Noname manuscript No.
(will be inserted by the editor)

Extending Sledgehammer with SMT Solvers

Jasmin Christian Blanchette · Sascha Böhme · Lawrence C. Paulson

the date of receipt and acceptance should be inserted later

Abstract Sledgehammer is a component of Isabelle/HOL that employs resolution-based first-order automatic theorem provers (ATPs) to discharge goals arising in interactive proofs. It heuristically selects relevant facts and, if an ATP is successful, produces a snippet that replays the proof in Isabelle. We extended Sledgehammer to invoke satisfiability modulo theories (SMT) solvers as well, exploiting its relevance filter and parallel architecture. The ATPs and SMT solvers nicely complement each other, and Isabelle users are now pleasantly surprised by SMT proofs for problems beyond the ATPs' reach.

1 Introduction

It is widely recognized that combining automated reasoning systems of different types can deliver huge rewards. There have been several attempts to combine interactive theorem provers (which are arguably better at formal modeling than at proving theorems) with a variety of automatic theorem provers (ATPs). One of the most successful such combinations is Sledgehammer [38, 45, 46], which interfaces Isabelle/HOL [43] with resolution provers for classical first-order logic. Sledgehammer is both effective, solving approximately one third of nontrivial goals arising in interactive proofs [17], and easy to use, since it is invoked with a single command. For these reasons, it has rapidly become indispensable to Isabelle users and has transformed the way Isabelle is taught to beginners [45].

Research partially supported by the Deutsche Forschungsgemeinschaft [grant numbers Ni 491/11-2 and Ni 491/14-1]. Sledgehammer was originally supported by the U.K.'s Engineering and Physical Sciences Research Council [grant number GR/S57198/01].

J. C. Blanchette · S. Böhme
Institut für Informatik, Technische Universität München, Munich, Germany
E-mail: {blanchette,boehmes}@in.tum.de

L. C. Paulson
Computer Laboratory, University of Cambridge, U.K.
E-mail: lp15@cam.ac.uk

Given an Isabelle/HOL conjecture, Sledgehammer heuristically selects a few hundred relevant lemmas from Isabelle's libraries, translates them to first-order logic along with the conjecture, and sends the resulting problem to four resolution provers (Section 2). The provers run in parallel, either locally or remotely via SystemOnTPTP [56]. If a proof is found, Sledgehammer delivers it to the user in the form of an explicit Isabelle command that can be inserted into the proof script. This command consists of a call to Isabelle's built-in prover *metis* [33, 46], supplying the facts that were used in the original proof. Typically, *metis* finds an equivalent proof; this final phase we call *proof reconstruction*.

First-order ATPs are powerful and general, but they can usefully be complemented by other technologies. Satisfiability modulo theories (SMT) solvers combine a satisfiability solver with decision procedures for first-order theories, such as equality, integer and real arithmetic, and bit-vector reasoning. SMT solvers are particularly well suited to discharging large proof obligations arising from program verification. Although SMT solvers are automatic theorem provers in a general sense, we find it convenient to reserve the abbreviation ATP for provers rooted in the tradition of TPTP (Thousands of Problems for Theorem Provers) [57].

There have also been several attempts to combine interactive theorem provers with SMT solvers, either as trusted oracles or with proof reconstruction. In previous work, we integrated the solvers CVC3 2.*x* [5], Yices 1.0.*x* [24] and Z3 2.*x*/3.*x* [41] with Isabelle as oracles, and implemented step-by-step proof reconstruction for Z3 [18]. The resulting *smt* proof method takes a list of problem-specific facts that are passed to the SMT solver along with the conjecture (Section 3).

While a motivated user can go a long way with the *smt* method [16], the need to specify facts and to guess that a conjecture could be solved by SMT makes it hard to use. As evidence of this, the Isabelle formalizations accepted in the *Archive of Formal Proofs* [34] in 2010 and early 2011, after *smt* was introduced in Isabelle, contain 7958 calls to the simplifier, 928 calls to the internal tableau prover, 219 calls to *metis* (virtually all generated using Sledgehammer), but not even one call to *smt*.

Can Isabelle users benefit from SMT solvers? We assumed so and took the obvious next step, namely to have Sledgehammer run SMT solvers in parallel with ATPs, reusing the existing relevance filter and parallel architecture (Section 4). This idea seems promising for a number of reasons:

- SMT solvers natively support linear integer and real arithmetic.
- Even for nonarithmetic problems, ATPs and SMT solvers have complementary strengths. The former handle quantifiers more elegantly, whereas the latter excel on large, mostly ground problems.
- Users should not have to guess whether a problem is more appropriate for ATPs or SMT solvers. Both classes of prover should be run concurrently.

The integration required extensive refactoring of Sledgehammer, a delicate piece of engineering developed by eight people in Cambridge and Munich over several years.

The Sledgehammer–SMT integration appears to be the first of its kind, and we had no clear idea of how successful it would be as we started the implementation work. Would the SMT solvers only prove conjectures already provable using the ATPs, or would they find original proofs? Would the decision procedures be pertinent to

interactive goals? Would the SMT solvers scale in the face of hundreds of quantified facts translated en masse, as opposed to carefully crafted axiomatizations?

The first results with Z3 were disappointing: Given a few hundred facts, the solver often ran out of memory or crashed. It took some tweaking and help from the Z3 developers to obtain decent results. We eventually added support for CVC3 and Yices, two solvers that, like Z3, support quantifiers via (automatically inferred) triggers—patterns that guide quantifier instantiations. Our evaluation on a large benchmark suite shows that SMT solvers (Z3 particularly) add considerable power to Sledgehammer (Section 6).

We presented an earlier version of this article at CADE-23 in Wrocław, Poland [11]. New research makes it possible to answer questions left open by the CADE paper. Other notable additions are a second example (Section 5.2) and further coverage of related work (Section 7).

2 Sledgehammer and Metis

Sledgehammer is Isabelle/HOL's subsystem for harnessing the power of other automated reasoning systems. Its processing steps include relevance filtering, translation to classical first-order logic, parallel ATP invocation, proof reconstruction, and proof minimization.

Relevance Filtering. Sledgehammer employs a simple relevance filter that examines the given conjecture and then extracts a few hundred lemmas from Isabelle's enormous libraries. The relevance test is based on how many symbols are shared between the conjecture and each candidate lemma [39], much in the style of SInE [30]. Introducing this filtering step greatly improves Sledgehammer's success rate, because most ATPs perform poorly in the presence of thousands of axioms. Although ATPs have improved a great deal since Meng and Paulson's original experiments [39], the relevance filter is still beneficial [13, § 7].

Translation into Untyped First-Order Logic. Isabelle/HOL's formalism, polymorphic higher-order logic [2, 62], is much richer than the ATPs' unsorted first-order logic. Sledgehammer's translation uses various techniques to translate HOL formulas to first-order logic [38]: Curried functions are passed varying numbers of arguments by means of an explicit, deeply embedded application operator, and λ-abstractions are rewritten to Turner combinators or transformed into explicit functions using λ-lifting [31]. These techniques help find proofs where no or little higher-order reasoning is needed.

