LEO-II and Satallax on the Sledgehammer Test Bench

Nik Sultana[a,*], Jasmin Christian Blanchette[b], Lawrence C. Paulson[a]

[a] *Computer Laboratory, University of Cambridge, United Kingdom*
[b] *Institut für Informatik, Technische Universität München, Germany*

Abstract

Sledgehammer is a tool that harnesses external first-order automatic theorem provers (ATPs) to discharge interactive proof obligations arising in Isabelle/HOL. We extended it with LEO-II and Satallax, the two most prominent higher-order ATPs, improving its performance on higher-order problems. To explore their usefulness, these ATPs are measured against first-order ATPs and built-in Isabelle tactics on a variety of benchmarks from Isabelle and the TPTP library. Sledgehammer provides an ideal test bench for individual features of LEO-II and Satallax, revealing areas for improvements.

1. Introduction

Most automatic theorem provers (ATPs) are restricted to first-order formalisms, whereas proof assistants typically support more expressive formalisms such as higher-order logic, type theory, and set theory. Until five years ago, there were only two higher-order ATPs, LEO [6] and TPS [1], both based on classical higher-order logic [18]. Since then, a new generation of higher-order ATPs has emerged: LEO-II by Benzmüller et al. [7, 37] and Satallax by Brown et al. [3, 17]. This development coincided with the extension of the TPTP (Thousands of Problems for Theorem Provers) infrastructure with a language for encoding problems in higher-order logic (THF0) [8], a collection of benchmark problems [5, 36], and a competition category for higher-order provers at CASC [31].

Also in recent years, the world of interactive theorem proving witnessed the development and adoption of Sledgehammer [10, 29], a bridge between the proof assistant Isabelle/HOL [28] and several first-order ATPs (including E [33], SPASS [38], Vampire [32], and Z3 [19]). When invoked on a proof goal, Sledgehammer heuristically selects a few hundred background facts, translates them to first-order logic, invokes the external provers in parallel, and reconstructs the proofs in Isabelle. Sledgehammer performs remarkably well in empirical evaluations [10, 14] and boosts user productivity [22]. But Paulson has remarked [29, § 4],

> 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.

[*] Corresponding author
Email addresses: `ns441@cam.ac.uk` (Nik Sultana), `blanchette@in.tum.de` (Jasmin Christian Blanchette), `lp15@cam.ac.uk` (Lawrence C. Paulson)

This paper presents a double materialisation of this vision: an extension of Sledgehammer with LEO-II and Satallax as additional backends (Section 3). The extension reuses many components of Sledgehammer, including the parallel architecture and the relevance filter, but communicates with the ATPs in the higher-order language THF0. Although LEO-II, Satallax, and Isabelle all support "higher-order logic", the translation from Isabelle to THF0 is nontrivial because THF0 does not cater for polymorphic types and axiomatic type classes, which are ubiquitous in Isabelle formalisations.

The integration is useful for proving goals where higher-order features predominate, as demonstrated by a few examples (Section 4). To ascertain more precisely the potential of LEO-II and Satallax, we let them compete on standard Isabelle benchmarks against first-order ATPs and built-in Isabelle tactics (Section 5.1). Although they are nowhere as powerful as the first-order ATPs, they can occasionally solve problems that no other provers or tactics can solve. To make the evaluation more informative, the Isabelle problems are complemented by a subset of the TPTP library, which emphasises the higher-order aspects of the logic. By tuning Sledgehammer's translation, we carried out a fine-grained evaluation (Section 5.2) of the higher-order ATPs' handling of types and λ-abstractions (two problem features we would expect them to handle well) and large background theories. Sledgehammer then acts as a test bench for LEO-II and Satallax, suggesting avenues for improvements.

2. Background

This paper combines several technologies—TPTP, LEO-II, Satallax, Isabelle/HOL, and Sledgehammer—that are amply described elsewhere. This section briefly outlines them.

2.1. TPTP Formats

The TPTP infrastructure defines a hierarchy of languages [8, 35]. Of interest to us are the *first-order form* (FOF) for first-order logic with equality over untyped terms, the core *typed first-order form* (TFF0) that extends FOF with simple types (sorts), and the core *typed higher-order form* (THF0) for higher-order logic. Ignoring minor syntactic differences, the strict inclusions FOF $\subset$ TFF0 $\subset$ THF0 hold.

THF0 types are either *type constants* κ or the function type $\sigma \to \tau$, where σ and τ are arbitrary types. The types of propositions o and of individuals ι are predefined. The intended semantics of THF0 is Henkin semantics with extensionality and Hilbert choice. We take some liberties with the syntax, preferring traditional notations and omitting the apply operator @; thus, we write $\mathsf{f}\ X\ Y$ rather than $\mathsf{f}\ @\ X\ @\ Y$ for the application of X and Y to the curried function f. We do honour the TPTP convention that variable names start with an uppercase letter and constants with lowercase. The use of sans serif for constants further emphasises this distinction.

2.2. LEO-II and Satallax

The higher-order automatic provers LEO-II [7] and Satallax [3, 16] have THF0 as their input language. Both attempt to find a refutation from the negated conjecture and the axioms, amounting to a proof of the original conjecture. To improve their effectiveness, both provers implement strategy scheduling, which involves trying a sequence of option settings, each for a fraction of the allotted time.

LEO-II implements a higher-order resolution calculus and periodically dispatches first-order subproblems to a first-order prover, usually E, with which it cooperates. LEO-II features several optimisations, notably shared indexed terms [37] and a simple relevance filter that is activated for problems that contain 100 axioms or more. Its proofs, expressed in the TSTP format, combine native inferences with embedded E proofs [34].

Satallax is a tableau-based instantiation prover. It builds propositional approximations of a THF0 problem and relies on the SAT solver MiniSat [20] to check them. In case of success, Satallax can return an unsatisfiable core (a list of formulas that suffice to obtain a contradiction), a Coq proof script, or a Coq proof term.

2.3. Isabelle/HOL

Isabelle [28] is a logical framework that provides a metalogic to encode the semantics of object logics and a collection of basic definitions to manage inference in those logics. Isabelle follows the LCF architecture: the logical kernel defines an abstract datatype of theorems, and theorems are proved using only the methods made available by the kernel.

HOL is Isabelle's most developed object logic. It is based on classical higher-order logic (simple type theory) [18], augmented with Hilbert choice, polymorphism, and Haskell-style axiomatic type classes [21, 40]. Users of Isabelle/HOL invariably work within a substantial body of formalised mathematics that has been constructed on the foundation of pure higher-order logic, including the concepts of orders, lattices, sets, functions, relations, numbers, and lists. We adhere to the Isabelle/HOL conventions for writing terms and rely on the reader's discernment to identify x in Isabelle with X in THF0, 0 with zero, () with unity, Cons with cons, and so on.

