Towards Strong Higher-Order Automation for Fast Interactive Verification

Jasmin Christian Blanchette[1,2,3], Pascal Fontaine[3],
Stephan Schulz[4], and Uwe Waldmann[2]

[1] Vrije Universiteit Amsterdam, The Netherlands
j.c.blanchette@vu.nl
[2] Max-Planck-Institut für Informatik, Saarbrücken, Germany
{jasmin.blanchette,uwe.waldmann}@mpi-inf.mpg.de
[3] Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
pascal.fontaine@loria.fr
[4] DHBW Stuttgart, Germany
stephan.schulz@dhbw-stuttgart.de

Abstract

We believe that first-order automatic provers are the best tools available to perform most of the tedious logical work inside proof assistants. From this point of view, it seems desirable to enrich superposition and SMT (satisfiability modulo theories) with higher-order reasoning in a careful manner, to preserve their good properties. Representative benchmarks from the interactive theorem proving community can guide the design of proof rules and strategies. With higher-order superposition and higher-order SMT in place, highly automatic provers could be built on modern superposition provers and SMT solvers, following a stratified architecture reminiscent of that of modern SMT solvers. We hope that these provers will bring a new level of automation to the users of proof assistants. These challenges and work plan are at the core of the Matryoshka project,[1] funded for five years by the European Research Council. We encourage researchers motivated by the same goals to get in touch with us, subscribe to our mailing list, and join forces.

1 Context and State of the Art

Proof assistants (also called interactive theorem provers) make it possible to develop computer-checked, formal proofs of theorems, usually expressed in some variant of higher-order (HO) logic. The primary advantage of formal proofs over their pen-and-paper counterparts is the high trustworthiness of the result; but increasingly, proof assistants are also used for their convenience, especially for program verification, where the proof obligations can be very large and unwieldy.

Compared with other formal methods, the hallmark of proof assistants is their wide applicability and expressiveness. They have been employed since the 1990s for hardware and software verification at AMD [20] and Intel [10]. Some mathematical proofs, such as the four-color theorem [7] and the Kepler conjecture [9], are so complex and require such massive computations that they could be fully trusted only after they had been conducted in a proof assistant. Vladimir Voevodsky, a leading homotopy theorist and Fields medalist, advocates the use of these systems as the only satisfactory answer to the unreliability of modern mathematics [18].

In computer science, two recent groundbreaking developments are the C compiler CompCert [12], verified using the Coq proof assistant, and the seL4 operating system microkernel [11], verified using Isabelle/HOL. CompCert is entering Airbus's tool chain because it generates trustworthy optimized code; without a formal proof, certification authorities would not condone the use of compiler optimizations [5]. The seL4 microkernel was developed jointly with a formal proof that it meets its specification.

[1] http://matryoshka.gforge.inria.fr/

By building on the verified kernel, it is now possible to design verified software chains, with formalized safety and security properties. There are ongoing projects in the automotive industry, aviation, space flight, and consumer devices based on seL4 [15].

So why is only a tiny fraction of software verified? There is a misconception that proof assistants can be mastered only by mathematical virtuosi, but this has been repeatedly refuted in practice. A much more serious issue that affects all users, whether novice or expert, is that proof assistants are very laborious to use. The formal verification of seL4 required about 20 person-years, compared with 2.2 person-years to develop the microkernel itself [11]. To formalize one step in an informal mathematical argument, users must often introduce intermediate properties, interacting with the proof assistant through specialized *tactics*. The need for users to engage in so many interactions—the lack of automation—stands out as the primary cause of low productivity.

The situation has improved substantially in recent years with the rise of first-order (FO) automatic theorem provers, especially superposition provers and SMT (satisfiability modulo theories) solvers based on CDCL(T). Systems such as Sledgehammer, HOLyHammer, Miz$\mathbb{AR}$, and Why3 provide a one-click connection from proof assistants to first-order provers, relieving the user from having to perform tedious tactic manipulations and to memorize lemma libraries [3]. According to Thomas Hales [8], the mathematician who proved the Kepler conjecture and who led the Flyspeck project that formalized it,

> Sledgehammers and machine learning algorithms have led to visible success. Fully automated procedures can prove 40% of the theorems in the Mizar math library, 47% of the HOL Light/Flyspeck libraries, with comparable rates in Isabelle. These automation rates represent an enormous saving in human labor.

