when I had questions or needed advice.

I thank Tobias Nipkow for drawing my interest to theoretical computer science. His exceptional lectures have left a deep impression on me and convinced me that I am studying the right subject whenever I was in doubt.

Cezary Kaliszyk, Andrei Popescu, Mark Summerfield, and three anonymous reviewers provided helpful feedback on a paper describing most contributions of this thesis. The feedback has also contributed to the text at hand.

Finally, I want to thank my family for their continuous support and for cheering me up in the lunch breaks during the final days on my thesis.

Abstract

Sledgehammer is a tool that integrates external automatic theorem provers (ATPs) into the Isabelle/HOL proof assistant, allowing users to delegate proof search to the computer. To guard against bugs, ATP proofs must be reconstructed in Isabelle. Reconstructing complex proofs involves translating them to detailed Isabelle proof texts, using suitable proof methods to justify the inference steps. This thesis addresses the main issues that have prevented previous implementations of this approach from working in practice: Sledgehammer now reconstructs skolemization inferences correctly, and it equips proof texts with the right amount of type annotations to ensure formulas are parsed correctly without overwhelming them with types. In addition, Sledgehammer now employs algorithms to test and optimize its output, resulting in simpler, faster, and more robust proofs.

Contents

1 Introduction

Sledgehammer [20] is a proof tool that connects the Isabelle/HOL proof assistant [17] with external automatic theorem provers (ATPs), including first-order resolution provers and SMT solvers. Given an interactive proof goal, it heuristically selects hundreds of facts (lemmas, definitions, and axioms) from Isabelle's vast libraries, translates them to first-order logic (FOL), and invokes the external provers. Although Sledgehammer may be trusted as an oracle,[1] most users are satisfied only once the proof has been reduced to Isabelle primitives and been verified by the kernel.

When Sledgehammer was originally conceived, the plan was to have it deliver detailed proofs in Isabelle's Isar language [30], a textual, human-readable format inspired by Mizar [14]. Paulson and Susanto [21] designed a prototype that performs inference-by-inference translation of ATP proofs into Isar proofs and justifies each Isar inference using *metis*, a proof method based on Hurd's Metis resolution prover [11]. This idea was abandoned for several reasons: The resulting proofs by contradiction were unpalatable, so that users were disinclined to insert them in their theory text; they were often syntactically incorrect due to technical issues; and a single *metis* call with the short list of needed lemmas, a *metis one-liner*, usually sufficed to re-find the proof.

Proof reconstruction with *metis* one-liners means that the proof must be re-found each time the Isabelle theory text 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 dedicated procedures for theory reasoning. The Z3-based *smt* proof method [5] is a powerful alternative to *metis*, but it depends on the availability of Z3 on the user's machine for proof replay, which hinders its acceptance among users. Moreover, due to its incomplete quantifier handling, it can fail to re-find a proof generated by a resolution prover.

[1]For many years, Sledgehammer employed type-unsound encodings by default [15], making it unsuitable as an oracle. Newer versions use optimized type-sound encodings [4].

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. The first issue is that the Isar proof, like the underlying ATP proof, is by contradiction. A paper by Blanchette describes an algorithm that turns such proofs around [3]. This thesis describes further enhancements that increase the intelligibility, efficiency, and robustness of the output and that are implemented in Sledgehammer's proof translation pipeline (Chapter 3).

- *Skolemization*: Sledgehammer communicates with ATPs in full FOL as opposed to quantifier-free clause normal form (CNF). Skolemization is performed by the external ATPs, but it must be reconstructed in Isar (Chapter 4).
- *Type annotations*: Isabelle can generate strings from formulas, 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 minimal, complete set of type annotations (Chapter 5).
- *Proof preplay*: 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 presenting them to users. If several proofs are available, users can choose the fastest one and insert it in their theory text (Chapter 6).
- *Proof optimization*: The generated proofs can be manipulated in several ways to obtain faster, simpler, and more robust proofs. Sledgehammer now compresses straightforward chains of deduction into single Isar inferences, replaces *metis* with faster methods were possible, and eliminates needless dependencies and proof steps (Chapter 7).

These enhancements are demonstrated in a case study in Chapter 8.

Most contributions of this thesis were published in a paper [24] at the PxTP workshop 2013. The material from [24] has been included in this thesis with the permission of the coauthor. The abstract, Chapters 1 - 6 and Chapter 9 were adapted from the paper with only little changes; Section 7.1 shares some text with the paper.

2 Isabelle/HOL

The Isabelle/HOL proof assistant is based on polymorphic higher-order logic (HOL) [8] extended with axiomatic type classes [29]. The types and terms of HOL are that of the simply-typed λ-calculus [6] 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 are also called type constants (e.g., *nat*). The binary type constructor $\alpha \to \beta$ is interpreted as the (total) function space from α to β. Type variables can carry type class constraints, which are essentially predicates on the types.

