Three Years of Experience with Sledgehammer, a Practical Link between Automatic and Interactive Theorem Provers

Lawrence C. Paulson
Computer Laboratory
University of Cambridge, U.K.
lp15@cam.ac.uk

Abstract

Sledgehammer is a highly successful subsystem of Isabelle/HOL that calls automatic theorem provers to assist with interactive proof construction. It requires no user configuration: it can be invoked with a single mouse gesture at any point in a proof. It automatically finds relevant lemmas from all those currently available. An unusual aspect of its architecture is its use of unsound translations, coupled with its delivery of results as Isabelle/HOL proof scripts: its output cannot be trusted, but it does not need to be trusted. Sledgehammer works well with Isar structured proofs and allows beginners to prove challenging theorems.

1 Introduction

Interactive theorem provers are widely used by researchers for modelling complex algorithms or systems using logic. They typically support rich formalisms that include recursive definitions of functions and types; it may even be possible to reason about a partial recursive function's domain of definition or to define a relation co-inductively. The greatest weakness of these tools is the actual theorem proving, which is extremely laborious; perhaps they should instead be called specification editors.

For nearly 20 years, researchers have sought to make interactive theorem provers better at proving theorems. Much of this effort has been devoted to providing decision procedures. Certainly decision procedures are essential, especially for arithmetic; without them, obvious identities can take hours to prove, with them, complicated facts can be proved instantly. But most of the time, our problem lies beyond the scope of any standard decision procedure. Automatic tools are needed that can work on any type of problem.

Automatic theorem provers (ATPs) are capable of creating long, incomprehensible chains of deduction. Many researchers have attempted to use them to support interactive theorem proving; particularly pertinent are Ahrendt et al. [1], Bezem et al. [5], Hurd [7] and Siekmann et al. [26]. But the only one to pass the test of time is Sledgehammer [14, 21], which links Isabelle/HOL to the automatic provers E, SPASS and Vampire. Isabelle users invoke Sledgehammer routinely when undertaking difficult proofs. In a recent study involving older Isabelle proof scripts, Böhme and Nipkow demonstrated that Sledgehammer could prove 34% of the nontrivial goals contained in those proofs [6].

Sledgehammer was first released in February 2007 to users daring enough to download an Isabelle nightly build. It was announced in November 2007 as a component of Isabelle2007. This paper outlines the design goal that made Sledgehammer successful (§2). It describes some of the lessons learnt in the past three years and its effect on the way Isabelle/HOL is taught (§3). Avenues for research are also discussed (§4).

2 Design Principles

The single most important design goal was one-click invocation. Some earlier systems required the user to gather up all facts that could be relevant to the problem, and furthermore to reduce it to first-order form. Problem preparation could easily take hours, with no guarantee that the call to a first order theorem prover would succeed. Such a tool would be of little value to users.

The two aspects of problem preparation (translation into first-order logic; identification of relevant facts) each required a substantial research effort. The numerous choices outlined below were made on the basis of innumerable experiments that consumed many thousands of hours of processor time.

2.1 Translation into First-Order Logic

Most interactive theorem provers support a language much richer than that of first-order logic. Isabelle/HOL [16] supports polymorphic higher-order logic, augmented with axiomatic type classes [32].[1] Many user problems contain no higher-order features, and might be imagined to lie within first-order logic; however, even these problems are full of typing information. Type information can take quadratic space [12] because every term must be labelled with its type, recursively, right down to the variables. Hurd [8] observed that omitting type information greatly improved the success rate of his theorem prover, Metis. This is hardly surprising, since the type information virtually buries the terms themselves. Hurd was able to omit type information because his proofs are reconstructed within the HOL4 system, which rejected any proofs that did not correspond to well-typed higher-order logic deductions.

Sledgehammer was always intended to rely on an analogous process of sound proof reconstruction, and from the outset it was clear that including complete type information would be unworkable. Completely omitting type information, although successful for Hurd, would not have worked for Isabelle because of its heavy use of type classes. Meng and I chose to include enough type information to enforce correct type class reasoning (the type class hierarchy is easily expressed using Horn clauses) but not to specify the type of every term [14, §4]. Colleagues have expressed horror at the very idea of using unsound translations; I have written a lengthy exploration of the salient issues [12, §2.8].