Sledgehammer has become an indispensable part of most Isabelle users' workflow. Using Isabelle without Sledgehammer has been compared to walking instead of running. When formalizing Gödel's incompleteness theorems [16], Lawrence Paulson estimated his improvement in productivity in a private communication as a "factor of at least three, maybe five"—an anecdotal but telling remark.

Inevitably, automatic provers are more successful on the easier parts of a formalization, leaving the more difficult—but also more interesting—aspects to humans. With computers and calculators, working mathematicians are no longer wasting their time performing tedious calculations; proof automation brings similar gains to developers of formal proofs. Nevertheless, all too often, tools like Sledgehammer fail to prove trivial-looking goals. This can happen for a number of reasons:

- *Lost in translation*: The translation from higher-order logic to first-order logic relies on clumsy encodings, often resulting in problems that are theoretically or practically unprovable.
- *No induction*: First-order provers generally cannot perform induction, a special case of higher-order reasoning. Among the main provers, only one (CVC4) supports a form of induction (structural induction on datatypes [19]).
- *Explosive search space*: First-order provers perform a systematic, breadth-first search for a proof. They usually fail on interactive goals that require a long proof, even if the proof is straightforward.

First-order automatic provers are very useful, but they are greatly hampered by their lack of support for higher-order constructs. More research is necessary to combine the most successful first-order methods with strong higher-order reasoning.

Native higher-order automated reasoning has been researched since the late 1960s. However, this work has not produced a viable alternative to the Sledgehammer-style HO-to-FO translation [22]. We see two reasons for this. First, higher-order reasoning has not yet assimilated the most successful first-order methods, namely superposition and SMT. Second, the existing provers are designed for tricky higher-order problems, whereas typical proof goals are only mildly higher-order but large. Moreover, these provers usually fail to discover even trivial proofs by induction [22].

2 Challenges and Objectives

Interactive theorem proving has grown considerably in recent years. Proof assistants are employed to build safety- and security-critical systems, and it is not uncommon for research papers to be accompanied by formal proofs. The memory models of Java and C++ have been mechanically verified [2, 13]. Proof assistants are even deployed in the classroom [17], replacing pen-and-paper proofs. These circumstances point to a future where these tools will be routinely used for critical computing infrastructure, for programming language design, and more broadly for research in computer science and mathematics—contributing to more reliable systems and science. But to make this a reality, we must take the hard, tedious labor out of interactive verification. Despite the success of Sledgehammer-style automation, interactive verification is rarely cost-effective. Much more ought to be done.

A fundamental problem is that automatic provers and proof assistants are developed by two mostly disjoint communities, pursuing their own goals. Generally speaking, research on automatic provers tends to focus on first-order logic, for which efficient proof calculi and algorithms have been developed. The tools are highly optimized and typically written in C or C++. Input formats are standardized. The annual competitions and the corresponding benchmarks occupy a prominent place in the developers' minds. The different systems support comparable sets of features, notably: a monomorphic (many-sorted) type system, equality reasoning, and linear arithmetic. In contrast, research on proof assistants focuses on specification mechanisms, user interfaces, specialized tactics, library development, and case studies (formalizations). Trustworthiness is guaranteed by an architecture built around a comparatively small inference kernel. The systems support variants of polymorphic higher-order logic, but the specific formalisms differ, leading to a fragmented community. Most systems are implemented in OCaml or Standard ML, which are considered safer and higher-level than C or C++.

