Beagle as a HOL4 external ATP method

Thibault Gauthier[1], Cezary Kaliszyk[1], Chantal Keller[2], and Michael Norrish[3]

[1] University of Innsbruck, Austria
{thibault.gauthier,cezary.kaliszyk}@uibk.ac.at
[2] Microsoft Research – Inria Joint Centre, France
Chantal.Keller@inria.fr
[3] Canberra Research Laboratory, NICTA, Australia[†]
michael.norrish@nicta.com.au

Abstract

This paper presents BEAGLE_TAC, a HOL4 tactic for using Beagle as an external ATP for discharging HOL4 goals. We implement a translation of the higher-order goals to the TFA format of TPTP and add trace output to Beagle to reconstruct the intermediate steps derived by the ATP in HOL4. Our translation combines the characteristics of existing successful translations from HOL to FOL and SMT-LIB; however, we needed to adapt certain stages of the translation in order to benefit form the expressiveness of the TFA format and the power of Beagle. In our initial experiments, we demonstrate that our system can prove, without any arithmetic lemmas, 81% of the goals solved by Metis.

1 Introduction

Interactive theorem provers (ITPs) help researchers certify large or complex proofs, such as the proof of the Kepler conjecture, or modeling complex algorithms and systems. Currently, their main drawback is that they need a lot of human guidance. To solve this issue, for a number of common tasks automation is provided, in particular in the context of higher-order logic internal automated theorem provers (ATPs) based on model elimination (MESON [10]), tableau (Blast [19]) and resolution (Metis [12]). The internal implementation may be limited, by the need to interact with the ITP for every proved step.

By contrast, external ATPs can easily be optimized for faster proof search. For this reason, many ITPs exploit their performance by transforming problems to the syntax of various ATPs and calling them on the translated formulas. In this setting, internal ATPs provide a complementary role as they are helping to reconstruct the proof. Such use of external ATPs for premise selection has been successfully used by Isabelle/HOL [15], HOL Light [14] Mizar [22], and Coq [1].

In this paper we investigate the use of the TFA (TPTP) format as an interface between ITPs and ATPs. In order to do so we define and implement a translation from the higher-order logic prover HOL4 to the many-sorted first-order logic with arithmetic defined by the TFA format [21]. This allows using Beagle [2] — an automated theorem prover for first-order logic with equality over linear integer and rational arithmetic — to discharge HOL4 goals. As Beagle is a recent project, we provide a first evaluation of its performance on translated higher-order goals. In order to verify (and possibly reconstruct) the found proofs, we implemented basic proof traces in Beagle. The code described in this paper provides BEAGLE_TAC, a HOL4 tactic that performs the translation, writes the TFA goals and calls Beagle is available at `https://github.com/barakeel/HOLtoTFF`.

[†]NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

Recently, there has been the emergence of using bridges to ATPs for premise selection. In particular Sledgehammer [15] has became an almost indispensable tool for many Isabelle/HOL users, often simply employed instead of library search. Similar tools have been developed for HOL Light [14] and Mizar [13]. In the context of HOL4, an export to SMT-solvers Yices and Z3 is provided [9, 23]. We are not aware of any HOL4 translation to a superposition based ATP, of any translation from an ITP to a superposition based ATP that use modulo theory decision procedures, such as done by SPASS+T [20] and Beagle. SPASS+T uses external SMT-solvers whereas the decision procedures are built inside Beagle, which should make proof reconstruction easier.

Another approach is to implement ATPs inside ITPs, like the implementation of a SMT solver inside Coq [16] or the Metis [12] prover. This method gives a prover which is correct by construction. However, experiments [1] show that you get a better efficiency for complex provers when using external ATPs and checking their answers.

Our translation builds on the ideas of Hurd [11] further enhanced by Meng and Paulson [17]. The most related translation is the one done in Isabelle/HOL towards the SMT-solver Z3 together with the proof reconstruction are fully described in the PhD thesis of Böhme [8]. The procedure is now integrated as a part of premise selection in Sledgehammer together with the corresponding proof reconstruction method `smt` [4]. In this work we adapt the translation in a few ways to support Beagle and linear integer arithmetic. Namely the handling of nested predicates, polymorphic axioms and the mapping of natural numbers need to be performed differently to efficiently use our targeted ATP.