Isabelle/HOL includes several automatic proof tools, including a term-rewriting engine (the *simplifier*), a tableau prover, and decision procedures for specific theories. It also works with external provers through Sledgehammer.

2.4. Sledgehammer

The purpose of Sledgehammer is to prove theorems with no user effort using automatic theorem provers—historically, first-order resolution provers and SMT solvers [10, 25]. When activated (by a single mouse click), it packages up the formula to be proved along with a collection of relevant facts (definitions, lemmas, or axioms) extracted from Isabelle's libraries using a simple relevance filter [26]. The problems are translated to the provers' respective input languages. The provers run in parallel, either locally or over the Internet.

If a proof is found, Sledgehammer minimises it to remove redundant facts, then inserts a *metis* or *smt* method call into the Isabelle formalisation to reconstruct the proof, with the short list of referenced facts. The *metis* method is based on the built-in resolution prover of the same name [30], whereas *smt* relies on the SMT solver Z3 [15]; both methods yield LCF-style proofs.

Translating Isabelle problems into the FOF and TFF0 languages supported by first-order provers is one of the main technical problems in Sledgehammer, *metis*, and *smt*. Although Isabelle/HOL is based on higher-order logic, many formulas are largely first-order, with only a few higher-order features. The translation is a two-step process:

1. *The higher-order constructs are eliminated* [25]. λ-abstractions are rewritten to combinators (I, K, S, B, C) or to supercombinators (λ-lifting). Functions are passed varying numbers of arguments via an explicit apply operator, hAPP. Boolean terms are converted to formulas using a unary predicate, hBOOL. Connectives and quantifiers are mapped to their first-order counterparts whenever possible; the remaining occurrences are embedded as uninterpreted function symbols, called *proxies*. All these artefacts are confined to the truly higher-order parts of the problem.

2. *The type information is encoded in the target logic* [9, § 6.5]. For TFF0, the problem is monomorphised, then its types are mapped to TFF0 types. For FOF, HOL types are encoded using a combination of type arguments and either guards or tags. A *type guard* takes the form of a predicate $\mathsf{g}(\sigma, X)$ that indicates whether variable X has type σ, where σ is encoded as a term; for example, $\mathsf{g}(\mathsf{list}(A), Xs)$ checks that Xs has type α *list*, where the term variable A encodes α. A *type tag* is a function $\mathsf{t}(\sigma, t)$ that wraps the term t with its type σ.

Example. Consider the following Isabelle/HOL conjecture, where $f :: \alpha \to \beta$ and $h :: \beta \to \gamma$:

$$\mathsf{map}\ (\lambda x.\ h\ (f\ x))\ xs = \mathsf{map}\ h\ (\mathsf{map}\ f\ xs)$$

The traditional guard- and combinator-based encoding, one of the many translations implemented in Sledgehammer and *metis*, produces the following untyped first-order (FOF) formulas:

$$\begin{array}{l}\mathsf{map}(\mathsf{a}, \mathsf{c}, \mathsf{combB}(\mathsf{h}, \mathsf{f}), \mathsf{xs}) = \mathsf{map}(\mathsf{b}, \mathsf{c}, \mathsf{h}, \mathsf{map}(\mathsf{a}, \mathsf{b}, \mathsf{f}, \mathsf{xs}))\\ \mathsf{g}(\mathsf{fun}(A, B), F) \wedge \mathsf{g}(\mathsf{fun}(B, C), G) \wedge \mathsf{g}(A, X) \longrightarrow\\ \qquad \mathsf{hAPP}(\mathsf{combB}(G, F), X) = \mathsf{hAPP}(G, \mathsf{hAPP}(F, X))\end{array}$$

The λ-abstraction is rewritten using the B combinator, which is characterised by the second equation—in HOL notation, $\mathsf{B}\ g\ f\ x = g\ (f\ x)$. Partial functions are passed arguments via the explicit apply operator (hAPP). Type and term variables in the conjecture are translated to Skolem constants: a, b, c, f, h, and xs. The polymorphic map function takes two type arguments encoded as terms; for example, the $(\alpha \to \beta) \to \alpha\ \mathit{list} \to \beta\ \mathit{list}$ instance of map is translated to $\mathsf{map}(\mathsf{a}, \mathsf{b}, \ldots)$. The encoding is complemented by typing axioms for the function symbols occurring in the problem, such as the following:

$$\begin{array}{c}\mathsf{g}(\mathsf{fun}(\mathsf{a}, \mathsf{b}), \mathsf{f}) \qquad\qquad \mathsf{g}(\mathsf{fun}(\mathsf{b}, \mathsf{c}), \mathsf{h})\\ \mathsf{g}(\mathsf{fun}(A, B), F) \wedge \mathsf{g}(\mathsf{fun}(B, C), G) \longrightarrow \mathsf{g}(\mathsf{fun}(A, C), \mathsf{comp}(F, G))\end{array}$$

The typing axioms are necessary to discharge the type guards occurring in the rest of the problem.

3. Bridging Isabelle and THF0

Sledgehammer's translation module generates problems in the TPTP untyped first-order form (FOF) and, since recently, monomorphic typed first-order form (TFF0). We have extended the tool to produce THF0 as well. Although both Isabelle/HOL and THF0 are higher-order logics, the translation is nontrivial because Isabelle features a metalogic and the HOL object logic supports polymorphism and axiomatic type classes.

3.1. The Metalogic

Isabelle's metalogic is an intuitionistic fragment of higher-order logic. The semantics interprets the type *prop* of propositions and the function type $\alpha \to \beta$. The metalogical operators are

$$\begin{array}{rll}\Longrightarrow :: & \mathit{prop} \to \mathit{prop} \to \mathit{prop} & \text{implication}\\ \bigwedge :: & (\alpha \to \mathit{prop}) \to \mathit{prop} & \text{universal quantification}\\ \equiv :: & \alpha \to \alpha \to \mathit{prop} & \text{equality}\end{array}$$

HOL provides a type *bool* of Booleans, the constants False, True, and $=$, the connectives $\neg$, $\wedge$, $\vee$, and $\longrightarrow$, and the quantifiers $\forall$ and $\exists$. HOL is embedded in the metalogic via the constant Trueprop :: $bool \to prop$, which is normally not shown to users. Isabelle provides methods to translate terms and theorems involving *prop*, $\Longrightarrow$, $\bigwedge$, and $\equiv$ into terms and theorems with *bool*, $\longrightarrow$, $\forall$, and $=$ instead. This allows Sledgehammer to work in HOL, avoiding the metalogic altogether.

3.2. Monomorphisation

THF0, the input logic of LEO-II and Satallax, only supports monomorphic types and terms. We must therefore encode Isabelle/HOL's polymorphism in THF0 or eliminate it somehow. Sound and complete encodings of polymorphic types in a monomorphic or untyped logic are well understood in a first-order context [9, 12]; unfortunately, their generalisations to higher-order logic are flawed, as we demonstrate in Section 3.6.

