Set of Support for Higher-Order Reasoning

Ahmed Bhayat and Giles Reger

University of Manchester, Manchester, UK

Abstract. Higher-order logic (HOL) is utilised in numerous domains from program verification to the formalisation of mathematics. However, automated reasoning in the higher-order domain lags behind first-order automation. Many higher-order automated provers translate portions of HOL problems to first-order logic (FOL) and pass them to FOL provers. However, FOL provers are not optimised for dealing with these translations. One of the reasons for this is that the axioms introduced during the translation (e.g. those defining combinators) may combine with each other during proof search, deriving consequences of the axioms irrelevant to the goal. In this work we evaluate the extent to which this issue affects proof search and introduce heuristics based on the set-of-support strategy for minimising the effects. Our experiments are carried out within the Vampire theorem prover and show that limiting how axioms introduced during translation can improve proof search with higher-order problems.

1 Introduction

Most automated theorem provers for higher-order logic (HOL) utilise a first-order (FO) theorem prover to discharge some of the proof burden [3]. However, this can lead to inefficiencies in practice due to FO provers not being optimised for the often unwieldy translations. The main goal of this work is to extend the Vampire theorem prover [9] to efficiently support higher-order reasoning. This involves extending proof search to handle the axioms arising from translation from HOL to FOL in a special way. Our main focus is a translation from a fragment of TPTP's higher-order form to FOL implemented within Vampire but we also look at a sample of output from the Sledgehammer tool [12].

This work builds on previous work [13] for controlling the use of theory axioms when using Vampire to reason about theories such as real and integer arithmetic. In this previous work (and the work presented here) the idea is to extend the set-of-support strategy to avoid inferences which cause the search space to grow too quickly with irrelevant consequences of the theory. Indeed, we can see the axioms introduced in the HOL translations as a theory of combinators and logical connectives (see later for further details).

Our preliminary results indicate that seeking to control the use of such axioms in proof search is important and that the set-of-support strategy can be helpful in doing this. The rest of the paper is organised as follows. We first introduce relevant background (Section 2), we then present the translation from higher-order logic to first-order logic (Section 3), next we report on some initial analysis

of the use of higher-order axioms in proof search within Vampire (Section 4), this is followed by a description of the set-of-support strategy (Section 5) and its evaluation (Section 6) before some final concluding remarks (Section 7).

2 Background

We introduce the necessary background of first-order and high-order logic, saturation-based theorem proving in Vampire, set of support reasoning, and the tools for automated higher-order reasoning.

2.1 First-order and Higher-order Logic

We consider a many-sorted first-order logic with equality. A term is either a variable, a constant, or a function symbol applied to terms. Function symbols are *sorted* i.e. their arguments and the return value have a unique *sort* drawn from a finite set of sorts S. We only consider well-sorted literals. There is an equality symbol per sort and equalities can only be between terms of the same sort. Here we consider a fragment of first-order logic where all atoms are equalities and literals are atoms or their negation. That is, we treat predicates as functions of boolean sort and include a *theory of booleans* (see [7,8]). Vampire supports the use of predicates natively but our translation does not make use of them. At this point we note that the theory of booleans requires the addition of the axioms: $\mathit{true} \neq \mathit{false}$ and $(\forall x : \mathit{bool})(x = \mathit{true} \vee x = \mathit{false})$ as well as congruence axioms for functions. Previous work [8] explores the addition of an additional inference rule to remove the need for these axioms. We draw attention to this as the translation from HOL to FOL introduces a lot of boolean structure and dealing with booleans becomes important when reasoning with the output of this translation.

Formulas (in FOL and HOL) may use the standard notions of quantification and logical connectives, but in this work we assume all formulas are *clausified* using standard techniques. A *clause* is a disjunction of literals where all variables are universally quantified (existentially quantified variables can be replaced by Skolem functions during clausification). We assume the reader is familiar with the standard semantics of FOL.

The main difference between FOL and HOL is that in the former variables can only range over individuals and not over functions. Below a version of HOL syntax based on the simply-typed lambda calculus is presented. The terms of HOL are defined over a signature $\Sigma = (F, S)$ where F is a set of function symbols and S is a set of atomic sorts containing o the sort of truth values, and ι the sort of individuals. For each type, it is assumed that there exists a countably infinite set of variables with that type.