The rest of this paper is organized as follows. In Section 2 we describe our translation from HOL4 to the TFA. In Section 3 we present our experiments with the translation. Our experiments with Beagle proof traces and an initial investigation of proof reconstruction from such traces is explained in Section 4. Finally we conclude in Section 5 and present an outlook on the future work.

2 Translation

In this section, we present the details of our translation, explaining the choices and differences from the existing ones. The HOL4 implementation follows the order presented in this section except for the mapping of monomorphic types and arithmetic constants which are both performed during the printing phase.

2.1 TFA format

The TFA format is part of the TPTP family. It is used to express typed first-order problems with arithmetics and is an extension of the existing FOF format for untyped first-order logic. It contains the predefined basic types: $\$o$ reserved for the return type of predicates, $\$i$ for individuals (used by default if no type is declared), $\$int$, $\$rat$ and $\$real$ for interpreted arithmetics. The equality $\$equal$ is a polymorphic predicate over the basic types. Arithmetic predicates and functions are supported by interpreted overloaded symbols such as $\$less$, $\$uminus$ or $\$sum$.

A TFA problem consists of three parts: basic types (alphanumerical strings interpreted as disjoint sets), function and predicate symbols with their types, and formulas with their role (either axiom or conjecture) and statement. The type of a function or predicate symbol of arity $n \geq 0$ is represented by $(t_1 * \ldots * t_n) > t_{n+1}$ where $t_1 \ldots t_{n+1}$ are basic types. The basic type of a bound variable is given in a formula, at the quantifier position. The TFA format does not

support currying or subtyping and overloading and polymorphism are restricted to the defined predicates or functions.

2.2 Polymorphic Types

There are many ways of encoding polymorphic types into first-order logic [5]. Most of them are limited by the fact that first-order provers mainly support one-sorted logic. As a consequence, it is necessary to modify the formula (for instance, by adding predicates or function symbols), thus making the problem harder to solve by the ATPs. These encodings preserve soundness and completeness but compromise efficiency.

Since we want to preserve the arithmetic types of TFA, it is not possible to add predicates that express polymorphism. Therefore, we use an approach similar to a hard type encoding that recently showed good results in an experiment combining SPASS and Isabelle/HOL [6]. As Beagle handles natively the many-sorted logic of the TFA format, we directly map the HOL4 type of monomorphic variables and constants. For instance, a constant c used with arity 2 having the type $list[int] \to int \to int \to int$ is translated to $(list\_int \ * \ \$int) > int\_F\_int$. The type $\$int$ is the TFA reserved type for integers. The types $list\_int$ and int_F_int are created basic types meant to represent lists of integers and functions from integers to integers used as arguments (see defunctionalization, section 2.5).

In order to achieve a complete translation, an instantiation of the polymorphic problem needs to be performed. It has been shown by Bobot and Paskevich [7] that the problem of computing a finite set of ground instances of the polymorphic formulas such that the resulting ground formulas are equivalent to the original polymorphic formulas is undecidable. Nevertheless, heuristic finite monomorphization using an iterative procedure usually preserves provability. In our implementation, for each constant c in a theorem thm, we search for a constant in the whole problem of the same name with a less general type. The derived substitution is used to create an instantiated copy of the theorem thm and added to the problem. We repeat this process for every theorems a maximum of 3 times, with a limited number of instantiations. These parameters are good enough for small problems, as in the practical experiments performed in Section 3 we reach a fixed point in 73 percent of the cases.

We can derive the maximum number of instantiations by looking at possible instantiations not only in the conjecture but also in other provided theorems. Our instantiation algorithm is similar to the ones used by MESON [10] or by Sledgehammer [4], however we use a different recursion scheme and we limit the explosion of the instantiation process by introducing different kinds of bounds.

