

Extending Sledgehammer with SMT Solvers

Jasmin Christian Blanchette^{1,*}, Sascha Böhme¹, and Lawrence C. Paulson²

¹ Institut für Informatik, Technische Universität München, Germany

² Computer Laboratory, University of Cambridge, U.K.

Abstract. Sledgehammer is a component of Isabelle/HOL that employs 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. Isabelle users are now pleasantly surprised by SMT proofs for problems beyond the ATPs' reach. Remarkably, the best SMT solver performs better than the best ATP on most of our benchmarks.

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 better at formal modeling than at proving theorems) with a variety of automatic theorem provers (ATPs) [1, 7, 20, 39, 42]. One of the most successful such combinations is Sledgehammer [27, 34], which interfaces Isabelle/HOL [31] with resolution provers for classical first-order logic. Sledgehammer is both effective, solving approximately one third of non-trivial goals arising in interactive proofs [9], and easy to use, since it is invoked with a single mouse gesture. It has become indispensable to Isabelle users and has transformed the way Isabelle is taught to beginners [33].

Given an Isabelle/HOL conjecture, Sledgehammer heuristically selects a few hundred relevant lemmas from Isabelle's libraries, translates them to unsorted first-order logic along with the conjecture, and sends the resulting problem to four theorem provers (Section 2). The provers run in parallel, either locally or remotely via SystemOnTPTP [40]. Users can keep working during the proof search, although most users find it hard to think while automatic provers are active in the background and prefer to wait up to 30 seconds for the responses. Isabelle's built-in prover Metis [21, 34] reconstructs resolution proofs in higher-order logic (HOL).

First-order ATPs are powerful and general, but they can usefully be complemented by other technologies. Satisfiability modulo theories (SMT) is a powerful technology based on combining 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

* Research supported by the Deutsche Forschungsgemeinschaft [grant number Ni 491/11-2]. Sledgehammer was originally supported by the UK's Engineering and Physical Sciences Research Council [grant number GR/S57198/01].

verification. Although they are automatic theorem provers in a general sense, they rely on techniques entirely different from classical resolution. In this paper, we will find it convenient to reserve the abbreviation ATP for resolution provers.¹

There have also been several attempts to combine interactive theorem provers with SMT solvers, either as oracles [5, 17, 37] or with proof reconstruction [18, 22, 26]. In previous work, we integrated the SMT solvers CVC3 [4], Yices [16] and Z3 [15] with Isabelle as oracles and implemented step-by-step proof reconstruction for Z3 [10]. 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* proof method [8], 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* [23] in 2010 and 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 *smt* call.

Can typical 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 seemed to be promising for a number of reasons:

- ATPs and SMT solvers have complementary strengths. The former handle quantifiers better, whereas the latter excel on large, mostly ground problems.
- The translation of higher-order constructs and types is done differently for the SMT solvers than for the ATPs—differences that should result in more proved goals.²
- 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.

Such an integration required extensive refactoring of Sledgehammer, a delicate piece of engineering developed by eight people in Cambridge and Munich over a period of seven years. The refactoring seemed worthwhile, especially since it also benefits other provers that we might want to interface with Sledgehammer, such as higher-order ATPs [3, 6].

The Sledgehammer–SMT integration is, to our knowledge, 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 typical 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 terminated due to a segmentation fault. 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 “triggers”—patterns that guide quantifier instantiations. Our evaluation on a large benchmark suite shows that SMT solvers add considerable power to Sledgehammer (Section 5).

¹ Instantiation-based provers such as Equinox [12] and iProver [24] are promising, but in case of success they currently do not deliver a proof, not even the list of used axioms.

² There are also many efficiency, readability, and robustness advantages of obtaining several proofs for the same goal from different sources [41].

2 Sledgehammer

Sledgehammer is Isabelle’s subsystem for harnessing the power of first-order ATPs. 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 to extract from Isabelle’s enormous libraries a few hundred lemmas that appear to be relevant to the problem at hand. The relevance test is based on how many constants (symbols) are shared between the conjecture and each candidate lemma [28]. Although crude, this filter greatly improves Sledgehammer’s success rate, because most ATPs perform badly in the presence of thousands of axioms.