Types	$\tau ::= \sigma \mid \tau_1 \to \tau_2$	where $\sigma \in S$ and τ_1 and τ_2 are types
Terms	$t ::= x_\tau \mid c_\tau \mid t1_{\tau_1 \to \tau_2} t2_{\tau_1} \mid \lambda x_\tau . t$	where c is a constant of type τ

It is assumed that the set of function symbols F contains at least the logical constants $\neg_{o\to o}$, $\vee_{o\to o\to o}$ and, for each type τ, constants $\Pi^{\tau}_{(\tau\to o)\to o}$ and $=^{\tau}_{\tau\to\tau\to o}$ for universal quantification and primitive equality. The remaining logical constants can be defined in terms of these. HOL formulas are defined as terms of type o.

In the semantics of higher-order logic, a model is a pair $(\{D_\tau|\tau \in T\}, I)$, where T is a set of sorts and $\{D_\tau\}$ is a set of frames, one for each sort $\tau \in T$. Each frame is itself a set with $D_o = \{\bot, \top\}$ and $D_{\tau\to\sigma}$ being a set of functions from D_τ to D_σ. The function I maps each $f_\sigma \in F$ to an element of D_σ and is chosen such that the logical constants have their expected denotation.

There are two main semantics for higher-order logic. In *standard semantics*, the set $D_{\tau\to\sigma}$ is taken to be the set of all possible functions with domain D_τ and range D_σ. In contrast, in *Henkin* or *general* semantics the frame $D_{\tau\to\sigma}$ is only assumed to contain the minimal necessary number of members.

A variable assignment is a function that maps each variable X_τ to a member of D_τ. A valuation function V from terms of higher-order logic to their denotations can be defined in the obvious manner. A term t_o is valid in a model M if, for all variable assignments ϕ, $V_\phi\ t_0 = \top$. For further details on the semantics of higher-order logic refer to [2].

In our work, we target a fragment of higher-order logic with Henkin semantics. The fragment targeted does not contain choice or extensionality. Henkin semantics is more or less standard in the automated higher-order logic community. The higher-order section of the TPTP library stipulates Henkin semantics as the intended semantics for all problems. Adding choice and extensionality axioms to a search space is known to have an adverse effect proof search. An option for future experiments would be to add a strategy to Vampire which runs with these axioms included.

2.2 Saturation-based Reasoning in the Vampire Theorem Prover

Theorem provers such as Vampire are refutational and saturation-based. The idea is that an input formula of the form *Premises* $\to$ *Conjecture* is negated, to give *Premises* $\wedge \neg$*Conjecture*, then clausified to produce a set of clauses S. This set is then *saturated* with respect to some inference system $\mathcal{I}$ meaning that for every inference from $\mathcal{I}$ with premises in S the conclusion of the inference is also in S. If the saturated set S contains a contradiction then the initial formula is necessarily valid. Otherwise, if $\mathcal{I}$ is a complete inference system, and importantly the requirements for this completeness have been preserved, then S is satisfiable and the input formula is not valid. Note that in our context of higher-order reasoning we are never complete but we mention this notion here as it is key to the approach. Finite saturation may not be possible and a lot of research over the past 50 years has focussed on how to control this saturation-based proof search to make finding a contradiction more likely. This is the focus of this paper.

The standard approach to saturation is the *given-clause* algorithm illustrated in Figure 1. The idea is to have an active set of clauses with the invariant that all inferences between active clauses have been performed and a passive set of

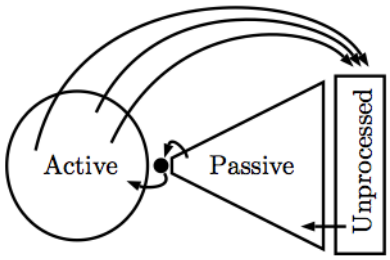

Fig. 1: Illustrating the Given Clause Algorithm.

clauses waiting to be activated. The algorithm then iteratively selects a *given clause* from passive and performs all necessary inferences to add it to active. The process of clause selection is important but not discussed here.

Vampire uses resolution and superposition as its inference system $\mathcal{I}$. A key feature of this calculus is the use of *literal selection* and *orderings* to restrict the application of inference rules, thus restricting the growth of the clause sets. Another very important concept related to saturation is the notion of *redundancy*. It is not key to this paper so we do not discuss the details but the idea is that some clauses in S are *redundant* in the sense that they can be safely removed from S without changing what can be derived. The notion of saturation then becomes saturation-up-to-redundancy.

