Premise Selection and External Provers for HOL4

Thibault Gauthier Cezary Kaliszyk
University of Innsbruck
{thibault.gauthier,cezary.kaliszyk}@uibk.ac.at

Abstract

Learning-assisted automated reasoning has recently gained popularity among the users of Isabelle/HOL, HOL Light, and Mizar. In this paper, we present an add-on to the HOL4 proof assistant and an adaptation of the HOL(y)Hammer system that provides machine learning-based premise selection and automated reasoning also for HOL4. We efficiently record the HOL4 dependencies and extract features from the theorem statements, which form a basis for premise selection. HOL(y)Hammer transforms the HOL4 statements in the various TPTP-ATP proof formats, which are then processed by the ATPs.

We discuss the different evaluation settings: ATPs, accessible lemmas, and premise numbers. We measure the performance of HOL(y)Hammer on the HOL4 standard library. The results are combined accordingly and compared with the HOL Light experiments, showing a comparably high quality of predictions. The system directly benefits HOL4 users by automatically finding proofs dependencies that can be reconstructed by Metis.

Categories and Subject Descriptors I.2.3 [*Artificial intelligence*]: Inference engines

Keywords HOL4; higher-order logic; automated reasoning; premise selection

1. Introduction

The HOL4 proof assistant [24] provides its users with a full ML programming environment in the LCF tradition. Its simple logical kernel and interactive interface allow safe and fast developments, while the built-in decision procedures can automatically establish many simple theorems, leaving only the harder goals to its users. However, manually proving theorems based on its simple rules is a tedious task. Therefore, general purpose automation has been developed internally, based on model elimination (MESON [9]), tableau (blast [19]), or resolution (Metis [11]). Although essential to HOL4 developers, the methods are so far not able to compete with the external ATPs [16, 22] optimized for fast proof search with many axioms present and continuously evaluated on the TPTP library [25] and updated with the most successful techniques. The TPTP (Thousands of Problems for Theorem Provers) is a library of test problems for automated theorem proving (ATP) systems. This standard enables convenient communication between different systems and researchers.

On the other hand, the HOL4 system provides a functionality to search the database for theorems that match a user chosen pattern. The search is semi-automatic and the resulting lemmas are not necessarily helpful in proving the conjecture. An approach that combines the two: searching for relevant theorems and using automated reasoning methods to (pseudo-)minimize the set of premises necessary to solve the goal, forms the basis of "hammer" systems such as Sledgehammer [20] for Isabelle/HOL, HOL(y)Hammer [15] for HOL Light or MizAR for Mizar [12]. Furthermore, apart from syntactic similarity of a goal to known facts, the relevance of a fact can be learned by analyzing dependencies in previous proofs using machine learning techniques [28], which leads to a significant increase in the power of such systems [17].

In this paper, we adapt the HOL(y)Hammer system to the HOL4 system and test its performance on the HOL4 standard library. The libraries of HOL4 and HOL Light are exported together with proof dependencies and theorem statement features; the predictors learn from the dependencies and the features to be able to produce lemmas relevant to a conjecture. Each problem is translated to the TPTP FOF format. When an ATP finds a proof, the necessary premises are extracted. They are read back to HOL4 as proof advice and given to Metis for reconstruction.

An adapted version of the resulting software is made available to the users of HOL4 in interactive session, which can be used in newly developed theories. Given a conjecture, the SML function computes every step of the interaction loop and, if successful, returns the conjecture as a theorem:

Example 1. (HOL(y)Hammer interactive call)

```
load "holyHammer";
  val it = (): unit
holyhammer ``1+1=2``;
  Relevant theorems: ALT_ZERO ONE TWO ADD1
  metis: r[+0+6]#
  val it = |- 1 + 1 = 2: thm
```

The HOL4 prover already benefits from export to SMT solvers such as Yices [31], Z3 [4] and Beagle [8]. These methods perform best when solving problems from the supported theories of the SMT solver. Comparatively, HOL(y)Hammer is a general purpose tool as it relies on ATPs without theory

CPP'15, January 12–14 2015, Mumbai, India..

ACM 978-1-4503-3296-5/15/01.
http://dx.doi.org/10.1145/2676724.2693173

reasoning and it can provide easily[1] re-provable problem to Metis.

The HOL4 standard distribution has since long been equipped with proof recording kernels [18, 32]. We first considered adapting these kernels for our aim. But as machine learning only needs the proof dependencies and the approach based on full proof recording is not efficient, we perform minimal modifications to the original kernel.

Contributions We provide learning assisted automated reasoning for HOL4 and evaluate its performance in comparison to that in HOL Light. In order to do so, we :

- Export the HOL4 data

  Theorems, dependencies, and features are exported by a patched version of the HOL4 kernel. It can record dependencies between theorems and keep track on how their conjunctions are handled along the proof. We export the HOL4 standard libraries (58 types, 2305 constants, 11972 theorems) with respect to a strict name-space rule so that each object is uniquely identifiable, preserving if possible its original name.

- Reprove

  We test the ability of a selection of external provers to reprove theorems from their dependencies.

- Define accessibility relations

  We define and simulate different development environments, with different sets of accessible facts to prove a theorem.

- Experiment with predictors

  Given a theorem and a accessibility relation, we use machine learning techniques to find relevant lemmas from the accessible sets. Next, we measure the quality of the predictions by running ATPs on the translated problems.

The rest of this paper is organized as follows. In Section 2 we describe the export of the HOL4 and HOL Light data into a common format and the recording of dependencies in HOL4. In Section 3, we present the different parameters: ATPs, proving environments, accessible sets, features, and predictions. We select some of them for our experiments and justify our choice. In Section 4 we present the results of the HOL4 experiments, relate them to previous HOL(y)Hammer experiments and explain how this affects the users. Finally in Section 5 we conclude and present an outlook on the future work.