2.3 λ-abstractions

There are two commonly used ways of removing λ-abstractions from higher-order terms in order to translate them to first-order logic: λ-lifting and combinator encoding. The encoding using combinators is a complete one, which means that using the definitions of the combinators a first-order prover can construct an equivalent of every λ-abstraction. Even though this encoding has been shown [15] to be reasonable for pure ATPs, it is not as efficient as λ-lifting for SMT-solvers [4]. therefore most systems use λ-lifting. An additional advantage of λ-lifting is that it transforms formulas into ones that are close to the originals. Therefore we chose to use the incomplete λ-lifting making the TFA output more readable.

In higher-order logic, a formula is a term of type $bool$ and naturally sub-formulas are sub-terms of type $bool$. Let $abs = \lambda x_1 \dots x_n.\ t[x_1 \dots x_n, v_1 \dots v_m]$ be the most top left lambda-abstraction in a formula f, where t is a term with variables $x_1,\dots x_n$ bound in abs and $v_1,\dots,v_m$

free in *abs*. Let $P[abs]$ be the smallest sub-formula containing *abs*. Keeping in mind that free variables may be captured, this sub-formula is locally rewritten in f to:

$$P[abs := Abs'] \wedge \forall v1 \ldots v_m x_1 \ldots x_n.\ Abs'\ x_1 \ldots x_n = t[x_1 \ldots x_n, v_1 \ldots v_m]$$

where $Abs' = Abs\ v_1 \ldots v_m$ and *Abs* is a fresh symbol in the whole problem. There a few variations of this approach: instead of abstracting a term on the subformula level, it can be done on the whole formula level [3] or even on the whole problem level [22]. This allows for sharing copies of the same abstraction, which is very useful in bigger problems for premise selection. As our problems so far have been rather small, we have refrained from such optimizations.

Example 1. *(Free variable captured)*

$$\forall v.\ P\ (\lambda x.\ x + v) \ \rightsquigarrow\ \forall v.\ P\ (Abs\ v) \wedge \forall x\ v.\ (Abs\ v)\ x = x + v$$

2.4 Nested Formulas

In higher-order logic predicates are identified with terms of type *bool* and can appear as arguments of functions and other predicates. This is not possible in the first-order TPTP formats and this step rewrites the formula in order to collapse all formula levels, effectively removing formulas as arguments of non logical operators.

The most common approach for transforming such formulas into FOF is to name each sub-formula with a boolean variable and substitute all occurences of this sub-formula by this variable and add the variable's definition to the problem. This is commonly used in transformations to first-order logic, as it results in smaller problems [14]. When translating to Beagle the isomorphic variant of $\forall x : \$o.\ x = T \vee x = F$ cannot be provided to the ATP, because it leads to a non-terminating proof search, therefore in our translation we perform disjunctive cases on sub-formulas used as arguments which is costly as it creates two copies for each of them.

Let t be the most top left sub-formula used as argument in a formula f. Let $P[t]$ be the smallest sub-formula containing the formula t. This sub-formula is locally rewritten in f to:

$$(t \Rightarrow P[t := T]) \wedge (\neg t \Rightarrow P[t := F])$$

This translation leaves only the boolean T or F as possible arguments. Since the type returned by predicates $\$o$ is reserved in TFA, we create an isomorphic type *bool* and isomorphic booleans *btrue* and *bfalse* complemented by the axiom $btrue \neq bfalse$.

2.5 Defunctionalization

In higher-order logic, the same function (either constant or variable) can be applied with different arities. Defunctionalization transforms such problems into equivalent problems, where each function has a fixed number of arguments. In order to minimize defunctionalization the problem is first rewritten to a clause set. This frees existentially quantified functions which reduces the number of necessary applications of defunctionalization.

The process of defunctionalization introduces an apply functor. In our implementation we use a separate functor *App* for each type. In higher-order logic each functor is equivalent to identity, making it possible to rewrite using the equation $f\ x = App\ f\ x$, so that every function symbol f is used as an argument in the translated formula. This transformation can be performed extensively, making the transformation complete (together with the extensionality principle for each type). This makes the problems quite inefficient.