Until recently, the translation of types was unsound: It provided enough information to enforce correct type class reasoning but not to prevent ill-typed instantiations. Soundness is not crucial because the proofs are rechecked by Isabelle's inference kernel. The current implementation safely erases most of the type information by inferring type monotonicity, resulting in a sound and efficient encoding [12]. Newer versions of SPASS [61] and Vampire [49] support a many-sorted logic, which would seem to make it more appropriate to encode HOL typing information than unsorted

first-order logic, but they do not support polymorphism. The solution is to monomorphize the formulas [12, § 3; 13]: Polymorphic formulas are iteratively instantiated with relevant ground instances of their polymorphic constants.

Parallel ATP Invocation. For a number of years, Isabelle has emphasized parallelism to exploit modern multi-core architectures [63]. Sledgehammer invokes several ATPs in parallel, with great success: Running E [54], SPASS, and Vampire in parallel for five seconds solves as many problems as running a single theorem prover for two minutes [17, § 8]. Recent versions of Sledgehammer also invoke E-SInE [30], a wrapper around E that is designed to cope with large axiom bases, and several other provers are also supported [10, § 6.6.4]. The automatic provers are executed in the background so that users can keep working on a proof, although many users prefer to take this opportunity to rest.

Proof Reconstruction. In keeping with the LCF philosophy [28], Isabelle theorems can only be generated within a small inference kernel. It is possible to bypass this safety mechanism if some external tool is to be trusted as an oracle, but all oracle inferences are tracked. Sledgehammer reconstructs proofs by running Metis, a first-order resolution prover written in ML [33]. Although Metis was designed to be interfaced with an LCF-style prover (HOL4), integrating it with Isabelle's inference kernel required significant effort [46]. The resulting *metis* proof method is given the typically short list of facts referenced in the proof found by the external ATP.[1] The *metis* call is delivered to the user in source form, so that it can then be replayed without running the external provers again. Given only a handful of facts, *metis* often succeeds almost instantly. Since *metis* has to re-find the proof, the external provers are essentially used as very precise relevance filters.

Proof Minimization. The final stage, attempting proof reconstruction using *metis*, fails about 5% of the time because *metis* takes too long. ATPs frequently use many more facts than are necessary. Sledgehammer's minimization tool takes a set of facts returned by an ATP and repeatedly calls that ATP with subsets of the facts to find a minimal set. Depending on the number of initial facts, it relies on either of these two algorithms:

- The naive linear algorithm attempts to remove one fact at a time. This can require as many ATP invocations as there are facts in the initial set.
- The binary algorithm recursively bisects the facts [21, § 4.3]. It performs best when a small fraction of the facts are actually required [17, § 7].

Example. In the Isabelle proof below, taken from a formalization of the Robbins conjecture [59], four of the five subproofs are discharged by a *metis* call generated automatically by Sledgehammer using an ATP:

[1] To avoid confusion between the Isabelle proof method and the underlying first-order prover, we consistently write *metis* for the former and Metis for the latter.

lemma $-(y \sqcup k \otimes (x \sqcup -(x \sqcup -y))) = -y$
proof –
  let $z = -(x \sqcup -y)$ **and** $ky = y \sqcup k \otimes (x \sqcup z)$
  have $-(x \sqcup -ky) = z$ **by** (*simp add*: *copyp0*)
  hence $-(-ky \sqcup -(-y \sqcup z)) = z$ **by** (*metis assms sup_comm*)
  also have $-(z \sqcup -ky) = x$ **by** (*metis assms copyp0 sup_comm*)
  hence $z = -(-y \sqcup -(-ky \sqcup z))$ **by** (*metis sup_comm*)
  finally show $-(y \sqcup k \otimes (x \sqcup -(x \sqcup -y))) = -y$ **by** (*metis eq_intro*)
qed

Briefly, **let** introduces term abbreviations; **have**, **hence**, and **also hence** announce intermediate facts; **finally show** announces the conclusion; and **by** discharges a proof obligation by invoking a method customized by arguments (typically, references to already proved lemmas).

The example is typical of the way Isabelle users employ the tool: If they understand the problem well enough to propose some intermediate properties and perform the necessary inductions, all they need to do is state a progression of properties in small enough steps and let Sledgehammer or an automatic tactic prove each one.

3 The *smt* Proof Method

SMT solvers are available in Isabelle through the *smt* proof method [15, 18]. It translates the conjecture and any user-supplied facts to the SMT solvers' many-sorted first-order logic, invokes a solver, and (depending on the solver) either trusts the result or attempts to reconstruct the proof in Isabelle.

Translation into Many-Sorted First-Order Logic. The translation maps HOL's arithmetic operators to the corresponding SMT-LIB 1.2 [47] concepts.[2] The translation is otherwise similar to that performed when communicating with many-sorted ATPs: Partial applications are translated using an explicit application operator, λ-abstractions are lifted, and polymorphic types are monomorphized and mapped to SMT-LIB sorts [14, § 2].

Proof Reconstruction. CVC3 and Z3 provide independently checkable proofs of unsatisfiability. The *smt* method reconstructs Z3's proofs and supports CVC3 and Yices as oracles. Reconstruction relies extensively on standard Isabelle proof methods such as the simplifier, the classical reasoner, and the arithmetic decision procedures. The theories of equality with uninterpreted functions and linear integer and real arithmetic are supported. Certificates make it possible to store Z3 proofs alongside Isabelle formalizations, allowing SMT proof replay without Z3; only if the formalizations change must the certificates be regenerated. Using oracles requires trusting both the solvers and the *smt* method's translation, so it is generally frowned upon.

[2] Support for SMT-LIB 2.0 [4] is future work.

Example. The periodic integer recurrence relation $x_{i+2} = |x_{i+1}| - x_i$ has period 9. This property can be proved in Isabelle using the *smt* method as follows [14, § 1.2]:

lemma $x_3 = |x_2| - x_1 \wedge x_4 = |x_3| - x_2 \wedge x_5 = |x_4| - x_3 \wedge x_6 = |x_5| - x_4 \wedge$
$x_7 = |x_6| - x_5 \wedge x_8 = |x_7| - x_6 \wedge x_9 = |x_8| - x_7 \wedge x_{10} = |x_9| - x_8 \wedge$
$x_{11} = |x_{10}| - x_9 \Longrightarrow x_1 = x_{10} \wedge x_2 = (x_{11} :: int)$
by *smt*

SMT solvers prove the formula almost instantly, and proof reconstruction (if enabled) takes a few seconds. In contrast, Isabelle's arithmetic decision procedure requires several minutes to establish the same result. This example does not require any problem-specific facts, but these would have been supplied as arguments in the *smt* call just like for *metis* in Section 2.

4 Extending Sledgehammer with SMT

Extending Sledgehammer with SMT solvers was to a large extent a matter of connecting existing components: Sledgehammer's relevance filter and minimizer with the *smt* method's translation and proof reconstruction. Figure 1 depicts the resulting architecture, omitting proof reconstruction and minimization.