Translation into Classical First-Order Logic. Isabelle’s formalism, polymorphic higher-order logic [2, 45], is much richer than the ATPs’ unsorted first-order logic. Sledgehammer uses various techniques to translate HOL formulas to first-order logic [27]. Many compromises are necessary here. The translation is unsound; the ATP proofs can be trusted only after they have been reconstructed. Higher-order features complicate the translation: λ -abstractions are rewritten to combinators, and curried functions are passed varying numbers of arguments by means of an explicit apply operator.

Parallel ATP Invocation. For a number of years, Isabelle has emphasized parallelism to exploit modern multi-core architectures [46]. Accordingly, Sledgehammer invokes several ATPs in parallel, with great success: Running E [38], SPASS [44], and Vampire [36] in parallel for five seconds solves as many problems as running a single theorem prover for two minutes [9, §8]. Recent versions of Sledgehammer also invoke SInE [19], a wrapper around E that is designed to cope with large axiom bases.

Proof Reconstruction. As in other LCF-style theorem provers, Isabelle theorems can only be generated within a small inference kernel. It is possible to bypass this safety mechanism, generally if some external tool is to be trusted as an oracle, but all oracle inferences are tracked. Sledgehammer performs true proof reconstruction by running Isabelle’s built-in resolution prover, Metis, supplying it with the short list of facts used in the proof found by the external ATP.

The Metis call with the identified facts is all that Sledgehammer includes in the Isabelle proof text, which can then be replayed without external provers. Since Metis is given only a handful of facts, it usually finds proofs within milliseconds.

Proof Minimization. Proof reconstruction using Metis loses about 10% of ATP proofs, partly because some of the proofs are unsound in a typed setting, but also because Metis times out [9, §3]. Automatic provers frequently use many more facts than are necessary. Sledgehammer’s minimization tool takes a set of facts returned by a prover and repeatedly calls it 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 prover invocations as there are facts in the initial set. A refinement is to inspect the ATP proofs to eliminate more facts at each iteration.

- The binary algorithm recursively bisects the facts [11, §4.3]. It performs best when a small fraction of the facts are actually required [9, §7].

Example. In the Isabelle proof below, taken from a formalization of the Robbins conjecture [43], four of the five subproofs are discharged by a Metis call generated automatically by Sledgehammer using an ATP:

```

proof –
  let  $z = \neg(x \sqcup \neg y)$  and  $ky = y \sqcup k \otimes (x \sqcup z)$ 
  have “ $\neg(x \sqcup \neg ky) = z$ ” by (simp add: cosp0)
  hence “ $\neg(\neg ky \sqcup \neg(\neg y \sqcup z)) = z$ ” by (metis assms sup_comm)
  also have “ $\neg(z \sqcup \neg ky) = x$ ” by (metis assms cosp0 sup_comm)
  hence “ $z = \neg(\neg y \sqcup \neg(\neg ky \sqcup z))$ ” by (metis sup_comm)
  finally show “ $\neg(y \sqcup k \otimes (x \sqcup \neg(x \sqcup \neg y))) = \neg y$ ” by (metis eq_intro)
qed

```

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

3 The SMT Proof Method

SMT solvers are available in Isabelle through the *smt* proof method. 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 equality and arithmetic operators to the corresponding SMT-LIB 1.2 [35] concepts. The theories of arrays, bit vectors, and algebraic datatypes are not yet exploited.

Many-sorted first-order logic’s support for sorts would seem to make it more appropriate to encode HOL typing information than classical first-order logic, but it does not support polymorphism. Several solutions have been proposed in the literature [14, 25]. Our current approach is to monomorphize the formulas: Polymorphic formulas are iteratively instantiated with relevant ground instances of their polymorphic constants. This process is iterated a bounded number of times to obtain the monomorphized problem.

Partial applications are translated using an explicit apply operator. In contrast with the combinator approach used by Sledgehammer when communicating with ATPs, the *smt* method lifts λ -abstractions into new rules, thereby introducing fresh constants.

Proof Reconstruction. CVC3 and Z3 provide independently checkable proofs of unsatisfiability. We have implemented proof reconstruction for Z3 and support 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.