A commonly used variant (Sledgehammer, HOLyHammer) is to preserve the function symbol if it is free and used with its lowest arity relative to the whole problem. In order to be as close as possible to the complete version we additionally perform defunctionalization on constants which share the type with a universally quantified variable. This procedure is quite effective, however, it cannot be performed for arithmetic functions as it would prevent us from mapping them to their TFA counterparts. The way partially applied arithmetic functions could be treated, is by adding definitions of such constants in terms of the TFA ones. For the unary minus operation this would amount to the reflexive HOL equation $uminus(x) = uminus(x)$ translated to the FOL equation $App(uminus, x) = \$uminus(x)$. This is analogous to the way partially applied logical operators are translated by Sledgehammer. In Sledgehammer such partially applied logical operators occur very rarely, and adding such encoding more likely hinders the proof, therefore we did not implement such transformation for partially applied arithmetic so far.

2.6 Linear Integer Arithmetic

The translation of the integer arithmetic of HOL4 to the TFA format is straightforward. Indeed, the TFA predicates $\$less, \ldots$, and functions $\$sum, \ldots$ can be directly mapped to the constants $<, \ldots$ and $+, \ldots$ used with the same arity. Partial application of an arithmetic operator and non linear multiplications are left uninterpreted in this evaluation. Since natural numbers are not supported in the TFA format, we inject them into integers using these rewrite rules:

$$\forall x : num.\ F[x] \ \rightsquigarrow\ \forall x' : int.\ (x' \geq 0) \Rightarrow F[x']$$

$$F[x : num] \ \rightsquigarrow\ F[x'] \ \wedge (x' : int \geq 0)$$

$$F[f] \ \rightsquigarrow\ F[f'] \wedge \forall (x' : int)\ y.\ (x' \geq 0) \Rightarrow f'(x', y) \geq 0$$

The third rule is only presented with a function f of arity 2 having one numeral as its argument. A generalization of this rule is applied for every function returning a numeral. There is one difference between the translation used in Sledgehammer for SMT-solvers and ours. We use the second rule only for free variables and we add the third rule to complement it, making the translation more complete and compact but leaving more work for the prover. Rational and real linear arithmetic are also supported by Beagle and a similar mapping could easily be added.

3 Experiments

All tests were performed with Beagle version 0.7 and HOL4 repository version on a dual-core processor 2.1 GHz CPU with 3.7GB RAM. Beagle's timeout was set to 15 seconds per goal. We consider all HOL4 standard library goals that have been solved by the tactic METIS_TAC. We recall that METIS_TAC calls a first-order prover based on resolution without any theory reasoning; as a consequence, it must be fed with the theory lemmas that are needed to solve the goal. On the contrary, BEAGLE_TAC calls Beagle which does handle linear integer arithmetic, and is consequently left without any hint for this theory.

In the first experiment, we test BEAGLE_TAC on 271 of these goals without providing any arithmetic lemmas. To measure the impact of the pseudo-monomorphization procedure (sec. 2.2), we launch BEAGLE_TAC with and without monomorphization. During the test without monomorphization, polymorphic lemmas are left uninstantiated. Thus, polymorphic types are mapped to newly created monomorphic types. The results presented in table 1 demonstrate that we could solve 81% of Metis provable problems.

Without monomorphization				With monomorphization			
Unsat.	Satisfiable	Unknown	Timeout	Unsat.	Satisfiable	Unknown	Timeout
70 %	15%	7%	8%	81 %	2%	8%	9%

Table 1: Percentage of goals solved by BEAGLE_TAC

Since Beagle is not complete, it can give answer "Unknown" for the problems which it cannot prove or disprove. For the problems that Beagle can disprove, it answers "Satisfiable". Such problems arise, since our translation is incomplete. The Table 2 compares the time taken by BEAGLE_TAC (split into translation time, problem writing time, and time taken by Beagle) with the time taken by METIS_TAC. The tactic METIS_TAC is a lot faster than BEAGLE_TAC. The translation takes more time than METIS_TAC but Beagle is the limiting factor. This is mostly due to the weakness of our translation. We have to recall that unlike Metis, Beagle has to perform the arithmetic reasoning itself (without any arithmetic axioms). We will compare the impact of arithmetic reasoning below.