Terms are either constants (e.g., map), 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 and quantifiers are predefined ($\neg$, $\wedge$, $\vee$, $\longrightarrow$, $\forall$, $\exists$). Constants can be polymorphic; for example, $\mathsf{map} :: (\alpha \to \beta) \to \alpha\ \mathit{list} \to \beta\ \mathit{list}$ applies a unary 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 $\mathsf{Trueprop} :: \mathit{bool} \to \mathit{prop}$, which is normally not printed. Some foundational properties can only be expressed in the metalogic, but they play no role in Sledgehammer. Readers unfamiliar with this two-layer design may safely pretend there is just one logic.

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 $2 + 2 = 4$. Without type annotations, the formula is parsed as $(2 :: \alpha) + (2 :: \alpha) = (4 :: \alpha)$, where α belongs to the *numeric* type class, which defines basic numeric operators and syntax but imposes no semantics on the "numbers." An annotation is necessary to make the formula provable—e.g., $(2 :: \mathit{int}) + 2 = 4$. A single annotation is sufficient because of the constraints arising from the most general types of the involved operators: $\mathrm{op} + :: \alpha \to \alpha \to \alpha$ and $\mathrm{op} = :: \alpha \to \alpha \to \mathit{bool}$.

3 The Translation Pipeline

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

3.1 ATP Proof

Paulson and Susanto had the foresight to choose TSTP (Thousands of Solutions for Theorem Provers) [26] as input format for their prototype. Among the automatic provers they wanted to integrate with Isabelle, only E [23] supported the format at the time. Nowadays, most provers feature some support for TSTP.

TSTP specifies the basic syntax for representing proofs as a directed acyclic graph of inferences. A single parser can be used to integrate all provers that can generate the syntax. However, the format does not mandate any proof system; hence, interfacing a new ATP usually requires some work, especially for processing inferences that introduce new symbols (e.g., skolemization). Isar proof construction is currently supported for the resolution provers E and Vampire [22] and the unit-equality prover Waldmeister [10].

SPASS generates proofs in its custom DFG format only (even though it can parse TPTP FOF [25]). Fortunately, DFG is based on similar concepts and can be represented using the same data structure as TSTP in memory, so it is also supported to a large extent.

3.2 Proof by Contradiction

The ATP proof is translated into an Isabelle proof by contradiction. This step preserves the graph structure of the proof, but the nodes are labeled by HOL formulas.

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

This translation corresponds largely to the work by Paulson and Susanto. We refer to their paper [21] for details. In particular, they describe how HOL terms, types, and type classes are reconstructed from their encoded FOL form. Their code had to be adapted to cope with the variety of type encodings supported by newer versions of Sledgehammer [4], but nonetheless their description fairly accurately describes the current state of affairs.

3.3 Direct Proof

The proof redirection algorithm [3] takes a proof by contradiction as the input and produces a direct proof. The latter can be regarded as a fragment of Isar proofs. The abstract syntax of proofs (π) and inferences (ι) is given by the production rules

$$\begin{aligned}\pi &::= (\texttt{fix}\ x^*)^*\ (\texttt{assume}\ l\!:\ \phi)^*\ \iota^* \\ \iota &::= \texttt{prove}\ q^*\ l\!:\ \phi\ \ l^*\ \pi^*\ m \\ &\quad\ \mid \texttt{obtain}\ q^*\ x^*\ \texttt{where}\ l\!:\ \phi\ \ l^*\ \pi^*\ m\end{aligned}$$

where x ranges over HOL variables (which may be of function types), ϕ over HOL formulas, l over Isar fact labels (names), and q over Isar qualifiers (`then` and `show`); m denotes a proof method and is initially set to *metis*. 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 `prove`. Its variant `obtain` proves the existence of HOL variables for which a property holds; the variables are added to the context.

Once the direct proof has been constructed, it is optimized and preplayed. Finally, qualifiers are introduced: `then` indicates that the previous fact is needed to prove the current fact, whereas `show` is required for the last inference in the top-level block. 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.

3.4 Isar Proof

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 correctly by

Isabelle. This is especially an issue for formulas, where type annotations might be needed.