Although resolution theorem proving is based on clause form, most modern ATPs accept problems in first-order format. Sledgehammer nevertheless translates problems into clause form itself, and using a naive application of distributive laws rather than a polynomial time algorithm based on formula renaming [17]. Moreover, the translation to clauses is performed using Isabelle's internal proof engine; this was thought to be essential to allow proof reconstruction within Isabelle. Sledgehammer's naive translation algorithm has caused real difficulties. Its exponential worst-case behaviour can be triggered by real-world examples. To prevent the naive translation from generating prohibitively many clauses, an arbitrary cutoff (currently 60) had to be introduced. Using first-order format could improve the success rate by exploiting the superior translation technology built into modern ATPs.

Higher-order problems posed special difficulties. We never expected first-order theorem provers to be capable of performing deep higher-order reasoning, but merely hoped to automate proofs where the higher-order steps were trivial. We examined several methods of translating higher-order problems into first-order logic, allowing for at least truth values to be used as the values of terms and for curried functions taking varying numbers of arguments [12]. We eventually adopted a translation based on the one that we used for first-order logic, modified to introduce higher-order mechanisms (such as an "apply operator" for function values) only when absolutely necessary. We thereby eliminated our original distinction between first-order and higher-order problems. A higher-order feature within a problem affects the translation locally, giving a smooth transition from purely first-order to heavily higher-order problems.

We also experimented with two methods of eliminating λ-abstractions in terms: by translating them into combinator form or by declaring equivalent functions. We ultimately opted for combinators, using fairly naive translation scheme. (More sophisticated schemes delivered no benefits.) Experience

[1]Note that Isabelle/HOL is the instantiation of Isabelle [20] to higher-order logic. Isabelle is a generic theorem prover, based on a logical framework [19].

suggests, unfortunately, that Sledgehammer is seldom successful on problems containing higher-order elements. Integration with a genuine higher-order automatic theorem prover, such as LEO-II [3], seems necessary. This would pose interesting problems for proof reconstruction: LEO-II's approach is to reduce higher-order problems to first-order ones by repeatedly applying specialised inference rules and then calling first-order ATPs. A LEO-II proof will therefore consist of a string of higher-order steps followed by a first-order proof. The latter part we know how to do; the crucial challenge is to devise a reliable way of emulating the higher-order steps within Isabelle.

Arithmetic remains an issue. A purely arithmetic problem can be solved using decision procedures, but what about problems that combine arithmetic with a significant amount of logic? In principle, Sledgehammer could solve such problems with the help of an ATP that combined arithmetic and logical reasoning, analogous to LEO-II's approach to higher-order logic. Current SMT solvers are probably of little value, because they do not handle quantified formulas well. But progress in that field is extremely rapid, and soon this option could become attractive.

2.2 Relevance Filtering

Our initial goals for Sledgehammer were modest: to improve upon Isabelle's built-in automatic tools, using only the lemma libraries used by those tools. There were two libraries, one consisting of facts useful for forward and backward chaining, the other consisting of rewriting rules for simplification. Each library contained hundreds of lemmas. Meng and I [13] discovered that automatic theorem provers could solve only trivial problems in the presence so many extraneous facts; by developing a lightweight, symbol-based relevance filter, we greatly improved the success rate. Users would still have to identify relevant facts that did not belong to these lemma libraries.

Tobias Nipkow made the crucial suggestion to dispense with the lemma libraries, substituting the full collection of Isabelle theorems (around 7000). This idea offered the enticing prospect that any relevant existing theorem, however obscure, could be located. I thought this goal to be unrealistic; it seemed to have too much in common with McAllester's Ontic system [9]. Ontic was intended to be able to prove mathematical results using known results that it identified automatically, and it seems fair to say that this objective was too ambitious. But Sledgehammer would only be part of the system rather than all-encompassing, and it could take advantage of 20 years of increasing hardware performance.

We were able to scale up the relevance filter to cope with the 20-fold increase in the number of facts to process. However, it relies on ad-hoc heuristics that sometimes deliver poor results. Briefly, it assigns a score to every available theorem based upon how many constants that theorem shares with the conjecture; this process iterates to include theorems relevant to those just accepted, but with a decay factor to ensure termination. The constants are weighted to give unusual ones greater significance. The relevance filter copes best when the statement to be proved contains some unusual constants; if all the constants are common then it is unable to discriminate among the hundreds of facts that are picked up. The relevance filter is also memoryless: it has no information about how many times a particular fact has been used in a proof, and it cannot learn.

