Cumulative Habilitation Thesis

# Learning Assisted Automated Reasoning

Cezary Kaliszyk

Innsbruck, June 15, 2015

Institute of Computer Science University of Innsbruck

# Contents

# Acknowledgements

I would like to thank my co-authors: Josef, Jasmin, Jiří, Daniel, and Thibault for the collaboration we had during the preparation of the publications that constitute this habilitation thesis.

I am grateful to everyone of the Computational Logic group in Innsbruck for creating a social and friendly working environment. I would especially like to thank Aart Middeldorp for giving me the freedom to pursue my research interests in the last years, motivation, and feedback. Parts of this work have been supported by the Austrian Science Fund (FWF) Project P26201 and the Austrian Agency for International Cooperation in Education and Research (OeAD).

I would also like to thank my friends at work and in Innsbruck: Tom, Bertram, Chris, Julian, René, Ruggi, and Martin. Our discussions brought many interesting ideas, and the coffee breaks provided welcome distraction, that allowed me to take a step back and to look at the seemingly unsolvable problems from a different point of view.

This work would not have been possible without the support of my family. Thank you for your support, understanding, and always being there.

Innsbruck, 2015

# Part I

# Preface

# 1 Introduction

The use of proof assistants is becoming a more accepted means of ascertaining the correctness of a computer program or of a mathematical theory. Practically all mathematical theorems can be formulated in the languages of these systems and the proofs can be written and verified formally. Reasoning over formal mathematics then becomes a general problem-solving technique for arbitrary problems that can be expressed in a sufficiently formal language and a non-contradictory setting.

The last two decades have brought significant progress in *interactive theorem proving* (ITP). Some graduate textbooks have been formalized [BR02, NK14], repositories of formal proofs have been created using various proof assistants [KNP], which were used to prove noteworthy programs correct. Important examples are the CompCert formalization of an optimizing compiler for the C programming language [Ler07] and the verification of the L4 operating system microkernel [KAE$^+$10]. Similarly impressive results have been obtained with the help of proof assistants in mathematics, such as the formal proof of the four color theorem in the Coq proof assistant [Gon07], the proof of the Kepler conjecture [HAB$^+$15] in HOL Light and Isabelle/HOL, and the development of the odd order theorem in Coq [GAA$^+$13].

These impressive results have so far not popularized proof assistants: they are still used mostly by the experts in the domain. There are many possible reasons for this [Wie01]: formalization is a nontrivial skill to learn, even routine formalizations are still quite laborious as working with a proof assistant often involves proving numerous steps manually, and a lot of cognitive processing is involved in formalization, which is not common to the majority of today's mathematicians. This means that the majority of mathematical knowledge remains to be inaccessible in a formal setting, which in itself is another obstacle to formalization.

Along with the development of proof libraries automated methods for proof assistants were improved. Many of the methods have been influenced by the field of *Automated Theorem Proving* (ATP), which focuses on developing stronger methods for finding proofs automatically. The best known result achieved by an ATP is the proof of the Robbins conjecture found by the EQP system [McC97]. At present, ATPs can find proofs that involve more than thousands of inference steps, such as the proofs in quasigroup and loop theories [PS10]. The cooperation between ATP and ITP has been discussed many times, including the famous QED Manifesto [Ano94].

Since 2003 the Mizar library has been translated to the formats used by ATP systems [Urb06c,Urb04,Urb03]. Together with a similar translation for Isabelle/HOL [MQP06], this created a number of challenges and opportunities for the developers of ATP systems. This led to the creation of the new field of *Automated Reasoning in Large Theories* (ARLT) [UV13], where ATP systems that can load the large number of theorems and proofs available in the ITP libraries are combined with AI methods that analyze such proofs, learn the relevance of the facts from them using various machine-learning methods, and are able to predict the most promising combinations of existing theorems for new conjectures. We call such AI/ATP systems *hammers*, after the popular Sledgehammer tool [PB12] for Isabelle/HOL and our own tool HOL(y)Hammer for HOL Light and HOL4. We also include the Mizar proof advisor, Miz$\mathbb{AR}$ [Urb08], in this category.

Hammers have become the strongest tools for providing proof advice in interactive proof formalization. They provide a brute force tool augmented with machine learning, which can tackle problems in large fact libraries. In his report on formal proof for the Bourbaki Seminar, Hales writes [Hal14a]:

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

In the development of our hammer system – HOL(y)Hammer – we have focused on HOL Light. HOL Light [Har96a] was the first well-known interactive theorem prover which integrated and extensively used a general ATP procedure, MESON [Har96c]. Harrison also implemented a bridge from HOL Light to Prover9 [McC10], which uses the detailed Ivy [MM00] proof objects. The existence of these two translation and reconstruction mechanisms made HOL Light a perfect candidate for investigating methods that attempt to automatically solve a new goal by selecting relevant subsets from the large knowledge library and running external ATPs with such relevant facts. Furthermore, thanks to the Flyspeck (*Formal Proof of Kepler*) project [Hal06], HOL Light is becoming a tool that is not only more interesting for general formalization, but also for the semantic AI methods. The proof of the Kepler Conjecture [Kep11, HHM$^+$10] involves the formalizations of a number of fields of mathematics (multivariate analysis, trigonometry, etc.), giving rise to a well-structured interconnected proof library.

A typical hammer consists of three components, which are used in order: () a *relevance filter* heuristically selects a subset of the accessible theorems that are most likely to lead to a successful proof of the given conjecture; (ii) *goal translation* encodes a proof assistant goal, typically expressed in a richer logic, to the logic and format of an ATP system; (iii) *reconstruction* extracts the information from the successful ATP proof that is necessary to construct the proof in the logic of the proof assistant. The way these components interact to process a given ITP goal is presented in Figure 1.1. The three components correspond to the three major problems listed in the next section.

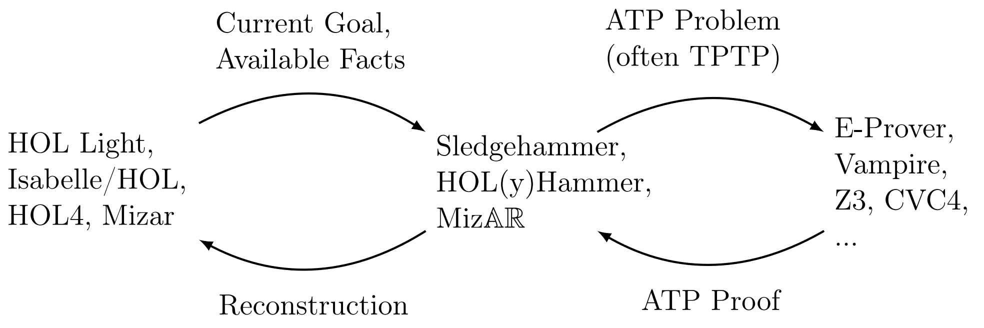


Figure 1.1.: The components of an AI/ATP system: given a goal in an interactive theorem prover and the accessible facts the relevance filter selects a subset of the facts that is likely to lead to a successful proof of the given conjecture. They are translated together with the conjecture to the logic and input format of the ATPs. If the ATP proof is successful, the reconstruction component builds the proof of the conjecture in the logic of the ITP.

## 1.1. Major Problems

In order to construct a hammer the following three problems need to be addressed:

- **Premise selection:** In order to choose the relevant premises, it is necessary to characterize them with meaningful features and to use suitable algorithms for choosing similar theorems augmented by the knowledge obtained by learning from previous proofs [KvLT+12, AHK+14].

- **Translation:** The logics of the interactive proof systems are usually significantly stronger from those available in the ATPs. Different translations trade the efficiency of the encoding for completeness, therefore most hammers include a number of different encodings. Such encodings differ for example in the treatment of polymorphic statements, encoding of lambda-abstractions, or the addition of higher-order apply functors [BBPS13].

- **Reconstruction:** The fact that a certain set of facts is sufficient to find an ATP proof is not enough to recreate the proof in the logic of the ITP. For this various reconstruction mechanisms are proposed, which starts with the use of available ITP automation techniques using just the information about the needed premises [Har96c, Hur03]; and ends with a complete replaying of each ATP or SMT inference [SB13, BBP11].

The rest of this document is organized as follows. Chapter 2 presents the contributions of this thesis. For each of the selected papers the publication details and the main achievements are briefly presented. Further contributions of my other work, that are beyond the scope of this thesis are listed. In Part II of the thesis (Chapters 3–11) the selected papers [1–11] are provided.

# 2 Contributions

We start this chapter by presenting how the selected papers included in Part II of this thesis contribute to solving the major problems presented in Chapter 1.1. The papers are ordered thematically by proof system, and within each theme they are listed in the order of their publication date. The chapter numbers indicated in the titles of section headings refer to the specific chapter(s) in Part II of this thesis wherein the papers are provided. The text of some conference papers (listed in gray) is not included in this thesis since these are subsumed by extended journal articles. At the end of the chapter further contributions not fully discussed in this thesis are briefly presented.

## 2.1. Automated Reasoning with HOL Light (Chapter 3)

### Publication details

[1] Cezary Kaliszyk and Josef Urban. Learning-assisted Automated Reasoning with Flyspeck. *J. Automated Reasoning*, 53(2):173–213, Springer, 2014. doi: `10.1007/s10817-014-9303-3`.

This paper introduces automatic learning from previous proofs in higher-order logic, combined with ATP techniques for relevance filtering. This is the initial paper introducing our HOL(y)Hammer system, which provides strong proof advice for HOL Light and is the basis for the further experiments presented in this thesis. We present the history of large-theory automated reasoning and discuss the automation already available in HOL Light and Flyspeck and propose the following components of an AI/ATP system:

- **Translation:** As the MESON translation uses different instances of equality for different types, it is not suitable for modern ATPs that implement paramodulation. We propose and implement sound and efficient translations of HOL Light formulas to several TPTP ATPs. This includes the untyped first-order (FOF) format [SSCG06], the monomorphic higher-order (THF0) format [GS06, SB10], and the polymorphic typed first-order (TFF1) format [BP13]. The different export formats allow the use of 17 different ATPs and SMTs.
- **Features:** In order to use machine learning algorithms to predict useful dependencies, it is necessary to characterize the formulas of higher-order logic. We charac-

terize each formula by the set of its (sub-)terms, which was already found useful in the MaLARea system [Urb07,USPV08], and adapt the scheme to higher-order logic by proposing various subterm variable and type variable normalization schemes.

- **Dependencies:** To learn from previous proofs we extract the theorem dependencies based on the complete export of theorems provided by HOL/Import [KK13]. We introduce several optimizations that improve the quality of advice, such as the splitting of conjunctions (including mini-scoping the quantifiers), in order to track the prover dependencies more precisely.
- **Prediction:** We integrate the external machine learning algorithms implemented by the SNoW [CCRR99] framework and a custom implementation of the $k$-nearest neighbor ($k$-NN) algorithm to predict premises likely to be useful in the proof of the given conjecture. We use Perceptron and Winnow based neural networks, as well as the Naive Bayes classifier.
- **Evaluation:** We propose an evaluation scenario that emulates the development of Flyspeck from the axioms and of higher-order logic up to the last theorem. Each time only the previous theorems and proofs are used. We evaluated the proposed AI/ATP system on a smaller subset of the Flyspeck theorems to define a portfolio of orthogonal strategies, and evaluated the 14 strongest strategies on the whole of Flyspeck. 39% of the 14185 theorems could be proved in a push-button mode (without any high-level advice and user interaction) in 30 seconds of real time per goal on a fourteen-CPU workstation.

Here my contributions are the definition and the implementation of the translation from HOL Light formulas to the logics and formats of the TPTP ATPs, the extraction and optimization of dependencies, the integration of the machine learning algorithms in HOL Light, and the ATP evaluation of the proposed methods.

## 2.2. Proof Reconstruction (Chapters 4 and 5)

### Publication details

[2] Cezary Kaliszyk and Josef Urban. PRocH: Proof Reconstruction for HOL Light. In Maria Paola Bonacina, editor. Proc. of the 24th *International Conference on Automated Deduction (CADE'13)*, volume 7898 of LNCS, pp. 267–274. Springer, 2013. doi: `10.1007/978-3-642-38574-2_18`

[3] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Certified Connection Tableaux Proofs for HOL Light and TPTP. In Xavier Leroy and Alwen Tiu, editors. *Proc. 4th Conference on Certified Programs and Proofs (CPP'15)*, pp. 59–66, ACM, 2015. doi: `10.1145/2676724.2693176`.

The proof advice system proposed in the previous chapter can automatically find a set of dependencies that is sufficient to finish the higher-order logic proof. This however does not give a user any intuition on how to proceed to complete the formal proof. In some cases the set of necessary dependencies returned by the system can involve ten or more required facts. Combining these facts in a proof using the available HOL tactics or decision procedures can sometimes still be a laborious task. In the two papers we propose and implement two complementary methods to reconstruct a HOL proof from the proofs found by the ATPs.

In the first paper [2], we implement a proof reconstruction method, PRocH, which relies on the complete TSTP proofs returned by the used automated theorem provers. Such proofs are produced for example by E-Prover [Sch13] and Vampire [KV13]. PRocH combines several reconstruction methods running in parallel. The main improvement over the tactics available in HOL Light is obtained by re-playing in the HOL logic the detailed inference steps recorded in the ATP (TPTP) proofs. This is done using several internal HOL Light inference methods. These methods range from fast variable matching and more involved rewriting, to full first-order theorem proving using the MESON tactic. Note that this technique is also supported by the `isar_proof` Sledgehammer function described in [PS07a]. There are two main differences: First, PRocH relies on the polymorphic encoding of types, and therefore in the reconstruction it can use the type annotations in the TSTP proof. Second, PRocH does not attempt to write a HOL proof script. The main contribution of the first paper is an algorithm to reverse the encoding used by HOL(y)Hammer. This algorithm can recover all the HOL types from the encoded polymorphic types, as long as no type variable has not been skolemized. Type checking in HOL is used to recover the types lost in the skolemization steps.

The second paper [3] presents a certified implementation of the calculus of the first-order theorem prover leanCoP [OB03, Ott10] inside HOL Light. We have implemented an OCaml version of the leanCoP compact connection prover and the reconstruction of its proofs inside HOL Light. This proof-reconstruction functionality can be used to certify arbitrary TPTP proofs produced by leanCoP. This means that leanCoP is now one of the few ATPs whose proofs can be verified in one of the safest LCF-based systems. The performance of the OCaml version on the benchmarks is comparable to the original Prolog version, while it always outperforms Metis and MESON, sometimes very significantly on the relevant ITP-related benchmarks. We implemented a HOL Light interface identical to that of MESON, namely a tactic using helper theorems and a conversion. The benchmarks show that the provided methods are likely the strongest single-step first-order proof-reconstruction tactics available in today's ITP systems.

My contributions in the first paper [2] are: (i) the algorithm to reverse the polymorphic encoding, (ii) the treatment of the skolemization steps, (iii) the choice and order of HOL Light rules to use for each single TSTP proof line, and (iv) the evaluation inside HOL Light. For the second paper [3], my contributions were the encoding of the Prolog semantics inside HOL Light to allow the implementation of the leanCoP algorithm, as well as proof reconstruction.

## 2.3. Re-use of Proof Knowledge (Chapter 6)

### Publication details

[4] Cezary Kaliszyk and Josef Urban. Automated Reasoning Service for HOL Light. In Jacques Carette, David Aspinall, Christoph Lange, Petr Sojka, and Wolfgang Windsteiger, editors. Proc. of the 6th *Conference on Intelligent Computer Mathematics (CICM'13)*, volume 7961 of LNCS, pp. 120–135. Springer, 2013. doi: 10.1007/978-3-642-39320-4_8

[5] Cezary Kaliszyk and Josef Urban. HOL(y)Hammer: Online ATP service for HOL Light. *Mathematics in Computer Science*, 9(1):5–22, Springer, 2015. doi: 10.1007/s11786-014-0182-0

The success rates of AI/ATP methods in large theories highly depend on the precision of the proof dependencies which the method learns from. Obtaining precise proof dependencies can be a costly process depending on the size of the development, the desired precision, and the available resources. The entire process can take from hours to weeks to perform. In a development where many changes are performed every day (often new proved theorems), such as Flyspeck [HAB$^+$15], recomputing the high-quality prediction knowledge every time a change is made may be impossible. In these papers [4,5], we propose a number of ways in which learned knowledge can be re-used between different versions of a HOL Light development, or even between different developments.

We propose and implement a recursive content hashing scheme adapted to higher-order logic. For each introduced term constant a hash of its (alpha-normalized) definition is computed, taking special care about the constants introduced by the automated definition mechanisms of HOL Light. The constant names are replaced by such hashes in the AI/ATP advice process. The hashed constants can be also used to compute the hashes of theorem statements, giving rise to normalized names of theorems. In particular, the HOL type introduction primitive rule requires an existence theorem. Using the hashes of such theorems also the names of types can be hashed.

This allows for assigning unique names to types, constants, and theorems. Such unique names permit the sharing and re-use of the knowledge about successful ATP proofs between different versions of Flyspeck, or even sharing of knowledge between different formalizations. In a large formal repository repetitions are unavoidable. The hashing lets us discover 39 Flyspeck constant symbols with the same content definition and more than 500 equivalent theorems that are given multiple names.

As the knowledge stemming from many developments may be shared most efficiently in a central environment, we created such an environment for HOL Light proof advice.

We combined the HOL(y)Hammer toolchain with the use of parallelized decision procedures available in HOL Light, ATP proof cross-minimization and the use of available reconstruction methods. To maximize re-use, we implemented caching of requests and of the tried ATP problems. The created online ATP advice service can re-use information across user developments. This significantly improved the advice quality on Flyspeck.

Here my contributions are the extension of the recursive hashing scheme to higher-order logic, the experiments comparing the knowledge present in the various versions of Flyspeck and the integration of the functionality into a common service.

## 2.4. External Provers for HOL4 (Chapter 7)

### Publication details

[6] Thibault Gauthier and Cezary Kaliszyk. Premise Selection and External Provers for HOL4. In Xavier Leroy and Alwen Tiu, editors. *Proc. 4th Conference on Certified Programs and Proofs (CPP'15)*, pp. 49–57, ACM, 2015. doi: 10.1145/2676724.2693173.

In this paper we extend HOL(y)Hammer to HOL4. The two provers share the same foundational logic. However, the implementations of the systems differ in a number of ways, such as the kernel inference rules, the proof terms, the way proved theorems are recorded, and the treatment of conjunctions. As HOL(y)Hammer was initially implemented as an add-on to HOL Light, we needed to generalize big parts of its functionality. The specific contributions of the paper are:

- We propose an extension of the TPTP language to rank-1 polymorphism, in order to allow the exchange of the data between HOL-based systems. HOL Light, HOL4, and Isabelle/HOL goal data can be exported in this format. The format combines the features of the already existing monomorphic higher-order THF0 and polymorphic first-order TFF1.

- We investigate the influence of accessibility relations on the quality of AI/ATP advice. As HOL Light (and Flyspeck) processes theorems in a linear order, the access to all previously obtained facts is natural. In HOL4 only the facts from the included theories, and previous theorems from the same theory are accessible. We also proposed an accessibility relation induced by the perfect division of the HOL4 library into theory files, induced by the little-theory approach [FGT92].

- In both HOL implementations, the theorems that are conjunctions of facts are often used to group properties into categories. In order to obtain more precise dependencies, we introduce a precise conjunct-tracing dependency mechanism in the HOL4 kernel. This allows to obtain more precise dependencies than those in HOL Light.

- Finally we make all the engineering work needed to record theorems, dependencies, export its library, extract HOL4 specific theory features, adapt HOL(y)Hammer to work with this external information, and combine it with Metis proof reconstruction. Our evaluations show that 50% of the HOL4 standard library can be reproved fully automatically without any dependency information.

My contributions were the proposed extension of the TPTP language, adaptation of HOL(y)Hammer to this input format, and the scenarios for the accessibility relation.

## 2.5. Machine Learning for Sledgehammer (Chapter 8)

### Publication details

[7] Daniel Kühlwein, Jasmin C. Blanchette, Cezary Kaliszyk, and Josef Urban. MaSh: Machine Learning for Sledgehammer. In Proc. of the 4th *International Conference on Interactive Theorem Proving (ITP'13)*, volume 7998 of LNCS, pp. 35–50. Springer, 2013. doi: `10.1007/978-3-642-39634-2_6`

Sledgehammer provides an AI/ATP method for the Isabelle/HOL proof assistant. The relevance filter of Sledgehammer, MePo [MP09], ranks the accessible facts and selects a subset, based on syntactic similarity to the current goal.

The main contribution of this paper is an alternate relevance filter for Sledgehammer (MaSh) that learns from previous successful proofs. Since the way a typical user interacts with Isabelle/HOL is different from the way this is done in HOL Light, we propose a way to integrate learning on top of the Isar loop, which permits deletion and renaming of facts. This means that the interface to MaSh must include learning and relearning operations. We specified the design goals of a "hammer" system integrated in an interactive proof assistant: it should require no configuration, no change in the proof workflow, no additional maintenance, and no overhead to the users that are not using the provided advice and automation.