3.5 Example

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

> `datatype` *state* = On | Off
>
> `fun` flip :: *state* $\rightarrow$ *state* `where`
> flip On = Off |
> flip Off = On
>
> `lemma` flip $x \neq x$

Invoking Sledgehammer launches a collection of ATPs (typically, E, SPASS, Vampire, and Z3). The conjecture is easy, so they rapidly return. Vampire delivers the following proof, presented in a slightly abbreviated TSTP-like format:

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	$\forall X_3\ (\neg\, \mathsf{state}(X_3) = \mathsf{on} \longrightarrow \mathsf{state}(X_3) = \mathsf{off})$	*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	$\forall X_0\ (\neg\, \mathsf{state}(X_0) = \mathsf{on} \longrightarrow \mathsf{state}(X_0) = \mathsf{off})$	57	rectify
893	plain	$\forall X_0\ (\mathsf{state}(X_0) \neq \mathsf{on} \longrightarrow \mathsf{state}(X_0) = \mathsf{off})$	892	flatten
1596	plain	$\forall X_0\ (\mathsf{state}(X_0) = \mathsf{on} \vee \mathsf{state}(X_0) = \mathsf{off})$	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	$\forall X_0\ (\mathsf{state}(X_0) = \mathsf{off} \vee \mathsf{state}(X_0) = \mathsf{on})$	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

The formulas used from the original problem are listed first. Each line gives a formula number, a role, and a FOL formula. Any problem formula that can be used to prove the conjecture is an axiom for the automatic prover, 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.

If Sledgehammer's *isar_proofs* option is enabled, 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	$\mathsf{flip\ On} = \mathsf{On} \vee \mathsf{Off} = s$	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	$\mathsf{flip\ Off} = \mathsf{Off}$	775, 3405
3409	False	3407, *flip.simps*(2), *state.distinct*(1)

Linear inference chains are drastically compressed, and the lemmas

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

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

`prove` [] 3407: "flip Off $\neq$ Off" [*flip.simps*(2), *state.distinct*(1)] [] *metis*
`prove` [] 3405: "flip $s \neq s \vee$ Off $\neq s$ [3407] [] *metis*
`prove` [] 3404: "flip $s \neq s \vee$ Off $\neq s \wedge$ Off $\neq$ On [3405, *state.distinct*(1)] [] *metis*
`prove` [] 3402: "flip $s \neq s \vee$ flip On $\neq$ On $\wedge$ Off $\neq s$ [3404, *flip.simps*(1)] [] *metis*
`prove` [show] 775: "flip $s \neq s$" [3402, *state.exhaust*] [] *metis*

Compression and cleanup simplify the proof further:[1]

```
prove [] $\epsilon$: "flip Off $\neq$ Off" [*flip.simps*(2), *state.distinct*(1)] []
prove [then, show] $\epsilon$: "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 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
```

[1]We omit the other optimizations from Chapter 7 from this example since applying them would result in a trivial "by simp"-proof

4 Skolemization

The typical architecture of modern first-order provers combines a clausifier and a CNF-based reasoning core. It is the clausifier's duty to skolemize the problem and move the nonskolemizable quantifiers to the front of the formulas, where they can be omitted. Sledgehammer historically performed clausification itself, using a naive exponential application of distributive laws. This was changed a few years ago to use the ATPs' native clausifiers, which generate a polynomial number of clauses [2, §6.6.1]. Skolemization transforms a formula into an equisatisfiable but not equivalent formula, introducing new symbols in the process. When translating ATP proofs to Isar proofs, this extension of the signature must be made explicit. Simply invoking *metis*, as done in Paulson and Susanto's prototype, will not work to replay skolemization inferences.

Conjecture and axioms are treated differently because of their different polarities. By convention, the axioms are positive and the conjecture is negative.[1] 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 [9].

E and Vampire explicitly record skolemization inferences in their proof, and fortunately they do it in the same way. On the other hand, SPASS's proofs are expressed in terms of the clausified problem, and Sledgehammer currently cannot reconstruct them corretly if they rely on skolemization.

4.1 The Positive Case

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

[1] 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.

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 are 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:

`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 only prove the formula 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 Vampire uses 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. An easy workaround would be to relaunch Sledgehammer with a monomorphizing type encoding [4, §3] to obtain a more suitable ATP proof. A more challenging alternative would involve detecting which monomorphic instances of the problematic lemmas are needed and re-engineer the proof accordingly.

4.2 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 *un-herbrandization*. In Isar, the `fix` command achieves a similar effect, as in the example below:

```
lemma ⋀x. R x
proof –
  fix x
  ⟨core of the argument⟩
  show R x ...
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 ∀v. ∃w. ∀x. ∃y. ∀z. Q v w x y z
proof –
  { fix v x z
    ⟨core of the argument⟩
    have ∃w y. Q v w (x w) y (z w y) by (metis ...) }
  thus ∀v. ∃w. ∀x. ∃y. ∀z. 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 arbitrarily. In the data structure that stores direct Isar-like proofs, the proof would be represented as

```
prove [] 41: "$\forall v.\ \exists w.\ \forall x.\ \exists y.\ \forall z.\ \mathsf{Q}\ v\ w\ x\ y\ z$" []
  [fix [$v, x, z$]
  $\langle$core of the argument$\rangle$
  prove [] 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.