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 SMT solvers as oracles requires trusting both the solvers and the *smt* method’s translation, so it is generally frowned upon.

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:

lemma “ $x_3 = |x_2| - x_1 \wedge x_4 = |x_3| - x_2 \wedge x_5 = |x_4| - x_3 \wedge x_6 = |x_5| - x_4 \wedge$
 $x_7 = |x_6| - x_5 \wedge x_8 = |x_7| - x_6 \wedge x_9 = |x_8| - x_7 \wedge x_{10} = |x_9| - x_8 \wedge$
 $x_{11} = |x_{10}| - x_9 \implies x_1 = x_{10} \wedge x_2 = (x_{11} :: int)$ ”

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 prove 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 the previous section.

4 Combining Sledgehammer and 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.

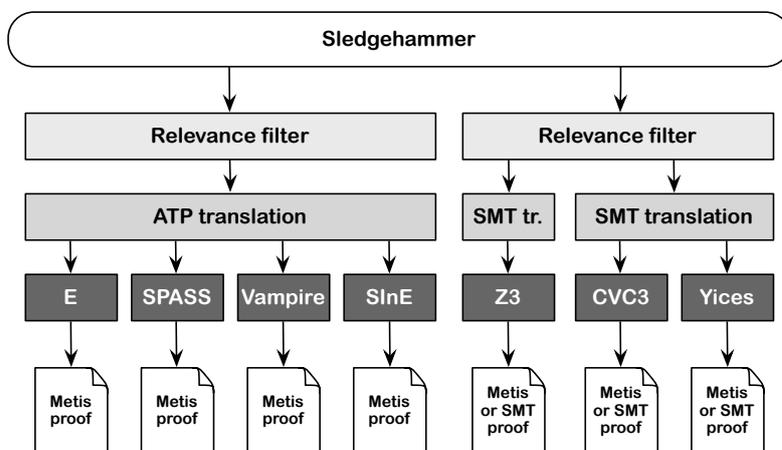


Figure 1. Sledgehammer’s extended architecture

Two instances of the relevance filter run in parallel, to account for different sets of built-in constants. 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. The translation for Z3 is done slightly differently than for CVC3 and Yices to take advantage of the former’s support for nonlinear arithmetic.

4.1 Relevance Filtering

In the old architecture, the available lemmas were rewritten to conjunctive normal form (CNF) using a naive application of distributive laws before the relevance filter was invoked [28]. To avoid clausifying thousands of lemmas on each Sledgehammer invocation, the CNF 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 [32].

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 constants in the CNF-based code, we keep track of polarities and detect quantifiers that give rise to Skolem constants.

The relevance filter gives more precise results if it ignores HOL constants that are translated to built-in constructs. For ATPs, this concerns equality, connectives, and quantifiers, as well as *let* and *if-then-else*. SMT solvers support a much larger set of built-in constructs, notably arithmetic operators. It was straightforward to generalize the filter code so that it performs its task appropriately for SMT solvers.

Observing that some provers cope better with large fact bases than others, we optimized the the maximum number of relevant facts to include in a problem independently for each prover (from a library of about 10000 facts). The maxima we obtained are 150 for CVC3 and Yices and 250 for Z3. In comparison, the filter currently selects up to 250 facts for E, 150 for SPASS, 450 for Vampire, and 500 for SInE.

4.2 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 of hundreds of facts took many seconds.
- It took us a while to get the bugs out of our translation code. Syntax errors in many generated problems 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 SMT solvers had involved only a handful of 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 4, to great effect.

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

We reported the segmentation faults to the Z3 developers, who released an improved version. The bug was located in Z3’s proof generation facility, which is disabled by default and hence not as well tested as the rest of the solver. 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.

4.3 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. For example:

```
by (smt assms copy0 sup_comm)
```

The proof method invokes Z3 to re-find the proof whenever and replays it step by step. The Z3 proof can also be stored alongside the Isabelle formalization as a certificate to avoid invoking Z3 each time the proof is rechecked. Proof 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 for one second. If Metis succeeds, Sledgehammer generates a Metis call rather than an *smt* call. Metis will of course fail if the proof requires theories other than equality.