BEAGLE_TAC	Translation	Writing	Beagle	METIS_TAC
4.55	0.82	0.18	3.55	0.11

Table 2: Mean time in seconds

We will now compare the efficiency of BEAGLE_TAC on different classes of problems, by splitting our problem set into categories and comparing the number of solved problems and the average solving time depending on the category.

Higher-order problems In Table 3, the performance of BEAGLE_TAC is measured on first-order and higher-order problems. Despite BEAGLE_TAC's reasonable performance on first-order problems, it solves only half of the higher-order problems. A lot could be done to improve the supports for higher-order in our translation. The processes of defunctionalization and boolean instantiations can increase dramatically the size of the formulas and λ-lifting lacks completeness.

	Proportion	BEAGLE_TAC	Beagle	METIS_TAC
first-order	91.9%	2.71	2.48	0.13
higher-order	55.8%	11.07	7.36	0.04

Table 3: Higher-order efficiency and run-time in seconds

Polymorphic problems In Table 4, we evaluate the effectiveness of the pseudo-monomorphization procedure. The proportion of problems solved are similar for monomorphic and polymorphic problems and the time taken by our translation does not change, which means that our heuristic for instantiating polymorphic types is efficient and well-suited to Beagle's capabilities.

	Proportion	BEAGLE_TAC	Beagle	METIS_TAC
Non-arith.	81.6%	5.91	4.15	0.08
Arithmetic	79.6%	3.62	3.15	0.13

Table 5: Arithmetic efficiency and run-time in seconds

	Proportion	BEAGLE_TAC	Beagle	METIS_TAC
Monomorphic	81.1%	5.83	4.28	0.12
Polymorphic	79.9%	3.32	2.85	0.09

Table 4: Monomorphization efficiency and run-time in seconds

The results discussed above in Table 1 have already been split in two categories: with and without monomorphization. The monomorphization step enable BEAGLE_TAC to solve 11% more goals. However, we have to note that this step increases the size of the problems, which results in more "Timeout"s. Concerning the algorithm itself, a fixed point has been found in 102 out of 139 (73%) polymorphic problems.

Arithmetic problems In Table 5, we separate the problems containing at least one arithmetic constant. Again, the proportion of problems solved are similar, which shows that Beagle handles TFA arithmetic well. We can further note that the time taken by BEAGLE_TAC is lower on arithmetic problems whereas the time taken by METIS_TAC increases, which indicates that Beagle's built-in arithmetic decision procedures are efficient.

Named theorems We can reprove 258 out of 270 (95%) HOL4 named theorems involving only arithmetics and/or pure higher-order terms (containing no uninterpreted constants). Some proofs fail because the original problems did not include extensionality, so that this property was not carried out to defunctionalized functions in the TFA translated version. Other proofs fail, since we do not declare the finiteness of the boolean type. This can be expressed in first-order logic, and in fact a HOL4 theorem that expresses this property could be added to the problems.

4 Reconstruction

An important characteristic of interactive theorem proof systems is the fact that every derived theorem is completely certified. An unverified call to an external prover compromises this property. It could be, that Beagle or the printing phase of our translation may happen to be unsound. However, reconstructing a proof that involves external theories is a challenging task for a number of reasons. The ATP may not output which small steps it actually performs. Replaying each of the small steps may be complicated as every single step may require a lot of specialized code and proofs to be simulated. Nevertheless, there are a number of successful examples of checking SMT-solver proofs: the integration of veriT into Coq [1] and reconstruction of Z3 proofs in Isabelle/HOL and HOL4 [9].

We have investigated the first steps towards reconstructing the proof found by Beagle in HOL4. We have implemented a minimal proof trace functionality in Beagle and added a parser

for trace in HOL4. The proof trace contains a list of clauses, as well as information about the usage of the split rule. The Beagle main loop maintains two sets of clauses, see Fig. 1. The old set contains clauses that have been used as a premise at least once, whereas the new set contains clauses that have been derived, but so far have not been used as a premise. Moreover, when the split rule is used Beagle creates derived loops in a DPLL way. In our output, we store every clause that is being added to the old set, together with the additional information about the current level of splitting, as well as tagging the clauses which were split. When Beagle successfully proves the conjecture, we can reconstruct a tree-like structure, where each vertex represents the application of a split rule and each edge stands for a list of clauses sorted in order of creation.