As a final note, from this discussion it may appear that completeness of $\mathcal{I}$ is important. But as we showed in our recent work on literal selection [6], breaking the conditions required for completeness can be helpful in proof search. Additionally, the focus of this paper is in higher-order reasoning, where the previous completeness arguments for FOL no longer hold.

2.3 Set of Support

The set of support strategy was introduced in [16,10] as a method for restricting the possible inferences. The idea is to split the clauses into a *set of support* and the rest, and then restrict inferences so that they must include at least one clause that is in the set of support, with new clauses being added to the set of support. Another way of phrasing this is that all inferences must have a clause in the initial set of support as an ancestor.

If this description feels familiar then it should as the above given-clause algorithm is based on the set of support strategy. Here the set of support is the passive set and the rest of the clauses must be immediately added to the active set, breaking the invariant that all inferences between clauses in this set have been performed. This immediately breaks the condition required for completeness. Note that our context of higher-order reasoning also introduces larger barriers for completeness. We are interested in how this strategy can be used to prevent the explosion of axioms coming from the HOL to FOL translation.

In the general set of support strategy it is not prescribed which clauses should be added to the set of support, but it is reasonable to assume that it should at least include the goal, thus encouraging goal-directed reasoning. This is the approach taken in Vampire where only the conjecture is added to the set of support. We note here that this means that set of support can only be used for problems containing a conjecture which excludes many problems in TPTP and all problems in SMTLIB [1].

In Vampire there are three set of support related options:

- `off`: do not perform set of support reasoning,
- `on`: perform set of support reasoning performing standard literal selection,
- `all`: perform set of support reasoning selecting all literals in those clauses initially added to active. This helps mitigate some of the incompleteness introduced by the set of support strategy.
- `theory`: apply set of support reasoning for theory axioms only (see below and Section 5).

Our previous work [13] in extending set of support for theory reasoning demonstrates that (i) set of support is a generally useful strategy, and (ii) it can be adapted to give more fine-grained control over the impact of including axioms from some theory in proof search.

2.4 Automated Higher-order Reasoning

There exists a rich literature in techniques for automating higher-order reasoning. A portion of this centres on techniques for translating higher-order to first-order logic. It is this previous work that motivates us as Vampire has often been used to discharge the first-order problems produced by such translations.

The higher-order theorem prover Isabelle is well-known as an interactive prover, but due to the possibility of chaining powerful automatic proof tactics together, it can be also run as an automatic prover. Notable amongst these tactics is Sledgehammer [12] which translates the current goal to first-order logic and then calls a number of first-order provers (including SMT solvers [4]) on this translation in parallel [11].

The fully automated higher-order prover LEO-III utilises a calculus based on paramodulation, equality and primitive substitution. It implements special inference rules to deal with choice, function synthesis and extensionality [14]. LEO-III also cooperates with first-order provers by regularly translating the clause set to first-order logic and passing it to a prover.

The translation utilised by both Sledgehammer and LEO-III differs minimally from that implemented in Vampire, details of which can be found in Section 3. One difference between the Vampire translation and the Isabelle/LEO translation is related to higher-order functions. If a higher-order function f always appears in the input problem with at least n arguments, then both Sledgehammer and LEO-III will translate it to a first order function of arity n. Vampire, on the other hand, deals with *all* higher-order application via the binary *app*

function discussed below. Vampire's translation results in a more nested structure (which can be problematic for proof search), but is more complete. For example, if the function symbol f always appears with two arguments in the non-Vampire translation this would appear as $f(t_1, t_2)$ and first-order unification with $\neg app(app(X, t_1), t_2)$ would fail.

Another important automated high-order prover is Satallax [5]. Satallax implements a tableaux calculus, attempting to ground the formulas on a branch. The proof search maintains a set of propositional clauses linked to the set of formulas. If this set of clauses ever becomes unsatisfiable, the original problem is unsatisfiable and Satallax records the input problem as a theorem.