4.3 Alternative Approaches

Given a HOL problem, the *metis* method clausifies it and translates it to FOL, invokes the first-order prover Metis, and replays the Metis inferences using suitable HOL tactics. Skolemization is simulated using Hilbert's choice operator ε [21]; 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 eliminate the dependency on Hilbert's choice [2, §6.6.7], only requiring the weak axiom of choice to move the existentials to the front. Whichever approach is used, Sledgehammer's textual proof construction exploits *metis*'s machinery (and the reduction of HOL to FOL) instead of replicating it textually.

Other ATP-based proof methods or tactics must also cope with skolemization. Isabelle's *smt* method [5] relies on Hilbert's choice, whereas HOL(y)Hammer's proof reconstructor [12] depends only on the weak axiom of choice. Another option is to trust the ATP's clausifier, leaving it to the user to inspect the generated clausification axioms; this is the approach implemented for reconstructing proofs found by Miz$\mathbb{AR}$ [1]. Finally, a radical approach, designed for textual proof reconstruction in Coq, is to replace Skolem function symbols by predicate symbols and adjust the proof accordingly, a process known as deskolemization [7].

5 Type Annotations

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

$$((\mathrm{op} = :: \textit{int list} \rightarrow \textit{int list} \rightarrow \textit{bool})\ (xs :: \textit{int list}) :: \textit{int list} \rightarrow \textit{bool})\ (ys :: \textit{int list}) :: \textit{bool}$$

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 correctly.

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 $\mathsf{ann}_\tau :: \tau \rightarrow \tau$ 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.

5.1 The Algorithm

Given a well-typed formula ϕ to annotate, the algorithm starts by replacing all the types in ϕ by the special placeholder _ (Isabelle's "dummy" type); this corresponds to printing the formular without annotations. It then emulates parsing the printed term back by infering the most general types for ϕ, resulting in a formula $\phi^\star$ in which the placeholders are instantiated. Next, it 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:

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

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. Each site t^τ is also associated with the set of type variables α_i it covers.

The goal is to compute a minimal set of sites that completely covers all type variables. Computing the globabl minimum amounts to solving the weighted set cover problem, which is NP-hard [13]. 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. It calculates a local minimum.

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 classical greedy approach could yield a too large set: For the term $\mathsf{h}^{\mathit{nat} \to \mathit{real}}\ \mathsf{c}^{\mathit{nat}}$ generalized to $\mathsf{h}^{\alpha \to \beta}\ \mathsf{c}^{\alpha}$, 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.

Since the names of fresh type variables are semantically irrelevant, ρ can be simplified in a preprocessing step. If the substitution contains fresh type variables that only occur once as a subtype in $\tau_1, \ldots, \tau_m$, they are replaced by _; type variables α_i in ρ's domain that map to _ can be removed from the subsitution. Thus, the formula $\mathsf{length}\ ([] :: \alpha\ \mathit{list}) = 0$ is printed as $\mathsf{length}\ [] = 0$ without undesirable gain of generality.

5.2 Example

Let $\mathsf{fst} :: \alpha \times \beta \to \alpha$ and $\mathsf{snd} :: \alpha \times \beta \to \beta$ be polymorphic constants that extract the components of a pair. Suppose the formula ϕ to annotate is

$$\forall x\ y.\ \exists p.\ \mathsf{fst}\ p = x \wedge \mathsf{snd}\ p = y$$

with x :: *nat* and y :: *real*. Inside Isabelle, the formula's subterms carry type information (except the bound variables):

$$\begin{aligned}&\mathsf{All}^{(nat\to bool)\to bool}\ (\lambda x^{nat}.\ \mathsf{All}^{(real\to bool)\to bool}\ (\lambda y^{real}.\ \mathsf{Ex}^{(nat\times real\to bool)\to bool}\ (\lambda p^{nat\times real}.\\ &\quad(\mathrm{op}\ \vee)^{bool\to bool\to bool}\ ((\mathrm{op}\ =)^{nat\to nat\to bool}\ (\mathsf{fst}^{nat\times real\to nat}\ p)\ x)\\ &\qquad((\mathrm{op}\ =)^{real\to real\to bool}\ (\mathsf{snd}^{nat\times real\to real}\ p)\ y))))\end{aligned}$$

Replacing the types with _ yields the formula

$$\mathsf{All}^{\_}\ (\lambda x^{\_}.\ \mathsf{All}^{\_}\ (\lambda y^{\_}.\ \mathsf{Ex}^{\_}\ (\lambda p^{\_}.\ (\mathrm{op}\ \vee)^{\_}\ ((\mathrm{op}\ =)^{\_}\ (\mathsf{fst}^{\_}\ p)\ x)\ ((\mathrm{op}\ =)^{\_}\ (\mathsf{snd}^{\_}\ p)\ y))))$$

from which type inference produces the formula $\phi^\star$:

$$\begin{aligned}&\mathsf{All}^{(\alpha\to bool)\to bool}\ (\lambda x^{\alpha}.\ \mathsf{All}^{(\beta\to bool)\to bool}\ (\lambda y^{\beta}.\ \mathsf{Ex}^{(\alpha\times\beta\to bool)\to bool}\ (\lambda p^{\alpha\times\beta}.\\ &\quad(\mathrm{op}\ \vee)^{bool\to bool\to bool}\ ((\mathrm{op}\ =)^{\alpha\to\alpha\to bool}\ (\mathsf{fst}^{\alpha\times\beta\to\alpha}\ p)\ x)\\ &\qquad((\mathrm{op}\ =)^{\beta\to\beta\to bool}\ (\mathsf{snd}^{\alpha\times\beta\to\beta}\ p)\ y))))\end{aligned}$$