We address this issue by monomorphising the problems before encoding them in THF0. This involves instantiating polymorphic types with heuristically selected ground types. Once this is done, THF0 types can be used to represent HOL types. The HOL Boolean type and function space are mapped to their THF0 counterparts, whereas the remaining types are mapped to distinct THF0 types. For example, $nat \to (nat \times nat \to bool) \to nat\ list$ becomes $nat \to (prod_{nat,nat} \to o) \to list_{nat}$ (where $prod_{nat,nat}$ and $list_{nat}$ are fresh atomic THF0 types).

Monomorphisation algorithms are necessarily incomplete [12, § 2], but our experience with first-order provers is that monomorphisation-based schemes outperform the best complete type encodings that rely on guards or tags [9, § 6.7.2]. Monomorphisation also relieves LEO-II and Satallax from having to reason about type classes, since these are attached to type variables, which are all instantiated by the monomorphiser.

Sledgehammer's monomorphiser iteratively instantiates polymorphic formulas with relevant monomorphic instances of their polymorphic symbols [13, § 2.2.1]. To ensure termination, the iterations are limited to a number K. An upper bound Δ on the number of new formulas curbs the exponential growth.

Experiments with first-order provers found $K = 3$ and $\Delta = 200$ suitable, so that a problem with 320 axioms will comprise at most 520 axioms after monomorphisation. For higher-order provers, we decreased these limits to $K = 2$ and $\Delta = 100$ based on experiments similar to those described in Section 5. Given formulas involving types *nat* and α *list*, two iterations suffice to produce *nat list* and *nat list list* instances; adding more layers of *list* rarely helps in practice. Increasing Δ does help solve additional problems, but it can rapidly overwhelm the provers.

3.3. Translation Pipeline

An unusual aspect of our work is that since Sledgehammer normally targets first-order provers, much of what we had to do was to tell it *not* to perform certain actions. Starting from Sledgehammer's existing translation to monomorphic typed first-order logic (TFF0), we targeted the THF0 format by following these steps:

1. Explicitly mark application using @, to comply with THF0.
2. Identify the HOL type *bool* with the THF0 type o, eliminating the need for hBOOL.
3. Identify the HOL function type with the THF0 function type and replace the explicit apply operator hAPP with @.
4. Map higher-order (unpolarised) occurrences of HOL connectives, quantifiers, and equality to the corresponding THF0 constructs, eliminating the need for proxies.

5. Let λ-abstractions pass through the translation.
6. For Satallax, identify the Hilbert choice constant from HOL with the corresponding THF0 operator. (LEO-II currently does not support Hilbert choice.)

The translation of HOL terms to THF0 is central to the extension of Sledgehammer with LEO-II and Satallax, but other parts of the machinery also needed some adjustments. The TPTP format allows us to label axioms as being *definitions*. This does not affect the semantics of the problem, but both LEO-II and Satallax tend to aggressively unfold definitions of the form $\mathsf{c} = t$, where c is a constant and t is a closed term. For all Isabelle definitions $\mathsf{c}\ x_1 \ldots x_n = t$ selected by the relevance filter, we experimented with having Sledgehammer generate a THF0 definition $\mathsf{c} = (\lambda x_1 \ldots x_n.\, t)$. This also requires reordering the formulas so that constants are defined before their first use, as expected by Satallax. Definitions had a clear positive impact on Satallax and an equally clear negative impact on LEO-II, so we made this the default for only Satallax.

Isabelle includes large background theories, which users can further extend. Sledgehammer's relevance filter heuristically selects up to N background facts, where N is carefully chosen for each prover. Although they make more goals provable, high values of N tend to overwhelm provers. Based on our evaluation (Section 5.2), we set the defaults to $N = 40$ for LEO-II and $N = 60$ for Satallax.

3.4. Proof Reconstruction

Unless we are prepared to trust Sledgehammer's translation and the external provers, any proofs found by LEO-II or Satallax should be validated by Isabelle.

For LEO-II, we extract the referenced facts from the TSTP proof returned and attempt to re-find the proof with a *metis* or *smt* call, in the hope that no deep higher-order reasoning is necessary. Since extensionality is built into THF0 but is an axiom in HOL, we detect applications of LEO-II's extensionality rule and supply the HOL axiom to the reconstructor. For Satallax, we take the unsatisfiable core as the list of facts to pass to *metis* or *smt*, together with the extensionality axiom and the definitions occurring in the problems (since they are omitted in the unsatisfiable core).

The *metis* and *smt* methods sometimes fail due to their incomplete, inefficient handling of higher-order constructs (Section 2.4). Work has started on a step-by-step proof reconstruction tactic for LEO-II as a more reliable option, similar in principle to the Z3-based *smt* method.

3.5. Problem Importer

Until recently, TPTP problems were imported into Isabelle using Sutcliffe's TPTP2X tool [36, § 4.3], which translated them into Isabelle theory files. This was inconvenient to Isabelle users, so we extended Isabelle to parse and interpret TPTP problems (whether they are expressed in CNF, FOF, TFF0, or THF0) directly as collections of HOL formulas. Once a problem has been imported, it can be processed by a variety of provers and counterexample generators. The version of Isabelle entered in CASC relies on this parser, as does the TPTP part of our evaluation in Section 5.1.

3.6. The Trouble with Polymorphic Type Encodings

There appears to be no sound, complete, and efficient way to encode the polymorphic type information of Isabelle/HOL in THF0 while identifying the HOL and THF0 function spaces and equality. The traditional approaches [9, 25], based on type guards or type tags, admit no generalisation to higher-order logic. Let us briefly see why.

Type Guards. Type guards are predicates that restrict the range of variables. In first-order logic, they take the form of a distinguished predicate $\mathsf{g}(\sigma, X)$ that checks whether variable X has type σ, where σ is encoded as a term. A guard-based translation of the HOL theorems

$$0 \neq (1 :: bit) \qquad\qquad f\,() = g\,() \implies f = (g :: unit \to \alpha)$$

that mimics the traditional first-order approach would yield the THF0 axioms

$$\mathsf{zero} \neq \mathsf{one} \tag{1}$$

$$\mathsf{g}\ (\mathsf{fun}\ \mathsf{unit}\ A)\ F \longrightarrow \mathsf{g}\ (\mathsf{fun}\ \mathsf{unit}\ A)\ G \longrightarrow F\ \mathsf{unity} = G\ \mathsf{unity} \longrightarrow F = G \tag{2}$$

where zero, one, unity have type ι and F, G have type $\iota \to \iota$ but are protected by g to guard against ill-typed instantiations.