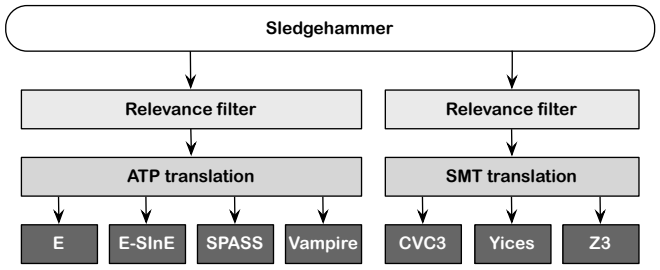

Figure 1 Sledgehammer's extended architecture

Two instances of the relevance filter run in parallel threads, to account for different sets of built-in constructs. The relevant facts and the conjecture are translated to the ATP or SMT version of first-order logic, and the resulting problems are passed to the provers.

4.1 SMT Solver Invocation

In our first experiments, we simply invoked Z3 as an oracle with the monomorphized relevant facts, using the same translation as for the *smt* proof method. The results were disappointing. Several factors were to blame. The translation to SMT-LIB format took many seconds. It took us some time to get the bugs out of our translation code; syntax errors in generated problems often caused Z3 to give up immediately. Z3 often ran out of memory after a few seconds or, worse, crashed. Latent issues both in our

translation and in Z3 were magnified by the number of facts involved. Our previous experience with the *smt* method had involved only a handful of carefully chosen facts.

The bottleneck in the translation was monomorphization. Iterative expansion of a few hundred HOL formulas yielded thousands of monomorphic instances. We reduced the maximum number of iterations from 10 to 3, to great effect. This choice, like many of the others reported below, was arrived at empirically, after extensive experimentation using our "Judgment Day" test harness [17].

The syntax errors were typically caused by confusion between formulas and terms or the use of a partially applied built-in symbol (both of which are legal in HOL). These were bugs in the *smt* proof method; we gradually eradicated them.

We reported the Z3 issues to its developers, who released an improved version. The crashes were caused by a bug in Z3's proof generation facility, which is disabled by default and hence not well tested. To handle the frequent out-of-memory conditions, we modified Sledgehammer to retry aborted solver calls with half the facts. This simple change was enough to increase the success rate dramatically.

But even in the absence of errors, we found it advantageous to divide the time available into slices, each given half the time and half the facts as the previous slice, rather than letting SMT solvers run undisturbed for the full time period. For example, given a time limit of 30 seconds, Z3 is first invoked with 350 facts for (up to) 15 seconds, then with 175 facts for 7.5 seconds, and so on. This smoothly generalizes the out-of-memory handling described above and increases the success rate further.

4.2 Proof Reconstruction

In case of success, Sledgehammer extracts the facts used in the SMT proof—the unsatisfiable core—and generates an *smt* proof method call with these facts supplied as arguments. The *smt* method invokes Z3 to re-find the proof and replays it step by step. The Z3 proof can also be stored alongside the Isabelle formalization as a certificate to avoid invoking the solver each time the proof is rechecked. Minimization can be done as for ATP proofs to reduce the number of facts.

To increase the success rate and reduce the dependency on external solvers or certificates, Sledgehammer first tries Metis internally for a few seconds. If Metis succeeds, Sledgehammer returns a *metis* call to the user rather than an *smt* call. (Recall that Sledgehammer always returns proofs in Isabelle source form.) Metis will of course fail if the proof requires theories other than equality. Conversely, Sledgehammer also tries *smt* for proofs found by ATPs when Metis takes too long.

One of the less academically rewarding aspects of integrating third-party tools is the effort spent on solving mundane issues. Obtaining an unsatisfiable core from the SMT solvers turned out to be surprisingly difficult:

1. CVC3 returns a full proof, but somehow the proof refers to all facts, whether they are actually needed or not, and there is no easy way to find out which facts are actually needed. We rely on Sledgehammer's proof minimizer and its binary algorithm to reduce the facts used to a reasonable number.

2. Yices can output a minimal core, but for technical reasons only when its native input syntax is used rather than SMT-LIB 1.2. We tried using off-the-shelf file format converters to translate SMT-LIB 1.2 to 2 then to Yices, but this repeatedly crashed. In the end, we settled for the same solution as for CVC3.
3. For Z3, we could reuse our existing proof parser, which we need to reconstruct proofs. The proof format is fairly stable, although new releases often include their share of minor incompatibilities.

The problem of interpreting the solvers' output is widely acknowledged in the SMT community. Efforts are under way to define a standard proof format for SMT solvers to complement SMT-LIB [8, 19].

4.3 Relevance Filtering

The relevance filter gives more precise results if it ignores HOL symbols that are translated to built-in constructs (logical and interpreted nonlogical symbols). For ATPs, this concerns equality, connectives, and quantifiers. SMT solvers support a much larger set of built-in constructs, notably arithmetic constants and operators. A fact such as $1+1=2$ on integers might be considered relevant for the ATPs but should always be left out of SMT problems, since it involves only built-in symbols—indeed, it is a tautology of linear arithmetic.

In the original Sledgehammer architecture, the available lemmas were rewritten to clause normal form using a naive application of distributive laws before the relevance filter was invoked [39]. To avoid clausifying thousands of lemmas on each invocation, the clauses were kept in a cache. This design was technically incompatible with the (cache-unaware) *smt* method, and it was already unsatisfactory for ATPs, which include custom polynomial-time clausifiers [44]. We rewrote the relevance filter so that it operates on arbitrary HOL formulas, trying to simulate the old behavior. To mimic the penalty associated with Skolem functions in the clause-based code, we keep track of polarities and detect quantifiers that give rise to Skolem functions.

Observing that some provers cope better with large fact bases than others, we optimized the maximum number of relevant facts to include in a problem independently for each prover (from a library of about 10 000 facts). The maxima we obtained are 400 for CVC3, 150 for Yices, and 350 for Z3, which are comparable to the figures we had previously obtained for the ATPs [39].

4.4 Redistribution and Distribution

Our goal with Sledgehammer is to help as many Isabelle users as possible. Third-party provers should ideally be bundled with Isabelle and ready to be used without requiring configuration. On the ATP side, Isabelle includes E and SPASS executables for Linux, Mac OS X, and Windows; users can download Vampire (whose license forbids redistribution), but most simply run Vampire remotely on SystemOnTPTP.

CVC3 allows redistribution and use by noncommercial and commercial users alike. Yices's license forbids redistribution, but noncommercial users can download

it from its official web site. Z3 requires a license for commercial uses but redistribution with Isabelle is allowed. Isabelle now includes CVC3 and Z3 executables for all major platforms, with Z3 disabled unless users confirm their noncommercial intentions via a configuration option.

For convenience, we additionally set up a server in Munich in the style of SystemOnTPTP for running CVC3 and Z3 remotely. Remote servers are satisfactory for proof search, at least when they are up and running and the user has Internet access. They also distribute the load: Unless the user's machine has eight processor cores, it would be reckless to launch four ATPs and three SMT solvers locally in parallel and expect the user interface to remain responsive. On an N-core machine, Sledgehammer runs at most N provers locally and up to four of the remaining provers remotely.

