Seventeen Provers Under the Hammer

Martin Desharnais
Graduate School of Computer Science, Saarland Informatics Campus, Saarbrücken, Germany
Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany

Petar Vukmirović
Vrije Universiteit Amsterdam, The Netherlands

Jasmin Blanchette
Vrije Universiteit Amsterdam, The Netherlands
Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany

Makarius Wenzel
Augsburg, Germany

Abstract

One of the main success stories of automatic theorem provers has been their integration into proof assistants. Such integrations, or "hammers," increase proof automation and hence user productivity. In this paper, we use Isabelle/HOL's Sledgehammer tool to find out how useful modern provers are at proving formulas in higher-order logic. Our evaluation follows in the steps of Böhme and Nipkow's Judgment Day study from 2010, but instead of three provers we use 17, including SMT solvers and higher-order provers. Our work offers an alternative yardstick for comparing modern provers, next to the benchmarks and competitions emerging from the TPTP World and SMT-LIB.

2012 ACM Subject Classification Computing methodologies → Theorem proving algorithms

Keywords and phrases Automatic theorem proving, interactive theorem proving, proof assistants

Digital Object Identifier 10.4230/LIPIcs.ITP.2022.8

Supplementary Material *Dataset*: `https://doi.org/10.5281/zenodo.5940084`

Funding This research has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka).
Jasmin Blanchette: has received funding from the Netherlands Organization for Scientific Research (NWO) under the Vidi program (project No. 016.Vidi.189.037, Lean Forward).

Acknowledgements We are grateful to the maintainers of StarExec for letting us use their service. Developers of automatic theorem provers, including Peter Baumgartner, Ahmed Bhayat, Pascal Fontaine, Konstantin Korovin, Giles Reger, Andrew Reynolds, Philipp Rümmer, Alexander Steen, and Martin Suda have helped us run their systems. Michael Färber, Thibault Gauthier, Konstantin Korovin, Lorenz Leutgeb, Jens Otten, Philipp Rümmer, Stephan Schulz, Hans-Jörg Schurr, Alexander Steen, Martin Suda, Mark Summerfield, Dmitriy Traytel, Josef Urban, and the anonymous reviewers suggested some textual improvements.

1 Introduction

Hammers [15] are tools that integrate automatic theorem provers in proof assistants, to automatically discharge proof obligations. CoqHammer [27], HOLyHammer [35], MizAℝ [58], and Sledgehammer (Section 2) are examples of hammers. They typically consist of three main components in addition to the automatic prover backends themselves:

- The *relevance filter* heuristically selects relevant lemmas (including definitions) based on the current goal, using either an iterative procedure or machine learning.

- The *problem translation* module encodes formulas from the proof assistant's logic to the automatic provers' usually less expressive logics. The encoding should ideally be sound and complete.
- The *proof reconstruction* module translates proofs generated by the automatic provers to proof texts expressed in the proof assistant's language.

There is ample evidence that hammers can be effective. The landmark Judgment Day study by Böhme and Nipkow [20] showed for the first time that a combination of three automatic provers (E [49], SPASS [61], Vampire [38]) could discharge about half of the goals that arise in typical Isabelle/HOL [43] developments. This came as a surprise to most Isabelle users, who had not yet integrated Sledgehammer into their workflow. Since then, it has been adopted by a large majority of users.

For over a decade, Judgment Day has been the most comprehensive evaluation of Sledgehammer, even though it considered only three provers and seven Isabelle theory files. Smaller evaluations in later papers [7, 8, 11–13, 17, 45, 50, 53] cover more provers, but they mostly predate the new generation of higher-order provers: cvc5 [1], Ehoh [60], Leo-III [51], Vampire [9], and Zipperposition [59]. We do not have a clear picture of how well these provers work in a hammer setting. We do not even know whether native support for higher-order logic outperforms encodings.

In this paper, we evaluate 17 modern provers (Section 3), both first- and higher-order. They support a wide range of formats from the TPTP [54] and SMT-LIB [2] families. We first generate problems using a tool called Mirabelle (Section 4), which invokes Sledgehammer on goals originating from 50 Isabelle proof developments. We use the MePo [42] relevance filter, which iteratively selects lemmas in a deterministic fashion based on the goal. To translate the problem, we use Sledgehammer's TPTP and SMT-LIB modules, targeting a wide range of provers, including SMT (satisfiability modulo theories) solvers. Once the problems are generated, we run the provers on them (Section 5). The evaluation yields a large collection of new benchmarks that can be used to tune automatic provers – much larger than Judgment Day, which currently consists of 1240 goals. Our collection of problems, called Seventeen, and the raw evaluation data are archived online.[1]

The evaluation is first and foremost an assessment of the provers and their features and of various encoding schemes. Our setup does not attempt to reconstruct proofs in Isabelle. To guard against incorrect proofs, we use only encodings known to be sound [12] and provers that have shown themselves to be trustworthy over the years. The question of how to reconstruct higher-order proofs in Isabelle is still partly open.

In addition, our evaluation provides raw data that can be used to fine-tune how Sledgehammer uses automatic provers. The evaluation also gives insights into which translation methods are useful in hammers, guiding the development of hammers.

Although it may be tempting to view this paper as a prover competition, there are important differences. In a competition, the developers have full control over how their provers are invoked, and there is a clear protocol for interacting with the organizers. We tried to pass meaningful command-line options and consulted prover developers, but there are no guarantees that we succeeded in finding the best options for all provers. Of course, the success rates we report are only lower bounds on what can be achieved.

[1] https://doi.org/10.5281/zenodo.5940084

Table 1 Output formats supported by Sledgehammer.

Format	Description
FOF	Untyped first-order logic
TF0	Many-sorted first-order logic
TX0	Many-sorted first-order logic with Booleans, if–then–else, and let
SMT2	Many-sorted first-order logic with Booleans, if–then–else, let, and linear arithmetic
TF1	Rank-1 polymorphic first-order logic
TX1	Rank-1 polymorphic first-order logic with Booleans, if–then–else, and let
TH0$^-$	Many-sorted higher-order logic
TH0$^+$	Many-sorted higher-order logic with Hilbert choice, if–then–else, and let
SMT3	Many-sorted higher-order logic with if–then–else, let, and linear arithmetic
TH1$^-$	Rank-1 polymorphic higher-order logic
TH1$^+$	Rank-1 polymorphic higher-order logic with Hilbert choice, if–then–else, and let

2 Sledgehammer

When we invoke Sledgehammer on a goal (i.e., a proof obligation, which might be provable or not), Sledgehammer's relevance filters and problem translation modules come into play. We briefly describe them below. The literature covers them in more detail.

2.1 The Relevance Filter

Given a goal and a cutoff number n, the relevance filter selects a subset of n lemmas among all the lemmas that are currently loaded (typically numbering in the thousands) and ranks them from 1 to n in order of decreasing expected relevance for proving the goal. Isabelle includes three relevance filters.