Already at this point, we face the issue that the second argument to the guard predicate g should in general be of type ι, not $\iota \to \iota$. This problem arises from our wish to identify the HOL function space with the THF one. (In contrast, the traditional first-order encoding treats higher-order arguments in the same way as first-order arguments, giving them the type ι.) Let us pretend this issue can be solved, perhaps via injections into ι.

Let ite ("if then else") be a constant such that

$$\mathsf{ite}\ \mathsf{true}\ X\ Y = X \qquad\qquad \mathsf{ite}\ \mathsf{false}\ X\ Y = Y$$

and consider the instantiation

$$A := \mathsf{bit} \qquad F := (\lambda U.\ \mathsf{zero}) \qquad G := (\lambda U.\ \mathsf{ite}\ (U = \mathsf{unity})\ \mathsf{zero}\ \mathsf{one})$$

in (2). Both $(\lambda U.\ \mathsf{zero})$ and $(\lambda U.\ \mathsf{ite}\ (U = \mathsf{unity})\ \mathsf{zero}\ \mathsf{one})$ correspond to well-typed HOL terms, of type $unit \to bit$. Assuming reasonable typing axioms for functions, the g guards in our instance of (2) should be dischargeable; otherwise, the encoding would be incomplete. The remaining assumption, $F\ \mathsf{unity} = G\ \mathsf{unity}$, reduces to $\mathsf{zero} = \mathsf{zero}$, i.e. true. Hence, from (2) we derive the THF0 theorem

$$(\lambda U.\ \mathsf{zero}) \;=\; (\lambda U.\ \mathsf{ite}\ (U = \mathsf{unity})\ \mathsf{zero}\ \mathsf{one}) \tag{3}$$

Although the sides of the equation encode HOL terms of type $unit \to bit$, they have type $\iota \to \iota$ in THF0. This gives us more than enough rope to derive a contradiction. Let

$$\mathsf{c} = (\mathsf{ite}\ (\mathsf{unity} = \mathsf{zero})\ \mathsf{one}\ \mathsf{zero})$$

By (1), we have $\mathsf{c} \neq \mathsf{unity}$. Hence,

$$(\lambda U.\ \mathsf{zero})\ \mathsf{c} \;=\; \mathsf{zero} \;=\; \mathsf{one} \;=\; (\lambda U.\ \mathsf{ite}\ (U = \mathsf{unity})\ \mathsf{zero}\ \mathsf{one})\ \mathsf{c}$$

by congruence and (3), contradicting (1). The encoding is unsound.

The above argument is admittedly rather technical. It may help to think of it in model-theoretic terms. Axiom (1) ensures that the domain associated with ι has at least two distinct elements, whereas (2) relies on the function space's built-in semantics to force a cardinality of 1 onto ι. Taken together, the two axioms are unsatisfiable. In contrast, the original HOL axioms are satisfiable because they operate on different types (bit versus unit).

It may be possible to repair the encoding by insisting that all functions that satisfy the guard $\mathsf{g}\ (\mathsf{fun}\ \sigma\ \tau)$ return some fixed element, $\mathsf{undefined}\ \tau$, for arguments that do not satisfy $\mathsf{g}\ \sigma$. (Explicit type arguments, such as τ in $\mathsf{undefined}\ \tau$, distinguish instances of polymorphic constants.)

This scheme would effectively rule out the problematic instantiation of G in the counterexample above. It also blends well with congruence of equality but forces us to translate an innocuous-looking formula such as $\mathsf{hd}\ (\mathsf{Cons}\ x\ xs) = x$ to something like

$$\mathsf{hd}\ A\ (\mathsf{cons}\ A\ X\ Xs) = \big(\mathsf{ite}\ A\ (\mathsf{g}\ A\ X \wedge \mathsf{g}\ (\mathsf{list}\ A)\ Xs)\ X\ (\mathsf{undefined}\ A)\big)$$

Any encoding based on this idea would be so cluttered as to be impractical.

Type Tags. Unlike type guards, type tags do not suffer from any obvious unsoundness, but they are hopelessly incomplete because of poor interactions with the built-in β and η rules of higher-order logic. To be effective, tags must generally appear around all terms and subterms, including function applications. For the η rule, $(\lambda x.\ f\,x) = f$, we could in principle supply the fully tagged version

$$\mathsf{t}\ (\mathsf{fun}\ A\ B)\ (\lambda X.\ \mathsf{t}\ B\ ((\mathsf{t}\ (\mathsf{fun}\ A\ B)\ F)\ (\mathsf{t}\ A\ X))) = \mathsf{t}\ (\mathsf{fun}\ A\ B)\ F$$

as an axiom along with the THF0 problem and hope that the built-in η rule will cause no harm. However, there is no adequate substitute for the β rule, $(\lambda x.\ t[x])\,a = t[a]$, because the variable x may occur at arbitrary positions in $t[x]$, a situation that cannot be captured by inflexible, stratified tags. Furthermore, the untypability of the guard predicate g mentioned above also plagues the tag function t.

4. Examples

The following examples illustrate the use of LEO-II and Satallax in Sledgehammer. They were chosen to demonstrate both the strengths and the weaknesses of the provers.

Finite Sums. The $\sum$ operator is formalised in Isabelle as a higher-order function of type $(\alpha \to \beta) \to \alpha\ set \to \beta$, with the constraint that β belongs to the type class of commutative monoids under addition. The proof goal below, where ω_n denotes a primitive nth root of unity, arises in a formalisation of the fast Fourier transform (FFT):

$$\sum_{j=0}^{n} \omega_{2n}^{i}\, \omega_{2n}^{i(2j)} f(2j+1) = \sum_{j=0}^{n} \omega_{2n}^{i} \big(\omega_{n}^{ij} f(2j+1)\big)$$

The first time we looked at this goal, none of the standard tactics or first-order ATPs could solve it within 30 seconds. In contrast, it took Satallax 22 seconds to find a proof relying on the associativity of multiplication and the following cancellation property:

$$\omega_{2n}^{i(2j)} = \omega_{n}^{ij}$$

Given these two lemmas, *metis* can re-find the proof almost instantly, but it requires that λ-lifting is used and that the extensionality axiom is supplied:

by (*metis* (*lifting*) *mult_ac*(1) *root_cancel1 ext*)

Users often edit and shorten *metis* proofs to use *auto* or *simp* instead, making the proof more idiomatic. The previous proof can be rewritten to use the simplifier as follows:

by (*simp only*: *mult_ac*(1) *root_cancel1*)

This example is one of many goals from the FFT theory that only Satallax can solve in reasonable time. It is difficult for first-order provers because equational reasoning takes place under λ-abstractions. One might think that λ-lifting, by eliminating all λs, would address this, but extensionality is then needed to compare lifted functions. First-order provers can deal with extensionality, encoded as the axiom

$$(\forall X.\ \mathsf{hAPP}(F, X) = \mathsf{hAPP}(G, X)) \longrightarrow F = G$$