The substitution entailed by ϕ and $\phi^\star$ is $\rho = \{\alpha \mapsto nat,\ \beta \mapsto real\}$. There are several possible ways to annotate the formula so as to cover both α and β, including

$$\begin{gathered}\forall x\ y.\ \exists p.\ \mathsf{fst}\ (p :: nat \times real) = x \wedge \mathsf{snd}\ p = y\\ \forall x\ y.\ \exists p.\ (\mathsf{fst}\ p :: nat) = x \wedge (\mathsf{snd}\ p :: real) = y\\ \forall x\ y.\ \exists p.\ \mathsf{fst}\ p = (x :: nat) \wedge \mathsf{snd}\ p = (y :: real)\end{gathered}$$

The third formula is the one produced by the reverse greedy algorithm. It is arguably the most aesthetically pleasing of the three, because both the annotated terms and the types are atomic.

Incidentally, the annotations could have been omitted in this example because the property holds generally for arbitrary types α and β, but this cannot always be relied upon. Moreover, omitting the type annotations is not completely harmless because of the poor interaction between skolemization and polymorphism.

6 Proof Preplay

Isar proofs generated from ATP proofs sometimes fail. We already mentioned that skolemization is not supported for polymorphic variables (cf. Chapter 4). The TSTP or DFG 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* fails to discharge a proof obligation within a reasonable amount of time.

As the experience with Sledgehammer-generated *metis* one-liners has shown, it is advantageous to try out, or *preplay*, the proofs before presenting them to users [2, §6.6.6]. 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 one-liners and structured Isar proofs. Based on the preplay information, they can make an informed decision while being spared the tedium of having to test the proofs manually.

Until recently, users had to enable Isar proof reconstruction to benefit from it. The preplay mechanism makes a new hybrid mode possible, in which an Isar proof is generated whenever the one-liner times out or fails. Even when the Isar proof is not entirely correct, it is easier to repair than the corresponding monolithic one-liner—for example, by adding some parentheses to guide the parsing or by replacing a failing or too slow *metis* call by a hand-written proof.

The following example, based on an Isabelle formalization of the Robbins conjecture [28], 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).
>
> Structured 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 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" on the first line. (Indeed, *metis* 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 $\Phi \vdash \phi$, it measures the time *metis* takes to deduce ϕ from Φ and caches 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 "may fail" in the banner.

As we will describe in Section 7.2, Sledgehammer now supports several other proof methods besides *metis*. The preplay framework has been adapted to work with these alternative methods as well. With this infrastructure in place, we are able to optimize proofs in several ways. The next chapter will describe them.

7 Proof Optimization

The proof preplay infrastructure established in Chapter 6 provides us with a precise measure[1] based on which we can optimize proofs: the time it takes to run a proof. This allows us to take a pragmatic approach: We can simply try out different manipulations and apply them only if they make the proof faster.

In contrast to speed, intelligibility is in the eye of the beholder and cannot simply be measured, so it is not our main focus. Nonetheless, we do use simple heuristics that aim at improving intelligibility whenever a second criterion aside from speed is useful in deciding what manipulation to apply next. Moreover, users may alter optimization parameters or disable certain optimizations altogether to obtain proofs that better fit their preferences.

7.1 Proof Compression

Isar proofs generated by Sledgehammer can consist of dozens or hundreds of steps. It is usually beneficial to compress them by eliminating intermediate steps. Compressed proofs can be faster to recheck; for example, when the proof related to Robbins conjecture from Chapter 6 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-liner is nothing but an Isar proof compressed to the extreme, and it can be both very slow and very cryptic.

In the simplest case, our compression procedure considers an intermediate step and the unique successor it is referenced by. It performs a merger if the resulting inference is fast enough—no more than 20% slower than the original inferences taken together. This 20% tolerance factor embodies a trade-off between processing speed and conciseness. Given the inferences $\Phi_1 \vdash \phi_1$ and $\{\phi_1\} \uplus \Phi_2 \vdash \phi_2$, where ϕ_1 is not referenced elsewhere in the proof, the merged inference is $\Phi_1 \cup \Phi_2 \vdash \phi_2$.

[1]For technical reasons, timings are subject to fluctutations. For our purposes however, they are a close enough approximation of "reality".