We also proposed a number of modifications the previously proposed AI/ATP, that are specifically tailored to exploit the Isabelle knowledge. We proposed to characterize Isabelle terms and formulas by non-trivial first-order patterns up to a given depth. Such a characterization can be computed fast while it preserves more sharing than subterms. The Isabelle features also include the names of locally bound variables and the name of a theory. We reimplemented the Naive Bayes relevance filter in order to adapt it to fact selection. This allows us to treat the known features and unknown features with different weights, which improves the quality of the machine learning. Experimentally we found out that positive features should have much higher weights than negative ones. The dependencies are extracted from the Isabelle proof terms. As some tactics (such as Presburger arithmetic) routinely pull more than 200 dependencies, a fixed limit is introduced, and proofs that include more dependencies than the limit are ignored.

We evaluated the proposed relevance filters on the top-level goals in three Isabelle/HOL developments: Cryptographic protocols [Pau98], Java-like language [KN05], and probability theory [HH11], as well as on the "Judgment Day" benchmark suite [BN10]. The new learning relevance filter performs better than the similarity based one, and combining the two yields a particularly strong relevance filter.

My contributions are the Sledgehammer internal machine learning algorithms. In order to efficiently predict relevant theorems I have modified the functional Naive Bayes and k-NN to use imperative data structures and partial sorting. I have also performed the evaluation on large formalizations.

## 2.6. Lemma Mining (Chapter 9)

### Publication details

[8] Cezary Kaliszyk and Josef Urban. Lemma Mining over HOL Light. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors. Proc. of the 19th *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'13)*, volume 8312 of LNCS, pp. 503–517. 2013. doi: `10.1007/978-3-642-45221-5_34`

[9] Cezary Kaliszyk and Josef Urban. Learning-assisted Theorem Proving with Millions of Lemmas. *J. Symbolic Computation*, 69:109–128, 2015. doi: `10.1016/j.jsc.2014.09.032`.

Large formal mathematical libraries contain millions of atomic inferences steps that build up the proved statements (lemmas). Similar to informal mathematical practice, only a fraction of such statements is named and later re-used in formal proofs. In this paper we propose and implement extraction of intermediate lemmas from HOL Light and Flyspeck, ways to estimate their predicted usefulness for further proofs, and evaluate the impact of such intermediate lemmas on an the quality of advice of an AI/ATP system. The specific contributions of the paper are:

- **Traces:** We have proposed ways to extract proof traces from the Flyspeck formalization. Since the formalization contains approximately 1.8 billion proof steps and 2 billion dependencies between such steps, various optimizations were proposed in the extraction process which reduce the extraction time and the size of the final graph. We have also extracted the graphs of tactic calls by modifying the tactic combinators. The graphs have been augmented by the information about the size and complexity of the steps, giving us a number of interesting statistics about the behaviour of the HOL system on a large formalization.

- **Lemma quality:** We have proposed several approaches to defining the notion of a useful/interesting lemma, such as: (i) a direct implementation of lemma quality metric based on the HOL Light proof-recording data structures that takes into account the numbers of dependencies, uses, and the size of each lemma; (ii) a quality metric based on Schulz's epcllemma [DS94,DS96] and our modified versions thereof; (iii) lemma quality based on the PageRank [PBMW98] (eigenvector centrality of a graph assigns weights to the nodes based on the weights of the neighboring nodes) of the dependencies and uses graphs; and (iv) graph cutting algorithms with modified weighting function.

- **Scenarios:** We have defined a number of scenarios in which the new lemmas can be used in a "hammer" system. Starting with scenarios that augment the automated proof knowledge by using the complete graph, through scenarios that use chains of conjectures to prove subsequent facts from the library, to fully-honest ATP scenarios, where the knowledge is not only limited to the previous lemmas, but also the graph is restricted and the complete lemma quality is recomputed at every step. We evaluated the methods experimentally obtaining a 5–20% improvement depending on the used trace, lemma quality metric, and scenario.

My contributions include the extraction and optimization of the HOL Light and Flyspeck proof traces, lemma quality metrics (i) and (iv), as well as the the non-incremental scenarios and their evaluation.

## 2.7. Automated Reasoning with Mizar (Chapter 10)

### Publication details

[10] Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *J. Automated Reasoning*, in press, 2015. doi: `10.1007/s10817-015-9330-8`.

So far we have focused on the application of AI/ATP methods to variants of higher-order logic. In this paper we show how the methods that efficiently learn from previous proofs can be extended to the Mizar system, whose logic is a dependently-typed set theory. As Mizar can be directly encoded in first-order logic with soft types, the main issue that we needed to solve in this paper is the sheer size of the MML (Mizar Mathematical Library). Indeed, the official release of MML includes 57897 named Mizar theorems, which depend on further 90000 unnamed Mizar formulas and formulas needed to encode the type system of Mizar in the TPTP FOF format. The five main contributions of this paper are:

- **Geometric progression relevance:** We proposed a new premise selector based on geometric progression of relevance for theorem neighbours combined with distance-weighted fact dependencies. Iteration from nearest and from farthest neighbours both give different complementary predictions.
- **Latent Semantic Indexing:** Using normalized subterm features gives 250 thousand distinct normalized features for the whole MML. To effectively deal with this number of features, we proposed methods to reduce the feature space. This is not only important for efficiency reasons, but the quality of some of the predictors may depend on the independence of features. This is the case for Naive Bayes: the application of the Bayes theorem which gives rise to the relevance formula used by the predictor requires the features to be independent. We used Latent Semantic Indexing [DDL+90] (LSI) of the feature space, which we tested with 800, 3200, and 6400 topics, as well as versions with TF-IDF [Jon72] feature scaling.
- **Precise dependency data:** Using the Mizar proof dependencies we constructed ATP problems corresponding to the named theorems, and by running ATPs we obtained a first set of ATP dependencies. By repeatedly training the machine learning algorithms on the previous sets of ATP dependencies, or the combination of the ATP dependencies and the Mizar proof dependencies we would get new sets of predictions. By running the ATPs on the problems corresponding to the more precise predictions, we would get more dependency information. Starting with initial 27842 training examples, by five additional passes we obtain 36122 training examples.
- **Efficient machine learners:** We proposed new efficient implementations of the machine learners that use low-level indexing structures. Both k-NN and Naive Bayes based relevance filters are implemented with IDF using arrays of maps and hashes, that associate features with the facts that appeared as dependencies when the feature was present for the conjecture. With this algorithm we can index the complete Mizar data for k-NN including IDF in three seconds and for Naive Bayes in five seconds.
- **Experiments:** We created a portfolio of orthogonal methods to achieve high success rates on any Mizar subgoal. We evaluated this strategy combination on the whole of MML in a scenario that simulates the development of the library. The 14 strongest strategies are able to automatically prove 40% of all the theorems.

I have proposed and implemented the geometric progression relevance filters, the low-level indexing for the new machine learners, and performed the experiments.

## 2.8. Semantic Features for Machine Learning (Chapter 11)

### Publication details

[11] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Efficient Semantic Features for Automated Reasoning over Large Theories. Accepted for the *Proc. International Joint Conference on Artificial Intelligence*, to appear, IJCAI/AAAI, 2015.

A crucial part of the learning methods are the features that characterize mathematical statements. So far we have considered syntactic features, i.e. features that textually represent a (sub-)formula or term. In this paper we proposed several types of features which are based on semantic concepts that are central to the logical frameworks.

The main contributions of this paper are new features based on first-order unification, first-order matching, and abstraction. We proposed mechanisms to efficiently compute such features for large datasets based on first-order term indexing structures, namely discrimination trees and substitution trees. Additionally all the nodes of the indexing structures that were created in the process of inserting (sub-)formulas and terms can serve as additional characteristics of the facts, and can be added as features.

We combined such semantic features with the previously considered characterizations such as symbols, (normalized) terms and term walks, and we measured the impact of such combinations on the performance of the learning algorithms on several corpora using standard machine learning evaluations and fully automated theorem proving evaluation.

My contributions were the features based on discrimination trees and discrimination nets, the proposed linearization of terms to provide more matching features, and the machine learning evaluation.

## 2.9. Further Contributions

After receiving my PhD, I have further (co-)authored the following publications which are not included in this thesis (only journal and international conferences with proceedings are listed). The papers [12–20] directly use or are used by AI/ATP techniques, whereas papers [21–28] are related to formalization of mathematics, nominal logic, and formalization of computer algebra.

[12] Jasmin C. Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *J. Formalized Reasoning*, in press, 2015.

[13] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Learning To Parse on Aligned Corpora. Accepted for the *Proc. 6h Conference on Interactive Theorem Proving (ITP'15)*, to appear in LNCS, 2015.

[14] Cezary Kaliszyk, Stephan Schulz, Josef Urban, and Jiří Vyskočil. System Description: E.T. 0.1. Accepted for *International Conference on Automated Deduction (CADE'15)*, to appear in LNCS, 2015.

[15] Cezary Kaliszyk, Josef Urban, Umair Siddique, Sanaz Khan-Afshar, Cvetan Dunchev, and Sofiene Tahar. Formalizing Physics: Automation, Presentation and Foundation Issues. Accepted for *Conference on Intelligent Computer Mathematics (CICM'15)*, to appear in LNCS, 2015.

[16] Cezary Kaliszyk, Lionel Mamane, and Josef Urban. Machine Learning of Coq Proof Guidance: First Experiments. In *Proc. 6th International Symposium on Symbolic Computation in Software Science (SCSS'14)*, volume 30 of EPiC, pp. 27—34, 2014.

[17] Thibault Gauthier and Cezary Kaliszyk. Matching Concepts across HOL Libraries. In Proc. of the 7th *Conference on Intelligent Computer Mathematics (CICM'14)*, volume 8543 of LNCS, pp. 267–281. 2014. doi: `10.1007/978-3-319-08434-3_20`

[18] Cezary Kaliszyk and Florian Rabe. Towards Knowledge Management for HOL Light. In Proc. of the 7th *Conference on Intelligent Computer Mathematics (CICM'14)*, volume 8543 of LNCS, pp. 357–372. 2014. doi: `10.1007/978-3-319-08434-3`

[19] Cezary Kaliszyk and Alexander Krauss. Scalable LCF-style proof translation. In Proc. of the 4th *International Conference on Interactive Theorem Proving (ITP'13)*, volume 7998 of LNCS, pp. 51–66. 2013. doi: `10.1007/978-3-642-39634-2_7`

[20] Cezary Kaliszyk, Josef Urban, Jiri Vyskocil, and Herman Geuvers. Developing Corpus-Based Translation Methods between Informal and Formal Mathematics: Project Description. In Proc. of the 7th *Conference on Intelligent Computer Mathematics (CICM'14)*, volume 8543 of LNCS, pp. 435–439. 2014. doi: `10.1007/10.1007/978-3-319-08434-3`

[21] Carst Tankink, Cezary Kaliszyk, Josef Urban, and Herman Geuvers. Formal Mathematics on Display: A Wiki for Flyspeck. In Proc. of the 6th *Conference on Intelligent Computer Mathematics (CICM'13)*, volume 7961 of LNCS, pp. 152–167. 2013. doi: `10.1007/978-3-642-39320-4`

[22] Carst Tankink, Cezary Kaliszyk, Josef Urban, and Herman Geuvers. Communicating Formal Proofs: The Case of Flyspeck. In Proc. of the 4th *International Conference on Interactive Theorem Proving (ITP'13)*, volume 7998 of LNCS, pp. 451–456. 2013. doi: `10.1007/978-3-642-39634-2_32`

[23] Christian Urban and Cezary Kaliszyk. General bindings and alpha-equivalence in Nominal Isabelle. *Logical Methods in Computer Science*, 8(2), 2012. doi: `10.2168/LMCS-8(2:14)2012`.

[24] Cezary Kaliszyk and Henk Barendregt. Reasoning about constants in Nominal Isabelle, or how to formalize the second fixed point theorem. In *Proc. of the First International Conference on Certified Programs and Proofs (CPP'11)*, volume 7086 of *LNCS*, pages 87–102. Springer, 2011. doi: `10.1007/978-3-642-25379-9_9`

[25] Cezary Kaliszyk and Christian Urban. Quotients revisited for Isabelle/HOL. In *Proc. of the 26th ACM Symposium on Applied Computing (SAC'11)*, pages 1639–1644. ACM, 2011. doi: `10.1145/1982185.1982529`

[26] Christian Urban and Cezary Kaliszyk. General bindings and alpha-equivalence in Nominal Isabelle. In G. Barthe, editor, *Proc. of the 20th European Symposium on Programming (ESOP'11)*, volume 6602 of *LNCS*, pp. 480–500, 2011. doi: `10.1007/978-3-642-19718-5`

[27] Cezary Kaliszyk and Tetsuo Ida. Proof assistant decision procedures for formalizing origami. In *Proc. of the 18th Symposium on the Integration of Symbolic Computation and Mechanised Reasoning (Calculemus'11)*, volume 6824 of *LNCS*, pages 45–57. Springer, 2011. doi: `10.1007/978-3-642-22673-1_4`

[28] Cezary Kaliszyk. Counting derangements, counting non bijective functions and the birthday problem. *Formalized Mathematics*, 18(4):197–200, 2010. doi: `10.2478/v10037-010-0023-9`.

## 2.10. Related Work

There are many systems that provide proof advice for ITPs based on either AI methods or ATP methods. The exact components of such advice systems vary. The Mizar Proof Advisor [Urb06a] started as an independent library search tool, that was later combined with automated reasoning techniques. Machine learning advice has also been integrated on the level of a proof assistant user interface by ML4PG [KHG13], or natural language proof system Naproche [CKKS10].

There have been many bridges from ITPs to ATPs, including the Otter and ACL2 Ivy bridge [MM00], the Faust prover integrated in HOL [KKS91], MESON in HOL Light [Har96c], longer lists of ATP provers in ILF [DGHW97] and KIV [ABH+98], leanTAP in Isabelle [Pau99], Metis in HOL4 and Isabelle/HOL [Hur03], and Bliksem in Coq [BHdN02]. The Omega framework [Mei00, SBF+03] integrated many provers.

The TPTP framework [Sut10] developed by Sutcliffe has played an important role in bridging some ITPs, ATPs, and SMTs. The Large Theory Batch category of the CASC competition organized since 2008 benchmarks the performance of ATPs on the large problems extracted from Mizar, Isabelle/HOL, HOL Light, HOL4, as well as SUMO [PS07b] and Cyc [RPG05]. This has stimulated the development of ATPs: Voronkov and Riazanov customized Vampire to answer queries over SUMO [RV02], Urban built MoMM [Urb06b] in E-prover, and Hoder and Voronkov included premise selection in Vampire [HV11] (later SiNE was also included in E-Prover [Sch13]).

A number of interactive proof systems are built around automation, this is the case for NQTHM [BM84] and ACL2 [Kau92]. Finally there are systems that resemble hammers, but have been tailored for software verification, such as Why3 [FP13] and the TLA+ environment [CDLM08, MV12].

## 2.11. Glossary

The following abbreviations and system names have been frequently used in the first part of this thesis:

**ATP** Automated Theorem Proving

**AR** Automated Reasoning

**ARLT** Automated Reasoning in Large Theories

**Coq** A proof assistant for the Calculus of Inductive Constructions
`http://coq.inria.fr/`

**Flyspeck** Formal Proof of the Kepler Conjecture project [HAB+15, HHM+10]
`https://code.google.com/p/flyspeck/`

**HOL** Higher-order logic [Pit93]

**HOL Light** A contemporary proof assistant based on the HOL logic [Har96a]
`https://www.cl.cam.ac.uk/~jrh13/hol-light/`

**HOL4** A contemporary proof assistant based on the HOL logic [SN08]
`http://hol.sourceforge.net/`

**Isabelle** A generic proof assistant, with main development focus on higher-order logic
`http://isabelle.in.tum.de/`

**ITP** Interactive Theorem Proving

**k-NN** k-Nearest Neighbours classification method [Dud76]

**leanCoP** Lean Connection-Based Prover [OB03, Ott10]
`http://leancop.de/`

**LSI** Latent Semantic Indexing [DDL+90]

**MESON** Model elimination ATP prover integrated in HOL Light [Har96c]

**Metis** Ordered paramodulation ATP prover integrated in Isabelle/HOL and HOL4 [Hur03]
`www.gilith.com/software/metis/`

**MML** Mizar Mathematical Library [RT99, Wie99]
`http://www.mizar.org/`

**SNoW** Sparse Network of Winnows [CCRR99]
`http://cogcomp.cs.illinois.edu/page/software_view/SNoW`

**SMT** Satisfiability Modulo Theories

**TPTP** Thousands of Problems for Theorem Proving [Sut10]
`http://tptp.org/`

# Part II

# Selected Papers

# 3 Automated Reasoning with HOL Light

## Publication Details

Cezary Kaliszyk and Josef Urban. Learning-assisted Automated Reasoning with Flyspeck. *J. Automated Reasoning*, 53(2):173–213, Springer, 2014. doi: 10.1007/s10817-014-9303-3.

## Abstract

The considerable mathematical knowledge encoded by the Flyspeck project is combined with external automated theorem provers (ATPs) and machine-learning premise selection methods trained on the Flyspeck proofs, producing an AI system capable of proving a wide range of mathematical conjectures automatically. The performance of this architecture is evaluated in a bootstrapping scenario emulating the development of Flyspeck from axioms to the last theorem, each time using only the previous theorems and proofs. It is shown that 39% of the 14185 theorems could be proved in a push-button mode (without any high-level advice and user interaction) in 30 seconds of real time on a fourteen-CPU workstation.

The necessary work involves: (i) an implementation of sound translations of the HOL Light logic to ATP formalisms: untyped first-order, polymorphic typed first-order, and typed higher-order, (ii) export of the dependency information from HOL Light and ATP proofs for the machine learners, and (iii) choice of suitable representations and methods for learning from previous proofs, and their integration as advisors with HOL Light. This work is described and discussed here, and an initial analysis of the body of proofs that were found fully automatically is provided.

## 3.1. Introduction and Motivation

*"It is the view of some of us that many people who could have easily contributed to project QED have been distracted away by the enticing lure of AI or AR."*

– The QED Manifesto

*"So it will take 140 man-years to create a good basic library for formal mathematics."*

– Freek Wiedijk [Wie01]

*"We will encourage you to develop the three great virtues of a programmer: laziness, impatience, and hubris."*

– Larry Wall, Programming Perl [Wal00]

*"And in demonstration itself logic is not all. The true mathematical reasoning is a real induction [...]"*

– Henri Poincaré, Science and Method [Poi13]

### 3.1.1. Large-Theory Automated Reasoning and HOL Light

Use of external first-order automated theorem provers (ATPs) like Vampire [KV13], E [Sch02], SPASS [WDF$^+$09], and recently also SMT (satisfiability modulo theories) solvers like Z3 [dMB08] for (large-theory) formalization has been developed considerably in the recent decade. Particularly in the Isabelle community, the Sledgehammer [BBN11, BBP11] bridge to such external tools is getting increasingly popular. This helps to further develop various parts of the technology involved. ATPs have recently gained the ability to quickly load large theories over large signatures and work with them [HV11]. Methods for automated selection of relevant knowledge and for proof guidance are actively developed [Urb11a], together with specialized automated systems targeted at particular mathematical domains [AP10,PW06,Bee01]. Formats and translation methods handling more formalization-friendly foundations are being defined [SSCB12,GS06,BP13], and metasystems that decide which ATP, translation method, strategy, parallelization, and premises to use to solve a given problem with limited resources are being designed [PB12,USPV08]. Cooperation of humans and computers over large corpora of formal knowledge is an interesting field, allowing exploration of new AI systems and combinations of different AI techniques that can attempt to encode concepts like analogy and intuition, and rigorously evaluate their usefulness. Perhaps not only Hilbert and Turing, but also the formality-opposing and intuition-oriented Poincaré[1] [Poi13] would have been interested to learn about the new "semantic AI paradise" of such large corpora of fully computer-understandable mathematics (from which we do not intend to be expelled).

The HOL Light [Har96a] system is probably the first among the existing well-known interactive theorem provers (ITPs) which has integrated and extensively used a general ATP procedure, the MESON tactic [Har96c]. Hurd has developed and benchmarked early bridges [Hur99,Hur02] between HOL and external systems, and his Metis system [Hur03] has also become a significant part of the Isabelle/Sledgehammer bridge to ATPs [PS07a]. Using the very detailed Otter/Ivy [MM00] proof objects, Harrison also later implemented a bridge from HOL Light to Prover9 [McC10]. HOL Light however does not yet have a

[1]2012 is not just the year of Turing [Hal14b], but also of Poincaré, whose ideas about creativity and invention involving random, intuition-guided exploration confirmed by critical evaluation quite correspond to what AI systems like MaLARea [USPV08] try to emulate in large formal theories.

general bridge to large-theory ATP/AI ("hammer"[2]) methods, similar to Isabelle/Sledgehammer or Miz$\mathbb{AR}$ [URS13, US10], which would attempt to automatically solve a new goal by selecting relevant knowledge from the large library and running (possibly customized/trained) external ATPs on such premise selections. HOL Light seems to be a natural candidate for adopting such methods, because of the amount of work already done in this direction mentioned above, and also thanks to HOL Light's foundational closeness to Isabelle/HOL. Also, thanks to the Flyspeck project [Hal06], HOL Light is becoming less of a "single, very knowledgable formalizer" tool, and is getting increasingly used as a "tool for interested mathematicians" (such as the Flyspeck team in Hanoi[3]) who may know the large libraries much less and have less experience with crafting their own proof tactics. For such ITP users it is good to provide a small number of strong methods that allow fast progress, which can perhaps also complement the declarative modes [Wie12] pioneered by HOL Light [Har96b] in the LCF world.