In order to successfully reconstruct all Beagle proofs, a lot more work is required, as all the rules need to be recorded and replayed. For instance, the simplification rule combines arithmetic with first-order logic reasoning, making the steps proved by this rule not easily automatically provable in HOL4. We have tried to reconstruct it by combining the application of the HOL4 arithmetic decision procedure COOPER_TAC [18] and METIS_TAC, however this fails in some cases. As Beagle is still in development and prone to many updates, we decided to postpone implementing code for the reconstruction of such rules. It may be possible to reconstruct the proof in two stages: in the first stage using Beagle for lemma filtering with arithmetic as done in HOLyHammer and Sledgehammer in their interaction with pure ATPs, and in the second stage turn off the arithmetic mapping and find also the necessary arithmetic lemmas to be given to Metis.

5 Conclusion

In this paper, we investigated the suitability of the TFA format as an interface between ITPs and ATPs, by implementing an interface between the interactive proof system HOL4 and an automated system Beagle. We tested the efficiency of Beagle on translated higher-order formulas, and showed that it could prove, without any arithmetic lemmas, 81% of the goals provable by Metis. The translation from HOL4 into a TPTP format is greatly inspired by similar translations (especially Sledgehammer), however we had to adapt certain stages of the translation to the capacities of Beagle (transformation of nested predicates, handling of polymorphic assumptions, etc.) and respecting the constraints of the TFA format. We mapped naturals and integers so that we could benefit form the linear integer arithmetic decision procedure of Beagle. Furthermore, we have introduced a minimal proof trace in Beagle, as a first step toward replaying the proof.

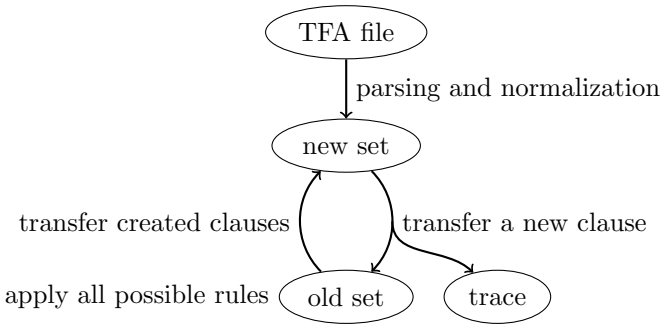

Figure 1: Beagle main loop augmented with the trace

The project helped reveal a few bugs in Beagle, which have been fixed by its authors now.

BEAGLE_TAC is more expressive than METIS_TAC, as it combines first-order logic with linear integer arithmetic, so we believe that HOL4 users could already benefit from our tactic. However many optimizations are still necessary for BEAGLE_TAC to compete with other proof methods available in HOL4. A mapping for real numbers and rationals should be provided in order to evaluate the capabilities of Beagle for these domains. We expect that like in other experiments an external ATP/SMT can outperform an internal one as we increase the number of lemmas, so a proper evaluation with a larger number of premises should be carried out. In a different line of work our translation could directly print its results to the TDFG format, so as to compare the strength of Beagle and SPASS+T. A long term future work is creating a "hammer-system" for HOL4 that would implement a relevance filter combined with ATP-based premise selection. This later project could also be combined with the bridge from HOL4 to Z3.

Acknowledgement

Peter Baumgartner and Josh Bax explained us the internals of Beagle and helped us find an issue in the initial version of our translation; Alexis Saurin reviewed a previous version of this paper; Jasmin Blanchette provided us with many useful references.
This work has been supported by the Austrian Science Fund (FWF): P26201.

References

[1] Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. A modular integration of SAT/SMT solvers to Coq through proof witnesses. In Jean-Pierre Jouannaud and Zhong Shao, editors, *CPP*, volume 7086 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2011.