Sledgehammer tries to compensate for the mismatch between these two worlds, but only so much can be done when using automatic provers as black boxes. A different approach is necessary for further progress. The Matryoshka project aims at designing and implementing such an approach. It is funded for five-year by the European Research Council (ERC). The project funds several postdoctoral and Ph.D. positions at Vrije Universiteit Amsterdam and Inria Nancy. Blanchette and Fontaine are directly funded by the project, whereas Schulz and Waldmann are external senior partners, whose expertise in the theory and practice of superposition will be vital for the project. The ERC grant already allowed us to hire Alexander Bentkamp, Johannes Hölzl, Robert Lewis, Hans-Jörg Schurr, and Petar Vukmirović, and we have an open position for a Ph.D. student. Close collaborators include Haniel Barbosa, Simon Cruanes, Mathias Fleury, Stephan Merz, Anders Schlichtkrull, Daniel Wand, and Christoph Weidenbach. See the project's web site for up-to-date information about the team and project-related publications.

Our grand challenge is to deliver very high levels of automation to users of proof assistants by fusing and extending two lines of research: automatic and interactive theorem proving. We believe that strong automation will arise only by building on the strengths of both communities. Higher-order reasoning, including induction, really belongs in high-performance automatic provers. The principles underlying the tactics of proof assistants, largely based on higher-order rewriting, must be integrated into the proof calculi of automatic provers, to solve problems that require both higher-order rewriting and systematic search. Briefly, we want to enrich state-of-the-art automatic methods with concepts motivated by interactive verification.

The first-order provers and calculi are sophisticated artifacts with fragile properties; combinations require careful theoretical and pragmatic considerations. The grand challenge will be met by pursuing four scientific objectives, presented below. Our starting point is that first-order provers are the best tools available for performing most of the logical work. First-order provers, in turn, may delegate this work to SAT (satisfiability) solvers, in a game of Russian dolls. Most problems that are higher-order are only mildly so—a little higher-order can go a long way. Often, expanding definitions and normalizing

λ-terms is all the higher-order reasoning that is needed; but the proof assistants are often too weak to carry out the remaining reasoning steps automatically, and first-order provers cannot manipulate the encoded λs efficiently. Typical proof goals arising in interactive verification are not very difficult, but they require a mixture of undirected proof search and long chains of straightforward reasoning. In his Vampire 2015 invited talk titled "Lost in Translation," Leonardo de Moura remarked: "We need provers/solvers that can understand HOL and perform proofs by induction. Even if it is just [a] thin layer."

Our first scientific objective is to design general, powerful methods to prove real-world interactive goals that lie beyond existing methods or require a sophisticated combination of techniques. Before looking into the details of specific proof calculi, we will develop high-level, technology-agnostic rules, heuristics, and strategies for instantiating higher-order variables (i.e., variables representing functions or formulas) and for reasoning about λ-terms, polymorphic types, (co)datatypes, and (co)induction.

A major part of this objective is to devise strong automation for higher-order logic (also called simple type theory). Given that higher-order automation has been researched for decades, with limited success, relying on a breakthrough is a high-risk aspect of our project. Yet several factors play in our favor. The past 20 years have seen the rise of superposition provers, SAT solvers, and SMT solvers. Most of the research on higher-order reasoning either predates these developments or fails to fully capitalize on them. Thanks to the wider adoption of proof assistants, we now have a clearer idea of the kinds of goals arising in practice and can count on a huge body of relevant benchmarks (e.g., the Archive of Formal Proofs, the Coq User Contributions). On interactive goals, Sledgehammer is demonstrably superior to previous approaches to higher-order reasoning [22]. Yet, its eager HO-to-FO translation views automatic provers as black boxes. Better performance will be possible if we open the black boxes and interleave FO and HO reasoning.

Moreover, first-order provers often fail because induction is necessary. Recent work on automating structural induction in superposition and SMT are welcome developments; but to reach its full potential, this work must be generalized to arbitrary induction schemas, including well-founded induction (for recursive functions) and rule induction (for inductive predicates). Codatatypes and coinduction are other examples of general, widely applicable theories that could be automated efficiently. Higher-order logic is also powerful enough to capture mathematical binder notations such as summations and integrals, which arise when studying quantitative properties of programs and protocols.