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:

- 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.
- Yices can output a minimal core, but for technical reasons only when its native input syntax is used rather than the standard SMT-LIB 1.2 format. 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.
- 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 come with various minor changes.

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. Today, 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.

For SMT solvers, the situation is similar. Only CVC3 allows redistribution and use by noncommercial and commercial users alike, and Z3 executables are not available for

Mac OS X. With the Z3 developers' express permission, we set up a server in Munich in the style of SystemOnTPTP for running Z3 (as well as CVC3) remotely.

Remote servers are satisfactory for proof search, at least when they are up and running and the user has Internet access. They also help 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 snappy.

4.5 Experiment: Generation of Weights and Triggers

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

To help guide quantifier instantiation and avert an explosion of the number of instances generated, some SMT solvers support extralogical annotations on their quantifiers. We have done some experiments with weights and triggers, which so far have been somewhat inconclusive.

Weights. Weights are specific to Z3. The greater the weight of the quantifier, the fewer instantiations are allowed. The instantiations that are allowed are those by terms that became active early, because they are more likely to be relevant to the problem at hand.

Intuitively, there is an easy way for Sledgehammer to fill in the weights meaningfully. The iterative relevance filter yields a list of facts sorted by likely relevance. We can give a weight of 0 to the most relevant fact included, N to the least relevant fact, and interpolate in between. If $N = 0$, we obtain Z3's default behavior. We currently use $N = 10$ with a quadratic interpolation, which seems to help more than it harms.

Triggers. 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. CVC3, Yices, and Z3 infer the 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/HOL lemmas. In particular, equations registered for use by the Isabelle simplifier typically define a function symbol applied to a constructor pattern in terms of a (possibly recursive) right-hand side. 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 [29], we found that the SMT solvers can be relied on to infer acceptable triggers and that our scheme for equations is too limited to help much. Perhaps we should try to add support for other common syntactic forms, such as introduction and elimination rules, to obtain greater benefits. This remains for future work.

4.6 Example

A gratifying example arose on the Isabelle mailing list [30] 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
```

He had defined an inductive predicate *step* that takes two *arith* values and wanted to prove the following simple property but did not know how to proceed:

```
lemma “step (Pred Z) m  $\implies$  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). Our colleague 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)
```

Although it 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 Evaluation

In their “Judgment Day” study, Böhme and Nipkow [9] evaluated Sledgehammer with E, SPASS, and Vampire on 1240 provable proof goals arising in seven representative formalizations from the Isabelle distribution and the *Archive of Formal Proofs*. To evaluate the SMT integration, we ran their benchmark suite with the latest versions of Sledgehammer on the same seven formalizations.³ We also added two formalizations (*QE* and *S2S*) that rely heavily on arithmetic to exercise the SMT decision procedures.

The formalizations are listed below. The last two columns give the percentage of the (now) 1591 proof goals that come from each formalization and the features it contains, where A means arithmetic, I means induction and recursion, L means λ -abstractions, and S means sets.

<i>Arrow</i>	Arrow’s impossibility theorem	L S	6.3%
<i>FFT</i>	Fast Fourier transform	A L	9.1%
<i>FTA</i>	Fundamental theorem of algebra	A	26.6%
<i>Hoare</i>	Completeness of Hoare logic with procedures	A I L	12.8%
<i>Jinja</i>	Type soundness of a subset of Java	I L	11.4%
<i>NS</i>	Needham–Schroeder shared-key protocol	I	6.2%
<i>QE</i>	DNF-based quantifier elimination	A L S	12.0%
<i>S2S</i>	Sum of two squares	A	8.1%
<i>SN</i>	Strong normalization of the typed λ -calculus	A I	7.2%

³ Our test data set is available at <http://www4.in.tum.de/~blanchet/cade2011-data.tgz>.

	<i>Arrow</i>	<i>FFT</i>	<i>FTA</i>	<i>Hoare</i>	<i>Jinja</i>	<i>NS</i>	<i>QE</i>	<i>S2S</i>	<i>SN</i>	All	Uniq.
E 1.2	25%	14%	61%	41%	33%	28%	23%	38%	59%	40.0%	.3%
SPASS 3.7	33%	14%	57%	50%	33%	29%	28%	37%	61%	41.4%	.4%
Vampire 1.0	31%	19%	62%	47%	35%	44%	24%	48%	62%	44.3%	1.0%
SInE 0.4	22%	17%	56%	38%	31%	26%	20%	38%	63%	38.0%	.3%
CVC3 2.2	33%	19%	51%	47%	37%	25%	21%	57%	55%	40.5%	.1%
Yices 1.0.28	29%	16%	50%	50%	38%	28%	21%	58%	59%	40.7%	.1%
Z3 2.15	47%	18%	62%	54%	49%	42%	25%	58%	63%	48.6%	3.1%
ATPs	40%	21%	68%	55%	37%	45%	31%	55%	70%	50.0%	6.8%
SMT solvers	50%	23%	65%	64%	50%	42%	26%	66%	63%	52.2%	9.0%
All provers	55%	28%	73%	65%	51%	51%	41%	73%	74%	59.0%	–

Figure 2. Success rates on all goals with proof reconstruction

	<i>Arrow</i>	<i>FFT</i>	<i>FTA</i>	<i>Hoare</i>	<i>Jinja</i>	<i>NS</i>	<i>QE</i>	<i>S2S</i>	<i>SN</i>	All	Uniq.
E 1.2	21%	10%	34%	25%	28%	17%	12%	9%	47%	23.1%	.1%
SPASS 3.7	29%	12%	31%	36%	27%	17%	16%	12%	53%	25.6%	.7%
Vampire 1.0	21%	17%	40%	33%	32%	36%	8%	19%	51%	28.7%	1.3%
SInE 0.4	19%	14%	29%	21%	26%	13%	9%	11%	53%	21.4%	.2%
CVC3 2.2	23%	14%	26%	29%	32%	13%	7%	25%	39%	22.8%	.2%
Yices 1.0.28	11%	11%	26%	37%	33%	16%	7%	26%	46%	23.6%	.1%
Z3 2.15	35%	13%	41%	45%	48%	34%	7%	28%	47%	33.2%	4.0%
ATPs	31%	18%	42%	41%	32%	38%	19%	26%	63%	33.7%	6.9%
SMT solvers	39%	17%	42%	52%	48%	34%	8%	33%	49%	35.9%	9.1%
All provers	42%	23%	50%	54%	48%	44%	23%	42%	64%	42.8%	–

Figure 3. Success rates on “nontrivial” goals with proof reconstruction

We ran the provers for 30 seconds, which corresponds to the default timeout in Sledgehammer. Even though the tool runs asynchronously, most users cannot think while it is active; they prefer to wait for it to return, hoping to get a proof for free. Böhme and Nipkow [9] considered timeouts of 60 and 120 seconds but found that these have a limited impact on the success rate.

If a proof is found, proof search is followed by reconstruction with a 30-second time limit. We allotted an extra 30 seconds to CVC3 and Yices to account for the expensive black-box proof minimization. This might not be entirely fair, but it reflects a compromise between the real power of these solvers and what Isabelle users currently perceive. Moreover, users are normally patient when they know that a proof has been found and has reached the minimizer.

Figure 2 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 59.0% of the goals, compared with 50.0% without SMT. Much to our surprise, the best SMT solver, Z3, beats the best ATP, Vampire, with 48.6% versus 44.3%. Z3 also contributes by far the most unique proofs: 3.1% of the goals are proved only by it, a figure that climbs to 8.5% if we exclude CVC3 and Yices.