2. Sharing HOL data between HOL4, HOL Light and HOL(y)Hammer

In order to process HOL Light and HOL4 data in a uniform way in HOL(y)Hammer, we export objects from their respective theories, as well as dependencies between theorems into a common format. The export is available for any HOL4 and HOL Light development. We shortly describe the common format used for exporting both libraries and present in more detail our methods for efficiently recording objects (types, constants and theorems) and precise dependencies in HOL4. We will refer to HOL(y)Hammer [15] for the details on recording objects and dependencies for HOL Light formalizations.

HOL Light and HOL4 share a common logic (higher-order logic with implicit shallow polymorphism), however their implementations differ both in terms of the programming language used (OCaml and SML respectively), data structures used to represent the terms and theorems (higher-order abstract syntax and de Bruijn indices respectively), and the exact inference rules provided by the kernel. As HOL(y)Hammer has been initially implemented in OCaml as an extension of HOL Light, we need to export all the HOL4 data and read it back into HOL(y)Hammer, replacing its type and constant tables. The format that we chose is based on the TPTP THF0 format [27] used by higher-order ATPs. Since formulas contains polymorphic constants which is not supported by the THF0 format, we will present an experimental extension of this format where the type arguments of polymorphic constants are given explicitly.

Example 2. (experimental template)

```
tt(name, role, formula)
```

The field name is the object's name. The field role is "ty" if the object is a constant or a type, and "ax" if the object is a theorem. The field formula is an experimental THF0 formula.

Example 3. (Object export from HOL4 to an experimental format)

- Type

```
(list,1) → tt(list, ty, $t > $t).
```

- Constant

```
(HD,‘‘:’a list -> :’a‘‘) →
  tt(HD, ty, ![A:$t]: (list A > A).
(CONS,‘‘:’a -> :’a list -> :’a list‘‘) →
  tt(CONS ,ty, ![A:$t]: (A > list A > list A).
```

- Theorem

```
(HD,‘‘∀ n:int t:list[int]. HD (CONS n t) = n‘‘) →
  tt(HD0, ax, (![n:int, t:(list int)]:
    ((HD int) ((CONS int) n t) = n).
```

In this example, `$t` is the type of all basic types.

All names of objects are prefixed by a namespace identifier, that allow identifying the prover and theory they have been defined in. For readability, the namespace prefixes have been omitted in all examples in this paper.

2.1 Creation of a HOL4 theory

In HOL4, types and constants can be created and deleted during the development of a theory. These objects are named at the moment they are created. A theorem is a SML value of type *thm* and can be derived from a set of basic rules, which is an instance of a typed higher-order classical logic. To distinguish between important lemmas and theorems created by each small steps, the user can name and delete theorems (erase the name). Each named object still present at the end of the development is saved and thus can be called in future theories.

There are two ways in which an object can be lost in a theory: either it is deleted or overwritten. As proof dependencies for machine learning get more accurate when more intermediate steps are available, we decided to record all created objects, which results in the creation of slightly bigger theories. As the originally saved objects can be called from

[1] reconstruction rate is typically above 90%

other theories, their names are preserved by our transformation. Each lost object whose given name conflicts with the name of a saved object of the same type is renamed.

Deleted objects The possibility of deleting an object or even a theory is mainly here to hide internal steps or to make the theory look nicer. We chose to remove this possibility by canceling the effects of the deleting functions. This is the only user-visible feature that behaves differently in our dependency recording kernel.

Overwritten objects An object may be overwritten in the development. As we prevent objects from being deleted, the likelihood of this happening is increased. This typically happens when a generalized version of a theorem is proved and is given the same name as the initial theorem. In the case of types and constants, the internal HOL4 mechanism already renames overwritten objects. Conversely, theorems are really erased. To avoid dependencies to theorems that have been overwritten, we automatically rename the theorems that are about to be overwritten.

2.2 Recording dependencies

Dependencies are an essential part of machine learning for theorem proving, as they provide the examples on which predictors can be trained. We focus on recording dependencies between named theorems, since they are directly accessible to a user. The time mark of our method slows down the application of any rules by a negligible amount.

Since the statements of 951 HOL4 theorems are conjunctions, sometimes consisting of many toplevel conjuncts, we have refined our method to record dependencies between the toplevel conjuncts of named theorems.

Example 4. (Dependencies between conjunctions)

```
ADD_CLAUSES: 0 + m = m ∧ m + 0 = m ∧
SUC m + n = SUC (m + n) ∧ m + SUC n = SUC (m + n)

ADD_ASSOC depends on:
  ADD_CLAUSES_c1: 0 + m = m
  ADD_CLAUSES_c3: SUC m + n = SUC (m + n)
  ...
```

The conjunct identifiers of a named theorem `T` are noted `T_c1`, ..., `T_cN`.

In certain theorems, a toplevel universal quantifier shares a number of conjuncts. We will also split the conjunctions in such cases recursively. This type of theorem is less frequent in the standard library (203 theorems).

Example 5. (Conjunctions under quantifier)

```
MIN_0: ∀ n. (MIN n 0 = 0) ∧ (MIN 0 n = 0)
```

By splitting conjunctions we expect to make the dependencies used as training examples for machine learning more precise in two directions. First, even if a theorem is too hard to prove for the ATPs, some of its conjuncts might be provable. Second, if a theorem depends on a big conjunction, it typically depends only on some of its conjuncts. Even if the precise conjuncts are not clear from the human-proof, the reproving methods can often minimize the used conjuncts. Furthermore, reducing the number of conjuncts should ease the reconstruction.