Our second objective is to integrate the abstract methods described above into today's most successful proof calculi: superposition and SMT. Despite some convergence, superposition and SMT remain very different technologies, with complementary strengths and weaknesses. The experience with Many goals can be proved only with superposition or with SMT. With multi-core processors being the norm, it is beneficial to let different kinds of provers run in parallel.

The challenge is to preserve the desirable properties of the underlying first-order calculi—whether theoretical (soundness and completeness) or practical (efficiency)—while extending them to perform higher-order rewriting and other higher-order reasoning. This is especially problematic for superposition, which relies on a term order to prune the search space. The order has sometimes been seen as an insurmountable obstacle for the extension of superposition to higher-order logic, but we believe that by treating β-equivalence as an inference, we can design suitable term orders [4].

Our third objective is to implement highly automatic higher-order provers building on modern superposition provers and SMT solvers, giving rise to the first generation of feature-rich, high-performance higher-order provers. In the past decade, we have seen the emergence of higher-order provers based on a *cooperative architecture* (Figures 1 and 2). The approach was pioneered by LEO-II; it has been subsequently adopted by newer versions of Satallax and is advertised as a key feature of the forthcoming Leo-III. These are full-fledged provers that regularly invoke an external first-order prover as a terminal procedure, in an attempt to finish the proof quickly. The external prover may succeed if all the necessary higher-order instantiations have been performed. But because the less efficient higher-order part of the

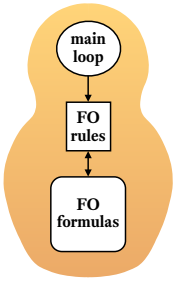

Figure 1 A first-order prover

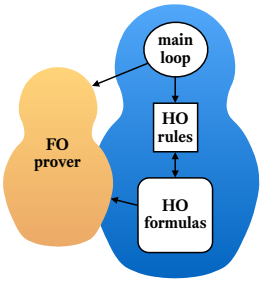

Figure 2 A "cooperative" prover

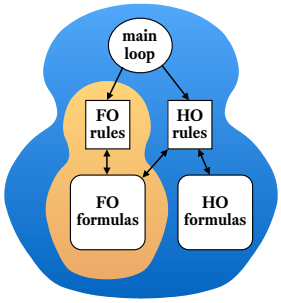

Figure 3 A Matryoshka prover

prover steers the search, this architecture leads to comparatively poor performance on problems with a substantial first-order component [22].

We propose a different stratified architecture for higher-order provers, code-named Matryoshka. At an abstract level, a first-order prover combines three ingredients: (1) a *collection of FO formulas* derived so far; (2) a set of *FO inference rules*; and (3) a *main loop* that applies the rules to derive more formulas (Figure 1). Our envisioned architecture will extend this setup with (1′) a *collection of HO formulas* and (2′) *HO inference rules* that operate on all formulas and put any derived formula in the appropriate collection (Figure 3). The principal modification to the underlying first-order prover is to the main loop, which must interleave FO and HO rule applications. Unlike with the cooperative architecture, a Matryoshka prover is a single program, written in one language—typically, C or C++.

With this new architecture, the automatic prover behaves exactly like a FO prover on FO problems, performs *mostly* like a FO prover on HO problems that are *mostly* first-order, and scales up to arbitrary HO problems, in keeping with the zero-overhead principle ("What you don't use, you don't pay for"). The architecture is reminiscent of the interaction between the SAT solver and quantifier instantiation inside an SMT solver, but there are many open questions because the two logics involved in a stratified higher-order prover are more expressive. Our vehicles of choice will be the superposition provers E [21] and SPASS [23] and the SMT solver veriT [6]. We will also continue our collaboration with the developers of CVC4 [1] and will happily work together with other teams.