	<i>Arrow</i>	<i>FFT</i>	<i>FTA</i>	<i>Hoare</i>	<i>Jinja</i>	<i>NS</i>	<i>QE</i>	<i>S2S</i>	<i>SN</i>	All
Arithmetic	0%	49%	7%	9%	0%	0%	21%	51%	3%	12.9%
Metis	80%	24%	89%	77%	80%	92%	60%	29%	100%	75.9%

Figure 4. Use of arithmetic in successful Z3 proofs and reconstructibility with Metis

	<i>Arrow</i>	<i>FFT</i>	<i>FTA</i>	<i>Hoare</i>	<i>Jinja</i>	<i>NS</i>	<i>QE</i>	<i>S2S</i>	<i>SN</i>	All
CVC3 2.2	+3%	-2%	+1%	+3%	-4%	+5%	+13%	-8%	-5%	+0.8%
Yices 1.0.28	-2%	-4%	+2%	0%	-2%	-4%	+11%	+2%	-1%	+0.8%
Z3 2.15	+2%	0%	-5%	-2%	0%	-1%	+10%	-1%	-3%	-0.9%
SMT solvers	+2%	-2%	-3%	-3%	-1%	0%	+10%	-3%	-3%	-1.6%

Figure 5. Absolute success rate differences between SMT solver runs without and with arithmetic on all goals with proof reconstruction

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 number of facts or the same options. Sledgehammer is not so much a competition as a *combination* of provers.

About one third 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 3: The ATPs solve 33.7% of these harder goals, and SMT solvers increase the success rate to 42.8%.

We also evaluated the extent to which the SMT decision procedures (other than equality) contribute to the overall result. To this end, we inspected the successful Z3 proofs 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 the Z3 proofs and passed them to Metis with a 30-second time limit. Figure 4 summarizes the results. 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.

These results prompted us to benchmark the SMT solvers with Isabelle’s arithmetic constants left uninterpreted, effectively disabling theory reasoning. We expected a loss comparable to the use of arithmetic in Z3 proofs, but the actual loss is much smaller. For some formalizations the success rates actually improved, as shown in Figure 5. Further experiments indicate that most of the 1.6% decrease in absolute success rates (from 52.2% to 50.6%) can be recovered by passing more facts to the SMT solvers. This is to be expected: When arithmetic constants are left uninterpreted, more facts are necessary to reason about them.

Arithmetic decision procedures are therefore not the main reason why the SMT solvers collectively outperform the ATPs. A more important reason is that many proofs found by ATPs are type-unsound in higher-order logic and cannot be replayed; in contrast, the SMT translation is designed to be sound by exploiting SMT sorts. Moreover, Metis sometimes fails to rediscover an ATP proof within a reasonable time, whereas proof reconstruction for Z3 is typically faster and more reliable.

Looking at the test data more closely, we also noticed that SMT solvers performed better on higher-order problems, suggesting that the *smt* method’s translation of λ -abstractions is better suited to the SMT solvers than combinators are to the ATPs. Remarkably, previous experiments found combinators superior to λ -lifting for ATPs [27]. We need to carry out new experiments to gain clarity on this point.

6 Conclusion

Sledgehammer has enjoyed considerable success since its inception in 2007 and has become indispensable to most Isabelle users, both novices and experts. It is possibly the only interface between interactive and automatic theorem provers to achieve such popularity. It owes 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.

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 non-arithmetic problems. In our evaluation, they solved about 36% of the nontrivial goals, increasing Sledgehammer’s success rate from 34% to 43% on these. Running the SMT solvers in parallel with the ATPs is entirely appropriate, for how is the user supposed to know which class of prover will perform best?

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 [8], 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.⁴

While the evaluation and user feedback show that the integration is a resounding success, much can still be improved. Work is under way to reconstruct Z3 proofs involving arrays, bit vectors, and algebraic datatypes. The heuristics for trigger generation are simplistic and would probably benefit from more research. The encoding of HOL types, based on monomorphization, was never meant to cope with hundreds of facts and could also benefit from new ideas.

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 [13] and PVS/SAL/Yices [37]), 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.