[2] Peter Baumgartner and Uwe Waldmann. Hierarchic superposition with weak abstraction. In Maria Paola Bonacina, editor, *CADE*, volume 7898 of *Lecture Notes in Computer Science*, pages 39–57. Springer, 2013.

[3] Jasmin C. Blanchette. *Automatic Proofs and Refutations for Higher-order Logic*. PhD thesis, Technische Universität München, 2012.

[4] Jasmin C. Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending Sledgehammer with SMT solvers. *J. Autom. Reasoning*, 51(1):109–128, 2013.

[5] Jasmin C. Blanchette, Sascha Böhme, Andrei Popescu, and Nicholas Smallbone. Encoding monomorphic and polymorphic types. In Nir Piterman and Scott A. Smolka, editors, *TACAS*, Lecture Notes in Computer Science, pages 493–507. Springer, 2013.

[6] Jasmin C. Blanchette, Andrei Popescu, Daniel Wand, and Christoph Weidenbach. More SPASS with Isabelle - Superposition with hard sorts and configurable simplification. In Lennart Beringer and Amy P. Felty, editors, *ITP*, volume 7406 of *Lecture Notes in Computer Science*, pages 345–360. Springer, 2012.

[7] François Bobot and Andrey Paskevich. Expressing polymorphic types in a many-sorted language. In Cesare Tinelli and Viorica Sofronie-Stokkermans, editors, *FroCoS*, volume 6989 of *Lecture Notes in Computer Science*, pages 87–102. Springer, 2011.

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

[9] Sascha Böhme and Tjark Weber. Fast LCF-style proof reconstruction for Z3. In Matt Kaufmann and Lawrence Paulson, editors, *Interactive Theorem Proving*, volume 6172 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2010.

[10] John Harrison. Optimizing proof search in model elimination. In Michael A. McRobbie and John K. Slaney, editors, *CADE*, volume 1104 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 1996.

[11] Joe Hurd. First-order proof tactics in higher-order logic theorem provers. *Design and Application of Strategies/Tactics in Higher Order Logics, number NASA/CP-2003-212448 in NASA Technical Reports*, pages 56–68, 2003.

[12] Joe Hurd. System description: The Metis proof tactic. *Empirically Successful Automated Reasoning in Higher-Order Logic (ESHOL)*, pages 103–104, 2005.

[13] Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *CoRR*, abs/1310.2805, 2013.

[14] Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *Journal of Automated Reasoning*, 2014. `http://dx.doi.org/10.1007/s10817-014-9303-3`.

[15] Jasmin C. Blanchette Lawrence C. Paulson. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In S. Schulz G. Sutcliffe, E. Ternovska, editor, *IWIL-2010*, 2010.

[16] Stéphane Lescuyer. *Formalizing and Implementing a Reflexive Tactic for Automated Deduction in Coq*. PhD thesis, Université Paris-Sud, 2011.

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

[18] Michael Norrish. Complete integer decision procedures as derived rules in HOL. In *Theorem Proving in Higher Order Logics, TPHOLs 2003, volume 2758 of Lect. Notes in Comp. Sci*, pages 71–86. Springer-Verlag, 2003.

[19] Lawrence C. Paulson. A generic tableau prover and its integration with Isabelle. *J. UCS*, 5(3):73–87, 1999.

[20] Virgile Prevosto and Uwe Waldmann. SPASS+T, 2006.

[21] Geoff Sutcliffe, Stephan Schulz, Koen Claessen, and Peter Baumgartner. The TPTP typed first-order form with arithmetic. In Nikolaj Bjørner and Andrei Voronkov, editors, *LPAR*, volume 7180 of *Lecture Notes in Computer Science*, pages 406–419. Springer, 2012.

[22] Josef Urban. Automated reasoning for Mizar: Artificial intelligence through knowledge exchange. In Piotr Rudnicki, Geoff Sutcliffe, Boris Konev, Renate A. Schmidt, and Stephan Schulz, editors, *LPAR Workshops*, volume 418 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[23] Tjark Weber. SMT solvers: new oracles for the HOL theorem prover. *International Journal on Software Tools for Technology Transfer*, 13(5):419–429, 2011.