2.3 Implementation of the recording

The HOL4 type of theorems *thm* includes a tag field in order to remember which oracles and axioms were necessary to prove a theorem. Each call to an oracle or axiom creates a theorem with the associated tag. When applying a rule, all oracles and axioms from the tag of the parents are respectively merged, and given to the conclusion of the rule. To record the dependencies, we added a third field to the tag, which consists of a dependency identifier and its dependencies.

Example 6. (Modified tag type)

```
type tag = ((dependency_id, dependencies),
            oracles, axioms)
type thm = (tag, hypotheses, conclusion)
```

Since the name of a theorem may change when it is overwritten, we create unmodifiable unique identifiers at the moment a theorem is named.

It consists of the name of the current theory and the number of previously named theorems in this theory. As a side effect, this enables us to know the order in which theorems are named which is compatible by construction with the pre-order given by the dependencies. Every variable of type *thm* which is not named is given the identifier *unnamed*. Only identifiers of named theorems will appear in the dependencies.

We have implemented two versions of the dependency recording algorithm, one that tracks the dependencies between named theorems, other one tracking dependencies between their conjuncts. For the named theorems, the dependencies are a set of identified theorems used to prove the theorem. The recording is done by specifying how each rule creates the tag of the conclusion from the tag of its premises. The dependencies of the conclusion are the union of the dependencies of the unnamed premises with its named premises.

This is achieved by a simple modification of the `Tag.merge` function already applied to the tags of the premises in each rule.

When a theorem $\vdash A \wedge B$ is derived from the theorems $\vdash A$ and $\vdash B$, the previously described algorithm would make the dependencies of this theorem the union of the dependencies of the two. If later other theorems refer to it, they will get the union as their dependencies, even if only one conjunct contributes to the proof. In this subsection we define some heuristics that allow more precise tracking of dependencies of the conjuncts of the theorems.

In order to record the dependencies between the conjuncts, we do not record the conjuncts of named theorems, but only store their dependencies in the tags. The dependencies are represented as a tree, in which each leaf is a set of conjunct identifiers (identifier and the conjunct's address). Each leaf of the tree represents the respective conjunct c_i in the theorem tree and each conjunct identifier represents a conjunct of a named goal to prove c_i.

Example 7. (An example of a theorem and its dependencies)

```
Th0 (named theorem): A ∧ B
Th1: C ∧ (D ∧ E)
     with dependency tree Tree([Th0],[Th0_c2])

This encodes the fact that:
  C depends on Th0.
  D ∧ E depends Th0_c2 which is B.
```

Dependencies are combined at each inference rule application and dependencies will contain only conjunct identifiers. If not specified, a premise will pass on its identifier if it is a named conjunct (conjunct of a named theorem) and its dependency tree otherwise. We call such trees passed dependencies. The idea is that the dependencies of a named conjunct should not transmit its dependencies to its children but itself. Indeed, we want to record the direct dependencies and not the transitive ones.

For rules that do not preserve the structure of conjunctions, we flatten the dependencies, i.e. we return a root tree containing the set of all (conjunct) identifiers in the passed dependencies. We additionally treat specially the rules used for the top level organization of conjunctions: `CONJ`, `CONJUNCT1`, `CONJUNCT2`, `GEN`, `SPEC`, and `SUBST`.

- `CONJ`: It returns a tree with two branches, consisting of the passed dependencies of its first and second premise.
- `CONJUNCT1` (`CONJUNCT2`): If its premise is named, then the conjunct is given a conjunct identifier. Otherwise, the first (second) branch of the dependency tree of its premise become the dependencies of its conclusion.
- `GEN` and `SPEC`: The tags are unchanged by the application of those rules as they do not change the structure of conjunctions. Although we have to be careful when using `SPEC` on named theorems as it may create unwanted conjunctions. These virtual conjunctions are not harmful as the right level of splitting is restored during the next phase.

  Example 8. (Creation of a virtual conjunction from a named theorem)

  ```
  ∀ x.x ⊢ ∀ x.x
                                  SPEC [A ∧ B]
  ∀ x.x ⊢ A ∧ B
                                  CONJUNCT1
  ∀ x.x ⊢ A
  ```

- `SUBST`: Its premises consist of a theorem, a list of substitution theorems of the form $(A = B)$ and a template that tells where each substitution should be applied. When `SUBST` preserves the structure of conjuncts, the set of all identifiers in the passed dependencies of the substitution theorems is distributed over each leaf of the tree given by the passed dependencies of the substituted theorems. When it is not the case the dependency should be flattened. Since the substitution of sub-terms below the top formula level does not affect the structure of conjunctions, it is sufficient (although not necessary) to check that no variables in the template is a predicate (is a boolean or returns a boolean).

The heuristics presented above try to preserve the dependencies associated with single conjuncts whenever possible. It is of course possible to find more advanced heuristics, that would give more precise human-proof dependencies. However, performing more advanced operations (even pattern matching) may slow down the proof system too much; so we decided to restrict to the above heuristics.

Before exporting the theorems, we split them by recursively distributing quantifiers and splitting conjunctions. This gives rise to conflicting degree of splitting, as for instance, a theorem with many conjunctions may have been used as a whole during a proof. Given a theorem and its dependency tree, each of its conjunctions is given the set of identifiers of its closest parent in this tree. Then, each of these identifiers is also split maximally. In case of a virtual conjunction (see the `SPEC` rule above), the corresponding node does not exist in the theorem tree, so we take the conjunct corresponding to its closest parent. Finally, for each conjunct, we obtain a set of dependencies by taking the union of the split identifiers.