The main open question is the extent to which the improvements we obtained by adding support for SMT provers are due to the *smt* method’s translation and proof reconstruction as opposed to the nature of SMT provers. To clarify this, we plan to carry out further experiments with SPASS’s support for sorts and Z3’s unsorted input format.

⁴ Indeed, we discovered a soundness bug in Yices and another in Z3 while preparing this paper.

Acknowledgment. Tobias Nipkow made this work possible and encouraged us throughout. Nikolaj Bjørner promptly fixed a critical bug in Z3’s proof generator, and Leonardo de Moura supplied a new Linux executable. Michał Moskal provided expert help on Z3 triggers. Mark Summerfield, Tjark Weber, and several anonymous reviewers provided useful comments on drafts of this paper. We thank them all.

References

1. W. Ahrendt, B. Beckert, R. Hähnle, W. Menzel, W. Reif, G. Schellhorn, and P. H. Schmitt. Integrating automated and interactive theorem proving. In W. Bibel and P. H. Schmitt, editors, *Automated Deduction—A Basis for Applications*, volume II of *Systems and Implementation Techniques*, pages 97–116. Kluwer, 1998.
2. P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof (2nd Ed.)*, volume 27 of *Applied Logic*. Springer, 2002.
3. J. Backes and C. E. Brown. Analytic tableaux for higher-order logic with choice. In J. Giesl and R. Hähnle, editors, *International Joint Conference on Automated Reasoning*, volume 6173 of *LNAI*, pages 76–90. Springer, 2010.
4. C. Barrett and C. Tinelli. CVC3. In W. Damm and H. Hermanns, editors, *Computer Aided Verification*, volume 4590 of *LNCS*, pages 298–302. Springer, 2007.
5. D. Barsotti, L. P. Nieto, and A. Tiu. Verification of clock synchronization algorithms: Experiments on a combination of deductive tools. *Formal Asp. Comput.*, 19(3):321–341, 2007.
6. C. Benzmüller, L. C. Paulson, F. Theiss, and A. Fietzke. LEO-II—a cooperative automatic theorem prover for higher-order logic. In A. Armando, P. Baumgartner, and G. Dowek, editors, *International Joint Conference on Automated Reasoning*, volume 5195 of *LNAI*, pages 162–170. Springer, 2008.
7. M. Bezem, D. Hendriks, and H. de Nivelle. Automatic proof construction in type theory using resolution. *J. Auto. Reas.*, 29(3–4):253–275, 2002.
8. S. Böhme, M. Moskal, W. Schulte, and B. Wolff. HOL-Boogie—an interactive prover-backend for the Verifying C Compiler. *J. Auto. Reas.*, 44(1–2):111–144, 2010.
9. S. Böhme and T. Nipkow. Sledgehammer: Judgement Day. In J. Giesl and R. Hähnle, editors, *International Joint Conference on Automated Reasoning*, volume 6173 of *LNAI*, pages 107–121. Springer, 2010.
10. S. Böhme and T. Weber. Fast LCF-style proof reconstruction for Z3. In M. Kaufmann and L. Paulson, editors, *Interactive Theorem Proving*, volume 6172 of *LNCS*, pages 179–194. Springer, 2010.
11. A. R. Bradley and Z. Manna. Property-directed incremental invariant generation. *Formal Asp. Comput.*, 20:379–405, 2008.
12. K. Claessen. Equinox, a new theorem prover for full first-order logic with equality. Presentation at Dagstuhl Seminar on Deduction and Applications, 2005.
13. E. Cohen, M. Dahlweid, M. A. Hillebrand, D. Leinenbach, M. Moskal, T. Santen, W. Schulte, and S. Tobies. VCC: A practical system for verifying concurrent C. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics*, volume 5674 of *LNCS*, pages 23–42. Springer, 2009.
14. J.-F. Couchot and S. Lescuyer. Handling polymorphism in automated deduction. In F. Pfenning, editor, *Conference on Automated Deduction*, volume 4603 of *LNAI*, pages 263–278. Springer, 2007.
15. L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

16. B. Dutertre and L. de Moura. The Yices SMT solver, 2006. Available at <http://yices.csl.sri.com/tool-paper.pdf>.
17. L. Erkök and J. Matthews. Using Yices as an automated solver in Isabelle/HOL. In J. Rushby and N. Shankar, editors, *Automated Formal Methods*, pages 3–13, 2008.
18. P. Fontaine, J.-Y. Marion, S. Merz, L. P. Nieto, and A. Tiu. Expressiveness + automation + soundness: Towards combining SMT solvers and interactive proof assistants. In H. Hermanns and J. Palsberg, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *LNCS*, pages 167–181. Springer, 2006.