4.5 Experiment: Generation of Weights and Triggers

SMT solvers work by incrementally building a model for the quantifier-free part of the problem, which consists of axioms and a negated conjecture. Quantifiers are instantiated at each iteration based on the set of active terms (i.e., the ground terms that the current partial model can interpret). These instances are conjoined with the quantifier-free part of the problem, helping to refine the model.

To guide quantifier instantiation and avert an explosion in the number of instances generated, some SMT solvers support extralogical annotations on their quantifiers. We have done some experiments with weights and triggers.

Weights are specific to Z3. The greater the weight of the quantifier, the fewer instantiations are allowed. The instantiations that take place are those by terms that became active early, because they are more likely to be relevant to the problem at hand. Sledgehammer's iterative relevance filter yields a list of facts sorted by likely relevance. This gives an easy way for Sledgehammer to fill in the weights meaningfully: Give a weight of 0 to the most relevant fact included, N to the least relevant fact, and interpolate in between. The case $N = 0$ corresponds to Z3's default behavior. We use $N = 10$ with a quadratic interpolation, which appears to help more than it harms.

A *trigger* is a set of patterns that must all match some active term for the instantiation to take place. Patterns are usually subterms of the quantified formula. All three supported SMT solvers infer triggers heuristically, but CVC3 and Z3 also provide a syntax for user-specified triggers. We tried to rely on this mechanism to exploit the form of Isabelle lemmas. In particular, equations registered for use by the simplifier often define a function symbol applied to a constructor pattern in terms of a (possibly recursive) right-hand side, as in ML function definitions. It then makes sense to take the entire left-hand side as the only trigger. When an instance of the left-hand side is active, the trigger enables the equation's instantiation.

In stark contrast with the SMT folklore that well chosen triggers are a prerequisite for success [40], we found that the SMT solvers can be relied on to infer suitable triggers and that our scheme for equations is too limited to help much. Future researchers could experiment with adding support for other common syntactic forms, such as introduction and elimination rules.

5 Examples

The Sledgehammer–SMT integration was introduced in late 2010, and since then we have received frequent confirmation of its usefulness, from novices and experts alike. The SMT integration played a prominent role in a library of relational and algebraic methods for modeling computing systems developed by Guttmann et al. [29, p. 630]:

> Our results show that algebraic formal methods can easily be developed by automated reasoning within Isabelle/HOL. A surprising observation is that the SMT solver Z3 often outperformed Metis and sometimes even the external ATP systems invoked by Sledgehammer.

This section relates two success stories: The first one arose on the Isabelle mailing list; the second one originates from a theory of setoids developed by colleagues.

5.1 An Arithmetic Datatype

This gratifying example arose on the Isabelle mailing list [42] barely one week after we had enabled SMT solvers in the development version of Sledgehammer. A new Isabelle user was experimenting with a simple arithmetic algebraic datatype:

datatype *arith* = Z | Succ *arith* | Pred *arith*

inductive isvalue :: *arith* → *bool* **where**
isvalue Z
isvalue *m* ⟹ isvalue (Succ *m*)

inductive step :: *arith* → *arith* → *bool* **where**
s_succ: step *m m'* ⟹ step (Succ *m*) (Succ *m'*)
s_pred_zero: step (Pred Z) Z
s_pred: step *m m'* ⟹ step (Pred *m*) (Pred *m'*)
s_pred_succ: isvalue *v* ⟹ step (Pred (Succ *v*)) *v*

He wanted to prove the following simple property about his inductive predicate step, which takes two *arith* values, but did not know how to proceed:

lemma step (Pred Z) *m* ⟹ *m* = Z

Our colleague Tobias Nipkow helpfully supplied a structured Isabelle proof:

using *assms* **proof** *cases*
  case *s_pred_zero* **thus** *m* = Z **by** *simp*
next
  case (*s_pred m'*)
  from 'step Z *m'*' **have** False **by** *cases*
  thus *m* = Z **by** *blast*
qed

The proof is fairly simple by interactive proving standards, but it nonetheless represents a few minutes' work to a seasoned user (and, as we saw, was too difficult for a novice). Nipkow then tried the development version of Sledgehammer and found a much shorter proof due to Z3:

by (*smt arith.simps*(2,4,5,8) *step.simps*)

The *arith.simps* facts express injectivity of the Pred constructor and distinctness of Z, Succ, and Pred, whereas *step.simps* captures step's introduction and elimination rules as a single equivalence of the form step $m\ m' \longleftrightarrow \exists m\, m'\, v \ldots$. Although the proof involves no theory reasoning beyond equality, the ATPs failed to find it within 30 seconds because of the presence of too many extraneous facts.

5.2 Setoids

A *setoid* $\mathcal{A} = (A, \sim)$ is a set A equipped with an equivalence relation $\sim$. Setoids are heavily used in the Coq theorem prover, where $\sim$ may be some form of extensional equality [7]. As part of a tool for transferring theorems across isomorphisms, Andreas Schropp and Andrei Popescu developed a formalization of setoids in Isabelle [53]. Rather than spend time developing a general theory of setoids, they were interested in deriving the needed results quickly, using automatic tools whenever possible.

Like Guttmann et al., they found that many of their proofs were found by Z3 but not by E, SPASS, or Vampire. Here is a typical example from their formalization, where the proof is found almost instantly by Z3:

lemma $f \in \mathcal{A} \rightsquigarrow \mathcal{B} \Longrightarrow \mathsf{inv}_{\mathcal{B},\mathcal{A}}\ (\mathsf{inv}_{\mathcal{A},\mathcal{B}}\ f) \sim_{\mathcal{A}\to\mathcal{B}} f$
by (*smt assms equiv_rel_trans iso_sDfunsp iso_sDsetoid*(2) *s_inv_eqOf s_inv_iso_s setoid_def sfun_elim sfun_eq_def ssym*)

Here, $\mathcal{A}$ and $\mathcal{B}$ are two setoids, and $\sim_{\mathcal{A}\to\mathcal{B}}$ is the lifting of their equivalence relations to the function space. The lemma states that $(f^{-1})^{-1}$ is equivalent to f if f is an isomorphism between $\mathcal{A}$ and $\mathcal{B}$. A setoid function is an isomorphism if it preserves and reflects the equivalence relations and is surjective modulo equivalence.

A detailed hand-written proof would involve tedious transitivity chains intermixed with applications of preservation and symmetry. This kind of reasoning has an algebraic flavor with mild higher-order aspects. The proof found by Z3 requires 10 facts, reflecting the lack of intermediate lemmas in the formalization (partly as a result of heavily relying on Sledgehammer). The need for so many facts probably explains why the problem was outside the ATPs' reach; 10 facts really are a lot, given the size and complexity of Sledgehammer problems.

6 Evaluation

In their "Judgment Day" study, Böhme and Nipkow [17] evaluated Sledgehammer with E, SPASS, and Vampire on 1240 provable proof goals arising in seven representative Isabelle formalizations. To evaluate the SMT integration, we ran an enlarged version of their benchmark suite with the latest versions of Sledgehammer, the ATPs, and of course the SMT solvers.[3]

[3] Our test data set is available at `http://www21.in.tum.de/~blanchet/jar-smt-data.tgz`.