In the general case, the intermediate step may be referenced by n successors. It can be eliminated by merging it into all of these successors. As n increases, however, it becomes less likely that the elimination will speed up the proof. Moreover, a high number of references suggests that an inference is important; removing it may spoil intelligibility. Therefore, we introduce an upper bound for n which defaults to 2.

Subproofs are always implicitly referenced by the single inference they belong to. If they consist of just one proof step, their elimination can be achieved in a similar manner. For example, consider the inference

$\texttt{prove}\ qs\ l_2\text{:}\ \phi_2\ \Phi_2\ [(\texttt{prove}\ []\ l_1\text{:}\ \phi_1\ \Phi_1\ []\ \textit{metis}), \ldots]\ \textit{metis}.$

Merging the one-step subproof into the inference, we obtain

$\texttt{prove}\ qs\ l_2\text{:}\ \phi_2\ (\Phi_1 \cup \Phi_2)\ [\ldots]\ \textit{metis}$

The compression algorithm performs the following steps:

1. Apply the algorithm recursively to all subproofs.

2. If there exist one-step subproofs, try eliminating them as described above.

3. While there are intermediate steps in the top level proof that have not been considered: Pick such a steps and try eliminating it as described above.

We take a depth-first approach since subproofs arguably impede intelligibility more than intermediate steps on the top level. In step 3, our implementation prefers inferences with a low number of references (because these are less likely to be important) and with long formulas (because these clutter the proof more). The process is guided by *metis*'s performance. Users who want to understand the proof may find that too many details have been optimized away. For them, there is a Sledgehammer option that controls the compression factor, which bounds the number of mergers before the algorithm stops in relation to the length of the uncompressed proof.

7.2 Beyond Metis: Alternative Proof Methods

Earlier versions of Sledgehammer have relied exclusively on *metis* to justify the steps of generated Isar proofs. This choice was due to the fact that *metis* is complete for first-order logic. Consequently, every inference performed by an external (first-order logic) ATP should in principle be reconstructable by *metis*. In contrast, other automatic proof methods integrated into Isabelle such as *simp*, *auto*, or *fastforce* are incomplete, so there is always the possibility that they may fail. However, they often succeed in practice, and in that case they are usually much faster than *metis*. With the proof preplay framework (Chapter 6) in place, we can simply try them and find out.

Our implementation is inspired by the *try0* tool in Isabelle [16]. For each inference, we run the methods *simp*, *auto*, *fastforce*, *force*, *arith*, and *blast* in parallel. As soon as one of the methods succeeds, we replace *metis* with that method in our Isar-like data structure, and interrupt the execution of all other methods. We also make sure the methods do not run longer than *metis* did; in that case, *metis* is already the best choice.

As we will demonstrate in Chapter 8, replacing *metis* with whichever proof method is fastest can lead to considerable speedups. In addition, this optimization may even fix broken Isar proofs. If for some reason *metis* is unable to reconstruct an inference or takes unacceptably long, there is a chance that one of the other methods can do better and solve the problem. Furthermore, the optimization can usually be applied without investing much additional time, since the involved methods succeed or fail quickly in most cases.

7.3 Fact Elimination

When trying to discharge a proof goal, *metis* uses nothing but logic and the facts (i.e. axioms, definitions, and lemmas) it is given. For instance, it cannot prove lemmas like $\mathsf{hd}\ [x] = x$ or $(1 :: \mathit{nat}) + 1 = 2$ without being supplied with adequate facts; in fact, *metis* knows nothing about lists or numbers. In contrast, many of the alternative proof methods mentioned in the previous section can prove the given lemmas without further hints. They are preconfigured to draw upon the necessary knowledge from the library. Therefore, we can typically reduce the set of facts used in a proof step after replacing metis with an alternative proof method.

Just as in the previous two sections, the optimization relies on the proof preplay infrastructure (Chapter 6). For each proof step and each fact it uses, we test if the associated proof method is still successful after removing the fact. If preplay fails or is not successful within the time limit defined by the previous successful preplay attempt, we consider the fact essential and keep it; otherwise, we drop it. In the process, we may eliminate referenes to facts established in previous proof steps. If a step is no longer referenced, we remove it.

The optimization often lead to faster, more concise proofs. Just as the previous optimization, it may even fix broken proofs: If a proof step fails, there is a chance it becomes dispensable through fact elimination and can be removed.