Example 9. (Recovering dependencies from the named theorem Th1)

```
Th0 (named theorem): A ∧ B
Th1 (named theorem): C ∧ (D ∧ E)
     with dependency tree Tree([Th0],[Th0_c1])

  Recovering dependencies of each conjunct
Th1_c0: Th0
Th1_c1: Th0_c1
Th1_c2: Th0_c1
  Splitting the dependencies
Th1_c0: Th0_c1 Th0_c2
Th1_c1: Th0_c1
Th1_c2: Th0_c1
```

Prover	Version	Premises
Vampire	2.6	96
E-prover	1.8	128
z3	4.32	32
CVC4	1.3	128
Spass	3.5	32
IProver	1.0	128
Metis	2.3	32

Table 1. ATP provers, their versions and arguments

3. Evaluation

In this section we describe the setting used in the experiments: the ATPs, the transformation from HOL to the formats of the ATPs, the dependencies accessible in the different experiments, and the features used for machine learning.

3.1 ATPs and problem transformation

HOL(y)Hammer supports the translation to the formats of various TPTP ATPs: FOF, TFF1, THF0, and two experimental TPTP extensions. In this paper we restrict ourselves to the first order monomorphic logic, as these ATPs have been the most powerful so far and integrating them in HOL4 already poses an interesting challenge. The transformation that HOL(y)Hammer uses is heavily influenced by previous work by Paulson [21] and Harrison [9]. It is described in detail in [15], here we remind only the crucial points. Abstractions are removed by β-reduction followed by λ-lifting, predicates as arguments are removed by introducing existentially quantified variables and the apply functor is used to reduce all applications to first-order. By default HOL(y)Hammer uses the tagged polymorphic encoding [3]: a special tag taking two arguments is introduced, and applied to all variable instances and certain applications. The first argument is the first-order flattened representation of the type, with variables functioning as type variables and the second argument is the value itself.

The initially used provers, their versions and default numbers of premises are presented in Table 1. The HOL Light experiments [15] showed, that different provers perform best

with different given numbers of premises. This is particularly visible for the ATP provers that already include the relevance filter SInE [10], therefore we preselect a number of predictions used with each prover. Similarly, the strategies that the ATP provers implement are often tailored for best performance on the TPTP library, for the annual CASC competition [26]. For ITP originating problems, especially for E-prover different strategies are often better, so we run it under the alternate scheduler `Epar` [29].

3.2 Accessible facts

As HOL(y)Hammer has initially been designed for HOL Light, it treats accessible facts in the same way as the accessibility relation defined there: any fact that is present in a theory loaded chronologically before the current one is available. In HOL4 there are explicit theory dependencies, and as such a different accessibility relation is more natural. The facts present in the same theory before the current one, and all the facts in the theories that the current one depends on (possibly in a transitive way) are accessible. In this subsection we discuss the four different accessible sets of lemmas, which we will use to test the performance of HOL(y)Hammer on.

Exact dependencies (reproving) They are the closest named ancestors of a theorem in the proof tree. It tests how much HOL(y)Hammer could reprove if it had perfect predictions. In this settings no relevance filtering is done, as the number of dependencies is small.

Transitive dependencies They are all the named ancestors of a theorem in the proof tree. It simulates proving a theorem in a perfect environment, where all recorded theorems are a necessary step to prove the conjecture. This corresponds to a proof assistant library that has been refactored into little theories [6].

Loaded theorems All theorems present in the loaded theories are provided together with all the theorems previously built in the current theory. This is the setting used when proving theorems in HOL4, so it is the one we use in our interactive version presented and evaluated in Section 4.5.

Linear order For this experiment, we additionally recorded the order in which the HOL4 theories were built, so that we could order all the theorems of the standard library in a similar way as HOL Light theorems are ordered. All previously derived theorems are provided.

3.3 Features

Machine learning algorithms typically use features to define the similarity of objects. In the large theory automated reasoning setting features need to be assigned to each theorem, based on the syntactic and semantic properties of the statement of the theorem and its attributes.

HOL(y)Hammer represents features by strings and characterizes theorems using lists of strings. Features originate from the names of the type constructors, type variables, names of constants and printed subterms present in the conclusion. An important notion is the normalization of the features: for subterms, their variables and type variables need to be normalized. Various scenarios for this can be considered:

- All variables are replaced by one common variable.
- Variables are replaced by their de Bruijn index numbers [30].
- Variables are replaced by their (variable-normalized) types [15].

The union of the features coming from the three above normalizations has been the most successful in the HOL Light experiments, and it is used here as well.

3.4 Predictors

In all our experiments we have used the modified k-NN algorithm [13]. This algorithm produces the most precise results in the HOL(y)Hammer experiments for HOL Light [15]. Given a fixed number (k), the k-nearest neighbours learning algorithm finds k premises that are closest to the conjecture, and uses their weighted dependencies to find the predicted relevance of all available facts. All the facts and the conjecture are interpreted as vectors in the n-dimensional feature space, where n is the number of all features. The distance between a fact and the conjecture is computed using the Euclidean distance. In order to find the neighbours of the conjecture efficiently, we store an association list mapping features to theorems that have those features. This allows skipping the theorems that have no features in common with the conjecture completely.

Having found the neighbours, the relevance of each available fact is computed by summing the weights of the neighbours that use the fact as a dependency, counting each neighbour also as its own dependency

4. Experiments

In this section, we present the results of several experiments and discuss the quality of the advice system based on these results. The hardware used during the reproving and accessibility experiments is a 48-core server (AMD Opteron 6174 2.2 GHz. CPUs, 320 GB RAM, and 0.5 MB L2 cache per CPU). In these experiments, each ATPs is run on a single core for each problem with a time limit of 30 seconds. The reconstruction and interactive experiments were run on a laptop with a Intel Core processor (i5-3230M 4 x 2.60GHz with 3.6 GB RAM).