Thy.	Description	Goals	Feats.
Arr	Arrow's impossibility theorem	101	L S
FFT	Fast Fourier transform	146	A L
FTA	Fundamental theorem of algebra	422	A
Hoa	Completeness of Hoare logic with procedures	203	A I L
Huf	Optimality of Huffman's algorithm	284	A I L S
Jin	Type soundness of a subset of Java	182	I L
NS	Needham–Schroeder shared-key protocol	99	I
QE	DNF-based quantifier elimination	191	A L S
S2S	Sum of two squares	130	A
SN	Strong normalization of the typed λ-calculus	115	A I

Figure 2 Isabelle theories

6.1 Setup

The Judgment Day benchmarks consist of all the proof goals arising in seven theories from the Isabelle distribution and the *Archive of Formal Proofs* [34]. We added two theories (*QE* and *S2S*) that rely heavily on arithmetic to exercise the SMT decision procedures and a third one (*Huf*) that combines all four features. These ten theories are listed in Figure 2 under short names. The third column lists the number of goals from each theory, and the last column specifies the features it contains, where A means arithmetic, I means induction/recursion, L means λ-abstractions, and S means sets. Our examples are representative of typical applications, with a bias toward arithmetic.

We used the following provers as backends: CVC3 2.4.1, E 1.4, E-SInE 0.4, SPASS 3.8ds, Vampire 1.8 (revision 1435), Yices 1.0.33, and Z3 3.2. E-SInE was run remotely via SystemOnTPTP; the other provers were installed on 32-bit Linux servers featuring 3.06 GHz Dual-Core Intel Xeon processors. Each prover was invoked with the set of options we had previously determined worked best.[4]

We ran the provers for 30 seconds each, which corresponds to the default time limit in Sledgehammer. Most proofs are found within a few seconds; Böhme and Nipkow considered timeouts of 60 and 120 seconds, with little impact [17]. Successful proof attempts are followed by a potential 30 second minimization phase and a 30 second reconstruction phase.

6.2 Proof Search

Figure 3 gives the success rates for each prover (or class of prover) on each formalization together with the unique contributions of each prover. Sledgehammer now solves 60.1% of the goals, compared with 54.2% without SMT. The best SMT solver, Z3,

[4] The setup for E was suggested by Stephan Schulz and includes the little known symbol offset weight function. The SPASS setup is described elsewhere [13]. Vampire was run in CASC mode, alternately with and without the set-of-support strategy, and with BDDs disabled. CVC3, E-SInE, Yices, and Z3 were run with the default configuration. All provers except E-SInE were invoked repeatedly with different options for a fraction of the total time limit [10, § 6.6.3].

	FFT AL	*QE* ALS	*Huf* AILS	*NS* I	*Jin* IL	*Arr* LS	*Hoa* AIL	*SN* AI	*FTA* A	*S2S* A	All	Uniq.
E	27	32	30	30	47	47	54	***71***	66	61	47.8	0.3
E-SInE	22	28	27	30	40	26	45	62	60	56	41.8	0.4
SPASS	27	29	29	***40***	43	46	***62***	70	***68***	***62***	***48.7***	0.9
Vampire	23	28	23	38	44	40	61	65	65	52	45.3	0.5
CVC3	***28***	30	36	29	36	39	59	54	65	61	46.3	0.7
Yices	25	***34***	26	28	40	41	56	59	55	61	43.1	0.2
Z3	27	31	***36***	39	***49***	***51***	54	63	62	58	48.1	***1.5***
ATPs	30	36	36	49	48	53	67	72	71	70	54.2	5.8
SMT Solvers	32	36	42	44	49	51	68	64	70	69	54.3	5.9
All Provers	38	45	44	52	52	58	72	74	76	79	60.1	–

Figure 3 Success rates on all goals per prover and theory (%)

	FFT AL	*QE* ALS	*Huf* AILS	*NS* I	*Jin* IL	*Arr* LS	*Hoa* AIL	*SN* AI	*FTA* A	*S2S* A	All	Uniq.
E	22	***21***	20	19	39	***35***	40	***63***	41	27	31.7	0.1
E-SInE	17	16	16	18	34	18	29	56	39	21	26.1	0.4
SPASS	22	17	20	***31***	38	32	***49***	59	***45***	***39***	***33.9***	1.2
Vampire	20	14	16	29	39	29	49	53	40	21	30.4	0.6
CVC3	***23***	12	25	16	29	27	46	41	43	30	29.8	0.8
Yices	19	16	17	16	36	24	47	47	32	23	27.5	0.3
Z3	20	9	***26***	30	***47***	***35***	44	47	40	25	32.1	***1.4***
ATPs	25	24	25	42	42	39	57	64	49	43	39.2	6.6
SMT Solvers	26	17	29	35	47	34	58	51	49	34	38.1	5.5
All Provers	32	26	32	43	47	40	63	64	56	55	44.7	–

Figure 4 Success rates on "nontrivial" goals per prover and theory (%)

outperforms both E and Vampire and sometimes even SPASS, which was specifically optimized for Isabelle [13]. Z3 contributes the most unique proofs: 1.5% of the goals are proved only by it, a figure that climbs to 4.1% if we exclude CVC3 and Yices. While it might be tempting to see this evaluation as a direct comparison of provers, recall that even provers of the same class are not given the same problems or the same options. Sledgehammer is not so much a competition as a *combination* of provers.

About 40% of the goals from the chosen Isabelle formalizations are "trivial" in the sense that they can be solved directly by standard Isabelle tactics invoked with no arguments. If we ignore these and focus on the "nontrivial" goals, which users are especially keen on seeing solved by Sledgehammer, the success rates are somewhat lower, as shown in Figure 4: The ATPs solve 39.2% of these harder goals, and SMT solvers increase the success rate to 44.7%.

It is instructive to compare these results with those obtained by Geoff Sutcliffe on the TPTP [57] benchmark suite. In a preliminary run over 7143 TPTP problems in the FOF (first-order form) division, he found that Z3 2.15 solved 2487 of these,

compared with 3848 for E 1.2 and 4511 for Vampire 0.6 [58]. The comparison is biased because E and Vampire have been extensively optimized against the TPTP, but it still suggests that Isabelle problems are better suited to SMT than typical TPTP problems are.

6.3 Decision Procedures

We also evaluated the extent to which the SMT decision procedures, beyond equality, contribute to the overall result. To this end, we inspected the successful Z3 proofs (including "trivial" ones) to determine the percentage of proofs that involve an arithmetic decision procedure. Theory-specific rewrite rules, which do not rely on any decision procedure, are not counted. Complementarily, we extracted the relevant facts from each Z3 proof and passed them to *metis* with a 30-second time limit.

Figure 5 summarizes the results. The two rows roughly add up to 100% in each column, since Metis is complete for first-order logic but has no support for arithmetic. For the formalizations under study, the vast majority of SMT proofs do not require any theory reasoning and can be reconstructed by a resolution prover.