It would obviously be preferable for the automatic theorem provers themselves to perform relevance filtering. Or we should use a sophisticated system based on machine learning, such as Josef Urban's MaLARea [28], where successful proofs provide information to guide other proofs. Unfortunately, any such approach will fail given Sledgehammer's use of unsound translations. In unpublished work by Urban, MaLARea easily proved the full Sledgehammer test suite by identifying an inconsistency in the translated lemma library; once MaLARea had found the inconsistency in one proof, it easily found it in all the others. Sledgehammer is successful only because its relevance filter generally selects too few lemmas to produce an inconsistent axiom set, even with the unsound translations.

To accomplish better relevance filtering, we must decide whether to adopt a general first-order approach or to build a sophisticated relevance filter directly into Isabelle. The former approach could take advantage of the efforts of the entire ATP community, but it would have to be good enough to cope with soundly translated (and presumably enormous) formulas. The latter approach would avoid translation issues, but it would impose the entire effort onto a few Isabelle developers.

2.3 Parallelism

Parallelism was another design objective, both to exploit the abundance of cheap processing power and so that users would not have to wait. Sledgehammer was intended to run in the background; Isabelle would continue to respond to commands, and users could keep working. This idea has turned out to be misconceived: thinking is difficult, and when users hope that a proof might be found for them, they stop and wait for Sledgehammer to report back. We do hope that eventually Sledgehammer will be configured to run spontaneously, without even the need for a mouse click. Then users will simply work and occasionally be delighted to have solutions displayed for them. Such a configuration would require a machine with enough processing power to support several ATP executions without becoming sluggish. An agent-based implementation of similar ideas, using a blackboard architecture, has for some time been part of the Ωmega system [4, 26].

The parallel invocation of different theorem provers is invaluable. Böhme and Nipkow [6] have demonstrated that running three different theorem provers (E [25], SPASS [30] and Vampire [22]) for five seconds solves as many problems as running the best theorem prover (Vampire) for two full minutes. It would be better to utilise even more theorem provers. I have undertaken informal, unpublished experiments involving many other systems.

- Gandalf [27] shows great potential, but unfortunately it does not output useful proofs; one cannot easily identify which axioms have taken part in the proof. A simple source code modification to improve the legibility of proofs would allow Gandalf to make useful contributions. Unfortunately, I was unable to identify the necessary changes. Gandalf has been found to be unsound,[2] but a small percentage of incorrect proofs would be tolerable.

- People sometimes suggest that we include Prover9 [10]. In my experiments, Prover9 performed poorly on the large problems generated by Sledgehammer. It could be effective in conjunction with an advanced and selective relevance filter.

- Another possibility is to run multiple instances of a theorem prover with different heuristics. This is not necessary with Vampire, which attempts a variety of heuristics in separate time slices. It could be particularly effective with the E prover, but designing suitable heuristics requires highly specialised skills.

2.4 Proof Reconstruction

Isabelle subscribes to the LCF philosophy: all proofs ultimately reduce to primitives executed by a logical kernel. Isabelle users would not trust a tool that uncritically accepted proofs from an external source (especially one coded in C++!). But Sledgehammer had an even stronger design objective: to deliver Isabelle proofs in source form. We envisaged that Sledgehammer runs would demand substantial computational resources; if somebody used Sledgehammer many times while constructing a proof, would it be feasible to run that proof again, perhaps to modify it using a laptop while at a conference? To be useful, Sledgehammer would have to return a piece of proof script that could be executed cheaply.

[2]See `http://www.cs.miami.edu/~tptp/TPTP/BustedAsUnsound.html`

2.4.1 Reconstruction of the Resolution Proof

The original plan was to emulate the inference rules of automatic theorem provers directly within Isabelle. We should have known better: Hurd [7] had noticed that the proofs delivered by automatic theorem provers (Gandalf, in his case) were not detailed and explicit enough. We made the same discovery [14] (in the case of SPASS), and despite considerable efforts, were only able to reconstruct a handful of proofs.