4.1 Reproving

We first try to reprove all the 9434 theorems in the HOL4 libraries with the dependencies extracted from the proofs. This number is lower than the number of exported theorems because definitions are discarded. Table 2 presents the success rates for reproving using the dependencies recorded without splitting. In this experiment we also compare many provers and their versions. For E-prover [23], we also compare its different scheduling strategies [29]. The results are used to choose the best versions or strategies for the selected few provers. Apart from the success rates, the unique number of problems is presented (proofs found by this ATP only), and CVC4 [1] seems to perform best in this respect. The translation used by default by HOL(y)Hammer is an incomplete one (it gives significantly better results than complete ones), so some of the problems are counter-satisfiable.

From this point on, experiments will be performed only with the best versions of three provers: E-prover, Vampire [16], and z3 [5]. They have a high success rate combined with an easy way of retrieving the unsatisfiable core. The same ones have been used in the HOL(y)Hammer experiments for HOL Light.

In Table 3, we try to reprove conjuncts of these theorems with the different recording methods described in Section 2.3. First, we notice that only z3 benefits from the track-

ing of more accurate dependencies. More, removing the unnecessary conjuncts worsen the results of E-prover and Vampire. One reason is that E-prover and Vampire do well with large number of lemmas and although a conjunct was not used in the original proof it may well be useful to these provers.Suprisingly, the percentage of reproved facts did not increase compared to Table 2, as this was the case for HOL Light experiments. By looking closely at the data, we notice the presence of the `quantHeuristics` theory, where 85 theorems are divided into 1538 conjuncts. As the percentage of reproving in this theory is lower than the average (16%), the overall percentage gets smaller given the increased weight of this theory. Therefore, we have removed the quantHeuristic theory in the Basic* and Optimized* experiments for a fairer comparison with the previous experiments. Finally, if we compare the Optimized experiment with the similar HOL Light reproving experiment on 14185 Flyspeck problems [15], we notice that we can reprove three percent more theorems in HOL4. This is mostly due to a 10 percent increase in the performance of z3 on HOL4 problems.

In Table 4 we have compared the success rates of reproving in different theories, as this may represent a relative difficulty of each theory and also the relative performance of each prover. We observe that z3 performs best on the theories `measure` and `probability`, `list` and `finite_map`, whereas E-prover and Vampire have a higher success rate on the theories `arithmetic`, `real`, `complex` and `sort`. Overall, the high success rate in the `arithmetic` and `real` theories confirms that HOL(y)Hammer can already tackle this type of theorems. Nonetheless, it would still benefit from integrating more SMT-solvers' functionalities on advanced theories based on `real` and `arithmetic`.

4.2 With different accessible sets

In Table 5 we compare the quality of the predictions in different proving environments. We recall that only the transitive dependencies, loaded theories and linear order settings are using predictions and that the number of these predictions is adapted to the ability of each provers. The exact dependencies setting (reproving), is copied from Table 3 for easier comparison.

Prover	Version	Theorem(%)	Unique	CounterSat
E-prover	`Epar` 3	44.45	3	0
E-prover	`Epar` 1	44.15	9	0
E-prover	`Epar` 2	43.95	9	0
E-prover	`Epar` 0	43.52	2	0
CVC4	1.3	42.71	44	0
z3	4.32	41.96	8	5
z3	4.40	41.65	1	6
E-prover	1.8	41.37	14	0
Vampire	2.6	41.10	14	0
Vampire	1.8	38.34	6	0
z3	4.40q	35.19	11	5
Vampire	3.0	34.82	0	0
Spass	3.5	31.67	0	0
Metis	2.3	29.98	0	0
IProver	1.0	25.52	2	35
total		50.96		38

Table 2. Reproving experiment on the 9434 unsplit theorems of the standard libary

	Basic	Optimized	Basic*	Optimized*
E-prover	42.43	42.41	46.23	45.91
Vampire	39.79	39.32	43.24	42.41
z3	39.59	40.63	43.78	44.18
total	46.74	46.76	50.97	50.55

Table 3. Success rates of reproving (%) on the 13910 conjuncts of the standard library with different dependency tracking mechanism.

We first notice the lower success rate in the transitive dependencies setting. There may be two justifications. First, the transitive dependencies provide a poor training set for the predictors; the set of samples is quite small and the available lemmas are all related to the conjecture. Second, it is very unlikely that a lemma in this set will be better than a lemma in the exact dependencies, so we cannot hope to perform better than in the reproving experiment.

We now focus on the loaded theories and linear order settings, which are the two scenarios that correspond to the regular usage of a "hammer" system in a development: given all the previously known facts try to prove the conjecture. The results are surprisingly better than in the reproving experiment. First, this indicates that the training data coming from a larger sample is better. Second, this shows that the HOL4 library is dense and that closer dependencies than the exact one may be found by the predictors. It is quite common that large-theory automated reasoning techniques find alternate proofs. Third, if we look at each ATP separately, we see a one percent increase for E-prover, a one percent decrease for Vampire, and 9 percent decrease for z3. This correlates with the number of selected premises. Indeed, it is easy to see that if a prover performs well with a large number of selected premises, it has more chance to find the relevant lemmas. Finally, we see that each of the provers enhanced the results by solving different problems.

We can summarize the results by inferring that predictors combined with ATPs are most effective in large and dense developments.