- MePo [42], named after *Me*ng and *P*aulso*n*, works iteratively starting from the goal. It is superficially similar to the SInE [31] algorithm implemented in several automatic provers (E, Vampire, Zipperposition) but was developed independently.
- MaSh [13], the *Ma*chine learner for *S*ledge*h*ammer, implements two fast machine learning algorithms: naive Bayes and k-nearest neighbors. It learns which lemmas are useful to reason about which symbols and ranks the lemmas accordingly.
- MeSh [13] is a combination of *Me*Po and Ma*Sh*.

While MaSh and MeSh give higher success rates than MePo [13], they are tricky to use properly in our complicated evaluation setup, with 50 different Isabelle developments. They also introduce nondeterminism. Since we are more interested in the relative performance of the provers than in the absolute success rates, we decided to use only MePo for this paper. The MaSh paper [13] has a detailed evaluation of the three filters.

MePo operates on a set of *known symbols*, initialized to consist of the symbols that occur in the goal. Roughly speaking, MePo works as follows:

1. Rank the (thousands of) available lemmas. The more symbols a lemma shares with the goal, and the fewer other symbols it contains, the higher its weight.
2. Select a number of lemmas based on their weights, removing them from the set of available lemmas. If no lemmas have suitably high weights, stop.

Table 2 Type encodings supported by Sledgehammer's TPTP module.

Encoding	Description
g	Polymorphism-preserving encoding based on type guards (predicates)
g?	Lightweight variant of g
g??	Featherweight variant of g
g@	Alternative polymorphism-preserving encoding based on type guards
t	Polymorphism-preserving encoding based on type tags (functions)
t?	Lightweight variant of t
t??	Featherweight variant of t
t@	Alternative polymorphism-preserving encoding based on type guards
$\tilde{\text{g}}$, $\tilde{\text{g}}$?, $\tilde{\text{g}}$??, $\tilde{\text{t}}$, $\tilde{\text{t}}$?, $\tilde{\text{t}}$??	Monomorphizing variants of g, g?, g??, t, t?, t??, respectively

3. Enlarge the set of known symbols to include all the symbols occurring in the newly selected lemmas.
4. If n lemmas have been selected, stop; otherwise, go to step 1.

A further refinement is that symbols are annotated with their types, to increase precision. For example, if the symbol nil of type *nat list* is known, lemmas containing nil of polymorphic type α *list* will be considered relevant, unlike lemmas containing nil of type *bool list*. (Type constructors are written in postfix notation.)

2.2 The TPTP Translation

The TPTP translation module [41] generates problems in TPTP formats FOF, TF0, TX0, TF1, TX1, TH0, and TH1. Moreover, we draw an unofficial distinction between TH0$^-$ and TH0$^+$ variants of TH0 and similarly for TH1. Table 1 presents a brief overview of all these formats, as well as the SMT-LIB formats described below. Note that all higher-order formats support interpreted Booleans. Moreover, although some of the TPTP formats provide interpreted arithmetic types and symbols, the TPTP translation module does not exploit this and maps Isabelle's arithmetic operators to uninterpreted symbols.

When translating Isabelle's logic to the automatic provers' possibly weaker logics, four types of constructs may need to be encoded: types, partial application, λ-abstractions, and Booleans. The encodings are naturally not used in formats that support the respective features; for example, TX0, TX1, TH0, and TH1 support interpreted Booleans, so there is no need to encode them.

- For the *types* and the type classes, some translation schemes encode Isabelle's entire rank-1 polymorphic type system using terms and clauses [12, 41]. An alternative that works particularly well in conjunction with the many-sorted formats TF0, TX0, and TH0 is to iteratively monomorphize the problem [12, Section 5.6]. Monomorphization is generally incomplete [18, Section 2]. Sledgehammer's sound type encodings are listed in Table 2 and described by Blanchette et al. [12].
- The *Boolean* constants False, True, the logical connectives, and the quantifiers are either mapped to their TPTP equivalents or translated to uninterpreted "proxy" symbols. Axioms are provided for reasoning about the proxies (e.g., False $\neq$ True).

- *Partial application* is encoded using a distinguished binary symbol app. For example, rev and rev xs are mapped to rev and app(rev, xs), respectively. As an optimization, if all occurrences of rev take at least one argument, it can be passed directly (e.g., rev(xs)).
- λ-*abstractions* are encoded using SKBCI combinators [57] or λ-lifting [33]. The generated problem may include axioms that define the introduced symbols or leave them *opaque*.

▶ **Example 1.** Consider the Isabelle symbol filter of type $(\alpha \Rightarrow \mathit{bool}) \Rightarrow \alpha\ \mathit{list} \Rightarrow \alpha\ \mathit{list}$ such that filter $p\ xs$ is defined as the sublist of xs that consists of the elements x for which $p\ x$ holds. The following lemma is part of Isabelle's list library:

$$\mathsf{filter}\ P\ (\mathsf{filter}\ Q\ xs) = \mathsf{filter}\ (\lambda x.\ Q\ x \wedge P\ x)\ xs$$

If the target is TF0, and both monomorphization and SKBCI are used, the encoding is

$$\mathsf{filter}_{nat}(P, \mathsf{filter}_{nat}(Q, xs)) = \mathsf{filter}_{nat}(\mathsf{S}_{nat,bool,bool}(\mathsf{B}_{nat,bool,bool\Rightarrow bool}(\mathsf{conj}, Q), P), xs)$$

where the subscripts identify monomorphic instances. In addition, axioms are included to characterize the combinators $\mathsf{S}_{nat,bool,bool}$ and $\mathsf{B}_{nat,bool,bool\Rightarrow bool}$ and the proxy conj. By contrast, if the target is TH1, the lemma is encoded as itself.

2.3 The SMT-LIB Translation

The SMT-LIB translation module [19, Chapter 2] was first developed as part of the *smt* proof method [21] and later added to Sledgehammer [11]. The SMT-LIB 2 standard roughly amounts to TPTP TX0 with arithmetic and other theories. The forthcoming SMT-LIB 3 is expected to support higher-order logic and polymorphism and thus be a superset of TPTP TH1. Both cvc5 and a fork of veriT support a preliminary version of the standard.

The SMT-LIB translation module monomorphizes polymorphic types. This is done even for SMT-LIB 3 output. In addition, for SMT-LIB 2, partial application is encoded using app, and λ-lifting is used. Isabelle's basic arithmetic operators on integers and reals are mapped to the corresponding SMT operators. We use the names SMT2 and SMT3 to refer to the fragments of SMT-LIB 2 and 3 used by Sledgehammer, respectively.

3 The Automatic Provers

For our evaluation, we selected 17 provers that support at least one of Sledgehammer's formats. The formats supported by each prover are listed in Table 3. All of the provers participated in the CASC [56] or the SMT-COMP [4] competition at some point. Except when mentioned otherwise, the provers were not specifically tuned for Sledgehammer problems.