### 3.1.2. Flyspeck as an Interesting Corpus for Semantic AI Methods

The purpose of the Flyspeck project is to produce a formal proof of the Kepler Conjecture [Kep11, HHM+10]. The Flyspeck development (which in this paper always means also the required parts of the HOL Light library) is an interesting corpus for a number of reasons. First, it formalizes considerable parts of standard mathematics, and thus exposes a large body of interconnected mathematical reasoning to all kinds of semantic AI methods and experiments. Second, the formalization is done in a relatively directed way, with the final goal of the Kepler conjecture in mind. For example, in the Mizar library[4] (and even more in other collections like the Coq contribs[5]), articles may be contributed as isolated developments, and only much later (or never) re-factored into a form that makes them work well with related developments. Such refactoring is often a nontrivial process [RT03]. In a directed development like Flyspeck, such integrity is a concern from the very beginning, and this concern should result in the theorems working better together to justify new conjectures that combine the areas covered by the development. Third, the language of HOL Light is in a certain sense simpler than the language of Mizar and Coq (and to a lesser extent also than Isabelle/HOL), where one typically first needs to set up the right syntactic/type-automation environment to be able to formulate new conjectures in advanced areas. This greater simplicity (which may come at a cost) makes it possible to write direct (yet advanced) queries to the AI/ATP ("hammer") system in the original language, without much additional need for specifying the context of the query. This could make such "hammer" more easy to try for interested mathematicians, and allow them to explore formal mathematics and Flyspeck. And fourth,

[2]Larry Paulson is guilty of introducing this "striking" terminology.

[3]`http://weyl.math.pitt.edu/hanoi2009/Participants/`

[4]`www.mizar.org`

[5]`http://coq.inria.fr/V8.2pl1/contribs/bycat.html`

Flyspeck is accompanied with an informal (LaTeX) text that is often cross-linked to the formal concepts and theorems developed in HOL Light. With sufficiently strong automated reasoning over the library, this cross-linking opens the way to experiments with alignment (and eventual semi-automated translation) between the informal and formal Flyspeck texts, using corpus-driven methods for language translation, assisted by such an AI/ATP "hammer" as an additional semantic filter/advisor.

### 3.1.3. The Rest of the Paper

The work reported here makes several steps towards the above goals:

1. Sound and efficient translations of the HOL Light formulas to several ATP (TPTP) formalisms are implemented (Section 3.2). This includes the untyped first-order (FOF) format [SSCG06], the polymorphic typed first-order (TFF1) format [BP13], and the typed higher-order (THF) format [GS06, SB10].

2. Dependency information is exported from the Flyspeck proofs (Section 3.3). This allows experiments with re-proving of theorems by 17 different ATPs/SMTs from their HOL Light dependencies, and provides an initial dataset for machine learning of premise selection from previous proofs.

3. Several feature representations characterizing HOL Light formulas are proposed, implemented, and used for machine-learning of premise selection. Several preprocessing methods are developed for the dependency data that are used for learning. The trained premise-selection systems are integrated as external advisors for HOL Light. A prototype system answering real-time mathematical queries by running various parallel combinations of the premise selectors and ATPs is built and made available as an online service. See Section 3.4.

The methods are evaluated in Section 3.5, and it is shown that by running in parallel the most complementary proof-producing methods on a 14-CPU workstation, one now has a 39% chance to prove the next Flyspeck theorem within 30 seconds in a fully automated push-button mode (without any high-level advice). 50% of the Flyspeck theorems can be re-proved within 30 seconds by a collection of 7 ATP methods (run in parallel) if the HOL Light proof dependencies are used. 56% of the theorems could be proved by the union of all methods tried in the evaluation. An initial analysis of these sets of proofs is given in Section 3.6. It is shown that the proofs produced by the learning-advised ATPs can occasionally develop ideas that are very different from the original HOL Light proofs, and that the learning-advised ATPs can sometimes produce simpler proofs and discover duplications in the library. Section 3.7 discusses related work and Section 3.8 suggests future directions.

## 3.2. Translation of HOL Light Formulas to ATP Formats

The HOL logic differs from the formalisms used by most of the existing ATP and SMT systems. The main differences to first-order logic are the use of the polymorphic type system, and higher-order features (guarded by the type system) such as quantification (abstraction) over higher-order objects and currying. On the other hand, the logic is made classical and comes with a straightforward intended interpretation in ZFC. Translation of this logic (and its type-class extension used by Isabelle/HOL) to ATP formalisms has been an active research topic started already in the 90s. Prominent techniques, such as lambda lifting, suitable type system translation methods, etc., have been described several times [Hur99, Hur02, Har96c, MP08, Bla12]. Therefore this section assumes familiarity with these techniques, and only briefly summarizes the logic and the translation approaches considered, and their particular suitability for the experiments over the HOL Light corpora. For a comprehensive recent overview and discussion of this topic and the issues related to the translation see Blanchette's thesis [Bla12]. In particular, it contains the arguments about the soundness and (in)completeness of the translation methods that we eventually chose.

### 3.2.1. Summary of the HOL Logic

HOL Light uses the *HOL logic* [Pit93]: an extended variant of Church's simple type theory [Chu40]. Type variables (implicitly universally quantified) are explicitly added to the language (providing polymorphism), together with arbitrary type operators (constructors of compound types like `'int list'` and `'a set'`). In the HOL logic, the terms and types are intended to have a standard set-theoretical interpretation in HOL universes. A *HOL universe* $U$ is a set of non-empty sets, such that $U$ is closed under non-empty subsets, finite products and powersets, an infinite set $I \in U$ exists, and a choice function $ch$ over $U$ exists (i.e., $\forall X \in U : ch(X) \in X$) . The subsets, products, and powersets together also yield function spaces. A frequently considered example of a HOL universe is the set $V_{\omega+\omega} \setminus \{0\}$,[6] with $ch$ being its (ZFC-guaranteed) selector, and $I = \omega$. The standard $U$-interpretation of a *monomorphic* (i.e., free of type variables) type $\sigma$ is a set $[\![\sigma]\!] \in U$, a *polymorphic* (i.e., containing type variables) type $\sigma$ with $n$ type variables is interpreted as a function $[\![\sigma]\!] : U^n \to U$, and the arrow operator observes the standard function-space behavior (lifted to appropriate mappings for polymorphic types) on the type interpretations. The standard interpretation of a closed monomorphic term $t : \sigma$ is an element of the set $[\![\sigma]\!] \in U$, and a closed polymorphic term (with $n$ type variables) $t : \sigma$ with $[\![\sigma]\!] : U^n \to U$ is interpreted as a (dependently typed) function assigning to each $n$-tuple $[X_1, \ldots, X_n] \in U^n$ an element of $[\![\sigma]\!]([X_1, \ldots, X_n])$. The HOL logic's

[6] $V_{\omega+\omega}$ is the $\omega + \omega$-th set of von Neumann's (cumulative) hierarchy of sets obtained by iterating the powerset operation starting with the empty set $\omega + \omega$ times. This shows that the HOL logic is in general weaker than ZFC.

type signature starts with the built-in nullary type constants `ind`, interpreted as the infinite set $I$, and `bool` (type of propositions), interpreted as a chosen two-element set in $U$ (its existence follows from the properties of a HOL universe). The term signature initially contains the polymorphic constants $=_{\alpha\to\alpha\to bool}$, and $\epsilon_{(\alpha\to bool)\to\alpha}$, interpreted as the equality and selector on each set in $U$. The inference mechanisms start with a set of standard primitive inference rules, later adding the axioms of functional extensionality, choice (implying the excluded middle in the HOL setting), and infinity. New type and term constructors can be introduced by simple definitional extension mechanisms, which are in HOL Light also used to introduce the standard logical connectives and quantifiers. The result is a classical logic system that is in practice quite close to set theory, differing from it mainly by the built-in type discipline (allowing also complete automation of abstraction) and by more frequent use of total functions to model mathematical objects. For example, predicates are modelled as total functions to `bool` on types, and sets are in HOL Light identified with (unary) predicates. The main issues for translation are the type system and the automated reification (abstraction) mechanisms that are not immediately available in first-order logic and may be encoded in more or less efficient and complete ways.

### 3.2.2. The MESON Translation

An obvious first idea for generating FOL ATP problems from HOL Light problems was to re-use parts of the already implemented MESON tactic. This tactic tries to justify a given goal $G$ with a supplied list of premises $P_1, ..., P_n$ by calling a customized first-order ATP implemented in HOL Light, which is based on the model elimination method invented by Loveland [Lov68], later combined with a Prolog-like search tree [Lov78]. The implementation of the MESON tactic in HOL Light first applies a number of standard translation techniques (such as $\beta$-reduction followed by lambda lifting, skolemization, introduction of the `apply` functor,[7] etc.) that transform the HOL goal (together with the supplied premises) to a clausal FOL goal (or multiple goals). An interesting (and MESON-specific) part of the transformation is a rather exhaustive and heuristic instantiation of the (often polymorphic) premises (called `POLY_ASSUME_TAC`), described below. The clausal FOL goal is then passed to the core ATP. If the core ATP succeeds, it returns a proof, which is then translated into HOL Light proof steps. The transformation from HOL to FOL is heuristic, incomplete, and tuned for relatively small problems. An interesting feature of MESON is that the core ATP does not treat equality specially (as is quite common in tableau provers), which in turn allows using multiple instantiated versions of equality (e.g., on lists and on real numbers) inside one problem. Such equational separation, when combined with the heuristic instantiation of other polymorphic constants done by MESON, then prevents the core ATP from doing ill-typed inferences

[7]Identity is used by MESON as the `apply` functor.

without the necessity for any additional type guards.

The most interesting part of the translation heuristically instantiates the (possibly polymorphic) premises $P_1, ..., P_n$ and adds them to the goal $G$. This is done iteratively, building a new temporary goal $G_i$ (where $G_0 = G$) from each premise $P_i$ and the previous goal $G_{i-1}$ as follows. All (possibly polymorphic) constants are collected from $P_i$ and $G_{i-1}$, and the set of all their pairs is created. When such a pair $\{c_P, c_G\}$ consists of two (symbolically) equal constants, the type of $c_P$ is matched to the type of $c_G$, and if a substitution $\sigma$ exists (i.e., $Type_{c_P}\sigma = Type_{c_G}$), it is added to the resulting set $\Sigma_i$ of type substitutions. Each type substitution from $\Sigma_i$ is then applied to $P_i$, and all such resulting instances of $P_i$ are added as assumptions to $G_{i-1}$, yielding $G_i$. The set of assumptions of the goal $G_i$ is thus typically greater than that of $G_{i-1}$, and the same typically holds for the set of constants in $G_i$, which will be in turn used to instantiate $P_{i+1}$.

This procedure is quite effective for the small problems that MESON normally handles. However, for problems with many premises and many polymorphic constants this turns out to be very inefficient. While re-using MESON allowed the quick initial exploration of using external ATPs and advisors described in [KU13b], this inefficiency practically excluded the (seemingly straightforward) use of the unmodified MESON procedure as an (at least basic) translation method for generating ATP problems with many premises. This is why the experiments presented here use different translations, described below.

Completeness of the translation from HOL to FOL is in general hard to achieve in an efficient way. The MESON translation is incomplete in several ways. The goal's proper assumptions are not monomorphised, and the free variables of polymorphic types are not used in the same way as the polymorphic constants. For example, given the premise:

$$\forall P : \alpha \to \text{bool}.\ \forall x : \alpha.\ Px$$

and a goal that does not mention $\alpha$, the premise will never be instantiated to the type present in the goal, and thus will not be usable for MESON.

### 3.2.3. Translation to the TFF1 and FOF Formats

There is a "simple" solution to the instantiation blow-up experienced with the MESON translation: avoid heuristic instantiation as a pre-processing step, and instead let the ATPs handle it as a part of the ATP problems. This technique is used in the Mizar/MPTP translation [Urb06c, Urb04, Urb03], where the (dependent and undecidable) soft type system cannot be separated from the core predicate logic. The relevant heuristics can instead be developed (and experimented with) on the level of ATPs. Indeed, for example the SPASS system includes a number of ATP techniques for both complete and incomplete work with (auto-detected) types [WAB+99, BPWW12]. This approach has been in the recent years facilitated by developing type-aware TPTP standards such as TFF0, TFF1, and THF, which – unlike related type-aware efforts like DFG [HKW96] and KIF [GF92] – seem to be more successful in being adopted by ATP and tool developers. In the case of the recent TFF1 standard [BP13] adding HOL-like polymorphic types to first-order logic, a translation tool to the FOF and SMT formats has been developed in 2012 by Andrei Paskevich as part of the Why3 system [FM07], simplifying the first experiments with the non-instantiating translation.

The translation to TFF1 proceeds similarly to the MESON translation, but without applying the `POLY_ASSUME_TAC`. The problem formulas are $\beta$-reduced, the remaining lambda abstractions are again removed using lambda lifting, and the `apply` functor is heuristically introduced. The particular heuristic for this is the one used by Meng and Paulson, i.e., for each higher-order constant $c$ the minimum arity $n_c$ with which it appears in a problem is computed, and the first $n_c$ arguments are always passed to $c$ directly inside the problem. If the constant is also used with more arguments in the problem, `apply` is used. Blanchette [Bla12] reports that this optimization works fairly well for Isabelle/Sledgehammer, and gives a simple example when it introduces incompleteness. As an example of the translation to the TFF1 format, consider the re-proving problem[8] for the theorem `Float.REAL_EQ_INV`[9] proved as part of the Jordan curve theorem formalization,[10] whose HOL Light proof is as follows:

```
let REAL_EQ_INV = prove(‘∀x y. ((x:real = y) ⟺ (inv(x) = inv (y)))‘,
((REPEAT GEN_TAC))
THEN (EQ_TAC)
THENL [((DISCH_TAC THEN (ASM_REWRITE_TAC[])));
 (* branch 2*) ((DISCH_TAC))
THEN ((ONCE_REWRITE_TAC [(GSYM REAL_INV_INV)]))
THEN ((ASM_REWRITE_TAC[]))]);;
```

The dependency tracking (see Section 3.3.1) has found the following dependencies of the theorem:[11]

```
AND_DEF FORALL_DEF IMP_DEF REAL_INV_INV REFL_CLAUSE TRUTH
Tactics_jordan.unify_exists_tac_example
```

From these dependencies, only `REAL_INV_INV` has nontrivial first-order content (a list of the trivial facts has been collected and is used for such filtering). The problem creation additionally adds three facts encoding properties of (HOL) booleans, and also the functional extensionality axiom (`EQ_EXT`). In the original HOL Light syntax the re-proving problem looks as follows:

[8] By a *re-proving problem*, we mean the ATP problem consisting of the translated HOL Light theorem together with the premises used in its original HOL Light proof.

[9] `http://mws.cs.ru.nl/~mptp/hh1/OrigDepsProbs/i/p/09895.p`

[10] `http://mws.cs.ru.nl/~mptp/hol-flyspeck/trunk/Jordan/float.html#REAL_EQ_INV`

[11] `Tactics_jordan.unify_exists_tac_example` is just ‘`T=T`‘. The name is accidental.

```
%   ORIGINAL: Float.REAL_EQ_INV
% Assm: EQ_EXT: !f g. (!x. f x = g x) ==> f = g
% Assm: BOOL_CASES_AX: !t. (t <=> T) \/ (t <=> F)
% Assm: NOT_CLAUSES_WEAK_conjunct1: ~F <=> T
% Assm: REAL_INV_INV: !x. inv (inv x) = x
% Assm: TRUTH: T
% Goal: !x y. x = y <=> inv x = inv y
```

After applying $\beta$-reduction, lambda lifting (none in the example), and introducing the `apply` functor (called here `happ`), this is transformed (still as HOL terms) into the following:

```
%   PROCESSED
% Assm: !f g. (!x. happ f x = happ g x) ==> f = g
% Assm: !t. (t <=> T) \/ (t <=> F)
% Assm: ~F <=> T
% Assm: !x. inv (inv x) = x
% Assm: T
% Goal: !x y. x = y <=> inv x = inv y
```

The application functor `happ` was only used for the function variables in the extensionality axiom (`EQ_EXT`). The function `inv` is always used with one argument in the problem, so it is never wrapped with `happ`. Finally, the TFF1 TPTP export declares the signature of the symbols and type operators, and adds the corresponding guarded quantifications to the formulas. The `apply` functor is called `i` in the TFF1 export (for concise output in case of many applications in a goal), and it explicitly takes also the type arguments (A and B in `aEQu_EXT`). This (making the implicit type variables explicit) is in TFF1 done for any symbol that remains polymorphic. We reserve the predicate `p` for translation between Boolean terms and formulas. This is done in the same way as in [MP08].

HOL Light allows one identifier to denote several different underlying constants. In the running example, `inv` is such an overloaded identifier and denotes the inverse operations on several different types. To deal with such identifiers different names are used for each underlying constant separately in the TFF1 export signature, so that the identifiers can be printed using their non-overloaded names like `real_inv`.

```
%   TYPES
tff(tbool, type, bool:$tType).
tff(tfun, type, fn:($tType * $tType) > $tType).
tff(treal, type, real:$tType).
%   CONSTS
tff(cp, type, p : (bool > $o)).
tff(chapp, type, i:!>[A:$tType,B:$tType]: ((fn(A,B) * A) > B)).
tff(cF, type, f:bool).
tff(crealu_inv, type, realu_inv:(real > real)).
tff(cT, type, t:bool).
%   AXIOMS
tff(aEQu_EXT, axiom, ![A : $tType,B : $tType]:
    ![F:fn(A,B),G:fn(A,B)]:(![X:A]:i(A,B,F,X) = i(A,B,G,X) => F = G)).
tff(aBOOLu_CASESu_AX, axiom, ![T:bool]:(T = t | T = f)).
tff(aNOTu_CLAUSESu_WEAKu_conjunct1, axiom, (~ (p(f)) <=> p(t))).
tff(aREALu_INVu_INV, axiom, ![X:real]:realu_inv(realu_inv(X)) = X).
tff(aTRUTH, axiom, p(t)).
tff(conjecture, conjecture, ![X:real,Y:real]:
    (X = Y <=> realu_inv(X) = realu_inv(Y))).
```

Problems in this format can be already given to the Why3 tool, which can translate them for various SMT solvers and ATP systems, and call the systems on the translated form. This was initially used both for ATPs working with the FOF format and for the SMTs. Currently, we only use Why3 for preparing problems for Yices, CVC3, and AltErgo. The translation to the FOF format was later implemented independently of Why3, to avoid an additional translation layer for the strongest tools, and in particular to be able to run the ATPs with different parameters and in a proof-producing mode. The procedure is however the same as in Why3, and the resulting FOF form will be as follows.

```
% Goal: !x y. x = y <=> inv x = inv y
fof(aEQu_EXT, axiom, ![A,B]: ![F, G]:
    (![X]: s(B,i(s(fun(A,B),F),s(A,X))) = s(B,i(s(fun(A,B),G),s(A,X)))
    => s(fun(A,B),F) = s(fun(A,B),G))).
fof(aBOOLu_CASESu_AX, axiom,
    ![T]: (s(bool,T) = s(bool,t) | s(bool,T) = s(bool,f))).
fof(aNOTu_CLAUSESu_WEAKu_conjunct1, axiom,
    (~ (p(s(bool,f))) <=> p(s(bool,t)))).
fof(aREALu_INVu_INV, axiom,
    ![X]: s(real,realu_inv(s(real,realu_inv(s(real,X))))) = s(real,X)).
fof(aTRUTH, axiom, p(s(bool,t))).
fof(conjecture, conjecture, ![X, Y]: (s(real,X) = s(real,Y) <=>
    s(real,realu_inv(s(real,X))) = s(real,realu_inv(s(real,Y))))).
```

This translation uses the (possibly quadratic) tagging of terms with their types (with “s” as the tagging functor), used, e.g., in Hurd’s work.

### 3.2.4. Translations to Higher-Order Formats

The recently developed TPTP THF standard can be used to encode problems in monomorphic higher-order logic. This allows experimenting with higher-order automated theorem provers like LEO2 [BPTF08] and Satallax [Bro12], in addition to the standard ATPs working in the first-order formalism. The translation to THF needs to perform only one step: monomorphisation. As explained in 3.2.2, this is however a nontrivial task, and the MESON tactic approach is already in practice too exhaustive for problems with many premises.

After developing the TFF1 and FOF translation, some initial experiments were done to produce a monomorphisation heuristic that behaves reasonably on problems with many premises. This heuristic is now as follows. The constants that can be used to instantiate the premises are extracted only once from the goal at the start of the procedure. Every premise can be instantiated using these goal constants, but the premises themselves are not further used to grow this set. This means that the procedure is even less complete than MESON, however the procedure is linear in the number of premises, and it is therefore possible to use it even with large numbers of advised premises. In practice, it is rarely the case that a premise can be instantiated in more than one way. A simple example when this happens is in the THF problem[12] created for the theorem `I_O_ID`,[13] where the particular goal and premise (both properties of the identity function) are as follows:

```
Assm: I_THM: !x. I x = x
Goal: I_O_ID: !f. I o f = f /\ f o I = f
```

The exact types inferred by the standard HOL (Hindley-Milner [Hin69]) type inference for the goal are as follows:

$$\forall f : A \to B.\ I_{B\to B}\ o\ f = f \wedge f\ o\ I_{A\to A} = f$$

Since the identity function appears in the goal both with the type `A→A` and with the type `B→B`, the following two instances of the premise `I_THM` are created by the THF translation:

```
% TYPES
thf(ta, type, a : $tType).
thf(tb, type, b : $tType).
thf(ci0, type, i0 : (a > a)).
thf(ci, type, i : (b > b)).
% AXIOMS
thf(aIu_THMu_monomorphized0, axiom, ![X:a]:((i0 @ X) = X)).
```

[12]http://mws.cs.ru.nl/~mptp/hh1/OrigDepsProbs/i/h/00119.p
[13]http://mws.cs.ru.nl/~mptp/hol-flyspeck/trunk/trivia.html#I_O_ID

```
thf(aIu_THMu_monomorphized1, axiom, ![X:b]:((i @ X) = X)).
```

Finally, while there is no TPTP standard yet for the polymorphic HOL logic, this logic is shared by a number of systems in the HOL family of ITPs. For the experiments described in 3.5.1 Isabelle is used in its CASC 2012 THF mode, but it should be possible to pass the problems to Isabelle directly in some (not necessarily TPTP) polymorphic HOL encoding. This is has been tried only to a small extent, and is still future work.