3 Translating HOL to FOL

Higher-order logic supports a number of features not supported in first-order logic. To translate from HOL to FOL we need to deal with each of these features. These features are as follows:

1. partial application: $f_{\sigma_1 \to \sigma_2 \to \tau}\ c_{\sigma_1}$
2. applied variables: $\forall x_{\tau \to \tau}.x\ a_\tau = x\ b_\tau$
3. lambda binders: $\lambda x.f\ x$
4. boolean sub-terms: $p_{o \to o}(x_o \vee y_o)$

The following sections explain how Vampire deals with each feature in its translation (which, as mentioned previously, is similar to that of Sledgehammer and Leo-III). Lambda expressions are translated using combinators rather than lambda-lifting for a couple of reasons. Firstly, the implementation of the combinator translation was relatively straightforward allowing the quick evaluation of various heuristics prior to the exploration of other translations. Secondly, lambda-lifting is incomplete. The idea behind the Vampire translation, is to achieve as complete a translation as possible, and then control the search space with heuristics and special inference rules.

Note that the translation results in a fully first-order set of formulas and therefore no modification is required to Vampire's data structures, proof procedures or heuristics.

3.1 Translating Partial Application and Applied Variables

All higher-order application, whether of constants or variables, is dealt with via a two place application function *app*. More precisely, let $[\![]\!]$ be our translation from higher-order to first-order logic. Then if st is any higher-order application term, $[\![st]\!] = app([\![s]\!], [\![t]\!])$. For example the higher-order term $f(Xa)$ would be translated to the first order term $app(f, app(X, a))$. Note that higher-order function symbols are translated to first-order constants.

All higher-order types are translated to first-order sorts. If, in the example given above, f has higher-order type $o \to \iota$ and $X\ a$ has type o then the outer *app* function would be of type $('o \to \iota' \times o) \to \iota$ where $'o \to \iota'$ is a first-order sort.

$$\begin{aligned}
\mathbf{I} &\equiv \lambda x.x \\
\mathbf{K} &\equiv \lambda x.\lambda y.x \\
\mathbf{C} &\equiv \lambda x.\lambda y.\lambda z.(x\ z\ y) \\
\mathbf{B} &\equiv \lambda x.\lambda y.\lambda z.(x\ (y\ z)) \\
\mathbf{S} &\equiv \lambda x.\lambda y.\lambda z.(x\ z\ (y\ z))
\end{aligned}$$

Fig. 2: Turner combinators.

3.2 Translating Lambdas

Lambda-expressions are translated to first-order utilising the extended set of Turner combinators given in Figure 2. The translation of a term t containing a lambda prefix is defined inductively:

1. if $t = \lambda x.x$ then $[\![t]\!] = iCOM$
2. if $t = \lambda x.t_1 x$ and x is not free in t_1 then $[\![t]\!] = [\![t_1]\!]$
3. if $t = \lambda x.t_1$ and x is not free in t_1 then $[\![t]\!] = app(kCOM, [\![t_1]\!]))$
4. if $t = \lambda x.t_1 t_2$ and x is free in t_2, not t_1 then $[\![t]\!] = app(app(bCOM, [\![t_1]\!]), [\![\lambda x.t_2]\!])$
5. if $t = \lambda x.t_1 t_2$ and x is free in t_1, not t_2 then $[\![t]\!] = app(app(cCOM, [\![\lambda x.t_1]\!])[\![t_2]\!])$
6. if $t = \lambda x.t_1 t_2$ and x is free in t_1 and t_2 then $[\![t]\!] = app(app(sCOM, [\![\lambda x.t_1]\!])[\![\lambda x.t_2]\!])$

For each combinator introduced in the translation, its defining axiom is added to the clause set. In particular instances of the following 5 axiom schemas are added as relevant.

1. $\forall X : \tau_1, Y : \tau_2, Z : \tau_3.app(app(app(sCOM, X), Y), Z) = app(app(X, Z), app(Y, Z))$
2. $\forall X : \tau_1, Y : \tau_2, Z : \tau_3.app(app(app(bCOM, X), Y), Z) = app(X, app(Y, Z))$
3. $\forall X : \tau_1, Y : \tau_2, Z : \tau_3.app(app(app(cCOM, X), Y), Z) = app(app(X, Z), Y)$
4. $\forall X : \tau_1, Y : \tau_2.app(app(kCOM, X), Y) = X$
5. $\forall X : \tau_1.app(iCOM, X) = X$

Note that the translation is to many-sorted first-order logic and thus the above axioms are in reality *axiom schemas*. In the actual translation, sorted versions of these axioms would be used.