(omitting types), but the literal $F = G$ dramatically increases the search space.[1]

Big O Notation. The "big O" notation can be defined as a polymorphic constant of type $(\alpha \to \alpha) \to (\alpha \to \alpha)$ *set* in HOL, where α must support various algebraic type classes. The constant is defined by

$$\mathrm{O}(f) = \{h : \exists c.\, \forall x.\ |h\, x| \le c \cdot |f\ x|\} = \mathsf{Collect}\ (\lambda h.\, \exists c.\, \forall x.\ |h\, x| \le c \cdot |f\ x|)$$

The constant $\mathsf{Collect}$ builds a simply typed set from a characteristic function. Big O membership is preserved by function composition in the following sense:

$$f \in \mathrm{O}(g) \implies (\lambda x.\ f\,(k\ x)) \in \mathrm{O}(\lambda x.\ g\,(k\ x))$$

To find the proof, we can unfold the definition of O and simplify using the equivalence $x \in \mathsf{Collect}\ P = P\ x$, yielding the trivial implication

$$\big(\exists c.\, \forall x.\ |f\, x| \le c \cdot |g\ x|\big) \implies \big(\exists c.\, \forall x.\ |f\,(k\ x)| \le c \cdot |g\ (k\ x)|\big)$$

If we call LEO-II and Satallax with the two necessary facts for 30 seconds, only LEO-II finds a proof. However, adding more facts quickly overwhelms it. Once we have proved the fact, we can try proving its instance

$$(\lambda x.\ f\, x + g\, x) \in \mathrm{O}(h) \implies (\lambda x.\ f\,(k\ x) + g\,(k\ x)) \in \mathrm{O}(\lambda x.\ h\,(k\ x))$$

Here, the situation is reversed. Satallax notices that the conjecture is a higher-order instance of the already proved lemma, despite the presence of dozens of extraneous facts. LEO-II cannot find a proof even if we leave out all unnecessary facts, which Benzmüller attributes to an explosion in the prover's "extensional pre-unification algorithm" [4].

Associativity of Append. The append operation on lists is associative. We were hoping that the higher-order ATPs would find a proof given only the induction rule for lists and the equational specification of append. The results were disappointing: even with a 300-second timeout, neither prover was able to guess the right instantiation for the higher-order variable in the induction schema, namely

$$\lambda xs.\ \forall ys\ zs.\ \mathsf{append}\ xs\ (\mathsf{append}\ ys\ zs) = \mathsf{append}\ (\mathsf{append}\ xs\ ys)\ zs$$

and carry out the proof from there. Both provers can find the proof if Sledgehammer preinstantiates the higher-order variable based on the goal, but then the problem is essentially first-order and well within the reach of the first-order ATPs.

[1]Resolution provers contain the proliferation of clauses by postponing paramodulations into variables (the only inference that can instantiate F and G), but this delays necessary applications of extensionality [39]. For SMT solvers, the main issue is the absence of obvious triggers (syntactic patterns that guide variable instantiations) [27].

5. Evaluation

In this section, we attempt to quantify LEO-II's and Satallax's performance along two main axes. First, we measure the provers' usefulness as backends to Sledgehammer (Section 5.1). This involves comparing them with the main first-order ATPs as well as Isabelle's automatic tactics. Second, we perform a more detailed analysis to determine how well LEO-II and Satallax cope with large background theories, types, and λ-abstractions (Section 5.2). The benchmark data is partitioned in three categories:[2]

- *Judgement Day* (1268 goals) consists of seven theories from the Isabelle distribution and the *Archive of Formal Proofs* [23]. These theories were selected by Böhme and Nipkow [14] and serve as the main benchmark suite for Sledgehammer.
- *Arithmetic Extension* of Judgement Day (616 goals) consists of three Isabelle/HOL theories that were used in evaluations of SMT solvers and type encodings [9, 10]. They involve both linear and nonlinear arithmetic.
- *TPTP THF0* (1000 goals) is a randomly chosen subset of the 2286 THF0 problems marked as theorems in the TPTP 5.3.0 library that do not originate from Isabelle [36]. These benchmarks have largely guided LEO-II's and Satallax's development.

The problems from all three categories were processed by Sledgehammer to generate input for the ATPs. For the two Isabelle categories, the Sledgehammer output includes heuristically selected background facts. For the TPTP THF0 category, all facts in the original problems were included, so that the THF0 problems generated for LEO-II and Satallax are nearly identical to the original problems.[3]

We used the following prover versions: LEO-II 1.3.4, Satallax 2.4, E 1.5, SPASS 3.8ds, Vampire 1.8 (revision 1435), and Z3 4.0. We relied on the default Sledgehammer setup for the first-order ATPs, as determined by previous evaluations, with time slicing to simulate strategy scheduling [10, 11]. The Isabelle tactics include the simplifier, the tableau prover, the resolution prover *metis*, and arithmetic decision procedures.

5.1. Usefulness as Backends

Table 1 presents the success rates of Isabelle's automatic tactics, first-order ATPs, and higher-order ATPs on the three benchmark categories. For each category and each prover (or class of provers), both the percentage of solved goals ("Solved") and the percentage of goals that were uniquely solved by that prover ("Uniq.") are shown.

The experiments were carried out on the same hardware as the original Judgement Day evaluation by Böhme and Nipkow [14, § 3]. Each ATP was run in a single thread with a wall-clock time limit of 30 seconds per problem, and each Isabelle tactic was given 5 seconds. The external provers were trusted.

LEO-II and Satallax solved among them 42.1% of Judgement Day, which is slightly more than the Isabelle tactics but much less than the first-order ATPs. Nonetheless, Satallax's unique contribution of 0.6%, or 8 goals, puts it in respectable company with E (0.8%), SPASS (0.6%),

[2]The dataset is available at `http://www.cl.cam.ac.uk/~ns441/files/testbench-data.tgz`.

[3]There are some minor differences: the Sledgehammer setup renames the constants, may reorder the axioms, and is more aggressive in labelling axioms as definitions.

	Judgement Day		Arith. Ext.		TPTP THF0	
	Solved	Uniq.	Solved	Uniq.	Solved	Uniq.
Isabelle tactics	40.4	6.9	40.1	9.9	62.7	0.9
First-order ATPs	**64.7**	**15.7**	**49.8**	**13.1**	70.2	4.5
E	54.3	0.8	36.0	0.6	49.6	0.0
SPASS	54.3	0.6	36.9	0.8	49.6	0.0
Vampire	54.3	0.8	33.1	0.3	58.0	1.9
Z3	53.4	2.4	41.4	4.1	56.7	0.9
Higher-order ATPs	42.1	0.6	25.2	0.2	**82.6**	**9.0**
LEO-II	30.0	0.0	15.6	0.0	62.8	1.1
Satallax	38.4	0.6	23.1	0.2	77.9	4.3
All provers	72.4	–	60.1	–	88.2	–

Table 1: Success rates (%) of proof search

and Vampire (0.8%) and raises the overall success rate from 71.8% to 72.4%. The goals uniquely solved by Satallax all involve reasoning about λs.