19. K. Hoder. Sine qua non for large theory reasoning. Presentation at Dagstuhl Seminar on Deduction at Scale, 2011.
20. J. Hurd. Integrating Gandalf and HOL. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *Theorem Proving in Higher Order Logics*, volume 1690 of *LNCS*, pages 311–321, 1999.
21. 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.
22. C. Keller. Cooperation between SAT, SMT provers and Coq. Presentation at the Synthesis, Verification and Analysis of Rich Models workshop, 2011.
23. G. Klein, T. Nipkow, and L. Paulson, editors. *The Archive of Formal Proofs*. <http://afp.sf.net/>.
24. K. Korovin. Instantiation-based automated reasoning: From theory to practice. In R. A. Schmidt, editor, *Conference on Automated Deduction*, volume 5663 of *LNAI*, pages 163–166. Springer, 2009.
25. K. R. M. Leino and P. Rümmer. A polymorphic intermediate verification language: Design and logical encoding. In J. Esparza and R. Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *LNCS*, pages 312–327. Springer, 2010.
26. S. McLaughlin, C. Barrett, and Y. Ge. Cooperating theorem provers: A case study combining HOL-Light and CVC Lite. *Electr. Notes Theor. Comput. Sci.*, 144(2):43–51, 2006.
27. J. Meng and L. C. Paulson. Translating higher-order clauses to first-order clauses. *J. Auto. Reas.*, 40(1):35–60, 2008.
28. J. Meng and L. C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. *J. Applied Logic*, 7(1):41–57, 2009.
29. M. Moskal. Programming with triggers. In B. Dutertre and O. Strichman, editors, *Satisfiability Modulo Theories*, 2009.
30. T. Nipkow. Re: [isabelle] A beginner’s questionu [sic], 26 Nov. 2010. Archived at <https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2010-November/msg00097.html>.
31. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
32. A. Nonnengart and C. Weidenbach. Computing small clause normal forms. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 335–367. Elsevier, 2001.
33. 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, *International Workshop on the Implementation of Logics*, 2010.
34. L. C. Paulson and K. W. Susanto. Source-level proof reconstruction for interactive theorem proving. In K. Schneider and J. Brandt, editors, *Theorem Proving in Higher Order Logics*, volume 4732 of *LNCS*, pages 232–245, 2007.
35. S. Ranise and C. Tinelli. The SMT-LIB standard: Version 1.2, 2006. Available at <http://goedel.cs.uiowa.edu/smtlib/papers/format-v1.2-r06.08.30.pdf>.

36. A. Riazanov and A. Voronkov. The design and implementation of Vampire. *AI Comm.*, 15(2–3):91–110, 2002.
37. J. M. Rushby. Tutorial: Automated formal methods with PVS, SAL, and Yices. In D. V. Hung and P. Pandya, editors, *Software Engineering and Formal Methods*, page 262. IEEE, 2006.
38. S. Schulz. System description: E 0.81. In D. Basin and M. Rusinowitch, editors, *International Joint Conference on Automated Reasoning*, volume 3097 of *LNAI*, pages 223–228. Springer, 2004.
39. J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, I. Normann, and M. Pollet. Proof development with Ω MEGA: The irrationality of $\sqrt{2}$. In F. Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, volume 28 of *Applied Logic*, pages 271–314. Springer, 2003.
40. G. Sutcliffe. System description: SystemOnTPTP. In D. McAllester, editor, *Conference on Automated Deduction*, volume 1831 of *LNAI*, pages 406–410. Springer, 2000.
41. G. Sutcliffe, C. Chang, L. Ding, D. McGuinness, and P. P. da Silva. Different proofs are good proofs. In D. McGuinness, A. Stump, G. Sutcliffe, and C. Tinelli, editors, *Workshop on Evaluation Methods for Solvers, and Quality Metrics for Solutions*, pages 1–10, 2010.
42. J. Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Auto. Reas.*, 37(1–2):21–43, 2006.
43. 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.
44. C. Weidenbach. Combining superposition, sorts and splitting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 1965–2013. Elsevier, 2001.
45. M. Wenzel. Type classes and overloading in higher-order logic. In E. L. Gunter and A. Felty, editors, *Theorem Proving in Higher Order Logics*, volume 1275 of *LNCS*, pages 307–322, 1997.
46. M. Wenzel. Parallel proof checking in Isabelle/Isar. In G. Dos Reis and L. Théry, editors, *Programming Languages for Mechanized Mathematics Systems*. ACM Digital Library, 2009.