The linear order experiments was also designed to make a valid comparison with a similar experiment where 39% of Flyspeck theorems were proved by combining 14 methods This number was later raised to 47% by improving the machine learning algorithm. Comparatively, the current 3

	arith	real	compl	meas
E-prover	61.29	72.97	91.22	27.01
Vampire	59.74	69.57	77.19	20.85
z3	51.42	64.46	86.84	31.27
total	63.63	75.31	92.10	32.70
	proba	list	sort	f_map
E-prover	42.16	23.56	34.54	33.07
Vampire	37.34	21.96	32.72	27.16
z3	54.21	25.62	25.45	43.70
total	55.42	26.77	40.00	45.27

Table 4. Percentage (%) of reproved theorems in the theories `arithmetic`, `real`, `complex`, `measure`, `probability`, `list`, `sorting` and `finte_map`.

	ED	TD	LT	LO
E-prover	42.41	33.10	43.58	43.64
Vampire	39.32	29.56	38.46	38.54
z3	40.63	24.66	31.22	31.20
total	46.76	37.54	50.54	50.68

Table 5. Percentage (%) of proofs found using different accessible sets: exact dependencies (ED), transitive dependencies (TD), loaded theories (LT), and linear order (LO)

methods can prove 50% of the HOL4 theorems. This may be since the machine learning methods have improved, since the ATPs are stronger now or even because the Flyspeck theories contain a more linear (less dense) development than the HOL4 libraries, which makes it harder for automated reasoning techniques.

4.3 Reconstruction

Until now all the ATP proved theorems could only be used as oracles inside HOL4. This defeats the main aim of the ITP which is to guarantee the soundness of the proofs. The provers that we use in the experiments can return the unsatisfiable core: a small set of premises used during the proof. The HOL representation of these facts can be given to Metis in order to reprove the theorem with soundness guaranteed by its construction. We investigate reconstructing proofs found by Vampire on the loaded theories experiments (used in our interactive version of HOL(y)Hammer). We found that Metis could reprove, with a one second time limit, 95.6% of these theorems. This result is encouraging for two reasons: First, we have not shown the soundness of our transformations, and this shows that the found premises indeed lead to a valid proof in HOL. Second, the high reconstruction rate suggest that the system can be useful in practice.

4.4 Case study

Finally, we present two sets of lemmas found by E-prover advised on the loaded libraries. We discuss the difference with the lemmas used in the original proof.

The theorem `EULER_FORMULE` states that any complex number can be represented as a combination of its norm and argument. In the human-written proof script ten theorems are provided to a rewriting tactic. The user is mostly hindered by the fact that she could not use the commutativity of multiplication as the tactic would not terminate. Free of these constraints, the advice system returns only three lemmas: the commutativity of multiplication, the polar representation `COMPLEX_TRIANGLE`, and the Euler's formula `EXP_IMAGINARY`.

Example 10. (In theory `complex`)

```
Original proof:
val EULER_FORMULE = store_thm("EULER_FORMULE",
  ``!z:complex. modu z * exp (i * arg z) = z``,
  REWRITE_TAC[complex_exp, i, complex_scalar_rmul,
  RE, IM, REAL_MUL_LZERO, REAL_MUL_LID, EXP_0,
  COMPLEX_SCALAR_LMUL_ONE, COMPLEX_TRIANGLE]);

Discovered lemmas:
COMPLEX_SCALAR_MUL_COMM COMPLEX_TRIANGLE
EXP_IMAGINARY
```

The theorem `LCM_LEAST` states that any number below the least common multiple is not a common multiple. This seems trivial but actually the least common multiple (lcm) of two natural numbers is defined as their product divided by their greatest common divisor. The user has proved the contraposition which requires two Metis calls. The discovered lemmas seem to indicate a similar proof, but it requires more lemmas, namely `FALSITY` and `IMP_F_EQ_F` as the false constant is considered as any other constant in HOL(y)Hammer and uses the combination of `LCM_COMM` and `NOT_LT_DIVIDES` instead of `DIVIDES_LE`.

Example 11. (In theory `gcd`)

```
Original proof:
val LCM_LEAST = store_thm("LCM_LEAST",
  ``0 < m ∧ 0 < n ==> !p. 0 < p ∧ p < lcm m n
  ==> ~(divides m p) ∨ ~(divides n p)``,
  REPEAT STRIP_TAC THEN SPOSE_NOT_THEN
  STRIP_ASSUME_TAC THEN `divides (lcm m n) p`
  by METIS_TAC [LCM_IS_LEAST_COMMON_MULTIPLE]
  THEN `lcm m n <= p` by METIS_TAC [DIVIDES_LE]
  THEN DECIDE_TAC);

Discovered lemmas:
LCM_IS_LEAST_COMMON_MULTIPLE LCM_COMM
NOT_LT_DIVIDES FALSITY IMP_F_EQ_F
```

4.5 Interactive version

In our previous experiments, all the different steps (export, learning/predictions, translation, ATPs) were performed separately, and simultaneously for all the theorems. Here, we compose all this steps to produce one HOL4 step, that given a conjecture proves it, usable in any HOL4 development in an interactive advice loop. It proceeds as follows: The conjecture is exported along with the currently loaded theories. Features for the theorems and the conjecture are computed, and dependencies are used for learning and selecting the theorems relevant to the conjecture. HOL(y)Hammer translates the problem to the formats of the ATPs and uses them to prove the resulting problems. If successful, the discovered unsatisfiable core, consisting of the HOL4 theorems used in the ATP proof, is then read back to HOL4, returned as a proof advice, and replayed by Metis.