The results for Arithmetic Extension are less impressive for the higher-order ATPs. This is not surprising since neither LEO-II nor Satallax is equipped with specialised reasoning engines for arithmetic. As one would expect, the best performers in this category are Z3 and Isabelle's tactics, both of which feature arithmetic decision procedures.

Satallax dominates the TPTP THF0 category. The strong performance of LEO-II and Satallax in this category was to be expected, since both provers have been tuned and tested against these problems. Conversely, the first-order ATPs' weaker results reflect the higher-order nature of these benchmarks.

Table 2 shows the success rate of proof reconstruction after minimisation for the two Isabelle categories as a percentage of all proofs found. Considering that the reconstructor methods *metis* and *smt* are essentially first-order provers and employ the same techniques as Sledgehammer to eliminate higher-order constructs, the reconstruction rates for LEO-II and Satallax are remarkably high. Unfortunately, reconstruction tends to fail precisely for the goals that are solved by only Satallax, leaving the user with a short list of facts but no actual proof.

	Judg. Day	Arith. Ext.
LEO-II	98.4	97.9
Satallax	97.5	97.9
E	97.2	98.2

	Judg. Day	Arith. Ext.
SPASS	99.1	99.6
Vampire	98.7	97.5
Z3	99.9	95.3

Table 2: Success rate (%) of proof reconstruction

5.2. Feature-by-Feature Analysis

To evaluate specific features of the higher-order provers, we found it useful to include Vampire and Z3 in our comparisons. These two provers can be seen as first-order cousins of the resolution prover LEO-II and the SAT-based Satallax, respectively.[4] The problems generated for Vampire and Z3 are in the TFF0 syntax, exploiting their native types (but not their support for arithmetic). The benchmarks used for this section are the union of Judgement Day and Arithmetic Extension, as generated by Sledgehammer.

Sledgehammer normally schedules multiple runs of an ATP, allotting a time slice to each run and varying the number of relevant facts, type encoding, and prover options. For these experiments, we disabled slicing and instead focused on three parameters, varying one at a time: the number of facts selected by the relevance filter, the encoding of types, and the translation scheme for λ-abstractions. Each prover invocation was assigned 6 GB RAM and a single core clocked at 2.1 GHz for 30 wall-clock seconds.

Number of Facts. Figure 3 plots the success rate of the ATPs as a function of the maximum number of facts selected by Sledgehammer's relevance filter. The first-order ATPs were given problems with TFF0 types and supercombinators (λ-lifting), whereas the higher-order ATPs received THF0 types and λs. Because of monomorphisation, the generated problems may include up to 100 more axioms than there are facts selected.

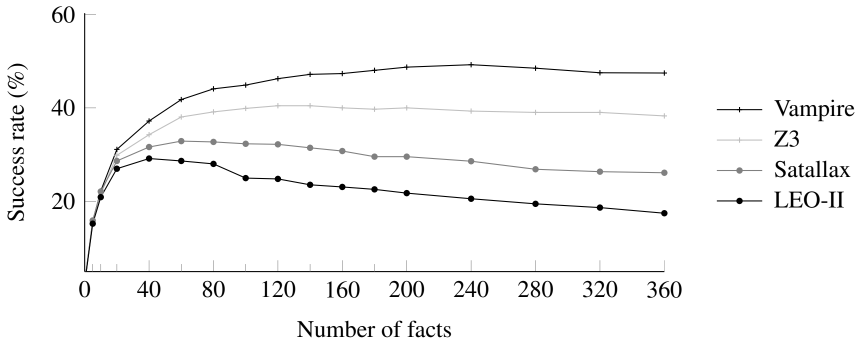

Figure 3: Scalability with the number of facts

Satallax's peak is at 60 facts, and performance degrades gracefully from that point. LEO-II's peak, at 40 facts, is lower and the degradation is slightly more rapid. Given that LEO-II is based on a highly optimised first-order prover and includes its own relevance filter, we could have expected it to scale better. We suspect that LEO-II's inefficient encoding of higher-order types in the untyped format supported by E is at cause. The next experiment will shed some light on this.

Types. Next, we want to measure the effectiveness of higher-order provers at handling higher-order formalisations and compare it to how they handle first-order translations of those same higher-order formalisations. In particular, higher-order problems are full of typing information. For a higher-order ATP to be successful, it must handle types efficiently. Although it is difficult to measure type handling directly, we can get a reasonably clear picture by exploiting Sledgehammer's type encoding machinery.

We consider three groups of type encodings, according to their target logic: untyped first-order logic (FOF), typed first-order logic (TFF0), and typed higher-order logic (THF0). For an

[4]Being LEO-II's backend, E arguably is a closer cousin than Vampire. However, it lacks support for types, which we use in our experiments.

untyped target, Sledgehammer provides a wide range of encodings [9, § 6.5]. Here we selected the four lightest sound encodings: polymorphic guards, polymorphic tags, monomorphic guards, and monomorphic tags.

The main difference between first-order (TFF0) types and higher-order (THF0) types concerns the function space: built-in rules such as β, η, and extensionality do not apply in the first-order case, nor can λ-abstractions be used to instantiate function variables.

		LEO-II		Satallax		Vampire		Z3	
		20 f	200 f	20 f	200 f	20 f	200 f	20 f	200 f
FO poly.	guards	22.1	7.0	22.2	20.2	30.7	41.6	27.7	39.9
	tags	20.2	7.5	19.3	15.4	28.4	41.6	27.7	**40.5**
FO mono.	guards	29.3	17.4	23.9	20.6	30.9	44.2	**29.5**	39.0
	tags	27.8	16.7	22.2	17.9	31.0	45.2	**29.5**	38.9
native FO mono. types		**29.7**	19.4	26.4	**26.4**	**31.1**	**48.0**	**29.5**	39.7
native HO mono. types		28.6	**21.9**	**27.0**	**26.4**	–	–	–	–

Table 4: Success rates (%) for the main type encodings

Table 4 shows the results of varying the type encoding. Each prover was given 20 and 200 facts (indicated as "20 f" and "200 f" in the heading), and the translation uniformly employed λ-lifting. We observe the following:

– Native first-order types perform better than any encoding into untyped first-order logic regardless of the prover, as one would expect.

– The gap between the first-order and higher-order ATPs on first-order problems indicates a need to focus on scalability to larger problems. Although LEO-II is based on E, it performs some reasoning of its own and relies on a heavy (tag-based) translation of types to first-order logic.