	FFT AL	*QE* ALS	*Huf* AILS	*NS* I	*Jin* IL	*Arr* LS	*Hoa* AIL	*SN* AI	*FTA* A	*S2S* A	All
Arithmetic	42	14	28	0	0	0	6	0	8	51	13.2
metis	36	69	62	95	92	88	89	92	93	24	78.8

Figure 5 Use of arithmetic in successful Z3 proofs and reconstructibility with *metis* (%)

These results prompted us to benchmark the SMT solvers with Isabelle's arithmetic symbols left uninterpreted, effectively disabling theory reasoning. Arithmetic reasoning is still possible using facts such as $1 + 1 = 2$, which are then considered relevant by the relevance filter. We expected a loss comparable to the use of arithmetic in Z3 proofs, but the actual loss is much smaller. For the *QE* formalization, the SMT solvers' support for arithmetic is actually harmful, as shown in Figure 6. Clearly, arithmetic decision procedures are not the main reason why the SMT solvers perform so well on Isabelle problems.

	FFT AL	*QE* ALS	*Huf* AILS	*NS* I	*Jin* IL	*Arr* LS	*Hoa* AIL	*SN* AI	*FTA* A	*S2S* A	All
CVC3	+5	−10	+4	−1	+1	−1	+3	0	+2	+2	+0.9
Yices	+6	−2	+2	+5	−1	+4	+4	−1	+1	+13	+2.5
Z3	+8	−7	+5	−1	−1	0	−2	+2	−1	+1	+0.1
All Solvers	+6	−6	+4	+1	0	+1	+1	0	+1	+5	+2.7

Figure 6 Absolute success rate differences between SMT solver runs with and without arithmetic on all goals with proof reconstruction (% points)

6.4 Discussion

When evaluating the provers for the CADE paper, we were surprised to find that Z3 outperformed the ATPs by a wide margin. We advanced several explanations, which can now be revisited in the light of new empirical results.

- *Unsound ATP encoding of types*: Until recently, Sledgehammer's encoding of types in unsorted first-order logic was unsound.[5] About 5% of the proofs found by ATPs were type-unsound in HOL and could not be replayed [17, § 4.1]. In contrast, the many-sorted SMT translation was designed to be sound. The passage to sound encodings and use of SPASS's and Vampire's newly introduced sorts improved the ATP success rate by a few percentage points [13, § 7].
- *Translation of λ-abstractions*: Looking at the test data closely, we noticed that SMT solvers seemed to perform better on higher-order problems, suggesting that the *smt* method's λ-lifting approach might be superior to the combinator approach used with the ATPs. However, new experiments reveal that the two λ translation schemes are comparable [10, § 6.7.3], corroborating earlier results [38, § 4].
- *Reliable reconstruction*: In the earlier Sledgehammer setup, as evaluated in the CADE paper, only *metis* was tried on (potentially type-unsound) ATP proofs, not the often more powerful *smt* method. The success rate of ATP proof reconstruction was 88.8%. Now that we have adopted type-sound encodings and added *smt* as a fallback if *metis* times out, 97.7% of the proofs found by ATPs are successfully reconstructed (which is comparable to 98.6% for SMT solvers).
- *Nature of SMT solvers*: Irrespective of the differences in problem encoding and proof reconstruction, SMT solvers appeared to be a good match for the proof obligations arising in Isabelle formalizations. This is largely confirmed by an evaluation where we gave exactly the same TPTP problem files to E, iProver, SPASS, Vampire, and Z3, exploiting Z3's support for the many-sorted TPTP syntax TFF0 [12, § 6].

7 Related Work

We know of very little other work that is comparable to Sledgehammer with regard to key features such as full automation (no need for the user to prepare the problem in any way), proof reconstruction (no assumption that externally found proofs are correct), and source code delivery (no need to run the external reasoners again when replaying a proof). And no other work ever enjoyed the same widespread deployment. Therefore, we use the phrase "related work" in a loose sense.

ATP Integrations. The most notable integrations of resolution and tableau provers in interactive provers are probably Otter in ACL2 [35]; Bliksem in Coq [9]; Gandalf in HOL98 [32]; DISCOUNT, SETHEO, and SPASS in ILF [23]; E and SPASS in Jahob [20]; Otter, PROTEIN, SETHEO, SPASS, and ${}_3T^AP$ in KIV [1, 48]; and Bliksem, EQP, LEO, Otter, PROTEIN, SPASS, TPS, and Waldmeister in ΩMEGA [37, 55].

[5] A sound encoding was also available, but its performance left much to be desired [17, § 4.2; 38, § 3].

ΩMEGA appears to have the most in common with our work, including background execution of external solvers and very powerful techniques for translating a proof found by one system into a proof intelligible to another. The chief difference is that ΩMEGA was created as a research project specifically to investigate proof planning and related ideas within a new, sophisticated architecture.

Few of these other tools seem to have won user acceptance or to have withstood the test of time. Sledgehammer's success ostensibly inspired the new Miz$\mathbb{AR}$ web service for Mizar [51], based on Vampire and SInE. An integration of the equational prover Waldmeister with Agda is under development [27].

SMT Solver Integrations. On the SMT side of things, PVS employs Yices as an oracle [52]. HOL Light integrates CVC Lite and reconstructs its proofs [36]. Jahob supports CVC3 and Z3 as oracles [50]. Isabelle/HOL enjoys two oracle integrations [6, 25] and two tactics with proof reconstruction [15, 18, 26]. HOL4 includes an SMT tactic [18, 60], and an integration of veriT in Coq, based on SMT proof validation, is in progress [3].

8 Conclusion

Sledgehammer has enjoyed considerable success since its inception in 2007 and has become indispensable to most Isabelle users. It is possibly the only interface between interactive and automatic theorem provers to achieve such popularity. It owes much of its success to its ease of use: Sledgehammer is integral to Isabelle and works out of the box, using a combination of locally installed provers and remote servers. It can even be configured to run automatically on all newly entered conjectures, in parallel with other proof tools and counterexample generators.

To Isabelle users, the addition of SMT solvers as backends means that they now get more proofs without effort. The SMT solvers, led by Z3, compete advantageously with the resolution-based ATPs and *metis* even on nonarithmetic problems. In our evaluation, they solved 38% of the nontrivial goals, increasing Sledgehammer's success rate from 39% to 45% on these. Running the SMT solvers in parallel with the ATPs is consistent with our objective of full automation.

To users of SMT solvers, the Sledgehammer–SMT integration eases the transition from automatic proving in first-order logic to interactive proving in higher-order logic. Other tools, such as HOL-Boogie [16], assist in specific applications. Isabelle/HOL is powerful enough for the vast majority of hardware and software verification efforts, and its LCF-style inference kernel provides a trustworthy foundation. Even the developers of SMT solvers profit from the integration: It helps them reach a larger audience, and proof reconstruction brings to light bugs in their tools, including soundness bugs, which might otherwise go undetected.[6]

The recent work by Guttmann et al. on a large Isabelle/HOL repository of algebras for modeling computing systems [29] constitutes an unanticipated validation of our work: They relied almost exclusively on Sledgehammer and the SMT integration

[6] Indeed, we discovered soundness bugs in Yices and Z3 while preparing the CADE paper.

to prove over 1000 propositions, including intricate refinement and termination theorems. To their surprise, they found that Sledgehammer can often automate algebraic proofs at the textbook level.