- *agsyHOL* [39] is a higher-order prover based on an intuitionistic sequent calculus and a narrowing engine. It was developed to showcase the technology behind the Agsy [40] proof tool for Agda. We use version 1.0.
- *Beagle* [5] is a first-order prover based on hierarchic superposition, a calculus that generalizes standard superposition (which in turn generalizes resolution) with theory reasoning – in Beagle's case, linear arithmetic reasoning. The main developer, Peter Baumgartner, points out that the prover is stronger on problems that require arithmetic reasoning and that it usually performs worse than state-of-the-art superposition provers on problems without theories. We use version 0.9.50.
- *cocATP*[2] is a Coq-inspired higher-order automatic theorem prover based on the calculus of constructions without inductive constructions. We use version 0.2.0.

[2] http://www.tptp.org/CASC/J7/SystemDescriptions.html#cocATP---0.2.0

Table 3 Input formats supported by the provers.

Prover	FOF	TF0	TX0	SMT2	TF1	TX1	TH0$^-$	TH0$^+$	SMT3	TH1$^-$	TH1$^+$
agsyHOL							✓				
Beagle	✓	✓		✓							
cocATP							✓				
cvc5	✓	✓		✓			✓		✓		
E	✓	✓	✓				✓				
ENIGMA	✓										
iProver	✓	✓	✓	✓							
leanCoP	✓										
Leo-III	✓	✓	✓		✓	✓	✓	✓		✓	✓
Princess	✓	✓		✓							
Satallax							✓				
SPASS	✓										
Vampire	✓	✓	✓	✓	✓	✓	✓			✓	
veriT				✓							
Z3	✓	✓		✓							
Zenon	✓										
Zipperposition	✓	✓			✓		✓			✓	

- *cvc5* is a first-order SMT solver based on CDCL(T) with some support for higher-order logic [1]. It is CVC4's [3] successor. The developers provided us with a portfolio of configurations that are tuned for low timeouts. We use version 0.0.4.
- E [49] is a first-order prover based on the superposition calculus. An extension called Ehoh supports some higher-order constructs [60]. We use E 2.6 with Ehoh. This version can parse TH0 but does not yet implement higher-order unification. Note that Vukmirović is a developer of E.
- *ENIGMA* [34] is a variant of E that uses machine learning for determining the order in which clauses are processed by E, thereby steering the search space traversal. ENIGMA's models were trained on the TPTP library, which could give suboptimal performance on Sledgehammer benchmarks. We use version 0.5.1.
- *iProver* [37] is a first-order prover based on instantiation. Recently, a superposition module was added [29]. The developers prepared a version of the prover trained using HOL-ML [32] on Sledgehammer benchmarks – version post-3.5 (git revision 04c55471083). Compared with 3.5, this version adds an extended parser that supports TF0, TX0, and SMT2 benchmarks.
- *leanCoP* [44] is a first-order prover based on the clausal connection calculus, a goal-oriented refinement of the clausal tableau calculus. Implemented in a few lines of Prolog, it combines extreme minimalism with decent performance. We use version 2.2.
- *Leo-III* [51] is a higher-order prover based on the higher-order paramodulation calculus. Paramodulation is a variant of superposition. The provers E and CVC4 are invoked as "end-game" backends at intervals on a first-order encoding of the current clause set. The main developer, Alexander Steen, provided command-line options suited for our experiments, added support for TX0, TX1, TH0$^+$, and TH1$^+$, and fixed a few issues we discovered. We use version 1.6.6.

- *Princess* [48] is a first-order SMT solver based on a tableau calculus and with support for several arithmetic and nonarithmetic theories. We use version 2021-11-15 with command-line options provided by the developer.
- *Satallax* [24] is a higher-order prover based on a tableau calculus guided by a SAT solver. Ehoh is invoked at regular intervals as an "end-game" backend on an applicative first-order (or λ-free higher-order) encoding of the current clause set. We use version 3.5.
- *SPASS* [61] is a first-order prover based on superposition. We use version 3.9.
- *Vampire* [38] is a higher-order prover based on superposition and instantiation. Its superposition calculus uses SKBCI combinators to represent λ-abstractions [9]. We use version 4.6.1.sl,[3] a customized version provided by the developers, with command-line options also provided by them. This version uses Z3 as a backend.
- *veriT* [23] is a first-order SMT solver based on the CDCL(T) calculus. A prototype supports the SMT3 format [1], but we use the standard version 2021.06-rmx. We invoke veriT with command-line options that were derived by the developers using a custom tool [50, Section 5.1] based on Sledgehammer problems.
- *Z3* [28] is a first-order SMT solver based on the CDCL(T) calculus. We use version 4.8.12. The solver was not optimized for these benchmarks.
- *Zenon* [22] is a first-order prover based on a tableau calculus. We use version 0.7.1.
- *Zipperposition* [59] is a higher-order prover based on λ-superposition, a higher-order variant of the superposition calculus. The E prover is invoked as an "end-game" backend at regular intervals on an applicative first-order (or λ-free higher-order) encoding of the current clause set. We run it in a custom mode designed for low timeouts. We use version 2.1. Note that Vukmirović and Blanchette are developers of Zipperposition.

4 Mirabelle

Mirabelle is a testing and evaluation tool included with Isabelle/HOL since version 2010. It was initially developed Sascha Böhme, Blanchette, and other Isabelle developers as a Perl script and some Standard ML code. It was used for many evaluations of Sledgehammer and automatic provers [6–8, 11–13, 16, 17, 20, 45, 47, 50, 53, 60], starting with Judgment Day [20]. It was also used to generate TPTP, CASC, and SMT-LIB benchmarks.

For Isabelle version 2021-1, Mirabelle was reimplemented by Wenzel and Desharnais. The new tool is implemented as part of Isabelle/Scala and is properly integrated with modern Isabelle concepts such as sessions and parallel proof checking.

The new Mirabelle is a command-line tool that takes four arguments as input:

- *An Isabelle session*: A session is a collection of Isabelle theory files forming a project. For example, each entry of the *Archive of Formal Proofs* (AFP) [14] constitutes a session.
- *A theory filter*: The filter selects a subset of the theory files from the session for further processing. For example, a theory filter may keep all theory files or only a specified one.
- *A goal filter*: The filter selects a subset of the goals within the selected theory files for further processing. For example, a goal filter may keep all goals or only the first n goals from the selected theories.
- *A list of actions*: Each action is applied to each selected goal and produces a final report.

[3] `https://github.com/vprover/vampire/releases/tag/v4.6.1.sl`

Mirabelle runs the specified session using Isabelle, collecting all selected goals. At the end, it applies the actions to the collected goals, using parallelism. Each action application produces some output, which can contain more details than the final reports. A Mirabelle *goal* consists of the Isabelle goals before and after a proof step, the type of proof step (e.g., one-line **by** proof), the theory file, and the goal's position in the file. A Mirabelle *action* consists of two functions: run performs the action, and finalize produces the final report.

Mirabelle includes a number of predefined actions. The one we use is Sledgehammer. It supports the same options as the **sledgehammer** command as well as some Mirabelle-specific ones. These can be specified on the command line. The action can either run the automatic provers or only generate the TPTP or SMT-LIB files without running the provers. It is this latter mode that we use for our evaluation. Other Mirabelle actions include *arith*, *metis* [46], and Quickcheck [26]. Users can register custom actions.