## 3.3. Exporting Theorem Problems for Re-proving with ATPs

In our earlier initial experiments [KU13b], it was found that the ATP problems created from the calls to the MESON tactic in the HOL Light and Flyspeck libraries are very easy for the state-of-the-art ATPs. Some of this easiness might have been caused by the (generally unsound) merging of different polymorphic versions of equality used by MESON into just one standard first-order equality.[14] However, after a manual random inspection it still seemed that the ratio of such unsound proofs is low, and the MESON problems are just too easy. That is why only the set of problems on the *theorem* level is considered for experiments here. The *theorem* level seems to be quite similar in the major ITPs: *theorem* is typically not corresponding to what mathematicians call a theorem, but it is rather a self-sufficient lemma with a formal proof of several to dozens (exceptionally hundreds) lines that can be useful in other formal proofs and hence should be named and exported. Since the ITP proofs can be longer (i.e., they can contain a number of MESON and other subproblems), proving such theorems fully automatically is typically a challenge, which makes such problems suitable for ATP benchmarks, challenges, and competitions.

### 3.3.1. Collecting Theorems and their Dependency Tracking

In Mizar/MPTP and in Isabelle (done by Blanchette in so far unpublished work) the ATP problems corresponding to theorems can be produced by collecting the dependencies (premises) from the proofs (by suitable tracking mechanisms), and then translating the $Premises \vdash Theorem$ problem using the methods described in Section 3.2. The recent work by Adams in exporting HOL Light to HOL Zero [Ada10] (with cross-verification as the main motivation) was initially used to obtain the theorem dependencies for the first experiments with HOL Light in [KU13b], and after that custom theorem-exporting and dependency-tracking mechanisms were implemented as described below.

[14]Note that the typed translation that we use here prevents deriving ill-typed equalities [Bla12].

### Collecting and Naming of Theorems

The first issue in implementing such mechanism is to decide what is considered to be a relevant theorem, and what should be its canonical name. In some ITPs, important statements have labels like `lemma`, `theorem`, `corollary`, etc. This is not the case in HOL Light, which is implemented in the OCaml toplevel. This means that every theorem or tactic is just an OCaml value. Some of those values are assigned names, while some are only created on the fly and immediately forgotten. In the relevant exporting work of Obua [OS06], every occurrence of the HOL Light command `prove` is replaced with a command that additionally records the name of the stored object. This strategy was used first, and extended to work with the whole Flyspeck library by also recording the names for the following commands: `prove_by_refinement`, `new_definition`, `new_recursive_-definition`, `new_specification`, `new_inductive_definition`, `define_type`, and `lift_-theorem`. This purely syntactic replacement method however turned out to be insufficient for a number of reasons. First, this method does not provide information about the scope of names with respect to the OCaml modules. Second, it does not provide the information whether a name given to a theorem has been declared on the top level, or inline inside a function (which makes such theorem unusable for proving other theorems on the toplevel), or even within a function called multiple times with different arguments (in which case the same name would be assigned to a number of different theorems). Finally, certain theorems accessible on the top level are created using other OCaml mechanisms, for example mapping a decision procedure over a list of terms. Recognizing syntactically theorems created in this way turned out to be impractical.

That is why a more robust method has been eventually used, based on the `update_-database`[15] recording functionality by Harrison and Zumkeller. This code accesses the basic OCaml data structures and makes it possible to record the name-value pairs for all OCaml values of the type `theorem` in a given OCaml state. Thus, it is sufficient to load the whole Flyspeck development, and then invoke this recording functionality.

After some initial experiments with ATP re-proving of the translated problems, this method was however further modified to be able to keep finer track of the use of theorems that are conjunctions of multiple facts. Such (often large) conjunctive theorems are used quite frequently in HOL Light, typically to package together facts that are likely to jointly provide a useful method for dealing with certain concepts or certain kinds of problems. For example the theorem `ARITH_EQ`[16] packages together ten facts about the equality of numerals as follows:

[15]`http://mws.cs.ru.nl/~mptp/hol-flyspeck/trunk/Examples/update_database.html`
[16]`http://mws.cs.ru.nl/~mptp/hol-flyspeck/trunk/calc_num.html#ARITH_EQ`

```
let ARITH_EQ = prove
 (‘(∀m n. (NUMERAL m = NUMERAL n) ⟺ (m = n)) ∧
   ((_0 = _0) ⟺ T) ∧
   (∀n. (BIT0 n = _0) ⟺ (n = _0)) ∧
   (∀n. (BIT1 n = _0) ⟺ F) ∧
   (∀n. (_0 = BIT0 n) ⟺ (_0 = n)) ∧
   (∀n. (_0 = BIT1 n) ⟺ F) ∧
   (∀m n. (BIT0 m = BIT0 n) ⟺ (m = n)) ∧
   (∀m n. (BIT0 m = BIT1 n) ⟺ F) ∧
   (∀m n. (BIT1 m = BIT0 n) ⟺ F) ∧
   (∀m n. (BIT1 m = BIT1 n) ⟺ (m = n))‘,
  REWRITE_TAC[NUMERAL; GSYM LE_ANTISYM; ARITH_LE] THEN
  REWRITE_TAC[LET_ANTISYM; LTE_ANTISYM; DENUMERAL LE_0]);;
```

An even more extreme example is the `ARITH` theorem which conjoins together all the basic arithmetic facts (there are 108 of them in the current version of HOL Light). The conjuncts of such theorems are now also named (using a serial numbering of the form `ARITH_EQ_conjunctN`), so that the dependency tracking can later precisely record which of the conjuncts were used in a particular proof. This significantly prunes the search space for ATP re-proving of the theorems that previously depended on the large conjunctive dependencies, and also makes the learning data extracted from dependencies of such theorems more precise.

This method can however result in the introduction of multiple names for a single theorem (which is just a HOL Light term of type `theorem`). If that happens (for this or other reasons), the first name that was associated with the theorem during the Flyspeck processing is always consistently used, and the other alternative names are never used. Such consistency is important for the performance of the machine learning on the recorded proof data.[17] The list of all theorems and their names obtained in this way is saved in a file, and subsequently used in the dependency extraction and problem creation passes.

### Dependency Recording

After the detection and naming of theorems, the recording of proof dependencies is performed, by processing the whole library again with a patched version of the HOL Light kernel. This patched version is the proof-recording component of the new HOL-Import [KK13], a mechanism designed to transfer proofs from HOL Light to Isabelle/HOL in an efficient way allowing the export of big repositories like Flyspeck. The code for every HOL inference step is patched, to record the newly created theorems. Each theorem is assigned a unique integer counter, and for every new theorem its dependencies on other

[17]For example, the MaLARea system does such de-duplication as a useful preprocessing step before learning and theorem-proving is started on a large number of related problems.

theorems (integers) are recorded and exported to a file. For every processed theorem it is also checked if it is one of the theorems named in the previous theorem-naming pass. If so, the association of this theorem's name to its number is recorded, and again exported to a file.

After this dependency-recording pass, the recorded information is further processed by an offline program to eliminate all unnamed dependencies (originating for example from having multiple names for a single theorem). For every named theorem its dependencies are inspected, and if a dependency $D$ does not have a name, it is replaced by its own dependencies (there is no unnamed dependency that could not be further expanded). This is done recursively, until all unnamed dependencies are removed. This produces for each named theorem $T$ a minimal (wrt. the original HOL Light proof) list of named theorems that are sufficient to prove $T$.

The numbering of theorems respects the order in which the theorems are processed in the Flyspeck development. This total ordering is compatible with (extends) the partial ordering induced by proof dependencies, and for the experiments conducted here it is assumed to be the *chronological order* in which the library was developed. The dependency information given in this chronological order for all 16082 named theorems (of which 1897 are (type) definitions, axioms, or their parts, and their dependencies are not exported) obtained by processing the Flyspeck library[18] (and its HOL Light prerequisites) is available online.[19] Together with suitably chosen characterizations of the theorems (see Section 3.4.1), this constitutes an interesting new dataset for machine-learning techniques that attempt to predict the most useful premises from the formal library for proving the next conjecture.

### 3.3.2. The Data Set of ATP Re-proving Problems

Analogously to Mizar and Isabelle, the re-proving ATP problems for the collected named theorems are finally produced by translating the $\mathit{Dependencies} \vdash \mathit{Theorem}$ problem to the ATP formalisms using the methods described in Section 3.2, together with basic filtering of dependencies that have trivial first-order content. 1897 of the 16082 named theorems do not have a proof (those are definitions and axioms). For all the remaining 14185 named theorems the corresponding re-proving ATP problems were created, and are available online[20] in the FOF, THF, and TFF1 formats. These problems are used for the ATP re-proving experiments described in Section 3.5.1. Smaller meaningful datasets will likely be created from this large dataset for ATP/AI competitions such as CASC LTB

[18]Flyspeck SVN revision 2887 from 2012-06-26 and HOL Light SVN revision 146 from 2012-06-02 are used for all experiments.

[19]`http://mws.cs.ru.nl/~mptp/hh1/deps.all`

[20]`http://mws.cs.ru.nl/~mptp/hh1/OrigDepsProbs/co_i_f_p_h.tgz`

and Mizar@Turing[21], analogously to the smaller MPTP2078[22] [AHK+14] ATP benchmark created from the ATP-translated Mizar library (MML), and the Judgement Day benchmark [BN10] created by ATP translation of a subset of the Isabelle/HOL library.

The average, minimum, and maximum sizes of problems in these datasets are shown in Table 3.1, together with the corresponding statistics for the problems expressed in the original HOL formalism. It can be seen that the number of formulas in the translated

| Format | Problems | Average size | Minimum size | Maximum size |
|---|---|---|---|---|
| HOL | 14185 | 42.7 | 4 | 510 |
| FOF | 14185 | 42.7 | 4 | 510 |
| TFF1 | 14185 | 71.9 | 10 | 693 |
| THF | 14185 | 78.8 | 5 | 1436 |

Table 3.1.: Sizes of the re-proving ATP problems (in numbers of formulas)

problems is typically at most twice the number of the original HOL formulas, i.e., the translations are indeed efficient for all the problems. This was not the case (and became a bottleneck) in the initial experiments using the more prolific MESON translation. There is no increase in the number of formulas when translating from the original HOL-formulated problem to the FOF translation. For the TFF1 and THF translation, formulas declaring the types of the symbols appearing in the problems are added, and for the THF translation multiple instances of the premises can additionally appear. Table 3.2 shows the total times needed for the various exporting phases run over the whole Flyspeck as explained above. For completeness, the time needed to export characterizations of the theorems for external (e.g., machine-learning) tools is also included (see Section 3.4.1 for the description of the characterizations that are used).

## 3.4. Premise Selection

Given a large library like Flyspeck, the interesting ATP/AI task is to prove new theorems without having to manually select the relevant premises. In the past decade, a number of premise selection methods have been developed and experimented with over large theories like Mizar/MML, Isabelle/HOL, SUMO, and Cyc. See [Urb11a, KvLT+12] for recent overviews of such methods.

ATP problems of this kind are created for Mizar/MML by consistent translation of the whole MML to TPTP, and then letting external premise selection algorithms find the most relevant premises for a given theorem *t* from the large set of *t*-allowed premises (typically those theorems and definitions that were already available when *t* was being

[21]`http://www.cs.miami.edu/~tptp/CASC/J6/Design.html#CompetitionDivisions`
[22]`http://wiki.mizar.org/twiki/bin/view/Mizar/MpTP2078`

| Phase | Time (minutes) |
|---|---|
| standard Flyspeck loading/verification | 180 |
| detection and naming of theorems | 1 |
| exporting theorem characterizations | 5 |
| dependency recording using patched kernel | 540 |
| offline post-processing of dependencies | 10 |
| creating re-proving ATP problems in FOF | 34 |
| creating re-proving ATP problems in TFF1 | 53 |
| creating re-proving ATP problems in THF | 32 |
| total | 855 |

Table 3.2.: Times of the exporting phases (total, for the whole Flyspeck)

proved, expressed, e.g., as TPTP include files). For Isabelle/Sledgehammer, the default premise selection algorithm is implemented inside Isabelle, i.e., it is working on the native Isabelle symbols. Only after the Sledgehammer premise selection chooses the suitable set of premises, the problem is translated to a given ATP formalism using one of the several implemented translation methods. In general, the symbol naming is in Isabelle consistent only before the translation is applied, and a particular symbol in two translated problems can have different meanings.

Both the Mizar and the Isabelle approach have some advantages and disadvantages. Optimizing the translation (or using multiple translations) as done in Isabelle can improve the ultimate ATP performance once the premises have been selected. On the other hand, if the whole library is not translated in a consistent manner to a common ATP format such as TPTP, ATP-oriented external premise selection tools like SInE cannot be directly used on the whole library. It could be argued that the SInE algorithm is relatively close to the Sledgehammer premise selection algorithm, and can be easily implemented inside Isabelle. However there are useful premise selection methods for which this is not so straightforward. For example in the MaLARea system, evaluation of premises in a large common pool of finite first-order models is an additional semantic premise-characterization method that improves the overall precision quite significantly [USPV08].[23] For such a pool of first-order models to be useful, the premises have to use symbols consistently also after the translation to first-order logic. Although various techniques can again be developed to lift this method to the current Sledgehammer translation setting, they seem less straightforward than for example a direct Isabelle implementation of SInE. This discussion currently applies also to the HOL Light ATP

[23]Recent evidence for the usefulness of model-based selection methods is the difference (64% vs. 50% problems solved) between the (otherwise quite similar) systems MaLARea and PS-E (`http://www.cs.ru.nl/~kuehlwein/CASC/PS-E.html`) in the 2012 Mizar@Turing competition.

translations described in Section 3.2. For example, the problem-specific optimization of the arity of symbols described in 3.2.3 will in general cause inconsistency on the symbol level between the FOF translations of two different HOL Light problems.

The procedure implemented for HOL Light is currently a combination of the external, internal, learning, and non-learning premise-selection approaches. This procedure assumes the common ITP situation of a large library of (also definitional) theorems $T_i$ and their proofs $P_i$ (for definitions the proof is empty), over which a new conjecture has to be proved. The proofs refer to other theorems, giving rise to a partial dependency ordering of the theorems extended into their total chronological ordering as described for Flyspeck in 3.3.1. For the experiments it will be assumed that the library was developed in this order. An overview of the procedure is as follows, and its details are explained in the following subsections.

1. Suitable characterizations (see Section 3.4.1) of the theorems and their proof dependencies are exported from HOL Light in a simple format.

2. Additional dependency data are obtained by running ATPs on the ATP problems created from the HOL Light proof dependencies, i.e., the ATPs are run in the re-proving mode. Such data are often smaller and preferable [KU13g]. These data are again exported using the same format as in (1).

3. The (global, first-stage) external premise selectors preprocess (typically train on) the theorem characterizations and the proof dependencies. Multiple characterizations and proof dependencies may be used.

4. When a new conjecture is stated in HOL Light, its characterization is extracted and sent to the (pre-trained) first-stage premise selectors.

5. The first-stage premise selectors work as rankers. For a given conjecture characterization they produce a ranking of the available theorems (premises) according to their (assumed) relevance for the conjecture.

6. The best-ranked premises are used inside HOL Light to produce ATP (FOF, TFF1, THF) problems. Typically several thresholds (8, 32, 128, 512, etc.) on the number of included premises are used, resulting in multiple versions of the ATP problems.

7. The ATPs are called on the problems. Some of the best ATPs run in a strategy-scheduling mode combining multiple strategies. Some of the strategies always use the SInE (i.e., local, second-stage) premise selection (with different parameters), and some other strategies may decide to use SInE when the ATP problem is sufficiently large.

*Loop to improve (2) and (3):* It is not an uncommon phenomenon that in the data-improving step (2) (ATP re-proving from the HOL Light proof dependencies) an

ATP proof could not be found for some theorem $T_i$, but an alternative proof of $T_i$ can be found from some other theorems preceding $T_i$ in the chronological order (which guards such alternative proofs against cycles). To achieve this, the trained premise selectors can be used also on all theorems that are already in the library, and the whole ATP/training process can be iterated several times to obtain as many ATP proofs as possible, and better (and differently) trained premise selectors for step (3). This is the same loop as in MaLARea.

### 3.4.1. Formula Characterizations Used for Learning

Given a new conjecture $C$, how do mathematicians decide that certain previous knowledge will be relevant for proving $C$? The approach taken in practically all existing premise-selection methods is to extract from such $C$ a number of suitably defined features, and use them as the input to the premise selection for $C$. The most obvious characterization that already works well in large libraries is the (multi)set of symbols appearing in the conjecture. This can be further extended in many interesting ways, using various methods developed, e.g., in statistical machine translation and web search, but also by methods specific to the formal mathematical domain. In this work, characterization of HOL formulas by all their subterms (found useful in MaLARea) was used, and adapted to the typed HOL logic. For example, the latest version of the characterization algorithm would describe the HOL theorem `DISCRETE_IMP_CLOSED`:[24]

```
∀s:real^N→bool e.
      &0 < e ∧ (∀x y. x IN s ∧ y IN s ∧ norm(y − x) < e ⟹ y = x)
      ⟹ closed s
```

by the following set of strings:

```
"real", "num", "fun", "cart", "bool", "vector_sub", "vector_norm",
"real_of_num", "real_lt", "closed", "_0", "NUMERAL", "IN", "=", "&0",
"&0 < Areal", "0", "Areal", "Areal^A", "Areal^A - Areal^A",
"Areal^A IN Areal^A->bool", " Areal^A->bool", "_0",
"closed Areal^A->bool", "norm (Areal^A - Areal^A)",
"norm (Areal^A - Areal^A) < Areal"
```

This characterization is obtained by:

1. Normalizing all type variables to just one variable `A`.
2. Replacing (normalizing) all term variables with their normalized type.
3. Collecting all (normalized) types and their component types recursively.

[24]`http://mws.cs.ru.nl/~mptp/hol-flyspeck/trunk/Multivariate/topology.html#DISCRETE_IMP_CLOSED`

4. Collecting all (normalized) atomic formulas and their component terms recursively.
5. Including all "proper" term and type constructors (logic symbols like conjunction are filtered out).

In the above example, `real` is a type constant, `IN` is a term constructor, `Areal^A -> bool` is a normalized type, `Areal^A` its component type, `norm (Areal^A - Areal^A) < Areal` is an atomic formula, and `Areal^A - Areal^A` is its normalized subterm.

The normalization of variable names is an interesting topic. It is good if the premise selectors can notice some similarity between two terms with variables,[25] which is hard (when using strings) if the variables have different names. On the other hand, total normalization to just one generic variable name removes also the information that the variables in a particular subterm were (not) equal. Also, terms with differently typed variables should be more distant from each other than those with the same variable types. In total, four versions of variable normalization were tested:

`syms0`: All free and bound variables are given the same name `A0`. This encoding is the most liberal, i.e., the resulting equality relation on the features is the coarsest one, allowing the premise selectors to see many similarities.

`syms`: First the free variables are numbered consecutively (`A0`, `A1`, etc). Then the bound variables are named with the subsequent numbers. This results in a finer notion of similarity than in `syms0`.

`symst`: Every variable is renamed to a textual representation of its type, for example `Anum` or `Areal`. This is again finer than `syms0`, but different from `syms`. This normalization is used in the above example, and also for most of the premise selection trainings.

`symsd`: In one `symst` implementation, the internal HOL Light type variable numbering was accidentally used, thus making most of such term features disjoint between different theorems. The performance was lower, but the method produces some unique solutions and is included in the evaluation.

In addition to that, several feature exports included also logic symbols. Various feature characterizations can have different performance on different datasets, and such characterizations can be also combined together in interesting ways. This is a rather large research topic that is left as future work for this newly developed dataset, along with other large-theory datasets. Just including the features encoding the validity in finite models will be interesting.

[25] One could require the similarity to also handle matching, etc. A simple way how to do it is to generate even more features. This is again left to further general research in this area.