In the last experiment, we evaluate the time taken by each steps on two conjectures, which are not already proved in the HOL4 libraries. The first tested goal C_1 is $gcd\ (gcd\ a\ a)\ (b + a) = (gcd\ b\ a)$, where $gcd\ n\ m$ is the greatest common divisor of n and m. It can be automatically proved from three lemmas about gcd. The second goal is C_2 is $Im(i * i) = 0$, where Im the imaginary part of a complex number. It can be automatically proved from 12 lemmas in the theories `real`, `transc` and `complex`.

In Table 6, the time taken by the export and import phase linearly depends on the number of theorems in the loaded libraries (given in parenthesis), as expected by the knowledge of our data and the complexity analysis of our code.

The time shown in the fourth column ("Predict") includes the time to extract features, to learn from the dependencies and to find 96 relevant theorems. The time needed for machine learning is relatively short. The time taken by Vampire shows that the second conjecture is harder. This is backed by the fact that we could not tell in advance what would be the necessary lemmas to prove this conjecture. The overall column presents the time between the interactive call and the display of advised lemmas. The low running times

support the fact that our tool is fast enough for interactive use.

	Export	Import	Predict	Vampire	Total
C_1 (2224)	0.38	0.20	0.29	0.01	0.97
C_2 (4056)	0.67	0.43	0.59	1.58	3.42

Table 6. Time (in seconds) taken by each step of the advice loop

5. Conclusion

In this paper we present an adaptation of the HOL(y)Hammer system for HOL4, which allows for general purpose learning-assisted automated reasoning. As HOL(y)Hammer uses machine learning for relevance filtering, we need to compute the dependencies, define the accessibility relation for theorems and adapt the feature extraction mechanism to HOL4. Further, as we export all the proof assistant data (types, constants, named theorems) to a common format, we define the namespaces to cover both HOL Light and HOL4.

We have evaluated the resulting system on the HOL4 standard library toplevel goals: for about 50% of them a sufficient set of dependencies can be found automatically. We compare the success rates depending on the accessibility relation and on the treatment of theorems whose statements are conjunctions. We provide a HOL4 command that translates the current goal, runs premise selection and the ATP, and if a proof has been found, it returns a Metis call needed to solve the goal. The resulting system is available at `https://github.com/barakeel/HOL`.

5.1 Future Work

The libraries of HOL Light and HOL4 are currently processed completely independently. We have however made sure that all data is exported in the same format, so that same concepts and theorems about them can be discovered automatically [7]. By combining the data, one might get goals in one system solved with the help of theorems from the other, which can then be turned into lemmas in the new system. A first challenge might be to define a combined accessibility relation in order to evaluate such a combined proof assistant library.

The format that we use for the interchange of HOL4 and HOL Light data is heavily influenced by the TPTP formats for monomorphic higher-order logic [27] and polymorphic first-order logic [2]. It is however slightly different from that used by Sledgehammer's `fullthf`. By completely standardizing the format, it would be possible to interchange problems between Sledgehammer and HOL(y)Hammer.

In HOL4, theorems include the information about the theory they originate from and other attributes. It would be interesting to evaluate the impact of such additional attributes used as features for machine learning on the success rate of the proofs. Finally, most HOL(y)Hammer users call its web interface [14], rather than locally install the necessary prover modifications, proof translation and the ATP provers. It would be natural to extend the web interface to support HOL4.

Acknowledgments

We would like to thank Josef Urban and Michael Färber for their comments on the previous version of this paper. This work has been supported by the Austrian Science Fund (FWF): P26201.

References

[1] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Proceedings of the 23rd International Conference on Computer Aided Verification*, CAV'11, pages 171–177, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-22109-5. URL `http://dl.acm.org/citation.cfm?id=2032305.2032319`.

[2] J. C. Blanchette and A. Paskevich. TFF1: The TPTP typed first-order form with rank-1 polymorphism. In M. P. Bonacina, editor, *CADE*, volume 7898 of *Lecture Notes in Computer Science*, pages 414–420. Springer, 2013. ISBN 978-3-642-38573-5. . URL `http://dx.doi.org/10.1007/978-3-642-38574-2`.

[3] J. C. Blanchette, S. Böhme, A. Popescu, and N. Smallbone. Encoding monomorphic and polymorphic types. In N. Piterman and S. A. Smolka, editors, *TACAS*, Lecture Notes in Computer Science, pages 493–507. Springer, 2013.

[4] S. Böhme and T. Weber. Fast LCF-style proof reconstruction for Z3. In M. Kaufmann and L. Paulson, editors, *Interactive Theorem Proving*, volume 6172 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2010. .

[5] L. De Moura and N. Bjørner. Z3: An efficient SMT solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-78799-2, 978-3-540-78799-0. URL `http://dl.acm.org/citation.cfm?id=1792734.1792766`.

[6] W. M. Farmer, J. D. Guttman, and F. J. Thayer. Little theories. In *Automated Deduction—CADE-11, volume 607 of Lecture Notes in Computer Science*, pages 567–581. Springer-Verlag, 1992.

[7] T. Gauthier and C. Kaliszyk. Matching concepts across HOL libraries. In S. Watt, J. Davenport, A. Sexton, P. Sojka, and J. Urban, editors, *Proc. of the 7th Conference on Intelligent Computer Mathematics (CICM'14)*, volume 8543 of *LNCS*, pages 267–281. Springer Verlag, 2014. .

[8] T. Gauthier, C. Kaliszyk, C. Keller, and M. Norrish. Beagle as a HOL4 external ATP method. In L. D. Moura, B. Konev, and S. Schulz, editors, *Proc. of the 4th Workshop on Practical Aspects of Automated Reasoning (PAAR'14)*, EPiC, 2014. to appear.