▶ **Example 2.** The following command launches Mirabelle on the `Compiler.thy` file from the `VeriComp` session and runs E for 30 seconds on the first 10 goals:

```
isabelle mirabelle -d '$AFP' -O output -T Compiler -m 10 \
  -A "sledgehammer[provers=e,timeout=30]" VeriComp
```

The Sledgehammer invocation on the first goal is identified by the label `0.sledgehammer goal.by VeriComp.Compiler 61:1935`. The associated action output is

```
succeeded (26348+464) [e]:
Try this: using L1.prog_behaves_def by auto (16 ms)
none (sledgehammer): succeeded (0)
```

The second line gives an Isabelle proof that reconstructs the automatic prover's proof in Isabelle and the time needed for reconstruction.

5 Evaluation

For our evaluation, we employed Mirabelle to generate problems from 5000 goals originating from 50 randomly selected entries of the AFP (100 goals per entry). Our objective was to include goals from various areas of mathematics and computer science representing diverse formalization styles and using various features. Within an entry, the goals were selected at regular intervals, alleviating the issue that consecutive goals tend to be similar. We used the repository revisions b87fcf474e7f of Isabelle (15 January 2022) and e2ae9549a7b0 of the AFP (19 December 2021). The experiments were run on the StarExec Iowa cluster [52] with a CPU timeout of 30 s per problem.

5.1 Base Configuration

Table 4 presents the number of proved problems (out of 5000) for the 17 provers and the different supported input formats. In this and later tables, the maximum of each row is shown in bold, and the maximum of each column is shown in italics.

The problems were generated using a *base configuration* of Sledgehammer, which sets the following options to provide a reasonable baseline for the experiments:

- The *fact_filter* option, which specifies the relevance filter, is set to MePo.
- The *max_facts* option, which controls the target number of Isabelle facts (definitions, lemmas, etc.) to include in generated problems, is set to 512 – a large number chosen to stress the provers.

Table 4 Proved problems for each prover and each input format, using the base configuration.

	FOF	TF0	TX0	SMT2	TF1	TX1	TH0$^-$	TH0$^+$	SMT3	TH1$^-$	TH1$^+$
agsyHOL							**814**				
Beagle	901	**1064**		853							
cocATP							**584**				
cvc5	*2619*	***2779***		*2631*			2522		*2557*		
E	2394	2456	2534				**2557**				
ENIGMA	**2301**										
iProver	2291	**2418**	2211	2144							
leanCoP	**1487**										
Leo-III	1938	**2136**	2084		503	459	1632	*1539*		608	*447*
Princess	1205	**1353**		1260							
Satallax							**1695**				
SPASS	**1550**										
Vampire	2495	**2608**	*2587*	2204	*2471*	*2212*	2179			*1814*	
veriT				**2463**							
Z3	2226	2249		**2252**							
Zenon	**812**										
Zipperposition	2551	2607			1702		***2718***			1674	

- The *induction_rules* option, which controls the translation of induction rules, is set so that these rules are excluded from generated problems.
- The *uncurried_aliases* option, which controls the translation of curried function applications in the TPTP module, is set to "false."
- The *lam_trans* option, which controls the translation scheme for λ-abstractions in the TPTP module, is set to use λ-lifting with defining equations for first-order formats and to keep the λ's as is for higher-order formats.
- The *type_enc* option, which controls the translation scheme for types in the TPTP module, is set as follows: For polymorphic formats, keep the types as is. For monomorphic formats, monomorphize the types. For FOF, use Sledgehammer's efficient `g??` encoding [12].
- The *max_mono_iters* and *max_new_mono_instances* options, which limit the number of axiom instances generated by monomorphization, are set to 3 and 100, respectively.

The other Sledgehammer options are left with their default values [10]. In the next subsections, we will deviate from this baseline to study the effect of various options.

In Table 4, we see that the best prover in the base configuration is cvc5. Remarkably, this success is achieved with the TF0 format and not with the arithmetic-capable SMT2. The situation is similar for iProver, Princess, and Vampire, but not Z3.

Intuitively, we would also expect support for higher-order logic to be beneficial, but this is not always true. E and Zipperposition perform better with the higher-order TH0$^-$ format than with the first-order TF0, but for cvc5, Leo-III, and Vampire the opposite is true. Similar effects are visible with formats supporting if–then–else and let (TX0, TX1, TH0$^+$, TH1$^+$).

Is higher-order logic useful at all? It would appear so. The data underlying Table 4 reveals that among the 3321 goals that are collectively proved by all provers and formats, 146 goals are proved with higher-order formats but not with first-order formats.

Table 5 Running time of each prover's best input format.

Prover	Format	Proved problems	Percentile 25	50	75	90	95	99	100
agsyHOL	TH0$^-$	814	0.5	1.5	4.2	9.0	15.3	24.9	29.5
Beagle	TF0	1064	5.5	7.0	9.6	12.3	13.7	15.4	16.8
cocATP	TH0$^-$	584	2.4	4.9	11.1	20.1	24.6	28.4	30.1
cvc5	TF0	2779	0.2	0.3	0.5	2.1	7.6	24.4	30.0
E	TH0$^-$	2557	0.2	0.4	1.1	9.5	13.5	23.6	29.2
ENIGMA	FOF	2301	2.7	2.9	3.4	4.5	6.2	9.0	15.1
iProver	TF0	2418	1.3	1.3	4.3	7.8	12.8	25.9	29.9
leanCoP	FOF	1487	1.4	1.4	3.5	11.6	12.7	26.3	29.2
Leo-III	TF0	2136	5.5	6.5	7.6	8.7	9.2	10.3	12.0
Princess	TF0	1353	6.8	7.6	8.5	9.2	9.5	9.8	11.0
Satallax	TH0$^-$	1695	2.6	5.8	12.7	20.0	21.1	24.4	28.6
SPASS	FOF	1550	1.7	3.7	10.0	18.1	22.9	28.8	29.7
Vampire	TF0	2608	0.2	0.3	5.3	11.0	12.8	23.0	29.4
veriT	SMT2	2463	0.1	0.1	0.2	1.1	4.6	19.6	29.2
Z3	SMT2	2252	0.2	0.2	0.3	0.6	2.3	17.4	29.8
Zenon	FOF	812	0.6	1.6	3.2	6.3	11.4	24.7	29.0
Zipperposition	TH0$^-$	2718	0.9	2.1	4.4	6.9	10.2	23.1	30.0

Finally, we notice that for all provers that support both TF0 and TF1, the monomorphized TF0 results are better than the TF1 results, and similarly for TX0 vs. TX1, for TH0$^-$ vs. TH1$^-$, and for TH0$^+$ vs. TH1$^+$.

5.2 Running Time

How quickly do the provers find the proofs? Table 5 answers this question for each prover, focusing on the most successful input format for the prover according to Table 4. The median, or 50th percentile, is shown, as well as other percentiles. (Percentiles are calculated using linear interpolation.) For example, the number 12.3 in the Beagle row indicates that 90% of the 1064 problems proved by Beagle are proved within 12.3 s. As in Table 4, the baseline configuration is used. Recall that the timeout is 30 s.