### 3.4.2. Machine Learning of Premise Selection

All the currently used first-stage premise selectors are machine learning algorithms trained in various ways on previous proofs. A number of machine learning algorithms can be experimented with today, and in particular kernel-based methods [AHK+14] and ensemble methods [KvLT+12] have recently shown quite good performance on smaller datasets such as MPTP2078. However, scaling and tuning such methods to a large corpus like Flyspeck and to quite a large number of incremental training and testing experiments is not straightforward.[26] That is why this work so far uses mostly the sparse implementation of a multiclass naive Bayes classifier provided by the SNoW system [CCRR99]. SNoW can incrementally train and produce predictions on the whole Flyspeck library presented in the chronological order in an hour (and often considerably faster on minimized data). I.e., one new prediction takes a fraction of a second. In addition to that, several other fast incremental (online) learning algorithms were briefly tried: the Perceptron and Winnow algorithms provided also by SNoW, and a custom implementation of the $k$-nearest neighbor ($k$-NN) algorithm. Only $k$-NN however produced enough additional prediction power. As already mentioned, the first-stage algorithms are often complemented by SInE as a second-stage premise selector when the ATP problem is written, and that is why some (in particular SInE-like) algorithms might look mostly redundant (in the overall ATP evaluation) when used at the first stage. This is obviously a consequence of the particular setup used here.

Two kinds of evaluation are possible in this setting and have been used several times for the Mizar data: a pure machine-learning evaluation comparing the predicted premises with the set of known sufficient premises, and an evaluation that actually runs an ATP on the predicted premises. While data are available also for the former, in this paper only the second evaluation is presented, see Section 3.5.2. The main reason for this is that alternative proofs are quite common in large libraries, and they often obfuscate the link between the pure machine-learning performance and the final ATP performance. Measuring the final ATP performance is more costly, however it practically stopped being a problem with the recent arrival of low-cost workstations with dozens of CPUs.

At a given point during the library development, the training data available to the machine learners are the proofs of the previously proved theorems in the library. A frequently used approach to training premise selection is to characterize each proof $P_i$ of theorem $T_i$ as a (multi)set of theorems $\{T_{i_1},...,T_{i_m}|T_{i_j}\ \textit{used in}\ P_i\}$. The training example will then consist of the input characterization (features) of $T_i$ (see 3.4.1), and the output characterization (called also output targets, classes, or labels) will be the (multi)set $\{T_i\} \cup \{T_{i_1},...,T_{i_m}|T_{i_j}\ \textit{used in}\ P_i\}$. Such training examples can be tuned in various ways. For example the output theorems may be further recursively expanded with their own dependencies, the input features could be expanded with the features of their definitions, various weighting schemes and similarity clusterings can be tried, etc.

[26]Such scaling up for the large Mizar library is work in progress at the time of writing this article.

This is also mostly left to future general research in premise-selection learning. Once the machine learner is trained on a particular development state of the library, it is tested on the next theorem $T$ in the chronological order. The input features are extracted from $T$ and given to the trained learner which then answers with a ranking of the available theorems. This ranking is given to HOL Light, which uses it to produce ATP problems for $T$ with varied numbers of the best-ranked premises.

### 3.4.3. Proof Data Used for Learning

An interesting problem is getting the most useful proof dependencies for learning. Many of the original Flyspeck dependencies are clearly unnecessary for first-order theorem provers. For example the definition of the $\wedge$ connective (`AND_DEF`) is a dependency of 14122 theorems. Another example are proofs done by decision procedures, which typically first apply some normalization steps justified by some lemmas, and then may perform some standard algorithm, again based on a number of lemmas. Often only a few of such dependencies are needed (i.e., the proofs found by decision procedures are often unnecessarily “complicated”).

Some obviously unhelpful dependencies were filtered manually, and this was complemented by using also the data obtained from ATP re-proving (see Section 3.5.1). Vampire, E, and Z3 can together re-prove 43.2% of the Flyspeck theorems (see Table 3.5), which is quite a high number, useful for trying to post-process automatically the remaining dependency data or even to completely disregard them. The following approaches to combining such ATP and HOL Light dependencies were initially tried, and combined with the various characterization methods to get the training data:

`minweight` (default): Always prefer the minimal ATP proof if available. On the ATP re-proved theorems collect the statistics about how likely a dependency in the HOL Light proof is really going to be used by the ATP proof, and use this likelihood as a weight when ATP proof is not available. When the weight is 0, use (cautiously) a minimal weight (0.001 or 0.000001) instead.

`nominweight:` As `minweight`, but without a minimal weight. Totally ignore ATP-irrelevant HOL Light dependencies.

`v_pref` (`e_pref, z_pref`): Instead of using the minimal ATP proof, always prefer the Vampire (E, Z3) proof. Can be combined with both weighting methods.

`symsonly:` Ignore all proofs. Learn only on examples saying that a theorem is good for proving itself, i.e., for its feature characterization. The trained system will thus recommend theorems similar to the conjecture, but not the dependencies of such theorems.

`atponly`: Use only the (minimal) ATP proofs for learning. Ignore the HOL Light proofs completely, and construct only the `symsonly` training examples for theorems that have no ATP proof. Can be combined with `v_pref`, `e_pref, z_pref`.

At some point, a *pseudo-minimization* procedure started to be applied first to the ATP proofs: each proof is re-run only with the premises needed for the proof, and if the number of needed premises decreases, this is repeated until the premise count stabilizes. Often this further removes unnecessary premises that appeared in the ATP proof, e.g., by performing unnecessary rewriting steps. This was later followed by adding *cross-minimization*: Each proof is re-run (pseudo-minimized) not just by the ATP that found the proof, but by all ATPs. This can further improve the training data, and also raise the number of proofs found by a particular ATP quite considerably, which in turn helps when proofs by a particular ATP are preferable for learning (see the `v_pref` approach above). Finally, the learning and proving can boost each other's performance: the proofs obtained by using the advice of the first-generation premise selectors can be added to the training data obtained from re-proving, and used to train the second generation of premise selectors. This process can be iterated, but only one iteration was done so far (using two different prediction methods).

The summary of the training data obtained by these procedures from the proofs is given in Table 3.3. Each of the ATP-obtained dependency sets (2–6 in Table 3.3) could be complemented by the HOL Light dependencies (1) as described above, producing differently trained advisors. For example, the best advising method based on (4) was only using the ATP proofs for training (no adding of HOL Light dependencies when the ATP proof was missing), preferring proofs by E (`e_pref`), using the `symst` (types instead of variables) characterizations, and choosing the best 128 premises. The new ATP proofs found using this method were added to (4), resulting in the dataset (5). The next most complementary advising method to that (measured before (5) became available) was combining the ATP dependencies from (2) with the HOL Light dependencies (1) using the `nominweight` and `v_pref` techniques, also using the `symst` features, and choosing the best 512 premises. The new ATP proofs found using this method were added to (5), resulting in (6). The performance of various premise selection methods is discussed in Section 3.5.

### 3.4.4. Communication with the Premise Advisors

There are several modes in which external premise selectors can be used. The main mode used here for experiments is the *offline (batch) mode.* In this mode, the premise selectors are incrementally trained and tested on the whole library dependencies presented as one file in the chronological order. *Incremental training/testing* means that the learning system reads the examples from the file one by one, for each theorem first producing an advice (ranking of previous theorems) based only on its features, and only after

| | | E | | Vampire | | Z3 | | **Union** | |
|---|---|---|---|---|---|---|---|---|---|
| Nr | Method | thm | dep ø | thm | dep ø | thm | dep ø | **thm** | **dep ø** |
| 1 | HOL deps | | | | | | | **14185** | **61.87** |
| 2 | ATP on (1), 300s | 5393 | 4.42 | 4700 | 5.15 | 4328 | 3.55 | **5837** | **3.93** |
| 3 | ATP on (1), 600s* | 5655 | 5.80 | 5697 | 5.90 | 4374 | 3.59 | **6161** | **5.00** |
| 4 | (3) minimized | 5646 | 4.52 | 5682 | 4.49 | 4211 | 3.47 | **6104** | **4.35** |
| 5 | (4) ∪ 1. advised | 6404 | 4.29 | 6308 | 4.17 | 5216 | 3.67 | **6605** | **4.18** |
| 6 | (5) ∪ 2. advised | 6848 | 3.90 | 6833 | 3.89 | 5698 | 3.48 | **6998** | **3.81** |

**thm:** Number of theorems proved by the given prover.

**dep ø:** The average number of proof dependencies in the proofs found by the prover.

**(1) - HOL deps:** Dependencies exported from HOL Light.

**(2),(3):** Dependencies obtained from proofs by ATPs run on HOL deps.

**(4):** Cross-minimization of (3).

**(5):** Added dependencies from new proofs advised by the best learning method (using (4)).

**(5):** Added dependencies from new proofs advised by the best complementary learning method (trained on (2) combined with (1)).

**(*):** The 6161 count in (3) is higher than in the final 900s experiments shown in Table 3.5. This is due to a incorrect (cyclic) dependency export for about 60 early HOL Light theorems used for (2)–(4). For training premise selection the effect of this error is negligible.

Table 3.3.: Improving the dependency data used for training premise selection

that learning on the theorem's dependencies and proceeding to the next example. The rankings are then used in HOL Light to produce all ATP problems in batch mode. This mode is good for experiments, because the learning data can be analyzed and pre-processed in various ways described above. All communication is fully file-based.

Another mode is used for *static online advice*. In this mode an (offline) pre-trained premise selector receives conjecture characterizations from HOL Light over a TCP socket, replies in real time with a ranking of theorems from which HOL Light produces the ATP problems and calls the ATPs to solve them. This mode has been initially implemented as a simple online service and can already be experimented with by interested readers.[27]

Finally, in a *dynamic online* mode the premise selector receives also training data in real time, and updates itself. The currently used learning systems support this dynamic mode, however, in an online service this mode of interaction requires some implementation of access rights, user limits, cloning of developments and services, etc. This is still future work, close to the recent work on formal mathematical editors and wikis [Kal07, ABMU11].

## 3.5. Experiments

The main testing set in all scenarios is constructed from the 14185 Flyspeck theorems. To be able to explore as many approaches as possible, a smaller subset of 1419 theorems is often used. This subset is stable for all such experiments, and it is constructed by taking every tenth theorem (starting with 0-th) in the chronological ordering.

A number of first- and higher-order ATPs and SMT solvers were tried on the problems. The most extensively used are Vampire 2.6 (called also V below), modified E 1.6, and Z3 4.0. Proofs are important in the ITP/learning scenarios, so Z3 and E are (unless otherwise noted) run in a proof-producing mode. In particular for Z3 this costs some performance. E 1.6 is not run in its standard auto-mode, but in a strategy-scheduling wrapper[28] used by the MaLARea system in the Mizar@Turing large-theory competition.

This wrapper (called either Epar or just E in the tables below) subsequently tries 14 strategies provided by the second author. These strategies were developed on the 1000 problems allowed for training large-theory AI/ATP systems before the Mizar@Turing competition [Urb14]. Some of these strategies have become available in E 1.6 when it was released after CASC 2012, but E's auto mode is still tuned for the TPTP library, and by default it always uses only one "best" (heuristically chosen) strategy on a problem, shunning so far the temptations of strategy scheduling. Epar outperforms the old auto-mode of E 1.4 by over 20% on the Mizar@Turing training problems, and seems to be

[27]The service runs at colo12-c703.uibk.ac.at on port 8080, example queries are:
`echo 'CARD {3,4} = 2' | nc colo12-c703.uibk.ac.at 8080` ,
`echo '(A DIFF B) INTER C = EMPTY <=> (A INTER C) SUBSET B' | nc colo12-c703.uibk.ac.at 8080` .

[28]`https://github.com/JUrban/MPTP2/blob/master/MaLARea/bin/runepar.pl`

competitive with Vampire 2.6. Fifteen more systems (or their versions) were tried to a lesser extent (typically on the 1419-problem subset) in the experiments. Some of these systems perform very well, and might be used more extensively later. Sometimes an additional effort is needed to make systems really useful; for example, proof/premise output might be missing, or additional mapping to a system's constructs needs to be done to take full advantage of the system's features. In this work such customizations are avoided. All the systems used are alphabetically listed in Table 3.4.

| System (short) | Format | Description |
|---|---|---|
| AltErgo | TFF1 | AltErgo version 0.94 |
| CVC3 | TFF1 | CVC3 version 2.2 |
| E 1.6 | FOF | E version 1.6pre011 Tiger Hill |
| Epar (E) | FOF | E version 1.6pre011 with the Mizar@Turing strategies |
| iProver | FOF | iProver version 0.99 (CASC-J6 2012) |
| Isabelle | THF | Isabelle 2012 used in CASC 2012 (no LEO2 or Satallax) |
| leanCoP | FOF | leanCoP version 2.2 (using eclipse prolog) |
| LEO2-po2 | THF | LEO2 v1.4.2 in the "po 2" proof mode[a] (OCaml-3.11.2) |
| LEO2-po1 | THF | LEO2 in the standard "po 1" mode (faster) |
| Metis | FOF | Metis version 2.3 (release 20101019) |
| Paradox | FOF | Paradox version 4.0, 2010-06-29. |
| Prover9 | FOF | Prover9 (32) version 2009-11A, November 2009. |
| SPASS | FOF | SPASS version 3.5 |
| Satallax | THF | Satallax version 2.6 |
| Vampire (V) | FOF | Vampire version 2.6 (revision 1649) |
| Yices | TFF1 | Yices version 1.0.34 |
| Z3 | FOF | Z3 version 4.0 |

Table 3.4.: Names and descriptions of the systems used in the evaluation

Unless specified otherwise, all systems are run with 30s time limit on a 48-core server with AMD Opteron 6174 2.2 GHz CPUs, 320 GB RAM, and 0.5 MB L2 cache per CPU. Each problem is always assigned one CPU. In the tables below, basic statistics are often computed about the population of the methods used: *Unique* solutions found by each method, and its *State of the art contribution (SOTAC)* as defined by CASC.[29]

[a]For the experiments that produce proof dependencies (useful, e.g., for learning), we have used the (so far experimental) version of LEO2 that fully reconstructs the original dependencies (the "po 2" option). For the rest of the experiments (where proofs are not needed), the standard version of LEO2 is used. This version is also proof-producing, but some additional work is still needed to extract the original proof dependencies. This work is currently being done by the LEO2 developers. The "po 1" version outperforms the "po 2" version.

[29]For each problem solved by a system, its SOTAC for the problem is the inverse of the number of systems

A system's CASC-defined SOTAC will be highest even if the system solved only one problem (which no other system solved). That is why also the Σ-SOTAC value is used: the sum of a system's SOTAC over all problems attempted. These metrics often indicate how productive it is to add a particular system or its version to a population of systems. Often it is interesting to know the best joint performance when running $N$ methods in parallel. Finding such a best combination is however an instance of the classical NP-hard Maximum Coverage problem. While it is often possible to use SAT solvers to get an optimal solution, a greedy algorithm is always consistently used to avoid problems when scaling to larger datasets. This also allows us to present this joint performance as a simple *greedy (covering) sequence*, i.e., a sequence that starts with the best system, and each next system in such sequence is the system that greedily adds most solutions to the union of solutions of the previous systems in the sequence.

### 3.5.1. Using External ATPs on ITP Dependencies

Table 3.5 shows the results of running Vampire, Epar, Z3, and Paradox on the 14185 FOF problems constructed from the HOL Light proof dependencies. The ATP success rate measured on such problems is useful as an upper estimate for the ATP success rate on the (potentially much larger) problems where all premises from the whole previous library are allowed to be used. This success rate can be used later to evaluate the performance of the algorithms that select a smaller number of the most relevant premises. The time limit for Vampire, Epar and Z3 was relatively high (900s), because particularly Vampire benefits from higher time limits (compare with Table 3.7) and the ATP proofs found by re-proving turn out to be more useful for training premise selectors than the original HOL Light dependencies (see Section 3.5.2). Paradox was run for 30 seconds to get some measure of the incompleteness of the FOF translation. The systems in Table 3.5 are already ordered using the *greedy covering sequence*, i.e., the joint performance of the top two systems is 41.9%, etc. The counter-satisfiability detected by Paradox is not by default included in the *greedy sequence*, since its goal is to find the strongest combination of proof-finding systems. The Paradox results are however included in the SOTAC and Unique columns.

Table 3.6 shows these results restricted to the 1419-problem subset. This provides some measure of the statistical error encountered when testing systems on the smaller problem set, and also a comparison of the systems' performance under high and low time limits used in Table 3.7.

Table 3.7 shows all tested systems on the 1419-problem subset, ordered by their absolute performance, and Table 3.8 shows the corresponding greedy ordering. The tested systems include also SMT solvers that use the TFF1 encoding and higher-order provers using the THF encoding. This is why it is no longer possible to aggregate the counter-

that solved the problem. A system's overall SOTAC is the average SOTAC over the problems it solves.

| Prover | Theorem (%) | Unique | SOTAC | Σ-SOTAC | CounterSat (%) | Greedy (%) |
|---|---|---|---|---|---|---|
| Vampire | 5641 (39.7) | 218 | 0.403 | 2273.58 | 0 ( 0.0) | 5641 (39.7) |
| Epar | 5595 (39.4) | 194 | 0.400 | 2238.58 | 0 ( 0.0) | 5949 (41.9) |
| Z3 | 4375 (30.8) | 193 | 0.372 | 1629.08 | 2 ( 0.0) | 6142 (43.2) |
| Paradox | 5 ( 0.0) | 0 | 0.998 | 2612.75 | 2614 (18.4) | 6142 (43.2) |
| any | 6142 (43.2) | | | | 2614 (18.4) | |

**Theorem (%):** Number and percentage of theorems proved by a system.

**Unique:** Number of theorems proved only by this system.

**SOTAC, Σ-SOTAC:** See the explanatory text for these metrics.

**CounterSat:** Number of problems found counter-satisfiable (unprovable) by this system.

**Greedy (%):** The joint coverage of all the previous systems and this one ordered in a greedy sequence (see the text).

Table 3.5.: Epar, V, Z3 re-proving with 900s and Paradox with 30s (14185 problems)

| Prover | Theorem (%) | Unique | SOTAC | Σ-SOTAC | CounterSat (%) | Greedy (%) |
|---|---|---|---|---|---|---|
| Vampire | 577 (40.6) | 19 | 0.403 | 232.25 | 0 ( 0.0) | 577 (40.6) |
| Epar | 572 (40.3) | 23 | 0.405 | 231.75 | 0 ( 0.0) | 608 (42.8) |
| Z3 | 436 (30.7) | 17 | 0.369 | 160.75 | 0 ( 0.0) | 625 (44.0) |
| Paradox | 1 ( 0.0) | 0 | 0.997 | 257.25 | 257 (18.1) | 625 (44.0) |
| any | 625 (44.0) | | | | 257 (18.1) | |

Table 3.6.: ATP re-proving with 900s time limit on 10% (1419) problems

satisfiability results (particularly found by Paradox) with the theoremhood results, and all the derived statistics are only computed using the Theorem column. While Vampire does well with high time limits in Table 3.5, it is outperformed by Z3 and E-based systems (Epar, E 1.6, LEO2-po1) when using only 30 seconds (which seem more appropriate for interactive tools than 300 or even 900 seconds). This suggests that the strategy scheduling in Vampire might benefit from further tuning on the Flyspeck data. Z3 is not run in the proof-producing mode in this experiment, which improves its performance considerably. It is not very surprising (but still evidence of solid integration work) that Isabelle performs best, as it already combines a number of other systems; see its CASC 2012 description[30] for details. An initial glimpse at Isabelle's unique solutions also shows that 75% of them are found by the recent Isabelle-specific additions (such as hard sorts) to SPASS [BPWW12] and its tighter integration with Isabelle. This is an evidence that pushing such domain knowledge inside ATPs (as done recently also with the MaLeCoP prototype [UVŠ11]) might be quite rewarding. The joint performance of all systems tested is 50.2% when Isabelle is included, and 47.4% when only the base systems are allowed. This is quite encouraging, and for example the counter-satisfiability results suggest that additional performance could be gained by further (possibly heuristic/learning) work on alternative translations. Pragmatically, the joint re-proving performance also tells us that when used in the MESON-tactic mode with premises explicitly provided by the users, a parallel 9-CPU machine running the nine systems from Table 3.8 will within 30 seconds (of real time) prove half of the Flyspeck theorems without any further interaction.

### 3.5.2. Using External ATPs with Premise Selection

As described in Section 3.4, there are a number of various approaches and parameters influencing the training of the premise selectors. These parameters were gradually (but not exhaustively) explored, typically on the 1419-problem subset. Several times the underlying training data changed quite significantly as a result of the data-improving passes described in Section 3.4.3. Some of these passes were evaluating the best prediction methods developed so far on all 14185 problems. All the experiments were limited to Vampire, Epar, and Z3. For most of the experiments (and unless otherwise noted) the first-stage premise selection is used to create problems with 8, 32, 128, and 512 premises. This slicing (i.e., taking the first $N$ premises) can be later fine-tuned, as done below in Section 3.5.2 for the best premise selection method.