We came up with a new plan: to use a general theorem prover, Metis, to reconstruct each proof step. Metis was designed to be interfaced with LCF-style interactive theorem provers, specifically HOL4. Integrating it with Isabelle's proof kernel required significant effort [21]. Metis then became available to Isabelle users, and it turned out to be capable of reconstructing proof steps easily. The output of Sledgehammer was now a list of calls to Metis, each of which proved a clause. This approach was inspired by the Otterfier proof transformation service [33].

Resolution proofs should ideally be translated to natural, intuitive Isabelle proofs. The best-known prior work on translating resolution proofs is TRAMP [11]; its applicability to Sledgehammer is unexplored. Preliminary work has commenced at Munich to see to what extent resolution proofs can be transformed into intelligible proofs.

2.4.2 One-Line Reconstruction, or ATPs as Relevance Filters

Having Metis available made possible an entirely different approach to proof reconstruction: to throw the proof away and allow Metis to find its own proof, using the lemmas that took part in the original resolution proof [21]. With this approach, Sledgehammer became merely a lemma finder, one that used automatic theorem provers merely as relevance filters. But this approach was generally effective, and had the great advantage that each Sledgehammer call now delivered a one-line result, rather than a lengthy and incomprehensible proof script in which all formulas were in clause form. At least one ATP implementer expressed disbelief that his system could be used merely as a relevance filter, but this approach allows any ATP to be used with Sledgehammer provided it returns the list of axioms used in its proof.

Metis sometimes fails to reconstruct the result of a Sledgehammer call. (Böhme and Nipkow [6] present detailed statistics.) Reconstruction necessarily fails if the resolution proof did not correspond to a well-typed Isabelle proof (recall that, normally, Sledgehammer omits most type information when translating Isabelle formulas into first-order logic). This type of failure could be eliminated by using sound translations, but the overall success rate would actually decrease considerably. The seasoned Sledgehammer user eventually learns to recognise unsound proofs (certain lemmas always seem to be mentioned). The number of such proofs could perhaps be reduced by ad hoc measures, such as removing those lemmas from the scope of Sledgehammer, possibly even with the help of machine learning.

Unfortunately, sometimes Metis is simply not powerful enough to prove a theorem that has already been proved by a more powerful system, despite being given a small list of axioms. ATPs frequently use many more axioms than are strictly necessary. The minimization tool developed by Philipp Meyer at TUM takes a set of axioms returned by a given ATP and repeatedly calls the same ATP with subsets of those axioms in order to find a minimal set. Reducing the number of axioms improves Metis's success rate, while also removing superfluous clutter from the proof scripts. ATPs themselves could return proofs using a minimum of axioms, or alternatively, proofs of a minimum length. Vampire's well-known limited resource strategy [23], although designed to cope with limited processor time, could probably be modified to minimise proofs efficiently.

3 Sledgehammer and Teaching

Sledgehammer was not designed specifically as an aid to novices. Experienced users have come to rely on it. But Sledgehammer seems to offer the greatest benefits to the least experienced users. It has certainly transformed the way Isabelle is taught. There are two reasons for this:

- Because it identifies relevant facts, users no longer need to memorise lemma libraries.
- Because it works in harmony with Isar structured proofs, users no longer need to learn many low-level tactics.

Demonstrations of interactive theorem provers necessarily involve deception. The implementers naturally want to show off their system in the best possible light, so they present examples that look more difficult than they really are. Typically they define some recursive functions and prove properties using obvious inductions followed by some sort of auto-tactic. The audience will be duly impressed, and some among them will decide to adopt that tool as the basis for their Ph.D. research. Too late, they encounter the crucial issue:

What do I do when the auto-tactic fails?

Typically, the answer is that one must write incomprehensible scripts that invoke a plethora of obscure commands. These generally include tactics to manipulate the set of assumptions in natural deduction or a sequent calculus. There may be tactics to transform an assumption by applying a rewrite rule or theorem, or to create a case split from an assumption. Some tactics substitute user-supplied terms into theorems.