Table 5 shows that if a prover finds a proof, it will likely find it quickly. Some provers are dazzlingly fast; the median time it takes to find a proof is 0.1 s for veriT, 0.2 s for Z3, and 0.3 s for cvc5 and Vampire. Remarkably, Princess proves its 1353 problems within 11 s; the remaining 19 s are unnecessary.

5.3 Number of Facts

An important Sledgehammer option is *num_facts*, which governs the number of facts to be selected by the relevance filter. These facts are included as axioms in the generated problems. The base configuration specifies 512 facts, but Table 6 shows what happens when we change this option (and keep the other options as in the base configuration). Figure 1 depicts the same information graphically for six provers, using a logarithmic x-axis.

The stronger provers tend to peak with more facts than the weaker ones. For example, the strongest prover, cvc5, peaks at 512 facts. Some of the stronger provers (E, Vampire, Zipperposition) implement the SInE [31] algorithm; others simply seem to scale well in the presence of extraneous axioms.

Table 6 Proved problems for each prover's best input format and different numbers of facts.

	Format	16	32	64	128	256	512	1024
agsyHOL	TH0$^-$	1106	**1199**	1128	1006	916	814	610
Beagle	TF0	1256	1544	**1672**	1611	1419	1064	526
cocATP	TH0$^-$	745	830	**837**	793	706	584	440
cvc5	TF0	1309	1704	2117	2496	*2736*	***2779***	2669
E	TH0$^-$	1346	1736	2116	2438	**2590**	2557	2368
ENIGMA	FOF	1297	1670	2004	2246	**2347**	2301	2137
iProver	TF0	1294	1667	2009	2300	**2431**	2418	2303
leanCoP	FOF	1150	1376	1510	**1581**	1562	1487	1356
Leo-III	TF0	1305	1680	2043	2333	**2385**	2136	1344
Princess	TF0	1230	1542	**1734**	1655	1636	1353	172
Satallax	TH0$^-$	1233	1459	1596	1702	**1725**	1695	1632
SPASS	FOF	1273	1578	1778	**1843**	1787	1550	973
Vampire	TF0	1303	1691	2049	2379	2569	**2608**	2576
veriT	SMT2	1312	1683	2020	2307	2446	**2463**	2382
Z3	SMT2	1314	1668	1964	2207	**2310**	2252	2150
Zenon	FOF	955	1048	**1067**	1009	899	812	712
Zipperposition	TH0$^-$	*1376*	*1778*	*2162*	*2504*	2699	**2718**	*2680*

5.4 Encoding of Lambdas

Not all provers support λ-abstractions. How practical are the encodings of λ's implemented in Sledgehammer, and is native support for λ's preferable when available? Table 7, which includes the nine provers that support TF0, presents a very fragmented picture: Some provers prefer λ-lifting ("lift") to SKBCI combinators ("combs"), while others prefer SKBCI combinators or a combination of λ-lifting and combinators. Some prefer opaque definitions, while others work better when the defining equations are present.

Particularly striking is that only two higher-order provers, E and Zipperposition, work best with TH0$^-$ and native λ's, whereas the other three prefer TF0 encodings. This is disappointing; we would have hoped that native higher-order support in a prover pays off.

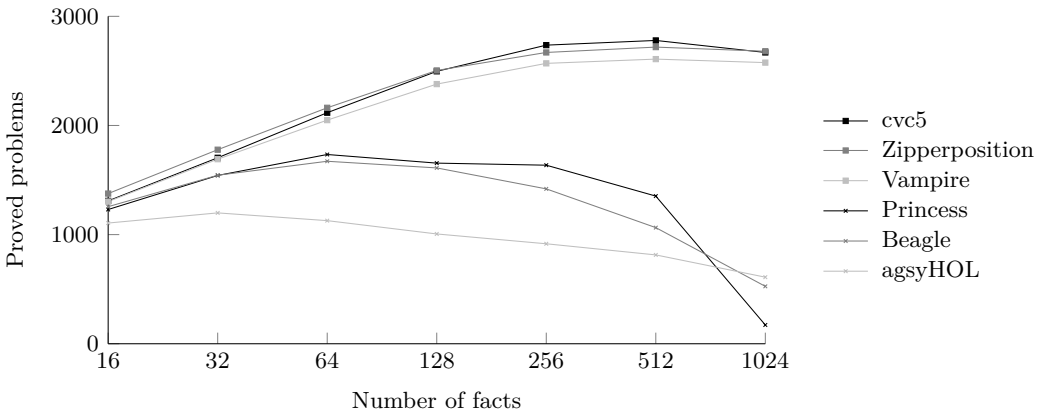

Figure 1 Proved problems for the top provers' best input formats and different numbers of facts.

Table 7 Proved problems by prover for the different encodings of λ-abstractions.

	TF0					TH0⁻
Prover	Lift	Opaque lift	Combs	Opaque combs	Lift & combs	Native
Beagle	1064	**1069**	998	1026	930	
cvc5	*2779*	*2628*	*2838*	*2793*	***2859***	2522
E	2456	2331	2471	2444	2434	**2557**
iProver	2418	2320	2423	**2430**	2388	
Leo-III	2136	2093	2133	**2152**	2061	1632
Princess	1353	**1369**	1319	**1369**	1119	
Vampire	2608	2461	2602	**2610**	2579	2179
Z3	2249	2154	**2326**	2299	2325	
Zipperposition	2607	2118	2635	2590	2642	***2718***

Table 8 Proved problems by prover for the different polymorphism-preserving encodings.

	FOF								TF1
Prover	g	g?	g??	g@	t	t?	t??	t@	native
Beagle	446	560	637	514	544	**650**	640	487	
cvc5	1660	*2289*	*2552*	1936	*2092*	1989	***2654***	*2216*	
E	1451	2080	**2195**	1646	1800	1820	2162	1390	
ENIGMA	1366	1974	**2063**	1531	1570	1599	2030	1299	
iProver	1587	2016	**2178**	1756	2087	*1992*	2167	1368	
leanCoP	1053	1276	1455	1212	**1565**	1356	1451	763	
Leo-III	871	1075	1504	1076	829	890	**1521**	1252	503
Princess	47	344	**835**	136	214	416	798	454	
SPASS	755	1032	**1194**	935	941	859	1168	784	
Vampire	*1918*	2172	2312	*2032*	1673	1975	2294	1528	***2471***
Z3	1472	2054	2238	1677	1765	1692	**2254**	1844	
Zenon	606	**634**	626	605	424	564	628	403	
Zipperposition	1689	1907	2042	1660	1941	1951	**2267**	1631	1702

For Leo-III and Satallax, which rely on backends, there is a potential explanation: The backends can work directly with TF0 problems but not with TH0. For Vampire, a possible explanation is that the higher-order version of the prover was not as extensively tuned as the first-order version.