– LEO-II performs better on problems where Sledgehammer eliminated the higher-order features than on higher-order problems when only 20 facts are passed, but the situation is reversed for 200 facts. The crossover is at around 100 facts. This suggests that LEO-II focuses too much on the higher-order features of Isabelle problems.

– Monomorphisation-based encodings generally outperform polymorphic ones. Extrapolating from the data, we would expect a hypothetical higher-order polymorphic type encoding (as considered in Section 3.6) to score a few percentage points above the best first-order polymorphic encoding, falling short of overtaking the monomorphic encodings.

Considering the success of monomorphisation, the lack of support for polymorphism in LEO-II and Satallax is not dramatic.

λ-Abstractions. Having tested the encoding of type information, we now turn to the translation of λ-abstractions. We fix the type encoding to native first-order or higher-order monomorphic types (whichever are available) and consider both 20-fact and 200-fact problems.

We evaluate five λ translation schemes. The *combinator* and *λ-lifting* schemes were described in Section 2.4. The *disabled* scheme replaces λs by unspecified fresh constants, effectively disabling all reasoning under λs. The *hybrid* scheme unites combinators and λ-lifting: it characterises each λ-lifted constant both using a lifted equation $\mathsf{c}\ x_1 \ldots x_n = t$ and via combinators. The *native* scheme keeps the λs in the generated problem.

In the context of THF0, the last four schemes are equally powerful: the ATPs can in principle unfold all occurrences of (super)combinators, yielding native λs. To avoid aggressive unfolding, the auxiliary equations that characterise (super)combinators are not labelled as THF0 definitions.

Table 5 presents the results. Satallax behaves as expected: the prover is more successful if λs are represented by λs in the problem or, failing that, if they appear as (super)combinators. It came as a surprise that, at 200 facts, LEO-II performs best when λ reasoning is disabled. This seems to indicate that LEO-II is mishandling higher-order constructs, as we observed already in connection with Table 4.

	LEO-II		Satallax		Vampire		Z3	
	20 f	200 f	20 f	200 f	20 f	200 f	20 f	200 f
Disabled	26.7	**22.5**	25.6	26.5	29.5	45.9	28.5	37.8
λ-lifting	**28.6**	21.9	27.0	26.4	31.1	**48.0**	29.5	39.7
Combinators	26.3	19.0	26.0	24.8	31.2	45.6	30.2	39.9
Hybrid	27.0	19.5	26.7	24.5	**31.6**	45.5	**30.4**	**40.3**
Native	27.0	21.6	**28.5**	**28.8**	–	–	–	–

Table 5: Success rates (%) for the λ translation schemes

For years we have wondered how much Sledgehammer and the first-order ATPs are penalised by their rudimentary handling of higher-order constructs. LEO-II and Satallax are not yet powerful enough to provide a direct answer, but Tables 4 and 5 give an indication: in each table, exploiting Satallax's higher-order features increases its success rates by a few percentage points; by extrapolating the Vampire and Z3 numbers, we get an idea of what a "highly optimised Satallax" or "higher-order Vampire" would be capable of.

6. Related Work

There have been at least a dozen attempts at integrating first-order ATPs in proof assistants over the past two decades. These are discussed in more detail elsewhere [10, 29]. Here we restrict our focus to higher-order ATPs and their use in an interactive context. The venerable system TPS forms the core of the student-friendly proof assistant ETPS (Educational Theorem Proving System) [2]. LEO was designed primarily as a backend for ΩMEGA; proofs found by LEO were reconstructed by TRAMP [24]. The development of LEO-II, LEO's successor, was in part motivated by a desire to extend Isabelle's proving arsenal. Satallax can output Coq proofs [17]: Coq users can invoke the ATP to produce proof scripts for problems in a monomorphic simply typed fragment of type theory augmented with the classical axioms. In another mode of operation, Satallax produces proof terms that can be checked by Coq.

On the evaluation side, the TPTP library is the de facto standard benchmark suite for first-order and higher-order ATPs alike. Version 5.3.0 of the TPTP includes nearly 3000 THF0 problems for testing higher-order reasoners (both provers and counterexample generators) from various sources [5, 36]. Specialised evaluations are carried out annually as competitions. Starting with its 2009 edition, CASC [36] includes a higher-order proving division where Isabelle, LEO-II, Satallax, and TPS take part. Competitions give developers an opportunity to show what their tools are capable of. Evaluations such as that of Section 5 are different in that they give an indication of how much mileage users can expect to get with the same tools but without extensive tuning by the tools' developers.

7. Conclusion

This paper described an integration of higher-order automatic theorem provers (ATPs) in Isabelle/HOL via Sledgehammer. Despite their lack of maturity, LEO-II and especially Satallax can occasionally discharge interactive goals that are beyond the effective reach of first-order ATPs and built-in automatic tactics.

Since LEO-II is based on the first-order prover E, we hoped it would cope well with the largely first-order problems produced by Sledgehammer, but our experiments revealed some weaknesses. Such observations are harder to make for Satallax, because it is directly based on a SAT solver and hence does not benefit from the optimisations implemented in state-of-the-art first-order provers.

Both sides of the integration could obviously benefit from further work. On the Isabelle side, a dedicated reconstruction method could work more reliably for truly higher-order proofs than *metis* and *smt*; moreover, both LEO-II and Satallax provide dozens of options that could be investigated. On the ATP side, much could be done to bring the performance of LEO-II and Satallax closer to that of first-order provers; although monomorphisation works reasonably well in practice, native support for polymorphism would be both more elegant and more complete.

Acknowledgement. Tobias Nipkow made this collaboration possible. Christoph Benzmüller and Chad Brown gave advice on their systems and adapted them to accommodate our needs. Geoff Sutcliffe drove the development of TPTP THF0 and was involved in many discussions in connection with this paper. Christoph, Chad, and Geoff, as well as Brian Huffman, Andrei Popescu, and the anonymous reviewers suggested many improvements to drafts of this paper. Daniel Wand helped us draw Figure 3. We thank them all. The second author's research was supported by the Deutsche Forschungsgemeinschaft (grant Ni 491/11-2). Sledgehammer was originally supported by the UK's Engineering and Physical Sciences Research Council (grant GR/S57198/01).

References

[1] P. B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi. TPS: A theorem-proving system for classical type theory. *J. Autom. Reasoning*, 16(3):321–353, 1996.

[2] P. B. Andrews, C. E. Brown, F. Pfenning, M. Bishop, S. Issar, and H. Xi. ETPS: A system to help students write formal proofs. *J. Autom. Reasoning*, 32(1):75–92, 2004.

[3] J. Backes and C. E. Brown. Analytic tableaux for higher-order logic with choice. *J. Autom. Reasoning*, 47(4):451–479, 2011.