3.3 Translating Logical Operators

In higher-order logic, it is possible for terms of boolean sort to be function arguments. Further, logical operators can be partially applied. Both of these are disallowed in first-order logic. Where high-order logical operators do not occur partially applied nor as function arguments, these are mapped to their first-order counterparts. Thus, $[\![f\ a \vee f\ b]\!] \rightarrow app(f, a) \vee app(f, b)$. Where logical constants appear as arguments, but are not within the scope of any lambdas, Vampire deals with this via a process known as *renaming* [7]. However, renaming within the scope of a lambda could cause bound variable to become free and thus logical constants appearing in this context are translated to uninterpreted first-order

constants. For example: $[\![\lambda X : o, Y : o \;.\; X \wedge Y]\!] = [\![\lambda X : o, Y : o \;.\; [\![X \wedge Y]\!]]\!] = [\![\lambda X : o, Y : o \;.\; app(app(vAND, X), Y)]\!]$. Here $[\![]\!]$ again represents our translation function and $vAND$ is a first-order constant of sort $o \to o \to o$.

Axioms are added to provide these constants with the desired semantics. The axioms are:

1. $\forall X : o, Y : o.\ app(app(vOR, X), Y) = true \iff X = true \vee Y = true$
2. $\forall X : o, Y : o.\ app(app(vAND, X), Y) = true \iff X = true \wedge Y = true$
3. $\forall X : o, Y : o.\ app(app(vIMP, X), Y) = true \iff X = true \implies Y = true$
4. $\forall X : o, Y : o.\ app(app(vIFF, X), Y) = true \iff (X = true \iff Y = true)$
5. $\forall X : o, Y : o.\ app(app(vXOR, X), Y) = true \iff X = true \oplus Y = true$
6. $\forall X : o, Y : o.\ app(app(vEQUALS, X), Y) = true \iff X = Y$
7. $\forall X : o.\ app(vNOT, X) = true \iff X \neq true$
8. $\forall X : \tau \to o.\ (app(vPI, X) = true \iff \forall Y : \tau.\ app(X, Y) = true)$
9. $\forall X : \tau \to o.\ (app(vSIGMA, X) = true \iff \exists Y : \tau.\ app(X, Y) = true)$

3.4 Examples of Translation

Consider the following higher-order formula:

$$mforall = \lambda Phi_{(i\to o)\to i\to o}, W_i.(\forall P_{i\to o} : Phi\ P\ W)$$

Let $[\![]\!]$ be the Vampire translation function. The first few steps the function would take to translate the above formula are given below.

$$\begin{aligned}
[\![mforall &= \lambda Phi_{(i\to o)\to i\to o}, W_i.(\forall P_{i\to o} : Phi\, P\, W)]\!] \\
[\![mforall]\!] &= [\![\lambda Phi_{(i\to o)\to i\to o}, W_i.(\forall P_{i\to o} : Phi\, P\, W)]\!] \\
mforall &= [\![\lambda Phi_{(i\to o)\to i\to o}.[\![\lambda W_i.(\forall P_{i\to o} : Phi\, P\, W)]\!]]\!] \\
mforall &= [\![\lambda Phi_{(i\to o)\to i\to o}.[\![\lambda W_i.[\![\forall P_{i\to o} : Phi\, P\, W]\!]]\!]]\!] \\
mforall &= [\![\lambda Phi_{(i\to o)\to i\to o}.[\![\lambda W_i.[\![vPI_{((i\to o)\to o)\to o}\,(\lambda P_{i\to o} : Phi\, P\, W)]\!]]\!]]\!] \\
mforall &= [\![\lambda Phi_{(i\to o)\to i\to o}.[\![\lambda W_i.app(vPI_{((i\to o)\to o)\to o}, [\![\lambda P_{i\to o} : Phi\, P\, W]\!])]\!]]\!] \\
&\vdots \\
mforall &= app(app(bCOM, app(bCOM, vPI)), cCOM)
\end{aligned}$$

The final formula above has not been annotated with sort and type information in order to keep it reasonably compact. However, it should be noted that the sorts of two $bCOM$ constants and the types of the two app functions are different.

Table 1: Maximum depth of pure descendants of HOL axioms in proofs.

Depth	Count
0	758
1	69
2	279
3	12
4	22
5	21
$6-8$	4
total (>0)	1165 (407)

Table 2: Percentage of clause search space consisting of clauses derived purely from HOL axioms.