[9] J. Harrison. Optimizing proof search in model elimination. In M. McRobbie and J. Slaney, editors, *Proceedings of the 13th International Conference on Automated Deduction*, number 1104 in LNAI, pages 313–327. Springer-Verlag, 1996.

[10] K. Hoder and A. Voronkov. Sine qua non for large theory reasoning. In N. Bjørner and V. Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wrocław, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2011. ISBN 978-3-642-22437-9.

[11] J. Hurd. First-order proof tactics in higher-order logic theorem provers. In M. Archer, B. D. Vito, and C. Muñoz, editors, *Design and Application of Strategies/Tactics in Higher Order Logics (STRATA 2003)*, number NASA/CP-2003-212448 in NASA Technical Reports, pages 56–68, Sept. 2003.

[12] C. Kaliszyk and J. Urban. Mizar 40 for Mizar 40. *CoRR*, abs/1310.2805, 2013. URL `http://arxiv.org/abs/1310.2805`.

[13] C. Kaliszyk and J. Urban. Stronger automation for Flyspeck by feature weighting and strategy evolution. In J. C. Blanchette and J. Urban, editors, *PxTP 2013*, volume 14 of *EPiC Series*, pages 87–95. EasyChair, 2013.

[14] C. Kaliszyk and J. Urban. HOL(y)Hammer: Online ATP service for HOL Light. *Mathematics in Computer Science*, 2014. URL `http://arxiv.org/abs/1309.4962`. to appear.

[15] C. Kaliszyk and J. Urban. Learning-assisted automated reasoning with Flyspeck. *Journal of Automated Reasoning*, 53(2):173–213, 2014.

[16] L. Kovács and A. Voronkov. First-order theorem proving and Vampire. In N. Sharygina and H. Veith, editors, *Proceedings of the 25th International Conference on Computer Aided Verification (CAV)*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2013. ISBN 978-3-642-39798-1. . URL `/pubpdf/First-Order_Theorem_Proving_and_Vampire.pdf`.

[17] D. Kühlwein, J. C. Blanchette, C. Kaliszyk, and J. Urban. MaSh: Machine learning for Sledgehammer. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Proc. of the 4th International Conference on Interactive Theorem Proving (ITP'13)*, volume 7998 of *LNCS*, pages 35–50. Springer Verlag, 2013.

[18] R. Kumar and J. Hurd. Standalone tactics using opentheory. In L. Beringer and A. P. Felty, editors, *Interactive Theorem Proving - Third International Conference, ITP 2012, Princeton, NJ, USA, August 13-15, 2012. Proceedings*, volume 7406 of *Lecture Notes in Computer Science*, pages 405–411. Springer, 2012. ISBN 978-3-642-32346-1. . URL `http://dx.doi.org/10.1007/978-3-642-32347-8_28`.

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

[20] L. C. Paulson and J. Blanchette. Three years of experience with Sledgehammer, a practical link between automated and interactive theorem provers. In *8th IWIL*, 2010. Invited talk.

[21] L. C. Paulson and K. W. Susanto. Source-level proof reconstruction for interactive theorem proving. In K. Schneider and J. Brandt, editors, *TPHOLs*, volume 4732 of *LNCS*, pages 232–245. Springer, 2007. ISBN 978-3-540-74590-7.

[22] S. Schulz. E - a brainiac theorem prover. *AI Commun.*, 15 (2-3):111–126, 2002.

[23] S. Schulz. System Description: E 1.8. In K. McMillan, A. Middeldorp, and A. Voronkov, editors, *Proc. of the 19th LPAR, Stellenbosch*, volume 8312 of *LNCS*. Springer, 2013.

[24] K. Slind and M. Norrish. A brief overview of HOL4. In O. A. Mohamed, C. A. Muñoz, and S. Tahar, editors, *TPHOLs*, volume 5170 of *Lecture Notes in Computer Science*, pages 28–32. Springer, 2008. ISBN 978-3-540-71065-3.

[25] G. Sutcliffe. The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, 43(4):337–362, 2009. ISSN 0168-7433. . URL `http://dx.doi.org/10.1007/s10817-009-9143-8`.

[26] G. Sutcliffe. The CADE-24 automated theorem proving system competition - CASC-24. *AI Commun.*, 27(4):405–416, 2014.

[27] G. Sutcliffe and C. Benzmüller. Automated reasoning in higher-order logic using the TPTP THF infrastructure. *J. Formalized Reasoning*, 3(1):1–27, 2010. . URL `http://dx.doi.org/10.6092/issn.1972-5787/1710`.

[28] J. Urban. MaLARea: a metasystem for automated reasoning in large theories. In G. Sutcliffe, J. Urban, and S. Schulz, editors, *ESARLT*, volume 257 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[29] J. Urban. Blistr: The blind strategymaker. *CoRR*, abs/1301.2683, 2013. URL `http://arxiv.org/abs/1301.2683`.

[30] J. Urban, G. Sutcliffe, P. Pudlák, and J. Vyskočil. MaLARea SG1 - Machine Learner for Automated Reasoning with Semantic Guidance. In A. Armando, P. Baumgartner, and G. Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 441–456. Springer, 2008. ISBN 978-3-540-71069-1.

[31] T. Weber. SMT solvers: new oracles for the HOL theorem prover. *International Journal on Software Tools for Technology Transfer*, 13(5):419–429, 2011. ISSN 1433-2779. . URL `http://dx.doi.org/10.1007/s10009-011-0188-8`.

[32] W. Wong. Recording and checking HOL proofs. In *Higher Order Logic Theorem Proving and Its Applications. 8th International Workshop, volume 971 of LNCS*, pages 353–368. Springer-Verlag, 1995.