In Isabelle, the simple combination of structured proofs and Sledgehammer takes the user surprisingly far. This is not the place to give a detailed tutorial on Isar structured proofs [15, 31]. In brief, they support natural deduction through local scopes that can introduce assumptions (using the keyword **assume**) as well as local variables and definitions. Moreover, while traditional tactic scripts contain only commands, a structured proof explicitly states the assumptions and goals. When proving a proposition, you can state intermediate properties that you believe to be helpful. If you understand the problem well enough to propose some intermediate properties, then all you have to do is state a progression of properties in small enough gaps for Sledgehammer to be able to prove each one.

```
proof -
  assume x: "x ∈ lambda_system M f"
  hence "x ⊆ space M"
    by (metis sets_into_space lambda_system_sets)
  hence "space M - (space M - x) = x"
    by (metis double_diff equalityE)
  thus "space M - x ∈ lambda_system M f" using x
    by (force simp add: lambda_system_def)
qed
```

Figure 1: A Structured Proof Completed with the Help of Sledgehammer

Figure 1 presents an example, where two intermediate facts (introduced by the keyword **hence**) assist in the proof of the conclusion (introduced by **thus**). Each of the intermediate facts is proved by a call to Metis that was generated using Sledgehammer. Nor need we restrict ourselves to a linear progression of facts. Because proofs are structured, you can nest the proofs of these lemmas to any depth.

Isar also supports calculational reasoning [2]. A chain of reasoning steps, connected by familiar relations such as $=$, $\leq$ and $<$, can be written with separate proofs for each step of the calculation. Once again, if you can see the intermediate stages of the transformation, then the proofs of each step can be found easily. Figure 2 presents an example of calculational reasoning, taken from a measure theory development.

```
have "f(u ∩ (x ∩ y)) + f(u - x ∩ y) =
     (f(u ∩ (x ∩ y)) + f(u ∩ y - x)) + f(u - y)"
  by (metis class_semiring.add_a ey)
also have "... =  (f((u ∩ y) ∩ x) + f(u ∩ y - x)) + f(u - y)"
  by (metis Int_commute Int_left_commute)
also have "... = f(u ∩ y) + f(u - y)" using fx Int y u
  by auto
also have "... = f u"
  by (metis fy u)
finally show "f(u ∩ (x ∩ y)) + f(u - x ∩ y) = f u" .
```

Figure 2: A Calculational Proof Completed with the Help of Sledgehammer

Top down proof development is greatly assisted by a trivial Isar feature: the ability to omit proofs. Where a proof is required, the user may simply insert the word **sorry**. Isabelle then regards the theorem as proved.[3] The user can then check that the newly introduced proposition indeed suffices to prove the next proposition in the development. A difficult proof can develop as a series of propositions, each initially "proved" using **sorry** but eventually using either Sledgehammer, another automated method, or a nested proof development of the same form. Progress in such a proof can be measured in terms of the difficulty of the propositions that lack real proofs. Although we can never be certain that a proof development can be completed until the very end, the ability to write **sorry** in place of a proof reduces the risk of discovering that a lemma is useless only after spending weeks proving it.

In January 2010, as part of its new MPhil. programme, the University of Cambridge offered a lecture course on Isabelle [18]. The course materials included almost no information about the low-level tactics that had been the mainstay of Isabelle proofs for nearly 20 years. Only two of the 12 lectures were devoted to Isar structured proofs, and they took a novel approach: rather than proceeding methodically through the Isar fundamentals, the lectures presented the outer skeleton of a proof, with crucial sections replaced by **sorry**. They described the idea of trying to eliminate each **sorry** using either Sledgehammer or some automatic tactic. Practical work submitted by the students later demonstrated that several of them had learned how to write complex, well-structured proofs. I was happy to reassure them that submitting work generated largely by Sledgehammer was by no means cheating!

I have taught Isabelle on a number of occasions, starting in the mid-1990s. Sledgehammer is obviously not the only thing to have changed in that expanse of time. The introduction of Isar, continual improvements to Isabelle's automated reasoners, and 15 years of Moore's Law have transformed the user experience. Interactive theorem proving has never been practical because it required far too much effort, even from highly specialised and experienced experts. For the first time, we can envisage the day when interactive theorem proving becomes straightforward enough to be adopted on a large scale.

4 Conclusions

Sledgehammer has been available for over three years, and in that time it has become an essential part of the Isabelle user's workflow. It is possibly the only interface between an interactive theorem prover

[3]The existence of **sorry** does not compromise Isabelle's soundness, because it is only permitted during interactive sessions. A theory file containing an occurrence of **sorry** may not be imported by another theory.

and automatic ones to achieve such popularity with users. It has transformed the way beginners perceive Isabelle.