	0	$>1\%$	$>2\%$	$>5\%$	$>10\%$	$>20\%$	$>30\%$	$>40\%$	$>50\%$
solved	672	443	192	98	56	40	36	31	24
unsolved	0	987	774	333	201	108	29	12	0

4 Analysing the Impact of Axioms on Proof Search

We analyse the impact of axioms introduced in the translation on proof search via the following experiment. We take 2,369 higher-order TPTP problems and run Vampire in default mode using the translation from HOL to FOL described in the previous section. We record (i) the maximum depth of pure descendants of HOL axioms in proofs, and (ii) the percentage of the clause search space consisting of clauses derived purely from HOL axioms in both successful and unsuccessful proof search attempts. By depth here we mean the maximum number of inference steps between a clause and a member of the initial clause set.

The results are given in Tables 1 and 2. Table 1 gives the number of problems solved with a certain depth of pure HOL axiom descendants. This shows us that the majority of cases (almost 70%) do not require us to combine any axioms introduced during proof search and most (almost 90%) only require combinations up to depth 2. This suggests that restricting the depth of such combinations will remove unnecessary work from proof search. Table 2 gives the number of problems where the percentage of pure HOL descendants in the search space is greater than a given percent for solved and unsolved problems (where zero is a special case). This shows us that in the majority of cases such combinations do not dominate the search space. However, in some cases they do and this is more prevalent in cases where Vampire does not solve the problem, perhaps suggesting that this could be a contributing factor. The evidence for combinations of HOL axioms significantly impacting proof search is not overwhelming but strong enough to be worthy of investigation. Looking at the 24 problems where the percentage is over 50% we have 16 problems with more than 90% of the search space being used with a maximum of 96%.

5 Set of Support for Reasoning with Higher-order Axioms

In this section we briefly describe how the notion of set of support is extended to control the explosive nature of axioms introduced during translation. This is broadly similar to the approach taken for theory axioms [13].

The general idea is to treat the whole input problem, with the exception of such axioms, as the set of support. However, as we saw in the previous section we sometimes need combinations of these axioms to find a proof. To allows this we implement an approach that attempts to saturate the axioms up to some depth during proof search (not preprocessing). The idea is to block all inferences that would produce a clause derived only from the HOL axioms at a certain depth. To achieve this we attach a flag and a counter to each clause where the flag indicates whether the clause is a pure consequence of HOL axioms and the counter counts the maximum number of inference steps between the clause and a HOL axiom, i.e. the clause's depth in the proof. Inferences are updated to update these values as appropriate. Whenever we generate a pure consequence of HOL axioms with depth larger than a given threshold we delete this clause.

Whilst testing the approach implemented above we noticed a number of problems where the proof contained very shallow reasoning with input clauses and deep reasoning with combinator axioms e.g. to derive a function to satisfy some existential claim. In such cases the inverse of the above approach could be helpful. To explore whether this was a useful approach we implemented an option which acted as above but with the roles of HOL axioms and other clauses reversed.

The techniques described above rely on the translation being carried out by Vampire. However, there is nothing here that relies on control over the translation process; we only need to be able to identify HOL axioms. It happens that problems produced by the Slegehammer tool in TPTP format use the "help_" prefix for the name of any axiom introduced during translation[1]. We add an option to detect axioms with such names and identify them as HOL axioms.

Finally, during testing we also observed that the axioms introduced for the theory of booleans also have a significant impact on proof search in some cases. Whilst a direct inference rule (FOOL paramodulation [8]) has been introduced to tackle this, we also noted that the set of support approach for theory reasoning introduced in [13] could be applied to this theory.

These discussion lead to the following changes/additions to Vampire options:

- The *sos* option is extended with three new values: `hol`, `hol_inverse`, and `both` where `both` turns both `hol` and `theory` on.
- A new option `sos_hol_limit` (`sshl`) is added to limit the depth of inferences with pure HOL axioms. This is in addition to the `sos_theory_limit` (`sstl`) introduced in [13].
- A new option `detect_sledgehammer_axioms` (`dsa`) will attempt to detect HOL axioms introduced by Sledgehammer.

[1] Thanks to Jasmin Blanchette for pointing this out to us.

Table 3: Results of running different set-of-support options on higher-order TPTP problems. Numbers in brackets are unique solutions. A - indicates the experiment was not run.