[4] C. Benzmüller. Private communication, January 2012.

[5] C. Benzmüller and C. E. Brown. A structured set of higher-order problems. In J. Hurd and T. F. Melham, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2005)*, volume 3603 of *LNCS*, pages 66–81. Springer, 2005.

[6] C. Benzmüller and M. Kohlhase. System description: LEO—A higher-order theorem prover. In C. Kirchner and H. Kirchner, editors, *Conference on Automated Deduction (CADE-15)*, volume 1421 of *LNAI*, pages 139–143. Springer, 1998.

[7] C. Benzmüller, L. C. Paulson, F. Theiss, and A. Fietzke. LEO-II—A cooperative automatic theorem prover for higher-order logic. In A. Armando, P. Baumgartner, and G. Dowek, editors, *International Joint Conference on Automated Reasoning (IJCAR 2008)*, volume 5195 of *LNAI*, pages 162–170. Springer, 2008.

[8] C. Benzmüller, F. Rabe, and G. Sutcliffe. THF0—The core of the TPTP language for higher-order logic. In A. Armando, P. Baumgartner, and G. Dowek, editors, *International Joint Conference on Automated Reasoning (IJCAR 2008)*, volume 5195 of *LNAI*, pages 491–506. Springer, 2008.

[9] J. C. Blanchette. *Automatic Proofs and Refutations for Higher-Order Logic*. Ph.D. thesis, Dept. of Informatics, T.U. München, 2012.

[10] J. C. Blanchette, S. Böhme, and L. C. Paulson. Extending Sledgehammer with SMT solvers. Submitted to *J. Autom. Reasoning*.

[11] J. C. Blanchette, A. Popescu, D. Wand, and C. Weidenbach. More SPASS with Isabelle—Superposition with hard sorts and configurable simplification. In *Interactive Theorem Proving (ITP 2012)*. Springer, 2012.

[12] F. Bobot and A. Paskevich. Expressing polymorphic types in a many-sorted language. In C. Tinelli and V. Sofronie-Stokkermans, editors, *Frontiers of Combining Systems (FroCoS 2011)*, volume 6989 of *LNCS*, pages 87–102. Springer, 2011.

[13] S. Böhme. *Proving Theorems of Higher-Order Logic with SMT Solvers*. Ph.D. thesis, Dept. of Informatics, T.U. München, 2012.

[14] S. Böhme and T. Nipkow. Sledgehammer: Judgement Day. In J. Giesl and R. Hähnle, editors, *International Joint Conference on Automated Reasoning (IJCAR 2010)*, volume 6173 of *LNAI*, pages 107–121. Springer, 2010.

[15] S. Böhme and T. Weber. Fast LCF-style proof reconstruction for Z3. In M. Kaufmann and L. Paulson, editors, *Interactive Theorem Proving (ITP 2010)*, volume 6172 of *LNCS*, pages 179–194. Springer, 2010.

[16] C. E. Brown. Reducing higher-order theorem proving to a sequence of SAT problems. In N. Bjørner and V. Sofronie-Stokkermans, editors, *Conference on Automated Deduction (CADE-23)*, volume 6803 of *LNAI*, pages 147–161. Springer, 2011.

[17] C. E. Brown. Satallax: An automated higher-order prover. In B. Gramlich, D. Miller, and U. Sattler, editors, *International Joint Conference on Automated Reasoning (IJCAR 2012)*. Springer, 2012.

[18] A. Church. A formulation of the simple theory of types. *J. Symb. Log.*, 5(2):56–68, 1940.

[19] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008)*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[20] N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *Satisfiability (SAT 2003)*, volume 2919 of *LNCS*, pages 502–518. Springer, 2004.

[21] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[22] W. Guttmann, G. Struth, and T. Weber. Automating algebraic methods in Isabelle. In S. Qin and Z. Qiu, editors, *International Conference on Formal Engineering Methods (ICFEM 2011)*, volume 6991 of *LNCS*, pages 617–632. Springer, 2011.

[23] G. Klein, T. Nipkow, and L. Paulson, editors. *The Archive of Formal Proofs*. `http://afp.sf.net/`.

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

[25] J. Meng and L. C. Paulson. Translating higher-order clauses to first-order clauses. *J. Autom. Reasoning*, 40(1):35–60, 2008.

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

[27] M. Moskal. Programming with triggers. In B. Dutertre and O. Strichman, editors, *Satisfiability Modulo Theories (SMT 2009)*, 2009.

[28] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[29] L. C. Paulson. Three years of experience with Sledgehammer, a practical link between automated and interactive theorem provers. In B. Konev, R. Schmidt, and S. Schulz, editors, *Practical Aspects of Automated Reasoning (PAAR-2010)*, 2010.

[30] L. C. Paulson and K. W. Susanto. Source-level proof reconstruction for interactive theorem proving. In K. Schneider and J. Brandt, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2007)*, volume 4732 of *LNCS*, pages 232–245. Springer, 2007.

[31] F. J. Pelletier, G. Sutcliffe, and C. Suttner. The development of CASC. *AI Commun.*, 15:79–90, 2002.

[32] A. Riazanov and A. Voronkov. The design and implementation of Vampire. *AI Commun.*, 15(2-3):91–110, 2002.

[33] S. Schulz. E—A brainiac theorem prover. *AI Commun.*, 15(2):111–126, 2002.

[34] N. Sultana and C. Benzmüller. Understanding LEO-II's proofs. In E. Ternovska, K. Korovin, and S. Schulz, editors,

International Workshop on the Implementation of Logics (IWIL-2012), 2012.

[35] G. Sutcliffe. The TPTP problem library and associated infrastructure—The FOF and CNF parts, v3.5.0. *J. Autom. Reasoning*, 43(4):337–362, 2009.

[36] G. Sutcliffe, C. Benzmüller, C. Brown, and F. Theiss. Progress in the development of automated theorem proving for higher-order logic. In R. Schmidt, editor, *Conference on Automated Deduction (CADE-22)*, volume 5663 of *LNAI*, pages 116–130. Springer, 2009.

[37] F. Theiß and C. Benzmüller. Term indexing for the LEO-II prover. In C. Benzmüller, B. Fischer, and G. Sutcliffe, editors, *International Workshop on the Implementation of Logics (IWIL-2006)*, volume 212 of *CEUR Workshop Proceedings*, 2006.

[38] C. Weidenbach. Combining superposition, sorts and splitting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 1965–2013. Elsevier, 2001.

[39] C. Weidenbach. Private communication, July 2012.

[40] M. Wenzel. Type classes and overloading in higher-order logic. In E. L. Gunter and A. Felty, editors, *Theorem Proving in Higher Order Logics (TPHOLs '97)*, volume 1275 of *LNCS*, pages 307–322. Springer, 1997.