8 Case Study

In this chapter, we illustrate some of the contributions of the thesis through a "real world example" that was posted on the Isabelle mainling list by a novice [27]. Given two real numbers from the open intervall between zero and one, we want to prove that the difference of the sum of the numbers and the product of the numbers is greater than zero. Here is the lemma in Isar syntax:

```
lemma
  fixes $a$ :: real and $b$ :: real
  assumes a0: $0 < a$ and a1: $a < 1$ and b0: $0 < b$ and b1: $b < 1$
  shows $a + b - a * b > 0$
```

When invoked without arguments, Sledgehammer produces a *metis* one-liner:

```
by (metis a0 a1 add_less_cancel_left b0 comm_monoid_add_class.add.right_neutral
    comm_monoid_mult_class.mult.left_neutral
    comm_semiring_1_class.normalizing_semiring_rules(24)
    diff_add_cancel pos_add_strict real_mult_less_iff1)
```

Metis can master the proof obligation sucessfully, but it took 798 ms to do so on our test system. This may be more time than users want to invest on such a simple lemma since the proof has to be rerun every time the theory is reloaded, which may result in unpleasant delays when working with large theories containing many lemmas. Furthermore, the *metis* one-liner is no help to users who want to understand why the lemma is true.

Running Sledgehammer with the *isar_proofs* option, but disabling the optimizations described in Chapter 7, we obtain a step-by-step Isar proof:

```
proof -
  have $\bigwedge x_2\, x_1.\ (x_2 :: real) + (x_1 - x_2) = x_1$
    by (metis comm_semiring_1_class.normalizing_semiring_rules(24) diff_add_cancel)
  hence f1: $\bigwedge x_1\, x_2\, x_3.\ (x_1 :: real) < x_2 - x_3 \ \vee\ \neg\, x_3 + x_1 < x_2$
    by (metis add_less_cancel_left)
  have f2: $\bigwedge x_1\, x_2.\ (x_1 :: \mathit{real}) * x_2 < x_2 \ \vee\ \neg\, 0 < x_2 \ \vee\ \neg\, x_1 < 1$
    by (metis comm_monoid_mult_class.mult.left_neutral real_mult_less_iff1)
  have $0 < b \wedge a < 1$
    by (metis a1 b0)
  hence $a * b < b$
    using f2 by metis
  hence $0 < a \wedge a * b < b$
    by (metis a0)
  hence $a * b < a + b$
    by (metis pos_add_strict)
  hence $a * b + 0 < a + b$
    by (metis comm_monoid_add_class.add.right_neutral)
  thus $0 < a + b - a * b$
    using f1 by metis
qed
```

Thanks to the type annotations inserted in the first three steps (Chapter 5), Isabelle parses the formulas correctly and the proof succeeds. The intermediate steps simplify *metis*'s task considerly: It is now able to verify the lemma in 74 ms. Although the proof is arguably tedious from a human perspective, it is much more insightful than the one-line proof.

By enabling proof compression with an appropriate conpression factor (Section 7.1), we can reduce the amount of detail to a more natural level:

```
  proof –
    have $a * b < b$
      by (metis a1 b0 comm_monoid_mult_class.mult.left_neutral real_mult_less_iff1)
    hence $a * b < a + b$
      by (metis a0 pos_add_strict)
    thus $0 < a + b - a * b$
      by (metis comm_monoid_add_class.add.right_neutral add_less_cancel_left)
          comm_semiring_1_class.normalizing_semiring_rules(24) diff_add_cancel
  qed
```

The resulting proof is very close to what a proof written by a human might look like, and it is even faster than the last one: It can now be checked in only 25 ms. Yet we can go one step further and try replacing *metis* with alternative proof methods (Section 7.2) and try eliminating unnecessary facts (Section 7.3), bringing us to the final result:

```
proof -
  have $a * b < b$
    using $a1$ $b0$ by $simp$
  hence $a * b < a + b$
    using $a0$ by $simp$
  thus $0 < a + b - a * b$
    by $simp$
qed
```

All proof steps are now justified by *simp*. The optimization algorithm was able to eliminate all references to external facts, producing an even cleaner proof. Running it takes only 5 ms.

9 Conclusion

The latest version of 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. Users who are interested in inspecting the proofs can force their generation by passing an option. Related options control preplay and optimization.

Sledgehammer-generated proofs cannot be expected to always turn out as nice as in the case study in Chapter 8. For example, some proofs exhibit a spaghetti-like structure. Users normally prefer sequential chains of reasoning. In such cases, it should be possible to minimize the number of jumps or introduce block structure to separate independent subproofs using appropriate algorithms. Similar work has been carried out for human-written proofs [18, 19].

While their is room for improvement regarding intelligibility, we expect the new features to be very helpful in enhancing efficiency: Slow *metis* one-liners should be replaceable with much faster step-by-step Isar proofs.

Bibliography

[1] J. Alama. Escape to Mizar from ATPs. In P. Fontaine, R. Schmidt, and S. Schulz, editors, *PAAR-2012*, pages 3–11, 2012.

[2] J. C. Blanchette. *Automatic Proofs and Refutations for Higher-Order Logic*. Ph.D. thesis, Dept. of Informatics, Technische Universität München, 2012.

[3] J. C. Blanchette. Redirecting proofs by contradiction. In J. C. Blanchette and J. Urban, editors, *PxTP 2013*, 2013.