Our fourth and final objective is to deliver strong automation in proof assistants by integrating the new automatic provers, with proof reconstruction to ensure trustworthiness. Ultimately, the crucial question is whether our methods will bring substantial benefits to end users. Most of these will work with a proof assistant, which offers a convenient interface for controlling automatic provers and for performing manual proof steps when necessary. Thus, it will be vital to integrate our new stratified provers in proof assistants, by developing or extending Sledgehammer-like tools (Figure 4). We will target Coq, Isabelle/HOL, Lean, and the TLA$^+$ Proof System, which cover the main higher-order formalisms in use today: dependent type theory, higher-order logic, and set theory. Coq and Isabelle are probably the two most popular general-purpose proof assistants, with hundreds of users. Coq's native automation is weaker than Isabelle's, and there is no equivalent to Sledgehammer. Lean is rapidly emerging as a strong rival to Coq. As for the TLA$^+$ Proof System, it is a newer tool for verifying concurrent and distributed systems based on Leslie Lamport's Temporal Logic of Actions, with potential in industry [14]. The proofs generated by the new provers will be reconstructed to yield self-contained proof texts.

Automated reasoning requires optimized data structures and algorithms. A separation of concerns means that the same automatic provers can be shared across many proof assistants. We contend that too much work has gone into engineering the individual proof assistants, and too little into developing compositional methods and tools with a broad applicability across systems.

Recently, Christoph Benzmüller and his colleagues have been working on Leo-III [24], a new prover

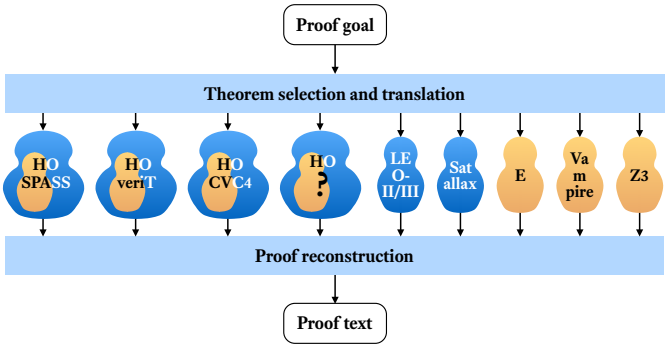

Figure 4 Integration of Matryoshka provers in a proof assistant

based on a multi-agent blackboard architecture. There are many links between the two projects, especially on the superposition front, and we are looking forward to fruitful exchanges and collaborations. We also expect to collaborate with the Lean developers, notably Jeremy Avigad and Leonardo de Moura. Indeed, two of the Matryoshka recruits—Johannes Hölzl and Robert Lewis—have experience with Lean.

The Matryoshka project aims to be a meeting point for researchers working for better automation in proof assistants. Do not hesitate to contact us if you want to keep up with the latest developments of the project and share your own research results. As of May 2017, there are already 30 researchers subscribed to the *matryoshka-devel* mailing list. Why not join them?[2]

Acknowledgment. We are thankful for the support and advice of the following friends and colleagues: Jeremy Avigad, Jean-Pierre Banâtre, Chad Brown, Jip Chong, Fabienne Elbar, Wan Fokkink, Carsten Fuhs, Kris de Jong, Cezary Kaliszyk, Steve Kremer, Leonardo de Moura, Anja Palatzke, Lawrence Paulson, Sylvain Petitjean, Andrei Popescu, Femke van Raamsdonk, Trenton Schulz, Mark Summerfield, Nicolas Tabareau, Dmitriy Traytel, Josef Urban, and Paul Zimmermann. The project receives funding from the European Research Council under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka).

References

[1] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification* (*CAV 2011*), volume 6806 of *LNCS*, pages 171–177. Springer, 2011.

[2] Mark Batty, Scott Owens, Susmit Sarkar, Peter Sewell, and Tjark Weber. Mathematizing C++ concurrency. In Thomas Ball and Mooly Sagiv, editors, *Principles of Programming Languages* (*POPL 2011*), pages 55–66. ACM, 2011.