Counterexample generators are worth mentioning at this point. Users waste much time attempting to prove statements that are not theorems. Tools that can refute invalid conjectures are every bit as helpful as those that prove theorems. Random testing is an obvious way to do this, but counterexample finding can also make use of automated deduction technology. An early example is refute, which uses a SAT solver to find counterexamples [29]; it is also a component of Isabelle.

Sledgehammer has a number of limitations, most of which open up suggestions for future work. The relevance filter is primitive, but an improved one will have to be part of Sledgehammer itself as long as unsound translations are used; only if a compact but sound translation is invented can we rely on automatic theorem provers doing relevance filtering for themselves. Unsound translations can be used safely because Sledgehammer does not trust the proofs that it receives from ATPs but merely uses them as hints to generate Isabelle proof scripts; proofs that violate Isabelle's typing rules are eliminated at this stage. The success rate for first-order problems might be improved by eliminating Sledgehammer's transformation to clause form, delegating that task to ATPs; the impact of such a change on proof reconstruction might be limited, since that is now done using Metis. Sledgehammer's performance on higher-order problems is unimpressive, and given the inherent difficulty of performing higher-order reasoning using first-order theorem provers, the way forward is to integrate Sledgehammer with an actual higher-order theorem prover, such as LEO-II [3]. Proof reconstruction would benefit from new ideas, especially so that it can deliver natural, intuitive proofs.

Acknowledgements

The Cambridge team that developed Sledgehammer included Jia Meng, Claire Quigley and Kong Woei Susanto. Sledgehammer has since been refined, improved and tested by the Isabelle team at T.U. Munich. Christoph Benzmüller and Jasmin Blanchette commented on this paper.

The research was supported by the Engineering and Physical Sciences Research Council [grant number GR/S57198/01], *Automation for Interactive Proof.*

References

[1] Wolfgang Ahrendt, Bernhard Beckert, Reiner Hähnle, Wolfram Menzel, Wolfgang Reif, Gerhard Schellhorn, and Peter H. Schmitt. Integrating automated and interactive theorem proving. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction— A Basis for Applications*, volume II. Systems and Implementation Techniques, pages 97–116. Kluwer Academic Publishers, 1998.

[2] Gertrud Bauer and Markus Wenzel. Calculational reasoning revisited (an Isabelle/Isar experience). In Richard J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics: TPHOLs 2001*, LNCS 2152, pages 75–90. Springer, 2001. Online at `http://link.springer.de/link/service/series/0558/tocs/t2152.htm`.

[3] Christoph Benzmüller, Lawrence C. Paulson, Frank Theiss, and Arnaud Fietzke. LEO-II - a cooperative automatic theorem prover for higher-order logic. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning — 4th International Joint Conference, IJCAR 2008*, LNAI 5195. Springer, 2008.

[4] Christoph Benzmüller and Volker Sorge. OANTS – an open approach at combining interactive and automated theorem proving. In Manfred Kerber and Michael Kohlhase, editors, *Symbolic Computation and Automated Reasoning*, pages 81–97. A. K. Peters, 2000.

[5] Marc Bezem, Dimitri Hendriks, and Hans de Nivelle. Automatic proof construction in type theory using resolution. *Journal of Automated Reasoning*, 29(3-4):253–275, 2002.

[6] Sascha Böhme and Tobias Nipkow. Sledgehammer: Judgement day. In J. Giesl and R. Hähnle, editors, *Automated Reasoning (IJCAR 2010)*, LNCS 6173. Springer, 2010. In press.

[7] Joe Hurd. Integrating Gandalf and HOL. In Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin, and Laurent Théry, editors, *Theorem Proving in Higher Order Logics: TPHOLs '99*, LNCS 1690, pages 311–321. Springer, 1999.

[8] Joe Hurd. First-order proof tactics in higher-order logic theorem provers. In Myla Archer, Ben Di Vito, and César Muñoz, editors, *Design and Application of Strategies/Tactics in Higher Order Logics*, number NASA/CP-2003-212448 in NASA Technical Reports, pages 56–68, September 2003.

[9] David McAllester. Ontic: A knowledge representation system for mathematics. In E. Lusk and R. Overbeek, editors, *9th International Conference on Automated Deduction*, LNCS 310, pages 742–743. Springer, 1988.