5.5 Encoding of Types

Some provers do not support types at all, or they support only monomorphic types. How practical are the type encodings implemented in Sledgehammer, and is native support for types preferable when available? Table 8, which includes the 13 provers that support FOF, presents the results for the *polymorphism-preserving* encodings – i.e., encodings that encode the rank-1 polymorphic type system in its full generality. In contrast, Table 9 presents the results for the *monomorphizing* encodings – i.e., encodings that first heuristically instantiate type variables to eliminate polymorphism. The last column of each table offers a comparison with native TF1 or TF0.

Table 9 Proved problems by prover for the different monomorphizing encodings.

	FOF						TF0
Prover	g̃	g̃?	g̃??	t̃	t̃?	t̃??	native
Beagle	473	842	901	544	887	880	**1064**
cvc5	2033	*2479*	*2619*	*2092*	*2525*	*2687*	***2779***
E	2133	2324	2394	1800	2303	2371	**2456**
ENIGMA	2065	2239	**2301**	1570	2220	2283	
iProver	1963	2171	2291	2087	2285	2278	**2418**
leanCoP	1136	1367	1487	**1565**	1533	1460	
Leo-III	1236	1744	1938	829	1814	1983	**2136**
Princess	159	923	1205	214	1010	1188	**1353**
SPASS	1318	1396	**1550**	941	1329	1533	
Vampire	2302	2426	2495	1673	2385	2484	**2608**
Z3	2127	2200	2226	1765	2048	2223	**2249**
Zenon	812	**818**	812	424	745	788	
Zipperposition	*2360*	2289	2551	1941	2455	2522	**2607**

Our findings are similar to those of Blanchette et al. [12]: Despite being incomplete, monomorphizing encodings outperform the polymorphism-preserving encodings. Overall, the best encodings are the monomorphizing g̃?? and t̃??. But an even better option is to monomorphize the problem and use the many-sorted TF0 format directly. We also see that Leo-III's and Zipperposition's native polymorphism underperforms compared with the polymorphic-preserving FOF encodings. The reason is probably that these provers rely on monomorphic backends. For example, Zipperposition disables its E backend when invoked on a polymorphic problem.

5.6 Portfolio

So far, we have evaluated only the performance of provers in isolation. What happens if we combine them? It turns out that a virtual portfolio consisting of all the provers in all the configurations of Tables 4 to 9, each invoked for 30 s, would prove 3508 goals. This corresponds to a success rate of 70.2%, which is clearly lower than the 76.7% figure obtained in the MaSh paper [13] on the Judgment Day benchmarks. Two possible explanations suggest themselves: The MaSh paper' evaluation used the machine learning filters MaSh and MeSh, which are stronger than MePo, and as observed elsewhere [14, Section 6] Judgment Day might consist of relatively easy goals.

Such a virtual portfolio is not very realistic, but it does give an idea of the state of the art in automated reasoning. A more realistic alternative is to consider the *greedy sequence* of length n. The sequence is obtained by iteratively (1) taking first the (or some) best prover configuration for the goals of interest, and (2) removing all goals proved by this configuration. These two steps are repeated n times, yielding n configurations. The greedy sequence is not necessarily optimal, but it can be computed efficiently.

Table 10 presents the greedy sequence of length 16 based on our experiments. Given 480 s – or 30 s and 16 threads – we can solve 3440 goals (68.8%). This shows how complementary the different provers and options are.

Table 10 A greedy sequence of provers and configurations.

Prover	Format	Configuration	Proved problems
cvc5	TF0	lift & combs	2859
Zipperposition	TH0$^-$	1024 facts	+230
Vampire	TF1	base	+77
veriT	SMT2	1024 facts	+60
E	TX0	base	+51
cvc5	TF0	128 facts	+38
Zipperposition	TH0$^-$	256 facts	+24
Beagle	SMT2	base	+20
cvc5	FOF	t??	+17
Vampire	TF0	1024 facts	+13
Z3	SMT2	1024 facts	+11
E	TH0$^-$	32 facts	+11
Vampire	TH0$^-$	base	+9
E	TH0$^-$	1024 facts	+9
Z3	SMT2	base	+6
Zipperposition	TH0$^-$	base	+5
Total			3440

6 Related Work

The various evaluations of Sledgehammer, cited at the beginning of Section 4, surely constitute the most closely related work. Among these, Böhme and Nipkow's Judgment Day study [20] stands out as the most comprehensive; it considers the following aspects: success rate, running time, proof complexity, and proof minimization. But it is over a decade old. Most of the other evaluations focus on a single new Sledgehammer feature and how it improves the success rate; for example, Sultana et al. [53] evaluate encodings of higher-order features (cf. Section 5.4), and Blanchette et al. [12] evaluate type encodings (cf. Section 5.5).

Similar evaluations for other hammers or hammer-like systems include a three-prover evaluation by Kaliszyk and Urban [36], called MizAR 40, using MizAR, a three-prover evaluation by Czajka and Kaliszyk [27] using CoqHammer, an eight-prover evaluation by Filliâtre [30] using Why3, and a 19-prover evaluation by Brown et al. [25], called GRUNGE, using a custom exporter from HOL4. Among these, GRUNGE is the most similar to our effort, as it includes a large number of provers and exploits support for higher-order logic and polymorphism where available. But there are also several important differences:

- *The benchmarks presented in the GRUNGE paper include only the set of lemmas needed for a proof in HOL4, whereas our benchmarks contain heuristically selected lemmas from Isabelle's libraries.* Including only a typically small set of lemmas makes the benchmarks easier to prove than selecting hundreds of lemmas heuristically. It is less representative of the typical hammer use case, where a proof is not available.
- *The GRUNGE benchmarks correspond to top-level lemmas or theorems in HOL4, whereas our benchmarks correspond to Isabelle goals or subgoals.* Proving a lemma is generally more difficult than proving a goal.
- *GRUNGE's encoding of polymorphism is simple but inefficient, whereas our evaluation relies on state-of-the-art monomorphization.* One of GRUNGE's two translation schemes [25, Section 4.3] corresponds to Sledgehammer's `t` encoding, whose performance was found to be very suboptimal in Section 5.5.

- *GRUNGE uses different provers to our evaluation.* In particular, GRUNGE does not generate SMT-LIB problems, nor does it exploit the recently added support for higher-order logic in cvc5, E, and Vampire. It also misses out on the improvements to Zipperposition over the past three years, which led it to win the higher-order division of the 2020 and 2021 editions of CASC [55, 56].

The top five provers on the GRUNGE benchmarks, with the number of proved problems out of 12 140, are Leo-III with 7090, Vampire with 5929, CVC4 with 5709, E with 5118, and HOLyHammer with 5059. The top prover is only the ninth best according to our Table 4. The discrepancy is easy to explain: For GRUNGE, Leo-III was, next to Zipperposition, the only system that fully supported polymorphic higher-order logic, and all the other systems saw versions of the benchmarks encoded using encodings that were not primarily tuned toward performance of particular provers, whereas for our evaluation we used state-of-the-art encodings of polymorphic types and higher-order features.