		Problems solved		
		sos=		
sshl	sstl	off	hol	both
0	0	1,163	1,145	1,190 (4)
1	0	-	960	-
2	0	-	1,114	1,161
5	0	-	1,158	-
10	0	-	1,162	-
0	2	-	-	1,146
1	1	-	-	992 (1)
2	0	-	-	1,161 (2)
2	2	-	-	1,121
10	10	-	-	1,164

6 Experimental Evaluation

The experiments described in this section use problems from TPTP v7.0.0 and were run on the StarExec cluster [15] where each node contains an Intel Xeon 2.4GHz processor. In experiments we run Vampire 4.2.1 extended with the options previously described.

6.1 Evaluating new Options

In this experiment we consider some combinations of the new options introduced in the previous Section and attempt to draw some conclusions about their benefit. We have not evaluated the full space of options as some were introduced late in the development and evaluation process. We select all relevant higher-order problems from TPTP e.g. those in higher-order form that are Theorems and exclude 93 problems that we cannot currently parse. This gives us a total of 2,369 problems. We run for 60 seconds with all options other than those specified set to their default values.

Table 3 presents the results of these experiments. Using both kinds of set-of-support where no descendants of HOL axioms are allowed improves on using no set-of-support at all. It is interesting to note that the best performance was achieved when boolean axioms were restricted along with HOL axioms. This is also where all unique solutions were found (solutions found by a single set of options).

The most significant result, not present in the above table, is that overall strategies employing set-of-support in some way solve 56 problems not solved by that without set-of-support (`off`). This is a strong indicator that including the set-of-support variations discussed here in a portfolio of techniques will be of benefit.

Table 4: Results of running different set-of-support options on TPTP Sledgehammer problems. Numbers in brackets are unique solutions. A - indicates the experiment was not run.

	`off`	`hol`	`hol_inverse`
0	276	277 (1)	273
1	-	273 (1)	273
5	-	276	-
10	-	276	240

Furthermore, there are 64 cases where the difference in solution times between the best set-of-support strategy and `off` is less than 1 second. If we just consider these 64 cases the average speedup using the set-of-support strategy is 179 times - 31 cases have a speedup of at least 100x. Effectively this means that solutions that were taking tens of seconds are now happening immediately.

As a comparison, this approach is currently not competitive with the leading automated higher-order provers. On the same set of benchmarks Satallax [5] solves 2,155 problems and Leo-III [14] solves 2,216 problems. However, there were 16 problems solved by Vampire not solved by either of these solvers.

6.2 Looking at Problems from Sledgehammer

We identified 364 problems inside the TPTP library that (i) make use of the Sledgehammer translation, (ii) use axioms that can be identified using the approach introduced previously, and (iii) can be parsed by Vampire. This is the point in our experiments where we introduced the `hol_inverse` option. Sledgehammer axioms include those describing booleans. Therefore, in these experiments we can view `hol` as meaning `both` with the same depth for both (and for combinations of the two, which did not apply previously).

This time we run in the CASC portfolio mode which runs a number of different strategies in sequence. The results are given in Table 4. In terms of overall problems solved the set-of-support strategies did not make much impact. However, strategies employing set-of-support in some way solve 10 problems not solved by that without set-of-support (`off`), out of these 8 were solved by strategies using the `hol_inverse` option. This time the average speedup was much more modest at around 2.1x with 29 problems being solved faster out of 53 problems where the solution time varied by more than 1 second (so 24 solutions were slower with set-of-support strategies).

Table 5 repeats the exercise of finding the depth of pure descendants of HOL axioms in proofs. Here we see two cases with very deep combinations. These were for `SWW246+1.p` and `SWW255+1.p`. This exercise reinforces our previous conclusion that the depth of such reasoning is generally small.

Table 5: Maximum depth of pure descendants of HOL axioms in proofs of problems from Sledgehammer.

Depth	Count
0	227
1	2
2	19
3	19
4	4
5	1
6	0
7	2
8	0
12	1
13	1
total > 0	40

7 Discussion and Conclusions

This paper has introduced a preliminary study aiming to understand the impact of axioms introduced during translation from HOL to FOL on proof search in first-order saturation-based theorem provers such as Vampire. We also presented some preliminary work using an extension of the set-of-support strategy (which was previously useful for theory reasoning) to improve proof search in such cases. Our results show that such approaches could help but indicate that our current implementation has room for improvement.