[10] William McCune. Prover9 and Mace4. `http://www.cs.unm.edu/~mccune/mace4/`.

[11] Andreas Meier. TRAMP: Transformation of machine-found proofs into natural deduction proofs at the assertion level (system description). In David McAllester, editor, *Automated Deduction — CADE-17 International Conference*, LNAI 1831, pages 460–464. Springer, 2000.

[12] Jia Meng and Lawrence C. Paulson. Translating higher-order clauses to first-order clauses. *Journal of Automated Reasoning*, 40(1):35–60, January 2008.

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

[14] Jia Meng, Claire Quigley, and Lawrence C. Paulson. Automation for interactive proof: First prototype. *Information and Computation*, 204(10):1575–1596, 2006.

[15] Tobias Nipkow. A tutorial introduction to structured Isar proofs. `http://isabelle.in.tum.de/dist/Isabelle/doc/isar-overview.pdf`.

[16] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer, 2002. LNCS Tutorial 2283.

[17] Andreas Nonnengart and Christoph Weidenbach. Computing small clause normal forms. In Robinson and Voronkov [24], chapter 6, pages 335–367.

[18] Lawrence C. Paulson. Interactive formal verification. `http://www.cl.cam.ac.uk/teaching/0910/L21/`. Lecture course materials.

[19] Lawrence C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.

[20] Lawrence C. Paulson. Tool support for logics of programs. In Manfred Broy, editor, *Mathematical Methods in Program Development: Summer School Marktoberdorf 1996*, NATO ASI Series F, pages 461–498. Springer, Published 1997.

[21] Lawrence C. Paulson and Kong Woei Susanto. Source-level proof reconstruction for interactive theorem proving. In Klaus Schneider and Jens Brandt, editors, *Theorem Proving in Higher Order Logics: TPHOLs 2007*, LNCS 4732, pages 232–245. Springer, 2007.

[22] Alexander Riazanov and Andrei Voronkov. Vampire 1.1 (system description). In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning — First International Joint Conference, IJCAR 2001*, LNAI 2083, pages 376–380. Springer, 2001.

[23] Alexandre Riazanov and Andrei Voronkov. Limited resource strategy in resolution theorem proving. *J. Symb. Comput.*, 36(1-2):101–115, 2003.

[24] Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier Science, 2001.

[25] Stephan Schulz. System description: E 0.81. In David Basin and Michaël Rusinowitch, editors, *Automated Reasoning — Second International Joint Conference, IJCAR 2004*, LNAI 3097, pages 223–228. Springer, 2004.

[26] Jörg Siekmann, Christoph Benzmüller, Armin Fiedler, Andreas Meier, Immanuel Normann, and Martin Pollet. Proof development with Ωmega: The irrationality of $\sqrt{2}$. In Fairouz Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, pages 271–314. Kluwer Academic Publishers, 2003.

[27] Tanel Tammet. Gandalf. *Journal of Automated Reasoning*, 18(2):199–204, 1997.

[28] Josef Urban. MaLARea: a metasystem for automated reasoning in large theories. In Geoff Sutcliffe, Josef

Urban, and Schulz Schulz, editors, *ESARLT 2007: Empirically Successful Automated Reasoning in Large Theories*, volume 257 of *CEUR Workshop Proceedings*, 2007.

[29] Tjark Weber. Bounded model generation for Isabelle/HOL. *Electron. Notes Theor. Comput. Sci.*, 125(3):103–116, 2005.

[30] Christoph Weidenbach. Combining superposition, sorts and splitting. In Robinson and Voronkov [24], chapter 27, pages 1965–2013.

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

[32] Markus Wenzel. Type classes and overloading in higher-order logic. In Elsa L. Gunter and Amy Felty, editors, *Theorem Proving in Higher Order Logics: TPHOLs '97*, LNCS 1275, pages 307–322. Springer, 1997.

[33] J. Zimmer, A. Meier, G. Sutcliffe, and Y. Zhang. Integrated proof transformation services. In C. Benzmüller and W. Windsteiger, editors, *Workshop on Computer-Supported Mathematical Theory Development, 2nd International Joint Conference on Automated Reasoning*, Electronic Notes in Theoretical Computer Science, 2004.