As a benchmark suite, Seventeen can be used as a complement to existing libraries, such as the TPTP [54], which includes both first- and higher-order problems in the TPTP formats, and SMT-LIB [2], which consists of first-order problems in SMT-LIB 2 syntax. Various collections of hammer-generated problems exist, such as GRUNGE, Judgment Day, and Miz$\mathbb{AR}$ 40. All these libraries are not necessarily mutually exclusive; for example, many Miz$\mathbb{AR}$- and Sledgehammer-generated problems are in the TPTP, and Sledgehammer-generated problems [47, Section 5] are also part of SMT-LIB.

7 Conclusion

In this paper, we evaluated 17 modern automatic provers, including SMT solvers and higher-order provers, thereby superseding the aging Judgment Day study; and we evaluated several aspects of a hammer, such as the encodings of types and λ-abstractions. In particular, we wanted to determine whether native implementations of various features perform better than encodings, which is a reasonable thing to hope for. We have now updated Sledgehammer's default setup to get the best out of the provers.

Specifically, we found that native support for monomorphic types was beneficial, but the current support for polymorphism is disappointing. Native support for features such as linear arithmetic and higher-order logic helps some provers and is detrimental to others. Overall, we see that simply implementing a feature in a prover is often not enough; to be useful, the feature must be fined-tuned based on benchmarks. Some provers seem to misbehave on Sledgehammer benchmarks, which indeed may look quite different from the problems used by the provers' developers to tune their systems. Since hammers are among the most useful applications of automatic provers, we contend that it is worthwhile to tune provers on hammer-generated benchmarks as a complement to the TPTP and SMT-LIB.

References

1 Haniel Barbosa, Andrew Reynolds, Daniel El Ouraoui, Cesare Tinelli, and Clark W. Barrett. Extending SMT solvers to higher-order logic. In Pascal Fontaine, editor, *CADE-27*, volume 11716 of *LNCS*, pages 35–54. Springer, 2019.

2 Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017. URL: `https://www.SMT-LIB.org/`.

3 Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV 2011*, volume 6806 of *LNCS*, pages 171–177. Springer, 2011.

4 Clark W. Barrett, Leonardo de Moura, and Aaron Stump. SMT-COMP: satisfiability modulo theories competition. In Kousha Etessami and Sriram K. Rajamani, editors, *CAV 2005*, volume 3576 of *LNCS*, pages 20–23. Springer, 2005.

5 Peter Baumgartner, Joshua Bax, and Uwe Waldmann. Beagle – A hierarchic superposition theorem prover. In Amy P. Felty and Aart Middeldorp, editors, *CADE-25*, volume 9195 of *LNCS*, pages 367–377. Springer, 2015.

6 Alexander Bentkamp, Jasmin Blanchette, Simon Cruanes, and Uwe Waldmann. Superposition for lambda-free higher-order logic. *Log. Meth. Comput. Sci.*, 17(2):1:1–1:38, 2021.

7 Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, and Petar Vukmirović. Superposition for full higher-order logic. In André Platzer and Geoff Sutcliffe, editors, *CADE-28*, volume 12699 of *LNCS*, pages 396–412. Springer, 2021.

8 Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, Petar Vukmirovic, and Uwe Waldmann. Superposition with lambdas. *J. Autom. Reason.*, 65(7):893–940, 2021.

9 Ahmed Bhayat and Giles Reger. A combinator-based superposition calculus for higher-order logic. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *IJCAR 2020, Part I*, volume 12166 of *LNCS*, pages 278–296. Springer, 2020.

10 Jasmin Blanchette. Hammering away: A user's guide to Sledgehammer for Isabelle/HOL, 2021. URL: `https://isabelle.in.tum.de/website-Isabelle2021-1/dist/Isabelle2021-1/doc/sledgehammer.pdf`.

11 Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending Sledgehammer with SMT solvers. *J. Autom. Reason.*, 51(1):109–128, 2013.

12 Jasmin Christian Blanchette, Sascha Böhme, Andrei Popescu, and Nicholas Smallbone. Encoding monomorphic and polymorphic types. *Log. Meth. Comput. Sci.*, 12(4), 2016.

13 Jasmin Christian Blanchette, David Greenaway, Cezary Kaliszyk, Daniel Kühlwein, and Josef Urban. A learning-based fact selector for Isabelle/HOL. *J. Autom. Reason.*, 57(3):219–244, 2016.

14 Jasmin Christian Blanchette, Maximilian Haslbeck, Daniel Matichuk, and Tobias Nipkow. Mining the archive of formal proofs. In Manfred Kerber, editor, *CICM 2015*, volume 9150 of *LNCS*, pages 1–15. Springer, 2015.

15 Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *J. Formaliz. Reason.*, 9(1):101–148, 2016.

16 Jasmin Christian Blanchette, Nicolas Peltier, and Simon Robillard. Superposition with datatypes and codatatypes. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *IJCAR 2018*, volume 10900 of *LNCS*, pages 370–387. Springer, 2018.

17 Jasmin Christian Blanchette, Andrei Popescu, Daniel Wand, and Christoph Weidenbach. More SPASS with Isabelle: Superposition with hard sorts and configurable simplification. In Lennart Beringer and Amy Felty, editors, *ITP 2012*, volume 7406 of *LNCS*, pages 345–360. Springer, 2012.

18 François Bobot and Andrei Paskevich. Expressing polymorphic types in a many-sorted language. In Cesare Tinelli and Viorica Sofronie-Stokkermans, editors, *FroCoS 2011*, volume 6989 of *LNCS*, pages 87–102. Springer, 2011.

19 Sascha Böhme. *Proving Theorems of Higher-Order Logic with SMT Solvers*. PhD thesis, Technische Universität München, 2012.

20 Sascha Böhme and Tobias Nipkow. Sledgehammer: Judgement Day. In Jürgen Giesl and Reiner Hähnle, editors, *IJCAR 2010*, volume 6173 of *LNCS*, pages 107–121. Springer, 2010.

21 Sascha Böhme and Tjark Weber. Fast LCF-style proof reconstruction for Z3. In Matt Kaufmann and Lawrence C. Paulson, editors, *ITP 2010*, volume 6172 of *LNCS*, pages 179–194. Springer, 2010.

22 Richard Bonichon, David Delahaye, and Damien Doligez. Zenon: An extensible automated theorem prover producing checkable proofs. In Nachum Dershowitz and Andrei Voronkov, editors, *LPAR 2007*, volume 4790 of *LNCS*, pages 151–165. Springer, 2007.

23 Thomas Bouton, Diego Caminha Barbosa De Oliveira, David Déharbe, and Pascal Fontaine. veriT: An open, trustable and efficient SMT-solver. In Renate A. Schmidt, editor, *CADE-22*, volume 5663 of *LNCS*, pages 151–156. Springer, 2009.

24 Chad E. Brown. Satallax: An automatic higher-order prover. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR 2012*, volume 7364 of *LNCS*, pages 111–117. Springer, 2012.