While the evaluation and user feedback show that the integration is a resounding success, much can still be improved. More work is needed to reconstruct Z3 proofs involving bit vectors and algebraic datatypes [15]. The heuristics for trigger generation are simplistic and would probably benefit from more research.

With the notable exceptions of triggers and weights, we treated the SMT solvers as black boxes. A tighter integration might prove beneficial, as has been observed with other verification tool chains (e.g., VCC/Boogie/Z3 [22] and PVS/SAL/Yices [52]), but it would also require much more work. Obtaining an unsatisfiable core from CVC3 and Yices would be a first small step in the right direction.

Acknowledgments Tobias Nipkow made this collaboration possible and encouraged us throughout. Andrei Popescu helped us write the subsection on setoids. Nikolaj Bjørner promptly fixed a critical bug in Z3's proof generator, and Leonardo de Moura supplied a new executable. Michał Moskal provided expert help on Z3 triggers. Viktor Kuncak, Mark Summerfield, Geoff Sutcliffe, Tjark Weber, and a number of anonymous reviewers provided useful comments on earlier versions of this article. We thank them all.

References

1. Ahrendt, W., Beckert, B., Hähnle, R., Menzel, W., Reif, W., Schellhorn, G., Schmitt, P.H.: Integrating automated and interactive theorem proving. In: W. Bibel, P.H. Schmitt (eds.) Automated Deduction—A Basis for Applications, *Systems and Implementation Techniques*, vol. II, pp. 97–116. Kluwer (1998)
2. Andrews, P.B.: An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof (2nd Ed.), *Applied Logic*, vol. 27. Springer (2002)
3. 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.) Certified Programs and Proofs (CPP 2011), *LNCS*, vol. 7086, pp. 135–150. Springer (2011)
4. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB standard—Version 2.0. In: A. Gupta, D. Kroening (eds.) Satisfiability Modulo Theories (SMT 2010) (2010)
5. Barrett, C., Tinelli, C.: CVC3. In: W. Damm, H. Hermanns (eds.) Computer Aided Verification (CAV 2007), *LNCS*, vol. 4590, pp. 298–302. Springer (2007)
6. Barsotti, D., Nieto, L.P., Tiu, A.: Verification of clock synchronization algorithms: Experiments on a combination of deductive tools. Formal Asp. Comput. **19**(3), 321–341 (2007)
7. Barthe, G., Capretta, V., Pons, O.: Setoids in type theory. J. Funct. Program. **13**(2), 261–293 (2003)
8. Besson, F., Fontaine, P., Théry, L.: A flexible proof format for SMT: A proposal. In: P. Fontaine, A. Stump (eds.) Proof Exchange for Theorem Proving (PxTP-2011), pp. 15–26 (2011)
9. Bezem, M., Hendriks, D., de Nivelle, H.: Automatic proof construction in type theory using resolution. J. Autom. Reasoning **29**(3-4), 253–275 (2002)
10. Blanchette, J.C.: Automatic proofs and refutations for higher-order logic. Ph.D. thesis, Dept. of Informatics, T.U. München (2012)
11. Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending Sledgehammer with SMT solvers. In: N. Bjørner, V. Sofronie-Stokkermans (eds.) Conference on Automated Deduction (CADE-23), *LNAI*, vol. 6803, pp. 207–221. Springer (2011)
12. Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. In: N. Piterman, S. Smolka (eds.) Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2013), *LNCS*, vol. 7795. Springer (2013)
13. 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.) Interactive Theorem Proving (ITP 2012), *LNCS*, vol. 7406, pp. 345–360. Springer (2012)
14. Böhme, S.: Proving theorems of higher-order logic with SMT solvers. Ph.D. thesis, Dept. of Informatics, T.U. München (2012)

15. Böhme, S., Fox, A.C.J., Sewell, T., Weber, T.: Reconstruction of Z3's bit-vector proofs in HOL4 and Isabelle/HOL. In: J.P. Jouannaud, Z. Shao (eds.) Certified Programs and Proofs (CPP 2011), *LNCS*, vol. 7086, pp. 183–198. Springer (2011)
16. Böhme, S., Moskal, M., Schulte, W., Wolff, B.: HOL-Boogie—An interactive prover-backend for the Verifying C Compiler. J. Autom. Reasoning **44**(1-2), 111–144 (2010)
17. Böhme, S., Nipkow, T.: Sledgehammer: Judgement Day. In: J. Giesl, R. Hähnle (eds.) International Joint Conference on Automated Reasoning (IJCAR 2010), *LNAI*, vol. 6173, pp. 107–121. Springer (2010)
18. Böhme, S., Weber, T.: Fast LCF-style proof reconstruction for Z3. In: M. Kaufmann, L. Paulson (eds.) Interactive Theorem Proving (ITP 2010), *LNCS*, vol. 6172, pp. 179–194. Springer (2010)
19. Böhme, S., Weber, T.: Designing proof formats: A user's perspective. In: P. Fontaine, A. Stump (eds.) Proof Exchange for Theorem Proving (PxTP-2011), pp. 27–32 (2011)
20. Bouillaguet, C., Kuncak, V., Wies, T., Zee, K., Rinard, M.C.: Using first-order theorem provers in the Jahob data structure verification system. In: B. Cook, A. Podelski (eds.) Verification, Model Checking, and Abstract Interpretation (VMCAI 2007), *LNCS*, vol. 4349, pp. 74–88. Springer (2007)
21. Bradley, A.R., Manna, Z.: Property-directed incremental invariant generation. Formal Asp. Comput. **20**, 379–405 (2008)
22. Cohen, E., Dahlweid, M., Hillebrand, M.A., Leinenbach, D., Moskal, M., Santen, T., Schulte, W., Tobies, S.: VCC: A practical system for verifying concurrent C. In: S. Berghofer, T. Nipkow, C. Urban, M. Wenzel (eds.) Theorem Proving in Higher Order Logics (TPHOLs 2009), *LNCS*, vol. 5674, pp. 23–42. Springer (2009)
23. Dahn, B., Gehne, J., Honigmann, T., Wolf, A.: Integration of automated and interactive theorem proving in ILF. In: W. McCune (ed.) Conference on Automated Deduction (CADE-14), *LNCS*, vol. 1249, pp. 57–60. Springer (1997)
24. Dutertre, B., de Moura, L.: The Yices SMT solver (2006). Available at `http://yices.csl.sri.com/tool-paper.pdf`
25. Erkök, L., Matthews, J.: Using Yices as an automated solver in Isabelle/HOL. In: J. Rushby, N. Shankar (eds.) Automated Formal Methods (AFM08), pp. 3–13 (2008)
26. Fontaine, P., Marion, J.Y., Merz, S., Nieto, L.P., Tiu, A.: Expressiveness + automation + soundness: Towards combining SMT solvers and interactive proof assistants. In: H. Hermanns, J. Palsberg (eds.) Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2006), *LNCS*, vol. 3920, pp. 167–181. Springer (2006)
27. Foster, S., Struth, G.: Integrating an automated theorem prover into Agda. In: M.G. Bobaru, K. Havelund, G.J. Holzmann, R. Joshi (eds.) NASA Formal Methods (NFM 2011), *LNCS*, vol. 6617, pp. 116–130 (2011)
28. Gordon, M.J.C., Milner, R., Wadsworth, C.P.: Edinburgh LCF: A Mechanised Logic of Computation, *LNCS*, vol. 78. Springer (1979)
29. Guttmann, W., Struth, G., Weber, T.: Automating algebraic methods in Isabelle. In: S. Qin, Z. Qiu (eds.) International Conference on Formal Engineering Methods (ICFEM 2011), *LNCS*, vol. 6991, pp. 617–632. Springer (2011)
30. Hoder, K., Voronkov, A.: Sine qua non for large theory reasoning. In: N. Bjørner, V. Sofronie-Stokkermans (eds.) Conference on Automated Deduction (CADE-23), *LNAI*, vol. 6803, pp. 299–314. Springer (2011)
31. Hughes, R.J.M.: Super-combinators: A new implementation method for applicative languages. In: LISP and Functional Programming (LFP 1982), pp. 1–10. ACM Press (1982)
32. Hurd, J.: Integrating Gandalf and HOL. In: Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, L. Théry (eds.) Theorem Proving in Higher Order Logics (TPHOLs '99), *LNCS*, vol. 1690, pp. 311–321 (1999)
33. 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)
34. Klein, G., Nipkow, T., Paulson, L. (eds.): The Archive of Formal Proofs. `http://afp.sf.net/`
35. McCune, W., Shumsky, O.: System description: IVY. In: D. McAllester (ed.) Conference on Automated Deduction (CADE-17), *LNAI*, vol. 1831, pp. 401–405. Springer (2000)
36. McLaughlin, S., Barrett, C., Ge, Y.: Cooperating theorem provers: A case study combining HOL-Light and CVC Lite. Electr. Notes Theor. Comput. Sci. **144**(2), 43–51 (2006)
37. Meier, A.: TRAMP: Transformation of machine-found proofs into natural deduction proofs at the assertion level (system description). In: D. McAllester (ed.) Conference on Automated Deduction (CADE-17), *LNAI*, vol. 1831, pp. 460–464. Springer (2000)

38. Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. J. Autom. Reasoning **40**(1), 35–60 (2008)
39. Meng, J., Paulson, L.C.: Lightweight relevance filtering for machine-generated resolution problems. J. Applied Logic **7**(1), 41–57 (2009)
40. Moskal, M.: Programming with triggers. In: B. Dutertre, O. Strichman (eds.) Satisfiability Modulo Theories (SMT 2009) (2009)
41. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: C.R. Ramakrishnan, J. Rehof (eds.) Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008), *LNCS*, vol. 4963, pp. 337–340. Springer (2008)
42. Nipkow, T.: Re: [isabelle] A beginner's questionu [sic] (26 November 2010). Archived at `https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2010-November/msg00097.html`
43. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, *LNCS*, vol. 2283. Springer (2002)
44. Nonnengart, A., Weidenbach, C.: Computing small clause normal forms. In: A. Robinson, A. Voronkov (eds.) Handbook of Automated Reasoning, pp. 335–367. Elsevier (2001)
45. Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: G. Sutcliffe, E. Ternovska, S. Schulz (eds.) International Workshop on the Implementation of Logics (IWIL-2010) (2010)
46. Paulson, L.C., Susanto, K.W.: Source-level proof reconstruction for interactive theorem proving. In: K. Schneider, J. Brandt (eds.) Theorem Proving in Higher Order Logics (TPHOLs 2007), *LNCS*, vol. 4732, pp. 232–245 (2007)
47. Ranise, S., Tinelli, C.: The SMT-LIB standard: Version 1.2 (2006). Available at `http://goedel.cs.uiowa.edu/smtlib/papers/format-v1.2-r06.08.30.pdf`
48. Reif, W., Schellhorn, G.: Theorem proving in large theories. In: W. Bibel, P.H. Schmitt (eds.) Automated Deduction—A Basis for Applications, vol. III: Applications, pp. 225–241. Kluwer (1998)
49. Riazanov, A., Voronkov, A.: The design and implementation of Vampire. AI Comm. **15**(2-3), 91–110 (2002)
50. Rinard, M.C.: Integrated reasoning and proof choice point selection in the Jahob system—Mechanisms for program survival. In: R.A. Schmidt (ed.) Conference on Automated Deduction (CADE-22), *LNCS*, vol. 5663, pp. 1–16. Springer (2009)
51. Rudnicki, P., Urban, J.: Escape to ATP for Mizar. In: P. Fontaine, A. Stump (eds.) Proof Exchange for Theorem Proving (PxTP-2011) (2011)
52. Rushby, J.M.: Tutorial: Automated formal methods with PVS, SAL, and Yices. In: D.V. Hung, P. Pandya (eds.) Software Engineering and Formal Methods (SEFM 2006), p. 262. IEEE (2006)
53. Schropp, A.: Instantiating deeply embedded many-sorted theories into HOL types in Isabelle. M.Sc. thesis, Dept. of Informatics, T.U. München (2012)
54. Schulz, S.: System description: E 0.81. In: D. Basin, M. Rusinowitch (eds.) International Joint Conference on Automated Reasoning (IJCAR 2004), *LNAI*, vol. 3097, pp. 223–228. Springer (2004)
55. Siekmann, J., Benzmüller, C., Fiedler, A., Meier, A., Normann, I., Pollet, M.: Proof development with ΩMEGA: The irrationality of $\sqrt{2}$. In: F. Kamareddine (ed.) Thirty Five Years of Automating Mathematics, *Applied Logic*, vol. 28, pp. 271–314. Springer (2003)
56. Sutcliffe, G.: System description: SystemOnTPTP. In: D. McAllester (ed.) Conference on Automated Deduction (CADE-17), *LNAI*, vol. 1831, pp. 406–410. Springer (2000)
57. 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)
58. Sutcliffe, G.: Private communication (2011)
59. Wampler-Doty, M.: A complete proof of the Robbins conjecture. In: G. Klein, T. Nipkow, L. Paulson (eds.) The Archive of Formal Proofs. Available at `http://afp.sf.net/entries/Robbins-Conjecture.shtml` (2010)
60. Weber, T.: SMT solvers: New oracles for the HOL theorem prover. J. Softw. Tools Technol. Transfer **13**(5), 419–429 (2011)
61. Weidenbach, C.: Combining superposition, sorts and splitting. In: A. Robinson, A. Voronkov (eds.) Handbook of Automated Reasoning, pp. 1965–2013. Elsevier (2001)
62. Wenzel, M.: Type classes and overloading in higher-order logic. In: E.L. Gunter, A. Felty (eds.) Theorem Proving in Higher Order Logics (TPHOLs '97), *LNCS*, vol. 1275, pp. 307–322 (1997)
63. Wenzel, M.: Parallel proof checking in Isabelle/Isar. In: G. Dos Reis, L. Théry (eds.) Programming Languages for Mechanized Mathematics Systems (PLMMS 2009). ACM Digital Library (2009)