[4] J. C. Blanchette, S. Böhme, A. Popescu, and N. Smallbone. Encoding monomorphic and polymorphic types. In N. Piterman and S. Smolka, editors, *TACAS 2013*, volume 7795 of *LNCS*, pages 493–507. Springer, 2013.

[5] S. Böhme and T. Weber. Fast LCF-style proof reconstruction for Z3. In M. Kaufmann and L. Paulson, editors, *ITP 2010*, volume 6172 of *LNCS*, pages 179–194. Springer, 2010.

[6] A. Church. A formulation of the simple theory of types. *J. Symb. Log.*, 5(2):56–68, 1940.

[7] H. de Nivelle. Translation of resolution proofs into short first-order proofs without choice axioms. *Inf. Comput.*, 199(1-2):24–54, 2005.

[8] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[9] J. Herbrand. *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.

[10] T. Hillenbrand, A. Buch, R. Vogt, and B. Löchner. WALDMEISTER—High-performance equational deduction. *J. Autom. Reasoning*, 18(2):265–270, 1997.

[11] J. Hurd. First-order proof tactics in higher-order logic theorem provers. In M. Archer, B. Di Vito, and C. Muñoz, editors, *Design and Application of Strategies/Tactics in Higher Order Logics*, number CP-2003-212448 in NASA Technical Reports, pages 56–68, 2003.

[12] C. Kaliszyk and J. Urban. PRocH: Proof reconstruction for HOL Light. In M. P. Bonacina, editor, *CADE-24*, volume 7898 of *LNAI*, pages 267–273. Springer, 2013.

[13] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, IBM Research Symposia Series, pages 85–103. Plenum Press, 1972.

[14] R. Matuszewski and P. Rudnicki. Mizar: The first 30 years. *Mechanized Mathematics and Its Applications*, 4(1):3–24, 2005.

[15] J. Meng and L. C. Paulson. Translating higher-order clauses to first-order clauses. *J. Autom. Reasoning*, 40(1):35–60, 2008.

[16] T. Nipkow. Programming and Proving in Isabelle/HOL. `http://isabelle.in.tum.de/documentation.html`, 2013.

[17] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[18] K. Pąk. The methods of improving and reorganizing natural deduction proofs. In *MathUI10*, 2010.

[19] K. Pąk. Methods of lemma extraction in natural deduction proofs. *J. Autom. Reasoning*, 50(2):217–228, 2013.

[20] L. C. Paulson and J. C. Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In G. Sutcliffe, E. Ternovska, and S. Schulz, editors, *IWIL-2010*, 2010.

[21] L. C. Paulson and K. W. Susanto. Source-level proof reconstruction for interactive theorem proving. In K. Schneider and J. Brandt, editors, *TPHOLs 2007*, volume 4732 of *LNCS*, pages 232–245. Springer, 2007.

[22] A. Riazanov and A. Voronkov. The design and implementation of Vampire. *AI Comm.*, 15(2-3):91–110, 2002.

[23] S. Schulz. System description: E 0.81. In D. Basin and M. Rusinowitch, editors, *IJCAR 2004*, volume 3097 of *LNAI*, pages 223–228. Springer, 2004.

[24] S. J. Smolka and J. C. Blanchette. Robust, semi-intelligible Isabelle proofs from ATP proofs. In J. C. Blanchette and J. Urban, editors, *PxTP 2013*, 2013.

[25] G. Sutcliffe. The TPTP problem library and associated infrastructure—The FOF and CNF parts, v3.5.0. *J. Autom. Reasoning*, 43(4):337–362, 2009.

[26] G. Sutcliffe, J. Zimmer, and S. Schulz. TSTP data-exchange formats for automated theorem proving tools. In W. Zhang and V. Sorge, editors, *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems*, volume 112 of *Frontiers in Artificial Intelligence and Applications*, pages 201–215. IOS Press, 2004.

[27] M. Vu. prove simple inequality. `https://lists.cam.ac.uk/mailman/htdig/cl-isabelle-users/2013-July/msg00018.html`, 2013.

[28] M. Wampler-Doty. A complete proof of the Robbins conjecture. In G. Klein, T. Nipkow, and L. Paulson, editors, *The Archive of Formal Proofs*. `http://afp.sf.net/entries/Robbins-Conjecture.shtml`, 2010.

[29] M. Wenzel. Type classes and overloading in higher-order logic. In E. L. Gunter and A. Felty, editors, *TPHOLs 1997*, volume 1275 of *LNCS*, pages 307–322. Springer, 1997.

[30] M. Wenzel. Isabelle/Isar—A generic framework for human-readable proof documents. In R. Matuszewski and A. Zalewska, editors, *From Insight to Proof—Festschrift in Honour of Andrzej Trybulec*, volume 10(23) of *Studies in Logic, Grammar and Rhetoric*. University of Białystok, 2007.