Table 3.9 shows an initial evaluation of 16 different learning combinations trained on ATP proofs obtained in 300s, complemented by the HOL Light proof dependencies (the second and first pass in Table 3.3). The two exceptions are the `symst+symsonly` combination, which ignores all proofs, and the `syms+old+000001` combination, which

[30] `www.cs.miami.edu/~tptp/CASC/J6/SystemDescriptions.html#Isabelle---2012`

| Prover | Theorem (%) | Unique | SOTAC | Σ-SOTAC | CounterSat (%) |
|---|---|---|---|---|---|
| Isabelle | 587 (41.3) | 39 | 0.201 | 118.09 | 0 ( 0.0) |
| Epar | 545 (38.4) | 9 | 0.131 | 71.18 | 0 ( 0.0) |
| Z3 | 513 (36.1) | 17 | 0.149 | 76.49 | 0 ( 0.0) |
| E 1.6 | 463 (32.6) | 0 | 0.101 | 46.69 | 0 ( 0.0) |
| LEO2-po1 | 441 (31.0) | 1 | 0.106 | 46.85 | 0 ( 0.0) |
| Vampire | 434 (30.5) | 3 | 0.107 | 46.44 | 0 ( 0.0) |
| CVC3 | 411 (28.9) | 4 | 0.111 | 45.76 | 0 ( 0.0) |
| Satallax | 383 (26.9) | 7 | 0.130 | 49.69 | 1 ( 0.0) |
| Yices | 360 (25.3) | 0 | 0.097 | 35.06 | 0 ( 0.0) |
| iProver | 348 (24.5) | 0 | 0.088 | 30.50 | 9 ( 0.6) |
| Prover9 | 345 (24.3) | 0 | 0.087 | 30.07 | 0 ( 0.0) |
| Metis | 331 (23.3) | 0 | 0.085 | 28.23 | 0 ( 0.0) |
| SPASS | 326 (22.9) | 0 | 0.081 | 26.46 | 0 ( 0.0) |
| leanCoP | 305 (21.4) | 1 | 0.092 | 27.96 | 0 ( 0.0) |
| AltErgo | 281 (19.8) | 1 | 0.100 | 28.14 | 0 ( 0.0) |
| LEO2-po2 | 53 ( 3.7) | 0 | 0.082 | 4.34 | 0 ( 0.0) |
| Paradox | 1 ( 0.0) | 0 | 0.059 | 0.06 | 259 (18.2) |
| any | 712 (50.1) | | | | 259 (18.2) |

Table 3.7.: All ATP re-proving with 30s time limit on 1419 problems (10%)

| Prover | Isabelle | Z3 | Epar | Satallax | CVC3 | Vampire | LEO2-po1 | leanCoP | AltErgo |
|---|---|---|---|---|---|---|---|---|---|
| Sum % | 41.3 | 46.9 | 48.8 | 49.3 | 49.6 | 49.9 | 50.0 | 50.1 | 50.1 |
| Sum | 587 | 666 | 693 | 700 | 705 | 709 | 710 | 711 | 712 |

| Prover | Epar | Z3 | Satallax | Vampire | LEO2-po1 | CVC3 | AltErgo | Yices | leanCoP |
|---|---|---|---|---|---|---|---|---|---|
| Sum % | 38.4 | 42.7 | 45.3 | 46.0 | 46.6 | 47.1 | 47.2 | 47.3 | 47.4 |
| Sum | 545 | 607 | 644 | 654 | 662 | 669 | 671 | 672 | 673 |

Table 3.8.: Greedy sequence for Table 3.7 (with and without Isabelle)

uses older ATP-re-proving data obtained by running each ATP only for 30s (about 700 ATP proofs less). Each row in Table 3.9 is a union of twelve 30s ATP runs: Vampire, Epar, and Z3 used on the 8, 32, 128, and 512 slices. After this initial evaluation, the `symst` (types instead of variables) characterization was preferred, trivial symbols were always pruned out, and Winnow and Perceptron were left behind. It is of course possible that some of these methods are useful as a complement of better methods. Preferring Vampire proofs helps the learning a bit, for reasons that are not yet understood. To get the joint 39.5% performance, in general 192-fold (= $16 \times 12$) parallelization is needed. This number could be reduced, but first better training methods were considered.

| `Method` | Theorem (%) | Unique | SOTAC | Σ-SOTAC |
|---|---|---|---|---|
| `B+symst+v_pref+nominweight` | 418 (29.4) | 2 | 0.093 | 38.98 |
| `B+symst` | 410 (28.8) | 0 | 0.089 | 36.65 |
| `B+syms0` | 406 (28.6) | 0 | 0.084 | 33.99 |
| `B+symst+triv` | 405 (28.5) | 3 | 0.094 | 37.98 |
| `B+syms` | 402 (28.3) | 0 | 0.083 | 33.48 |
| `B+syms+triv+nominweight` | 397 (27.9) | 1 | 0.083 | 32.98 |
| `B+syms0+triv` | 397 (27.9) | 0 | 0.078 | 30.82 |
| `B+syms+triv+v_pref` | 396 (27.9) | 0 | 0.081 | 31.99 |
| `B+syms+triv` | 393 (27.6) | 0 | 0.077 | 30.13 |
| `B+syms+triv+z_pref` | 392 (27.6) | 1 | 0.082 | 32.09 |
| `B+syms+triv+e_pref` | 392 (27.6) | 0 | 0.078 | 30.45 |
| `B+symsd` | 382 (26.9) | 3 | 0.097 | 37.08 |
| `B+syms+old+000001` | 376 (26.4) | 14 | 0.129 | 48.34 |
| `B+symst+symsonly` | 302 (21.2) | 17 | 0.141 | 42.55 |
| `W+symst` | 251 (17.6) | 9 | 0.141 | 35.37 |
| `P+symst` | 195 (13.7) | 6 | 0.144 | 28.13 |
| `any` | 561 (39.5) | | | |

**`B,W,P:`** Naive Bayes, Winnow, Perceptron.

**`triv:`** Logical (trivial) symbols like conjunction are included.

**`old+000001:`** Using older 30-second ATP data, and a minimal weight of 0.000001 for irrelevant HOL dependencies (the default weight was 0.001).

Table 3.9.: 16 premise selection methods trained on ATP proofs complemented with HOL proofs evaluated on 1419 selected problems

### Further Premise Selection Improvements

Complementing the ATP dependency data with the (possibly discounted) HOL Light dependencies seems to be a plausible method. Even if the HOL Light dependencies are very redundant, the redundancies should be weighted down by the information learned from the large number of ATP proofs, and the remaining HOL Light dependencies should be in general more useful than no information at all. A possible explanation of why this approach might still be quite suboptimal (in the ATP setting) is that the HOL Light proofs are often not a good guidance for the ATP proofs, and may push the machine learners in a direction that is ATP-infeasible. A small hint that this might be the case is the good performance of the `nominweight` method in Table 3.9. This method completely ignores all HOL Light dependencies that were never used in previous ATP proofs. This suggested to test the more radical `atponly` approach, in which only the ATP proofs are used for training. This approach improved the best method from 29.4% to 31.9%, and added 25 newly solved problems (1.8%) to those solved by the 16 methods in Table 3.9. These results motivated further work on getting as many (and as minimal) ATP proofs as possible, producing the methods tagged as `m10, m10u` and `m10u2` in the tables below. These methods were trained on the proofs obtained by 10-minute (hence `m10`) ATP runs that were further upgraded by the advised proofs as described in Section 3.4.3. The best `m10u2` method raised the performance by further 0.5%, and the learning on the advised proofs in different ways made these methods again quite orthogonal to the previous ones.

Even though Winnow and Perceptron performed poorly (as expected from earlier unpublished experiments with MML), they added some new solutions. This motivated one simple additional experiment with the classic $k$-nearest neighbor ($k$-NN) learner, which computes for a new example (conjecture) the $k$ nearest (in a given feature distance) previous examples and ranks premises by their frequency in these examples. This is a fast ("lazy" and trivially incremental) learning method that can be easily parameterized and might for some parameters behave quite differently from naive Bayes. For large datasets a basic implementation gets slow in the evaluation phase, but on the Flyspeck dataset this was not yet a problem and full training/evaluation processing took about the same time as naive Bayes. Table 3.10 shows the performance of three differently parameterized $k$-NN instances, and Table 3.11 shows 8 different $k$-NN-based methods that together prove 29% of the problems. As expected, $k$-NN performs worse than naive Bayes, but much better than Winnow and Perceptron. The 160-NN and 40-NN methods indeed produce somewhat different solutions, and they are sufficiently orthogonal to the previous methods and both contribute to the performance of the final best mix of 14 prediction/ATP methods.

| `Method` | Theorem (%) | Unique | SOTAC | Σ-SOTAC | Processed |
|---|---|---|---|---|---|
| `KNN160+m10u+atponly` | 391 (27.5) | 84 | 0.512 | 200.33 | 1419 |
| `KNN40+m10u+atponly` | 330 (23.2) | 10 | 0.403 | 132.83 | 1419 |
| `KNN10+m10u+atponly` | 244 (17.1) | 6 | 0.360 | 87.83 | 1419 |
| `any` | 421 (29.6) | | | | 1419 |

Table 3.10.: Three instances of $k$-nearest neighbor on the 1419-problem subset

| `Method` | `KNN160+512e` | `KNN40+32e` | `KNN160+512v` | `KNN40+512e` |
|---|---|---|---|---|
| Sum % | 21.7 | 25.5 | 26.7 | 27.4 |
| Sum | 309 | 363 | 379 | 390 |
| `...` | `KNN160+128z` | `KNN160+128e` | `KNN10+512v` | `KNN10+512e` |
| ... | 28.1 | 28.4 | 28.8 | 29.0 |
| ... | 399 | 404 | 409 | 412 |

Table 3.11.: Greedy sequence using $k$-NN-based premise selectors

### Performance of different premise slices

Fig. 3.1 shows how the ATP performance changes when using different numbers of the best-ranked premises. This is again evaluated in 30 seconds on the 1419-problem subset, i.e., Vampire's performance is likely to be better (compared to Epar) if a higher time limit was used. To a certain extent this graph serves also as a comparison of the first-stage premise selection (in this case naive Bayes trained on the minimized proofs) with the second-stage premise selection (various SInE strategies tried by the ATPs). Z3 has no second-stage premise selection, and after 250 premises the performance drops quite quickly (12.4% with 256 premises vs. 6.2% with 740 premises). For Vampire this drop is more moderate (18.1% vs. 12.9%). Epar stays over 20% with 512 premises, and drops only to 15.6% with 2048 premises. Thus, 512 premises seems to be the current "margin of error" for the first-stage premise selection that can be (at least to some extent) offset by using SInE at the second stage.

Table 3.12 shows for this premise selection method the joint performance (in greedy steps) of all premise slicings, when for each slice the union of all ATPs' solutions is taken. Only 17 slices are necessary (when using the greedy approach); the remaining 8 slices do not contribute more solutions. In general, this union would take $3 * 17 = 51$ ATP runs, however only 28 ATP runs are actually required to achieve the maximum 36.4% performance. These runs are not shown in full here, and instead only the first 14 runs that yield 35% are shown in Table 3.13. Assuming a 14-CPU server, 35% is thus the 30-second performance when using only one (the best) premise selection method.

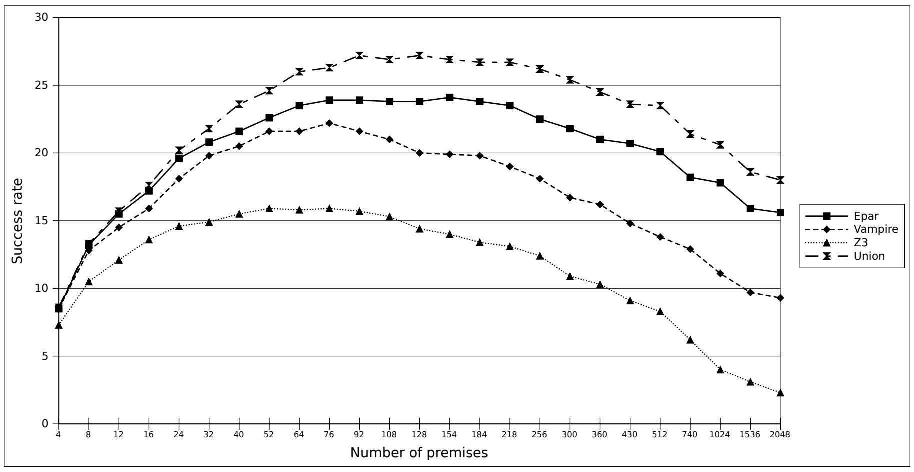


Figure 3.1.: Performance of different premise slices (in % of the 1419 problems)

| **Method** | **128** | **1024** | **40** | **256** | **64** | **24** | **740** | **430** | **...** |
|---|---|---|---|---|---|---|---|---|---|
| Sum % | 27.2 | 30.2 | 32.9 | 33.9 | 34.5 | 34.9 | 35.3 | 35.5 | ... |
| Sum | 386 | 429 | 467 | 482 | 490 | 496 | 501 | 504 | ... |
| **...** | **92** | **2048** | **184** | **218** | **154** | **52** | **32** | **12** | **4** |
| ... | 35.7 | 35.8 | 36.0 | 36.0 | 36.1 | 36.2 | 36.2 | 36.3 | 36.4 |
| ... | 507 | 509 | 511 | 512 | 513 | 514 | 515 | 516 | 517 |

Table 3.12.: Greedy covering sequence for `m10u2` slices (joining ATPs)

| **Method** | **154e** | **52v** | **1024e** | **16e** | **300v** | **92e** | **64z** |
|---|---|---|---|---|---|---|---|
| Sum % | 24.1 | 27.4 | 29.8 | 31.1 | 32.0 | 32.6 | 33.1 |
| Sum | 343 | 390 | 423 | 442 | 455 | 464 | 471 |
| **...** | **256z** | **256e** | **184v** | **32v** | **2048e** | **128v** | **24z** |
| ... | 33.5 | 33.8 | 34.1 | 34.3 | 34.6 | 34.8 | 35.0 |
| ... | 476 | 480 | 484 | 488 | 491 | 494 | 497 |

Table 3.13.: Greedy covering sequence for `m10u2` limited to 14 slicing/ATP methods

### The Final Combination and Higher Time Limits

It is clear that the whole learning/ATP AI system can (and will) be (self-)improved in various interesting ways and for long time.[31] When the number of small-scale evaluations reached several hundred and the main initial issues seemed corrected, an overall evaluation of the (greedily) best combination of 14 methods was done on the whole set of 14185 Flyspeck problems using a 300s time limit. These 14 methods together prove 39% of the theorems when given 30 seconds in parallel (see Table 3.14), which is also how they are run in the online service. The large scale evaluation is shown in Table 3.15 and Table 3.16. Table 3.15 sorts the methods by their 300s performance, and Table 3.16 computes the corresponding greedy covering sequence. Comparison with Table 3.14 shows that raising the CPU time to 300s helps the individual methods (2.7% for the best one), but not so much the final combination (only 0.3% improvement).

| `Prover` | Sum % | Sum |
|---|---|---|
| `B+symst+m10u2+atponly+154e` | 24.1 | 343 |
| `KN40+symst+m10u+atponly+32e` | 28.6 | 407 |
| `B+symst+m10u2+atponly+1024e` | 31.2 | 443 |
| `B+v_pref+triv+128e` | 33.1 | 471 |
| `B+symst+m10u2+atponly+92v` | 34.4 | 489 |
| `W+symst+128e` | 35.3 | 501 |
| `B+symst+m10u+atponly+32z` | 35.9 | 510 |
| `B+syms0+triv+512e` | 36.5 | 519 |
| `B+all000001+old+triv+128v` | 37.2 | 528 |
| `KN160+symst+m10u+atponly+512z` | 37.6 | 534 |
| `W+symst+32z` | 37.9 | 539 |
| `B+symst+m10u2+atponly+128e` | 38.3 | 544 |
| `B+symst+m10+vs+pref+atponly+128z` | 38.6 | 549 |
| `B+symst+32z` | 39.0 | 554 |

Table 3.14.: The top 14 methods in the greedy sequence for 30s small-scale runs

### 3.5.3. Union of Everything

Tables 3.17 shows the “union of everything”, i.e., the union of problems (limited to the 1419-problem subset) that could either be proved by an ATP from the HOL Light dependencies or by the premise selection methods. Together with Table 3.7 and Table 3.8 this also shows how much the ATP proofs obtained by premise selection methods complement

[31] In 2008, new proofs were still discovered after a month of running MaLARea on the whole MML. Analyzing the proofs and improving such AI over an interesting corpus gets addictive.

| Prover | Theorem (%) | Unique | SOTAC | Σ-SOTAC |
|---|---|---|---|---|
| B+symst+m10u2+atponly+154e | 3810 (26.8) | 33 | 0.157 | 597.89 |
| B+symst+m10u2+atponly+128e | 3799 (26.7) | 25 | 0.153 | 580.70 |
| B+symst+m10u2+atponly+92v | 3740 (26.3) | 95 | 0.167 | 623.03 |
| B+symst+m10u2+atponly+1024e | 3280 (23.1) | 206 | 0.198 | 649.52 |
| B+syms0+triv+512e | 3239 (22.8) | 101 | 0.170 | 551.90 |
| B+v_pref+triv+128e | 2814 (19.8) | 36 | 0.143 | 401.27 |
| B+all000001+old+triv+128v | 2475 (17.4) | 50 | 0.149 | 367.86 |
| KN40+symst+m10u+atponly+32e | 2417 (17.0) | 78 | 0.160 | 386.90 |
| B+symst+m10+v_pref+atponly+128z | 2257 (15.9) | 33 | 0.138 | 311.74 |
| B+symst+m10u+atponly+32z | 2191 (15.4) | 43 | 0.139 | 304.77 |
| KN160+symst+m10u+atponly+512z | 1872 (13.1) | 37 | 0.145 | 270.58 |
| W+symst+128e | 1704 (12.0) | 56 | 0.164 | 279.34 |
| B+symst+32z | 1408 ( 9.9) | 16 | 0.118 | 166.09 |
| W+symst+32z | 711 ( 5.0) | 9 | 0.124 | 88.40 |
| any | 5580 (39.3) | | | |

Table 3.15.: The top 14 methods from Table 3.14 evaluated in 300s on all 14185 problems

| Prover | Sum % | Sum |
|---|---|---|
| B+symst+m10u2+atponly+154e | 26.8 | 3810 |
| B+symst+m10u2+atponly+1024e | 30.1 | 4273 |
| B+symst+m10u2+atponly+92v | 33.0 | 4686 |
| KN40+symst+m10u+atponly+32e | 34.8 | 4938 |
| B+syms0+triv+512e | 36.2 | 5148 |
| B+all000001+old+triv+128v | 36.9 | 5247 |
| B+symst+m10u+atponly+32z | 37.5 | 5332 |
| W+symst+128e | 38.1 | 5411 |
| KN160+symst+m10u+atponly+512z | 38.4 | 5454 |
| B+v_pref+triv+128e | 38.7 | 5492 |
| B+symst+m10+v_pref+atponly+128z | 38.9 | 5528 |
| B+symst+m10u2+atponly+128e | 39.1 | 5553 |
| B+symst+32z | 39.2 | 5571 |
| W+symst+32z | 39.3 | 5580 |

Table 3.16.: The greedy sequence for Table 3.15 (300s runs on all problems)

the ATP proofs based only on the HOL Light dependencies. The methods' running times are not comparable: the re-proving used 30s for each system, while the data for advised methods are aggregated across E, Vampire and Z3, and across the four premise slicing methods. This means that they in general run in $12 \times 30$ seconds (although typically only one or two slices are needed for the final joint performance). The number of Flyspeck theorems that were proved by any of the many experiments conducted is thus 56.5% when Isabelle is considered, and 54.7% otherwise.

| Prover | Sum % | Sum |
|---|---|---|
| Epar | 38.4 | 545 |
| B+syms+triv | 44.7 | 635 |
| Z3 | 48.2 | 684 |
| Satallax | 50.7 | 720 |
| B+v_pref+atponly | 52.1 | 740 |
| LEO2-po1 | 52.6 | 747 |
| CVC3 | 53.0 | 753 |
| B+symsonly+triv | 53.4 | 758 |
| B+symst+m10+nominwght | 53.6 | 762 |
| KN40+symst+m10u+atponly | 53.9 | 765 |
| B+syms0 | 54.1 | 768 |
| W+symst | 54.2 | 770 |
| B+symst+m10u2+atponly | 54.4 | 772 |
| Vampire | 54.4 | 773 |
| leanCoP | 54.5 | 774 |
| B+symst | 54.6 | 775 |
| B+symst+triv | 54.6 | 776 |
| AltErgo | 54.7 | 777 |

| Prover | Sum % | Sum |
|---|---|---|
| Isabelle | 41.3 | 587 |
| B+symst+m10u2+atponly | 48.4 | 687 |
| Z3 | 52.0 | 738 |
| Epar | 53.0 | 753 |
| B+syms | 54.0 | 767 |
| B+symsonly+triv | 54.5 | 774 |
| Satallax | 54.9 | 780 |
| B+symst+m10+nominwght | 55.3 | 785 |
| CVC3 | 55.6 | 790 |
| W+symst | 55.8 | 793 |
| KN40+symst+m10u+atponly | 56.0 | 796 |
| B+v_pref+atponly | 56.2 | 798 |
| LEO2-po1 | 56.3 | 799 |
| leanCoP | 56.3 | 800 |
| B+symst | 56.4 | 801 |
| B+symst+triv | 56.5 | 802 |

Table 3.17.: Covering sequence (without and with Isabelle) for all methods used

## 3.6. Initial Comparison of the Advised and Original Proofs

There are 6162 theorems that can be proved by either Vampire, Z3 or E from the original HOL dependencies. Their collection is denoted as *Original*. There are 5580 theorems (denoted as *Advised*) that can be proved by these ATPs from the premises advised automatically. It is interesting to see how these two sets of ATP proofs compare. In this section, a basic comparison in terms of the number of premises used for the ATP proofs is provided. A more involved comparison and research of the proofs using the proof-complexity metrics developed for MML in [AKU12] is left as an interesting future work. The intersection of *Original* and *Advised* contains 4694 theorems. Both sets of proofs are already minimized as described in Section 3.4.3. The proof dependencies were extracted[32] and the number of dependencies was compared. The complete results of this comparison are available online,[33] sorted by the difference between the length of the *Original* proof and the *Advised* proof. To make it easier to explore the differences described in the next subsections, the Flyspeck and HOL Light Subversion repositories were merged into one (git) repository, and (quite imperfectly) HTML-ized[34] by a simple heuristic Perl script. A simple CGI script[35] can be used to compare the dependencies needed for the (minimized) advised ATP proof with the dependencies needed for the ATP proof from the original HOL Light premises, and also with the actual HOL Light proof.