This works leaves a number of open questions, which suggest possible future directions:

Detecting HOL axioms. For our work on Sledgehammer problems we were able to take advantage of the fact that the axioms we wanted to treat differently were annotated. What can we do if this is not the case? Many of the axioms have a particular syntactic form and it seems plausible that this would be enough in many cases.

Are all HOL axioms equal. In this work we have treated all of the different classes of axioms arising from the FOL to HOL translation in the same way. For example, combinator axioms and axioms for logical symbols. Do they have the same impact on proof search? Related experiments with theory axioms suggest that separating into different clauses would have a positive impact but comes with the added cost of a larger space of proof search parameters.

What about non-pure axioms. In this work we have ignored all axioms that are not pure HOL axiom descendants but perhaps this is a spectrum – perhaps axioms that are 'close' to the pure set should be treated differently from those that are 'far'. Does it help to quantify this notion and refine our notion of depth

accordingly? The technical solution to quantifying closeness does not seem too complex but it is unclear whether this will help.

When to do this. In this work we have chosen to restrict the depth of the pure set of descendants during proof search. The alternatives would be to perform a pre-saturation of this set or to statically (offline) produce such a set. The offline approach will not work in general as the set of axioms is not known statically due to their parameterisation by sort. The pre-saturation approach may involve unnecessary computation as we saw earlier that many problems required no pure descendants. We have no data on how many 'unnecessary' descendants are produced but it is almost certain that we would do not normally saturate to the given depth as the normal proof search heuristics will guide exploration of the search space. Whilst we still believe doing this during proof search makes the most sense, we should explore the alternatives.

Common patterns. A final point is that there may be some common consequences of pure HOL axioms that are used frequently in proofs and should be added automatically. When we looked at the same idea with theory axioms we found that such consequences were very shallow and adding them had no measurable impact but the case may be different in this setting and should be explored.

References

1. Clark Barrett, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010.
2. Christoph Benzmüller, Chad E. Brown, and Michael Kohlhase. Higher-order semantics and extensionality. *Journal of Symbolic Logic*, 69, 12 2004.
3. Christoph Benzmüller, Nik Sultana, Lawrence C. Paulson, and Frank Theib. The higher-order prover leo-ii. *Journal of Automated Reasoning*, 55(4):389–404, 2015.
4. Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending sledgehammer with SMT solvers. *J. Autom. Reasoning*, 51(1):109–128, 2013.
5. Chad E. Brown. Satallax: An automatic higher-order prover. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning*, pages 111–117, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
6. Kryštof Hoder, Giles Reger, Martin Suda, and Andrei Voronkov. Selecting the selection. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning: 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 – July 2, 2016, Proceedings*, pages 313–329, Cham, 2016. Springer International Publishing.
7. Evgenii Kotelnikov, Laura Kovács, Giles Reger, and Andrei Voronkov. The vampire and the FOOL. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, Saint Petersburg, FL, USA, January 20-22, 2016*, pages 37–48, 2016.
8. Evgenii Kotelnikov, Laura Kovács, and Andrei Voronkov. A first class boolean sort in first-order theorem proving and TPTP. In *Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*, pages 71–86, 2015.

9. Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV 2013*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35, 2013.
10. William McCune. OTTER 2.0. In *10th International Conference on Automated Deduction, Kaiserslautern, FRG, July 24-27, 1990, Proceedings*, pages 663–664, 1990.
11. Jia Meng and Lawrence C. Paulson. Translating higher-order clauses to first-order clauses. *Journal of Automated Reasoning*, 40(1):35–60, Jan 2008.
12. Lawrence C Paulson and Jasmin Christian Blanchette. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. *IWIL-2010*, 1, 2010.
13. Giles Reger and Martin Suda. Set of support for theory reasoning. In Thomas Eiter, David Sands, Geoff Sutcliffe, and Andrei Voronkov, editors, *IWIL Workshop and LPAR Short Presentations*, volume 1 of *Kalpa Publications in Computing*, pages 124–134. EasyChair, 2017.
14. Alexander Steen and Christoph Benzmüller. The higher-order prover leo-iii. *CoRR*, abs/1802.02732, 2018.
15. Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. StarExec, a cross community logic solving service. `https://www.starexec.org`, 2012.
16. Lawrence Wos, George A. Robinson, and Daniel F. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *J. ACM*, 12(4):536–541, October 1965.