[3] Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *J. Formalized Reasoning*, 9(1):101–148, 2016.

[4] Jasmin Christian Blanchette, Uwe Waldmann, and Daniel Wand. A lambda-free higher-order recursive path order. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures* (*FoSSaCS 2017*), volume 10203 of *LNCS*, pages 461–479, 2017.

[5] Sandrine Blazy. Compiling avionics software with the CompCert formally verified compiler. Dagstuhl seminar on Qualification of Formal Methods Tools, 2015. http://materials.dagstuhl.de/files/15/15182/15182.SandrineBlazy.Slides.pdf.

[2] http://matryoshka.gforge.inria.fr/#Contact

[6] Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, and Pascal Fontaine. veriT: An open, trustable and efficient SMT-solver. In Renate A. Schmidt, editor, *Conference on Automated Deduction* (*CADE-22*), volume 5663 of *LNCS*, pages 151–156. Springer, 2009.

[7] Georges Gonthier. Formal proof—The four-color theorem. *Notices AMS*, 55(11):1382–1393, 2008.

[8] Thomas C. Hales. Developments in formal proofs. *CoRR*, abs/1408.6474, 2014. Séminaire Bourbaki, 66ème année, 2013–2014, n° 1086. http://arxiv.org/abs/1408.6474.

[9] Thomas C. Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal proof of the Kepler conjecture. *CoRR*, abs/1501.02155, 2015. http://arxiv.org/abs/1501.02155.

[10] John Harrison. Formal verification at Intel. In *Logic in Computer Science* (*LICS 2003*), pages 45–54. IEEE Computer Society, 2003.

[11] Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an operating-system kernel. *Commun. ACM*, 53(6):107–115, 2010.

[12] Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, 2009.

[13] Andreas Lochbihler. Making the Java memory model safe. *ACM Transactions on Programming Languages and Systems*, 35(4):12:1–65, 2014.

[14] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How Amazon Web Services uses formal methods. *Commun. ACM*, 58(4):66–73, 2015.

[15] NICTA. seL4 community projects. http://sel4.systems/Community/Projects/.

[16] Lawrence C. Paulson. A mechanised proof of Gödel's incompleteness theorems using Nominal Isabelle. *J. Automated Reasoning*, 55(1):1–37, 2015.

[17] Benjamin C. Pierce. Lambda, the ultimate TA: Using a proof assistant to teach programming language foundations. In Graham Hutton and Andrew P. Tolmach, editors, *International Conference on Functional Programming* (*ICFP 2009*), pages 121–122. ACM, 2009.

[18] Julie Rehmeyer. Voevodsky's mathematical revolution. *Scientific American* Blogs, 2013. http://blogs.scientificamerican.com/guest-blog/voevodskye28099s-mathematical-revolution/.

[19] Andrew Reynolds and Viktor Kuncak. Induction for SMT solvers. In Deepak D'Souza, Akash Lal, and Kim Guldstrand Larsen, editors, *Verification, Model Checking and Abstract Interpretation* (*VMCAI 2014*), volume 8931 of *LNCS*, pages 80–98. Springer, 2014.

[20] David M. Russinoff. A mechanically checked proof of correctness of the AMD K5 floating point square root microcode. *Formal Methods in System Design*, 14(1):75–125, 1999.

[21] Stephan Schulz. System description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence and Reasoning (LPAR-19)*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.

[22] 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.

[23] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS version 3.5. In Renate A. Schmidt, editor, *Conference on Automated Deduction* (*CADE-22*), volume 5663 of *LNCS*, pages 140–145. Springer, 2009.

[24] Max Wisniewski, Alexander Steen, Kim Kern, and Christoph Benzmüller. Effective normalization techniques for HOL. In Nicola Olivetti and Ashish Tiwari, editors, *International Joint Conference on Automated Reasoning* (*IJCAR 2016*), volume 9706 of *LNCS*, pages 362–370. Springer, 2016.