### 3.6.1. Theorems Proved Only with Advice

The list of 885 theorems proved only with advice is available online[36] sorted by the number of necessary premises. The last theorem in this order (`CROSS_BASIS_NONZERO`)[37] used 34 premises for the advised ATP proof, while its HOL Light proof is just a single invocation of the `VEC3_TAC` tactic[38] (which however brings in 121 HOL Light dependencies, making re-proving difficult). The following two short examples show how the advice can sometimes get simpler proofs.

1. Theorem `FACE_OF_POLYHEDRON_POLYHEDRON` states that a face of a polyhedron (defined in HOL Light generally as a finite intersection of half-spaces) is again a polyhedron:

```
∀s:real^N→bool c. polyhedron s ∧ c face_of s ⟹ polyhedron c
```

The HOL Light proof[39] takes 23 lines and could not be re-played by ATPs, but a much simpler proof was found by the AI/ATP automation, based on (a part of) the `FACE_-OF_STILLCONVEX` theorem: a face $t$ of any convex set $s$ is equal to the intersection of $s$ with the affine hull of $t$. To finish the proof, one needs just three “obvious” facts: Every polyhedron is convex (`POLYHEDRON_IMP_CONVEX`), the intersection of two polyhedra is again a polyhedron (`POLYHEDRON_INTER`), and affine hull is always a polyhedron (`POLYHEDRON_-AFFINE_HULL`):

[32]See `http://mws.cs.ru.nl/~mptp/hh1/ATPdeps/deps_of_atp_proofs_from_hol_deps.txt` and `http://mws.cs.ru.nl/~mptp/hh1/ATPdeps/deps_from_advised_atp_proofs.txt`.
[33]`http://mws.cs.ru.nl/~mptp/hh1/ATPdeps/deps_comparison.txt`
[34]`http://mws.cs.ru.nl/~mptp/hol-flyspeck/index.html`
[35]e.g., `http://mws.cs.ru.nl/~mptp/cgi-bin/browseproofs.cgi?refs=COMPLEX_MUL_CNJ`
[36]`http://mws.cs.ru.nl/~mptp/hh1/ATPdeps/aonly_by_length.txt`
[37]`http://mws.cs.ru.nl/~mptp/cgi-bin/browseproofs.cgi?refs=CROSS_BASIS_NONZERO`
[38]An interesting future work is to integrate calls to such tactics into the learning/ATP framework, or even to learn their construction (from similar sequences of lemmas used on similar inputs). The former task is similar to optimizing SMT solvers and tools like MetiTarski.
[39]`http://mws.cs.ru.nl/~mptp/cgi-bin/browseproofs.cgi?refs=FACE_OF_POLYHEDRON_POLYHEDRON`

```
FACE_OF_STILLCONVEX:
∀s t:real^N→bool. convex s ⟹
(t face_of s ⟺ t SUBSET s ∧ convex(s DIFF t) ∧ t = (affine hull t) INTER s)
POLYHEDRON_IMP_CONVEX: ∀s:real^N→bool. polyhedron s ⟹ convex s
POLYHEDRON_INTER:
∀s t:real^N→bool. polyhedron s ∧ polyhedron t ⟹ polyhedron (s INTER t)
POLYHEDRON_AFFINE_HULL: ∀s. polyhedron(affine hull s)
```

2. Theorem FACE_OF_AFFINE_TRIVIAL states that faces of affine sets are trivial:

```
∀s f:real^N→bool. affine s ∧ f face_of s ⟹ f = ∅ ∨ f = s
```

The HOL Light proof[40] takes 19 lines and could not be re-played by ATPs. The advised proof finds a simple path via previous theorem FACE_OF_DISJOINT_RELATIVE_INTERIOR saying that nontrivial faces are disjoint with the relative interior, and theorem RELATIVE_-INTERIOR_UNIV saying that any affine hull is equal to its relative interior. The rest is again just use of several "basic facts" about the topic (skipped here):

```
FACE_OF_DISJOINT_RELATIVE_INTERIOR:
  ∀f s:real^N→bool. f face_of s ∧ ¬(f = s) ⟹ f INTER relative_interior s = ∅

RELATIVE_INTERIOR_UNIV:
  ∀s. relative_interior(affine hull s) = affine hull s
```

### 3.6.2. Examples of Different Proofs

Finally, several examples are shown where the advised ATP proof differs from the ATP proof reconstructed from the original HOL Light dependencies.

1. Theorems COMPLEX_MUL_CNJ[41] and COMPLEX_NORM_POW_2 stating the equality of squared norm to multiplication with a complex conjugate follow easily from each other (together with the commutativity of complex multiplication COMPLEX_MUL_SYM). The proof of COMPLEX_MUL_CNJ in HOL Light (below) re-uses the longer proof of COMPLEX_NORM_-POW_2. The advised ATP proof directly uses COMPLEX_NORM_POW_2, but (likely because COMPLEX_MUL_SYM was never used before) first unfolds the definition of complex conjugate and then applies commutativity of real multiplication.

[40] http://mws.cs.ru.nl/~mptp/cgi-bin/browseproofs.cgi?refs=FACE_OF_AFFINE_TRIVIAL
[41] http://mws.cs.ru.nl/~mptp/cgi-bin/browseproofs.cgi?refs=COMPLEX_MUL_CNJ

```
let COMPLEX_MUL_CNJ = prove
 (‘∀z. cnj z * z = Cx(norm(z)) pow 2 ∧ z * cnj z = Cx(norm(z)) pow 2‘,
  GEN_TAC THEN REWRITE_TAC[COMPLEX_MUL_SYM] THEN
  REWRITE_TAC[cnj; complex_mul; RE; IM; GSYM CX_POW; COMPLEX_SQNORM]
  THEN REWRITE_TAC[CX_DEF] THEN AP_TERM_TAC THEN BINOP_TAC THEN
  CONV_TAC REAL_RING);;

COMPLEX_NORM_POW_2: ∀z. Cx(norm z) pow 2 = z * cnj z
COMPLEX_MUL_SYM: ∀x y. x * y = y * x
```

2. Theorem `disjoint_line_interval`[42] states that the left endpoints of two unit-long integer-ended intervals on the real line have to coincide if the intervals share a point strictly inside them. This suggests case analysis, which is what the longer HOL Light proof (omitted here) seems to do. The advisor instead gets the proof in a single stroke by noticing a previous theorem saying that the left endpoint is the $floor$ function which is constant for the points inside such intervals:

```
disjoint_line_interval:
∀(x:real) (y:real). integer x ∧ integer y ∧
  (∃ (z:real). x < z ∧ z < x + &1 ∧ y < z ∧ z < y + &1) ⟹ x = y

FLOOR_UNIQUE: ∀x a. integer(a) ∧ a ≤ x ∧ x < a + &1 ⟺ (floor x = a)
```

3. Theorem `NEGLIGIBLE_CONVEX_HULL_3`[43] states that the convex hull of three points in $\mathbb{R}^3$ is a negligible set. In HOL Light this is proved from the general theorem `NEGLIGIBLE_-CONVEX_HULL` stating this property for any finite set of points in $\mathbb{R}^n$ with cardinality less or equal to $n$. Instead of justifying this precondition, a shorter proof is found by the advised ATP that saw an analogous theorem about the affine hull, the inclusion of the convex hull in the affine hull, and the preservation of negligibility under inclusion.

[42] http://mws.cs.ru.nl/~mptp/cgi-bin/browseproofs.cgi?refs=Vol1.disjoint_line_interval
[43] http://mws.cs.ru.nl/~mptp/cgi-bin/browseproofs.cgi?refs=NEGLIGIBLE_CONVEX_HULL_3

```
let NEGLIGIBLE_CONVEX_HULL_3 = prove
 (`∀a b c:real^3. negligible (convex hull a,b,c)`,
  REPEAT GEN_TAC THEN MATCH_MP_TAC NEGLIGIBLE_CONVEX_HULL THEN
  SIMP_TAC[FINITE_INSERT; CARD_CLAUSES; FINITE_EMPTY; DIMINDEX_3] THEN
  ARITH_TAC);;

NEGLIGIBLE_CONVEX_HULL:
  ∀s:real^N→bool. FINITE s ∧ CARD(s) ≤ dimindex(:N)
     ⟹ negligible(convex hull s)

NEGLIGIBLE_AFFINE_HULL_3:
  ∀a b c:real^3. negligible (affine hull a,b,c)
CONVEX_HULL_SUBSET_AFFINE_HULL:
  ∀s. (convex hull s) SUBSET (affine hull s)
NEGLIGIBLE_SUBSET:
  ∀s:real^N→bool t:real^N→bool. negligible s ∧ t SUBSET s
    ⟹ negligible t
```

4. Theorem BARV_CIRCUMCENTER_EXISTS[44] says that under certain assumptions, a particular point (circumcenter) lies in a particular set (affine hull). The HOL Light proof unfolds some of the assumptions and takes 14 lines. The advisor just found a related theorem MHFTTZN3 which under the same assumptions states that the singleton containing the circumcenter is equal to the intersection of the affine hull with another set. The rest are two "obvious" facts about elements of intersections (IN_INTER) and elements of singletons (IN_SING):

```
BARV_CIRCUMCENTER_EXISTS: ∀V ul k. packing V ∧ barV V k ul ⟹
  circumcenter (set_of_list ul) IN affine hull (set_of_list ul)

MHFTTZN3:
  ∀V ul k. packing V ∧ barV V k ul ⟹
  ((affine hull (voronoi_list V ul)) INTER (affine hull (set_of_list ul))
    =  circumcenter (set_of_list ul)  )

IN_SING: ∀x y. x IN y:A ⟺ (x = y)
IN_INTER: ∀s t (x:A). x IN (s INTER t) ⟺ x IN s ∧ x IN t
```

5. An example of the reverse phenomenon (i.e., the advised proof is more complicated than the original) is theorem BOUNDED_CLOSURE_EQ[45] saying that a set in $R^n$ is bounded iff its closure is bounded. The harder direction of the equivalence was already available

[44] http://mws.cs.ru.nl/~mptp/cgi-bin/browseproofs.cgi?refs=Rogers.BARV_CIRCUMCENTER_EXISTS
[45] http://mws.cs.ru.nl/~mptp/cgi-bin/browseproofs.cgi?refs=BOUNDED_CLOSURE_EQ

as theorem `BOUNDED_CLOSURE`, and was used both by the HOL Light and the advised proof. The easier direction was in HOL Light proved by theorems `CLOSURE_SUBSET` and `BOUNDED_SUBSET` saying that any set is a subset of its closure and any subset of a bounded set is bounded. The advised proof instead went through a longer path based on theorems `CLOSURE_APPROACHABLE`, `IN_BALL` and `CENTRE_IN_BALL` to show that every element in a set is also in its closure, and then unfolded the definition of `bounded` and showed that the bound on the norms of closure elements can be used also for the original set.

```
let BOUNDED_CLOSURE_EQ = prove
 (`∀s:real^N→bool. bounded(closure s) ⟺ bounded s`,
  GEN_TAC THEN EQ_TAC THEN REWRITE_TAC[BOUNDED_CLOSURE] THEN
  MESON_TAC[BOUNDED_SUBSET; CLOSURE_SUBSET]);;

BOUNDED_CLOSURE: ∀s:real^N→bool. bounded s ⟹ bounded(closure s)
BOUNDED_SUBSET: ∀s t. bounded t ∧ s SUBSET t ⟹ bounded s
CLOSURE_SUBSET: ∀s. s SUBSET (closure s)

CLOSURE_APPROACHABLE:
  ∀x s. x IN closure(s) ⟺ ∀e. &0 < e ⟹ ∃y. y IN s ∧ dist(y,x) < e
IN_BALL: ∀x y e. y IN ball(x,e) ⟺ dist(x,y) < e
CENTRE_IN_BALL: ∀x e. x IN ball(x,e) ⟺ &0 < e
bounded: bounded s ⟺ ∃a. ∀x:real^N. x IN s ⟹ norm(x) ≤ a
```

### 3.6.3. Remarks

The average number of HOL Light proof dependencies restricted to the set of theorems reproved by ATPs is 34.54, i.e., there are on average about nine times more dependencies in a HOL Light proof than in the corresponding ATP proof (see Table 3.3). This perhaps casts some light on how learning-assisted ATP currently achieves its performance. A large human-constructed library like Flyspeck is often dense/redundant enough[46] to allow short proofs under the assumption of perfect (and thus inhuman) premise selection. Such short proofs can be found even by the quite exhaustive methods employed by most of the existing ATPs. The smarter the premise selection and the stronger the search inside the ATPs, the greater the chance that such proofs will end up inside the ATP's time-limited search envelope. The outcome of using such advisors extensively could be "better-informed" mathematics that has shorter proofs which use a variety of lemmas much more than the basic definitions and theorems. Whether such mathematics is easier for human consumption is not clear. Already now mathematical texts sometimes optimize proof

[46] As long as such libraries are human-constructed, they will remain imperfectly organized and redundant. No "software engineering" or other approach can prevent new shortcuts to be found in mathematics, unless an exhaustive (and infeasible) proof minimization is applied.

length by lemma re-use to an extent that may make the underlying ideas less visible. Perhaps this is just another case where the strong automation tools will eventually help to understand how human cognition works.

The ATP search is quite unlike the much less exhaustive search done by decision procedures, and also unlike the human proofs, where the global economy of dependencies is not so crucial once a fuzzy high-level path to the goal gets some credibility. Both the human and the decision-procedure proofs result in more redundant ("sloppier") proofs, which can however be more involved (complicated) than what the ATPs can achieve even with optimal premise selection. Learning such (precise or fuzzy) high-level pathfinding is an interesting next challenge for large-theory AI/ATP systems. With the number of proofs and theory developments to learn from available now in the HOL/Flyspeck, Mizar/MML, and Isabelle corpora, and the already relatively strong performance of the "basic" AI/ATP methods that are presented in this paper, these next steps seem to be worth a try.

## 3.7. Related Work and Contributions

Related work has been mentioned throughout the paper, and some of the papers cited provide recent overviews of various aspects of our work. In particular, Blanchette's PhD thesis and [BBPS13] give a detailed overview of the translation methods for the (extended) HOL logic used in Isabelle. See [Urb11a, KvLT$^+$12] for recent overviews of large-theory ATP methods, and [UV13] for a summary of the work done over MML and its AI aspects.

Automated theorem proving over large theories goes back at least to Quaife's large developments [Qua92] with Otter [MW97][47] (continued to some extent by Belinfante [Bel99]). Most of the ATP/ITP combinations developed in 1990s used ATPs on user-restricted search space. Examples include the ATPs for HOL (Light) by Harrison and Hurd mentioned above, similar work for Isabelle by Paulson [Pau99], integration of CL$^A$M with HOL [SGBB98] and integration of ATPs with the Omega proof assistant [Mei00]. Dahn, Wernhard and Byliński exported Mizar/MML into the ILF format [DW97], created (small) ATP problems from several Mizar articles, and researched ATP-friendly encodings of Mizar's dependent and order-sorted type system [Dah98]. Large-theory ATP reappeared in 2002 with Voronkov's and Riazanov's customized Vampire answering queries over the whole SUMO ontology [PS07b], and Urban's MoMM (modified E) authoring tool [Urb06b] using all MML lemmas for dependently-typed subsumption of new Mizar goals. Since 2003, experiments with (unmodified) ATPs over large libraries have been carried out for MML [Urb04] (using machine learning for premise selection) and for Isabelle/HOL (using the symbol-based Sledgehammer heuristic for premise se-

[47] An interesting case is McAllester's Ontic [McA89]. The whole library is searched automatically, but the automation is fast and intentionally incomplete.

lection). A number of large-theory ATP methods and systems (e.g., SInE, MaLARea, goal-oriented heuristics inside ATPs) have been developed recently and evaluated over large-theory benchmarks and competitions like CASC LTB and Mizar@Turing. A comprehensive comparison of ATP and Mizar proofs was recently done in [AKU12]. As here, the average number of Mizar proof dependencies is higher than the number of ATP dependencies, however, the difference is not as striking as for HOL Light (a very different method is used to get the Mizar dependencies).

The work described here adds HOL Light and Flyspeck to the pool of systems and corpora accessible to large-theory AI/ATP methods and experiments. A number of large-theory techniques are re-used, sometimes the Mizar, Isabelle and CASC LTB approaches are combined and adapted to the HOL Light setting, and some of the techniques are taken further. The theorem naming, dependency export, problem creation, and advising required newly implemented HOL Light functions. The machine learning adds $k$-nearest neighbor, and the feature characterization was improved by replacing variables in terms with their HOL types. A MaLARea-like pass interleaving ATP with learning was used to obtain as many ATP proofs as possible, and the proofs were postprocessed by pseudo- and cross-minimization. Unlike in MaLARea, this was done in a scenario that emulates the growth of the library, i.e., no information about the proofs of later theorems was used to train premise selection for earlier theorems. Motivated by the recent experiments over the MPTP2078 benchmark, the machine learning was complemented by various SInE strategies used by E and Vampire. The strategy-scheduling version of E using the strategies developed for Mizar@Turing was tested for the first time in such large evaluation. A significant effort was spent to find the most orthogonal ingredients of the final mix of premise selectors and ATPs: in total 435 different combinations were tested. The resulting 39% chance of proving the next theorem without any user advice is a landmark for a library of this size. While a similar number was achieved in [KvLT$^+$12] on the much smaller MPTP2078 benchmark with a lower time limit, only 18% success rate was recently reported in [AKU12] for the whole MML in this fully push-button mode.[48] None of those evaluations however combined so many methods as here. The improvement over the best method (proving 24.1% theorems in 30s and 26.8% in 300s) shows that such combinations significantly improve the usability of large-theory ATP methods for the end users.

## 3.8. Future Work

Stronger machine learning (kernel/ensemble, etc.) methods and more suitable characterizations (e.g., addition of model-evaluation features and more abstract features) are likely to further improve the performance. The prototype online service could be made

[48]A similar large-scale evaluation for Isabelle would be interesting. It is not clear whether the current "Judgement Day" benchmark contains goals on the same (theorem) level of granularity.

customizable by learning from users' own proofs. So far only three ATPs are used by the service, but many other systems can eventually be added, possibly with various custom mappings to their logics. The translation methods can be further experimented with: either to get a symbol-consistent first-order translation (to allow, e.g., the model-evaluation features), or to get less incomplete translations. Proof reconstruction is currently work in progress. A simple and obvious approach is to try MESON with the minimized set of dependencies. When it is ready, unsound translations can be added to the pool of methods as was originally done in Isabelle Isabelle [MP08]. Training ATP-internal guidance on the corpus for prototype learning/ATP systems like MaLeCoP will be interesting, and perhaps also further tuning of ATP strategies for systems like E.

The power of the combined system probably already now makes it interesting as a complementary semantic aid/filter for first experiments with statistical translation methods between the informal Flyspeck text and the Flyspeck formalization. The cases of machine translation (as in Google Translate) and natural-language query answering (as in IBM Watson) have recently demonstrated the power of large-corpus-driven methods to automatically learn such translation/understanding layers from uncurated imperfect resources such as Wikipedia. In other words, large bodies of mathematics (and exact science) such as arXiv.org are unlikely to become computer-understandable by the current painstaking human encoding efforts and additions of further and further logic complexity layers that increase the formalization barrier both for humans and AI systems. Large-scale (world-knowledge-scale) formalization for (mathematical) masses is hard to imagine as one large "perfectly engineered" knowledge base in which everyone will know perfectly well where their knowledge fits. Such attempts seem to be as doomed as the initial attempts (in the Stone Age of Internet) to manually organize the World Wide Web in one concise directory. Gradual world-scale formalization seems more likely to happen through simpler logics that can be reasonably crowd-sourced (e.g., as Wikipedia was), assisted by AI (learning/ATP) methods continuously training and self-improving on cross-linked formal/semiformal/informal corpora expressed in simple formalisms that can be reasonably explained to such automated/AI methods.

## Acknowledgments

Tom Hales helped to start the MESON exporting work at CICM 2011, and his interest as a leader of Flyspeck has motivated us. Thanks to John Harrison for discussing MESON and related topics. Mark Adams gave us his HOL Light proof export data for HOL Zero, which made it easy to start the initial re-proving and learning experiments. Piotr Rudnicki has made his Mizar workstation available for a number of experiments (and his enthusiasm and support will be sorely missed). Andrei Paskevich has implemented the Why3 bridge to the TFF1 format just in time for our experiments. Finally, this work stands on the shoulders of many ATP and ITP (in particular HOL and Isabelle) developers, and tireless ATP competition organizers and standards producers. We are thankful to the JAR referees, PAAR 2012 referees and also to Jasmin Blanchette for their extensive comments on the early versions of this paper.