25 Chad E. Brown, Thibault Gauthier, Cezary Kaliszyk, Geoff Sutcliffe, and Josef Urban. GRUNGE: A grand unified ATP challenge. In Pascal Fontaine, editor, *CADE-27*, volume 11716 of *LNCS*, pages 123–141. Springer, 2019.

26 Lukas Bulwahn. The new Quickcheck for Isabelle – Random, exhaustive and symbolic testing under one roof. In Chris Hawblitzel and Dale Miller, editors, *CPP 2012*, volume 7679 of *LNCS*, pages 92–108. Springer, 2012.

27 Łukasz Czajka and Cezary Kaliszyk. Hammer for Coq: Automation for dependent type theory, 2018.

28 Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS 2008*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

29 André Duarte and Konstantin Korovin. Implementing superposition in iProver (system description). In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *IJCAR 2020, Part II*, volume 12167 of *LNCS*, pages 388–397. Springer, 2020.

30 Jean-Christophe Filliâtre. One logic to use them all. In Maria Paola Bonacina, editor, *CADE-24*, volume 7898 of *LNCS*, pages 1–20. Springer, 2013.

31 Krystof Hoder and Andrei Voronkov. Sine qua non for large theory reasoning. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE-23*, volume 6803 of *LNCS*, pages 299–314. Springer, 2011.

32 Edvard K. Holden and Konstantin Korovin. Heterogeneous heuristic optimisation and scheduling for first-order theorem proving. In Fairouz Kamareddine and Claudio Sacerdoti Coen, editors, *CICM 2021*, volume 12833 of *LNCS*, pages 107–123. Springer, 2021.

33 R. J. M. Hughes. Super-combinators: A new implementation method for applicative languages. In *LFP 1982*, pages 1–10. ACM Press, 1982.

34 Jan Jakubův and Josef Urban. ENIGMA: efficient learning-based inference guiding machine. In Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe, and Olaf Teschke, editors, *CICM 2017*, volume 10383 of *LNCS*, pages 292–302. Springer, 2017.

35 Cezary Kaliszyk and Josef Urban. HOL(y)Hammer: Online ATP service for HOL Light. *Math. Comput. Sci.*, 9(1):5–22, 2015.

36 Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *J. Autom. Reason.*, 55(3):245–256, 2015.

37 Konstantin Korovin. iProver – An instantiation-based theorem prover for first-order logic (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR 2008*, volume 5195 of *LNCS*, pages 292–298. Springer, 2008.

38 Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV 2013*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.

39 Fredrik Lindblad. A focused sequent calculus for higher-order logic. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *IJCAR 2014*, volume 8562 of *LNCS*, pages 61–75. Springer, 2014.

40 Fredrik Lindblad and Marcin Benke. A tool for automated theorem proving in Agda. In Jean-Christophe Filliâtre, Christine Paulin-Mohring, and Benjamin Werner, editors, *TYPES 2004*, volume 3839 of *LNCS*, pages 154–169. Springer, 2004.

41 Jia Meng and Lawrence C. Paulson. Translating higher-order clauses to first-order clauses. *J. Autom. Reason.*, 40(1):35–60, 2008.

42 Jia Meng and Lawrence C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. *J. Applied Logic*, 7(1):41–57, 2009.

43 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

44 Jens Otten. leanCoP 2.0 and ileanCoP 1.2: High performance lean theorem proving in classical and intuitionistic logic (system descriptions). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR 2008*, volume 5195 of *LNCS*, pages 283–291. Springer, 2008.

45 Lawrence C. Paulson and Jasmin Christian Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska, editors, *IWIL-2010*, volume 2 of *EPiC*, pages 1–11. EasyChair, 2012.

46 Lawrence C. Paulson and Kong Woei Susanto. Source-level proof reconstruction for interactive theorem proving. In Klaus Schneider and Jens Brandt, editors, *TPHOLs 2007*, volume 4732 of *LNCS*, pages 232–245. Springer, 2007.

47 Andrew Reynolds and Jasmin Christian Blanchette. A decision procedure for (co)datatypes in SMT solvers. *J. Autom. Reason.*, 58(3):341–362, 2017.

48 Philipp Rümmer. A constraint sequent calculus for first-order logic with linear integer arithmetic. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *LPAR 2008*, volume 5330 of *LNCS*, pages 274–289. Springer, 2008.

49 Stephan Schulz, Simon Cruanes, and Petar Vukmirovic. Faster, higher, stronger: E 2.3. In Pascal Fontaine, editor, *CADE-27*, volume 11716 of *LNCS*, pages 495–507. Springer, 2019.

50 Hans-Jörg Schurr, Mathias Fleury, and Martin Desharnais. Reliable reconstruction of fine-grained proofs in a proof assistant. In André Platzer and Geoff Sutcliffe, editors, *CADE-28*, volume 12699 of *LNCS*, pages 450–467. Springer, 2021.

51 Alexander Steen and Christoph Benzmüller. Extensional higher-order paramodulation in Leo-III. *J. Autom. Reason.*, 65(6):775–807, 2021.

52 Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. StarExec: A cross-community infrastructure for logic solving. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *IJCAR 2014*, volume 8562 of *LNCS*, pages 367–373. Springer, 2014.

53 Nik Sultana, Jasmin Christian Blanchette, and Lawrence C. Paulson. LEO-II and Satallax on the Sledgehammer test bench. *J. Applied Logic*, 11(1):91–102, 2013.

54 Geoff Sutcliffe. The TPTP problem library and associated infrastructure – From CNF to TH0, TPTP v6.4.0. *J. Autom. Reason.*, 59(4):483–502, 2017.

55 Geoff Sutcliffe. The 10th IJCAR automated theorem proving system competition – CASC-J10. *AI Commun.*, 34(2):163–177, 2021.

56 Geoff Sutcliffe and Martin Desharnais. The CADE-28 automated theorem proving system competition – CASC-28. *AI Communications*, 34(4):259–276, 2022.

57 David A. Turner. A new implementation technique for applicative languages. *Softw. Pract. Exper.*, 9:31–49, 1979.

58 Josef Urban, Piotr Rudnicki, and Geoff Sutcliffe. ATP and presentation service for Mizar formalizations. *J. Autom. Reason.*, 50(2):229–241, 2013.

59 Petar Vukmirović, Alexander Bentkamp, Jasmin Blanchette, Simon Cruanes, Visa Nummelin, and Sophie Tourret. Making higher-order superposition work. In André Platzer and Geoff Sutcliffe, editors, *CADE-28*, volume 12699 of *LNCS*, pages 415–432. Springer, 2021.

60 Petar Vukmirović, Jasmin Christian Blanchette, Simon Cruanes, and Stephan Schulz. Extending a brainiac prover to lambda-free higher-order logic. In Tomas Vojnar and Lijun Zhang, editors, *TACAS 2019*, volume 11427 of *LNCS*, pages 192–210. Springer, 2019.

61 Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS version 3.5. In Renate A. Schmidt, editor, *CADE-22*, volume 5663 of *LNCS*, pages 140–145. Springer, 2009.