# 4 TSTP Proof Reconstruction

## Publication Details

Cezary Kaliszyk and Josef Urban. PRocH: Proof Reconstruction for HOL Light. In Maria Paola Bonacina, editor. Proc. of the 24th *International Conference on Automated Deduction (CADE'13)*, volume 7898 of LNCS, pp. 267–274. Springer, 2013. doi: `10.1007/978-3-642-38574-2_18`

## Abstract

PRocH is a proof reconstruction tool that imports in HOL Light proofs produced by ATPs on the recently developed translation of HOL Light and Flyspeck problems to ATP formats. PRocH combines several reconstruction methods in parallel, but the core improvement over previous methods is obtained by re-playing in the HOL logic the detailed inference steps recorded in the ATP (TPTP) proofs, using several internal HOL Light inference methods. These methods range from fast variable matching and more involved rewriting, to full first-order theorem proving using the MESON tactic. The system is described and its performance is evaluated here on a large set of Flyspeck problems.

## 4.1. Introduction, Motivation, and Related Work

Independent verification of proofs found by Automated Theorem Provers (ATPs) is not an uncommon topic in automated reasoning research. Systems like IVY [MM00] rely on very detailed proof output from ATPs (Otter, Prover9), which is then independently replayed and checked by a trusted system (ACL2). This technique has been used several times, e.g., for replaying the MESON and Otter/Prover9 proofs in HOL Light, replaying the Otter/Prover9 proofs in Mizar, and replaying the Metis proofs in HOL and Isabelle. The GDV [Sut06] tool is even parametric: any ATP system understanding TPTP can be used for independent verification. The Metis/Isabelle combination has been used also as a part of the Sledgehammer [PS07a] tool that uses arbitrary ATPs to discharge Isabelle proof obligations. If an ATP proof is found, and Metis can reconstruct the proof $\mathit{NeededLemmas} \vdash \mathit{Conjecture}$, then `metis(NeededLemmas)` is a valid Isabelle tactic, that is in practice used as an ATP proof importer. In HOL Light, MESON can be used in the same way for importing the proofs found by ATPs on FOL problems produced by the recently developed HOL(y)Hammer tool [KU14].

However, Metis and MESON are on average weaker than state-of-the art ATPs like Vampire and E. As ATPs and premise-selection tools get stronger, the ability of Metis and MESON to reconstruct (in short time usable in large ITP libraries) the ATP proof just from the proof premises decreases. Also, proofs in ITP systems like Isabelle, HOL Light and Mizar should (eventually) strive for human readability. Even if the strength of Metis and MESON grew, a single call to them would get hard to understand, requiring further explanation. These reasons motivate our work on a general tool that reconstructs TPTP proofs in HOL Light, using not just the proof premises, but also the steps recorded in the TPTP proof format.

## 4.2. Using Existing Approaches on HOL Light Problems

In total, the experiments with ATP-proving of HOL Light and Flyspeck theorems described in [KU14] have produced 7247 proofs when using Vampire, E (run under the Epar [Urb14] scheduler) and Z3, sometimes with high timelimits (900s). Only Vampire and E produce full TPTP proofs, but Z3 also prints the necessary premises (unsat core). These proofs are pseudo/cross-minimized, i.e., each proof was re-run by all ATPs using the proof premises only, while the number of proof premises was decreasing. Using the resulting sets of premises, Epar can find 6318 proofs in 30s. This set is used for further evaluations here. Table 4.1 shows the performance of the potential proof-importing tools, i.e., MESON, Metis, and Prover9 run with 300s time limit. Note that (unlike MESON) both Metis and Prover9 are just run externally, i.e., not reconstructing a valid HOL proof. As mentioned above, low proof times are important for working with large ITP libraries

| method | MESON | Metis | Prover9 |
|---|---|---|---|
| replayed | 5255 | 4595 | 4672 |
| replayed (%) | 83.1 | 72.7 | 73.9 |

Table 4.1.: MESON, Metis, and Prover9 with 300s on the 6318 Epar proofs

containing (tens of) thousands of theorems, each typically proved using several of the low-level (MESON, Metis, Mizar “by”, etc.) “atomic” calls. Table 4.2 therefore shows the performance of the above methods when using only 1 second for reconstruction.

Particularly the numbers obtained for Metis are considerably worse than the numbers obtained so far with Metis-based proof reconstruction in Sledgehammer [BN10], where only 10% of ATP proofs are lost by Metis. One possible reason is that Metis has been well-integrated with Sledgehammer, e.g., by using customized Sledgehammer-generated term orderings. Another part of explanation could be that the proofs found

| method | MESON | Metis | Prover9 |
|---|---|---|---|
| replayed | 5014 | 2803 | 4111 |
| replayed (%) | 79.3 | 44.3 | 65.0 |

Table 4.2.: MESON, Metis, and Prover9 with 1s on the 6318 Epar proofs.

by HOL(y)Hammer on Flyspeck are on average harder than the proofs found by Sledgehammer on the Judgement Day benchmark. The reasons can be that the Judgement Day benchmark consists of goals that are on average easier, the HOL(y)Hammer premise selection might be more precise (allowing more involved ATP proofs), and also ATP systems like Vampire and E have been strengthened since the time of the Judgement Day evaluation.

## 4.3. PRocH System Description

The HOL(y)Hammer tool runs in parallel several (now 14) AI/ATP combinations on a given HOL problem, and if a proof is found, it is pseudo/cross-minimized by further parallel running of the ATPs (and their strategies). PRocH then follows by trying in parallel several (old and new) proof reconstruction methods. Unlike the above methods that can only use the ATP proof premises to find their own detailed proof, the most complicated of the PRocH's methods (`hh_recon`) also tries to reconstruct in HOL Light the TPTP proofs created by the ATP systems. In this its closest relative is the `isar_proof` Sledgehammer function described in [PS07a], from which it probably differs by complete reliance on type annotations. In some sense, PRocH's `hh_recon` is so far less ambitious than `isar_proof`, because it does not yet attempt to write a HOL proof script. This also allows to treat some constructs (e.g., higher-order application) differently from `isar_proof` during the reconstruction. PRocH's use is now similar to HOL's MESON tactic, i.e., a call to `hh_recon[HOLPremises]` will try to justify a given HOL conjecture by going through the following stages (described more in the following subsections):

1. *Translation to FOL:* A HOL Light problem in the form

   $$HOLPremises \vdash HOLConjecture$$

   is translated to an untyped FOF TPTP problem, where part of the FOF encoding of terms are annotations encoding their HOL type.

2. *Running ATPs:* An external ATP is run on the first-order problem producing a TPTP proof.

3. *Parsing:* The untyped FOF and CNF formulas in the TPTP proof are parsed back into typed HOL terms (making use of the encoded type annotations). This part also has to handle skolemization.

4. *Replaying:* The justification structure of the TPTP proof is replayed on the parsed HOL Light terms, resulting in a valid HOL Light proof.

### 4.3.1. Translation to FOL and Producing FOL Proofs

The translation to FOL is described in [KU14], but we show a brief example here. The translation has to encode higher-order features like lambda abstraction, currying, quantification over function variables and their application. As a leading example, consider the following higher-order theorem `FORALL_ALL`[1]

$\forall$`P l.` ($\forall$`x. ALL (P x) l`) $\Longleftrightarrow$ `ALL (`$\lambda$`s.` $\forall$`x. P x s) l`

saying that each `x`-image of a binary relation (predicate) `P(x,y)` contains (is true for) all elements of a list `l` iff for all elements `s` of `l` the unary predicate "`P(x,s) is true for all x`" is true. To express this in FOL, first the lambda function is lifted from the context (its definition is created and used as an antecedent) and the higher-order applications are made explicit as follows:

$\forall$`l P F.` ($\forall$`s. happ F s` $\Longleftrightarrow$ ($\forall$`x. happ (happ P x) s`))
$\Longrightarrow$ (($\forall$`x. ALL (happ P x) l`) $\Longleftrightarrow$ `ALL F l`)

The formulas before and after the lambda-lifting and `happ`-introduction conversions are logically equivalent in the HOL logic,[2] and such conversions are used to change the initial HOL proof state into a form *ConvHOLPremises* $\vdash$ *ConvHOLConjecture* that will later correspond to the formulas reconstructed from the ATP proof. After these conversions, the implicit polymorphic HOL type domains (`A,B`) are explicitly introduced as variables and quantified over, and type annotations are added for all HOL terms using the `s` and `p` wrappers (and `happ` is shortened to `i`), resulting in the following FOF formula:

```
![A,B,L,P,F]:
 (![S]:(p(s(bool,i(s(fun(B,bool),F),s(B,S))))
   <=> ![X]:p(s(bool,i(s(fun(B,bool),i(s(fun(A,fun(B,bool)),P),s(A,X))),
                      s(B,S)))))
 =>
 (![X]:p(s(bool,all(s(fun(B,bool),i(s(fun(A,fun(B,bool)),P),s(A,X))),
                   s(list(B),L))))
   <=> p(s(bool,all(s(fun(B,bool),F),s(list(B),L))))))
```

[1]`http://mws.cs.ru.nl/~mptp/hol-flyspeck/trunk/lists.html#FORALL_ALL`

[2]For HOL speakers, e.g., the `happ` functor is just the HOL identity, i.e., we use:

```
happ_def = new_definition `(happ :  ((A -> B) -> A -> B)) = I`;;
happ_conv_th = prove (`!(f:A->B) x.  f x = happ f x`, ...  );;
happ_conv = REWR_CONV happ_conv_th;;
```

This way the HOL problem $ConvHOLPremises \vdash ConvHOLConjecture$ is translated to an untyped FOF TPTP problem $FOLPremises \vdash FOLConjecture$, on which ATPs like E and Vampire are run, producing derivations in the TPTP format [SSCG06]. Unlike the fixed and very detailed Otter/Prover9IVY format, the TPTP proof steps may be justified by arbitrary inference method, and thus may in theory be arbitrarily hard. In practice, for E and Vampire the (overwhelming number of) proof steps are detailed and easy to check with weak ATPs. Several interesting steps from E's proof of `FORALL_ALL` are as follows:

```
fof(2,axiom,p(s(bool,t)), file('f1', aTRUTH)).
fof(4,axiom, (~(p(s(bool,f)))<=>p(s(bool,t))), file('f1', aBOOL_CASES_AX)).
fof(5,conjecture, (![A,B,L,P,F]: ... ), file('f1', cFORALL_ALL)).
fof(6,negated_conjecture, ~(![A,B,L,P,F]: ... ),
    inference(assume_negation,[status(cth)],[5])).
...
fof(25,negated_conjecture,?[X10]:?[X11]:?[X12]:?[X13]:?[X14]: ...,
    inference(variable_rename,[status(thm)],[24])).
fof(26,negated_conjecture,![X15]:(~(p(s(bool,i(s(fun(esk3_0,bool),esk6_0)...,
    inference(skolemize,[status(esa)],[25])).
...
cnf(33,plain,(~p(s(bool,f))),inference(cn,[status(thm)],[32,theory(equality)])).
...
cnf(6393,negated_conjecture,(p(s(bool,f))),
    inference(spm,[status(thm)],[6320,5512,theory(equality)])).
cnf(6404,negated_conjecture,($false),
    inference(sr,[status(thm)],[6393,33,theory(equality)])).
cnf(6405,negated_conjecture,($false),6404,['proof']).
```

Such TPTP proofs produced by ATPs on the type-annotated input are the starting point for the HOL proof reconstruction. This is done in two stages: reconstruction of HOL terms/formulas, and reconstruction of the justification structure (HOL proof).

### 4.3.2. Reconstructing Terms and Formulas

The TPTP proof format is first parsed into suitable ML data structures using a lexer/parser combination created with `ocamllex/ocamlyacc`. For terms/formulas, parts of the HOL Light parsing mechanisms are re-used. In particular we gradually construct the intermediate HOL Light `preterm` structure, on whose final form the HOL Light `retypeckeck` function is called to obtain a HOL term. The preterm is constructed using variable/constant constructors (`Varp`), binary applications (`Combp`), abstractions (`Absp`), and type annotations (`Typing`).

Initially, the preterm just mirrors the FOL term structure, and in several passes the HOL structure is recovered from the type annotations. The recovery process might fail if the ATPs did proof-relevant operations that break the type annotation, however, at least with the resolution/paramodulation inferences done by E this practically does not happen. The first step is discovery of HOL type variables in formulas. For every type annotation `s(type, term)` all variables (and skolem constants) that appear in the left argument are considered to be type variables. In the next step, quantifications over such type variables are removed (they are implicitly universal in the HOL logic). During skolemization, type variables might have become arguments to newly introduced skolem functors. Such type arguments are removed, they are implicit in the HOL logic.

After that the `s` and `p` annotations are changed into `Typing` constructors with the appropriate types, and HOL Light's `retypeckeck` is called on the transformed preterm to obtain a HOL term.

### 4.3.3. Replaying ATP Proofs in HOL Light

As mentioned above, the problem $HOLPremises \vdash HOLConjecture$ is in HOL Light first converted (packaging the conversions in `HH_TAC`) to the equivalent problem $ConvHOLPremises \vdash ConvHOLConjecture$. This problem (proof state) is then further transformed using the HOL formulas reconstructed from the ATP proof, and using mechanisms implemented by the HOL Light subgoal package to handle the ATP proof steps. Given the topologically sorted list of proof steps, for every proof step a HOL tactic is applied, depending on the type of the step. Axioms are looked up among the HOL goal assumptions (using their name) and proved using these assumptions. The negated conjecture is introduced by transforming the goal using HOL's `REFUTE_TAC` ("R") (proof by contradiction). Skolemization steps are justified using HOL's `CHOOSE_TAC` ("C"), and for plain inference steps (SZS status THM) we gradually try three increasingly complex methods: matching (`MATCH_-ACCEPT_TAC` - "m"), rewriting (`REWRITE_TAC` - "r"), and HOL's full first-order ATP (`MESON_-TAC` - "1" or "2" depending on the number of premises). The final contradiction concludes the HOL proof using HOL's `ACCEPT_TAC` ("A"). For the reconstruction of the proof of `FORALL_ALL`, the sequence of this steps is as follows: `mR1rrC1111111mC1111122222221222A`.

## 4.4. Evaluation

Table 4.4 shows 1s evaluation of the reconstruction methods tried on the 6318 Epar proofs. The methods (tactics) are described in Table 4.3.[3] PRocH tries (after the HOL(y)Hammer conversion) three methods in parallel, i.e., its total CPU time can be 3s. This is not very significant for the comparison, see the 300s results of MESON and Prover9 in Table 4.1. The methods in Table 4.4 are ordered from top to bottom by a greedy covering sequence (the last column), where the next method always adds most to the previous methods. The unique number of solutions, SOTAC and $\Sigma$−SOTAC [KU14] are metrics that show the usefulness in the whole population.

[3]We tried more tactics, but they did not find more solutions. Higher times help very little, see: `http://cl-informatik.uibk.ac.at/users/cek/recon_stats.html`

| Method | Description |
|---|---|
| PRocH | HOL(y)Hammer conversion, parallel `HH_RECON`, MESON, and Prover9. |
| MESON | Standard `MESON_TAC` conversion then MESON and its replay. |
| `SIMP` | `SIMP_TAC`: Simplification by repeated conditional contextual rewriting. |
| Prover9 | Standard Prover9 conversion then Prover9 and its proof replay. |
| `REWRITE` | `REWRITE_TAC`: goal simplification by repeated unconditional rewriting. |
| `INT_ARITH` | Basic algebra and linear arithmetic over the integers. |
| `COMPLEX_FIELD` | Basic "field" facts over the complex numbers. |

Table 4.3.: Names and descriptions of the tactics tried for proof reconstruction.

| Prover | Theorem (%) | Unique | SOTAC | Σ−SOTAC | Greedy (%) |
|---|---|---|---|---|---|
| PRocH | 5687 (90.0) | 418 | 0.404 | 2298.50 | 5687 (90.0) |
| MESON | 5014 (79.3) | 118 | 0.367 | 1839.30 | 5862 (92.7) |
| `SIMP` | 2384 (37.7) | 54 | 0.290 | 692.30 | 5968 (94.4) |
| `INT_ARITH` | 407 ( 6.4) | 4 | 0.236 | 95.95 | 5972 (94.5) |
| `REWRITE` | 1540 (24.3) | 3 | 0.249 | 382.87 | 5975 (94.5) |
| `COMPLEX_FIELD` | 84 ( 1.3) | 2 | 0.270 | 22.68 | 5977 (94.6) |
| Prover9 | 2208 (34.9) | 1 | 0.293 | 646.40 | 5978 (94.6) |

Table 4.4.: Performance of reconstruction tactics run in 1s on 6318 Epar proofs.

The performance of the three submethods used by PRocH are shown in Table 4.5. They are again ordered by their greedy covering sequence. The HOL(y)Hammer preprocessing significantly improves the Prover9-based replay, but more important for the overall performance gain is the large number (406, i.e., 6.4% of 6318) of unique solutions contributed by `HH_RECON`. Finally, the performance of PRocH and MESON is compared in Figure 4.1 depending on the count of premises in the reconstructed proof. As the number of premises goes up (ATP proofs get more involved), PRocH becomes more and more necessary.

| Prover | Theorem (%) | Unique | SOTAC | Σ−SOTAC | Greedy (%) |
|---|---|---|---|---|---|
| HOL(y)Hammer + Prover9 | 4737 (74.9) | 253 | 0.412 | 1954.00 | 4737 (74.9) |
| HOL(y)Hammer + `HH_RECON` | 4299 (68.0) | 406 | 0.421 | 1811.50 | 5499 (87.0) |
| HOL(y)Hammer + MESON | 4737 (74.9) | 188 | 0.406 | 1921.50 | 5687 (90.0) |

Table 4.5.: Performance of the three submethods used by PRocH.

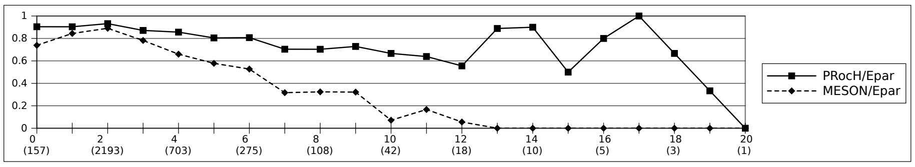


Figure 4.1.: PRocH and MESON dependence on premise nr. (Epar proof nr. in brackets).

## 4.5. Conclusion and Future Work

96.4% of the 6318 Epar proofs are reconstructed in 300s by some of the methods. To a great extent this validates the AI/ATP proof methods developed in [KU14], which were so far only verified by manual checking that the most striking (much shorter) AI/ATP proofs are really valid. 94.4% of the 6318 Epar proofs are reconstructed in 1s by one of the five (sub)methods run in parallel, i.e., the top three methods from Table 4.4, where PRocH consists of the three parallel submethods from Table 4.5. This makes the replay of more involved proofs fast and practical.

It would be good to postprocess the verbose ATP proofs into more compact, structured [Wie12], and human-readable proofs that would be stored directly as HOL Light code. Running proof-shortening tools in a loop is a simple method that already helps a lot, e.g., when importing Otter/Prover9 proofs into Mizar. Tools for lemma and concept introduction [Pą13, VSU10] can be experimented with, and with stronger AI/ATP assistance are becoming more and more important.

# 5 Certified Connection Tableaux Proofs

**Publication Details**

Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Certified Connection Tableaux Proofs for HOL Light and TPTP. In Xavier Leroy and Alwen Tiu, editors. *Proc. 4th Conference on Certified Programs and Proofs (CPP'15)*, pp. 59–66, ACM, 2015. doi: 10.1145/2676724.2693176.

**Abstract**

In the recent years, the Metis prover based on ordered paramodulation and model elimination has replaced the earlier built-in methods for general-purpose proof automation in HOL 4 and Isabelle/HOL. In the annual CASC competition, the leanCoP system based on connection tableaux has however performed better than Metis. In this paper we show how the leanCoP's core algorithm can be implemented inside HOL Light. leanCoP's flagship feature, namely its minimalistic core, results in a very simple proof system. This plays a crucial role in extending the MESON proof reconstruction mechanism to connection tableaux proofs, providing an implementation of leanCoP that certifies its proofs. We discuss the differences between our direct implementation using an explicit Prolog stack, to the continuation passing implementation of MESON present in HOL Light and compare their performance on all core HOL Light goals. The resulting prover can be also used as a general purpose TPTP prover. We compare its performance against the resolution based Metis on TPTP and other interesting datasets.

## 5.1. Introduction and Related Work

The leanCoP [OB03] automated theorem prover (ATP) has an unusually good ratio of performance to its implementation size. While its core algorithm fits on some twenty lines of Prolog, starting with CASC-21 [Sut08] it has regularly beaten Otter [MW97] and Metis [Hur03] in the FOF division of the CASC ATP competitions. In 2014, leanCoP solved 158 FOF problems in CASC-J7,[1] while Prover9 solved 95 problems. On the large-theory (chainy) division of the MPTP Challenge benchmark[2], leanCoP's goal-directed

[1]http://www.cs.miami.edu/~tptp/CASC/J7/WWWFiles/DivisionSummary1.html

[2]http://www.cs.miami.edu/~tptp/MPTPChallenge/

calculus beats also SPASS 2.2 [WAB+99], and its further AI-style strengthening by integrating into leanCoP's simple core learning-based guidance trained on such larger ITP corpora is an interesting possibility [UVŠ11].

Compact ATP calculi such as leanTAP [BP95] and MESON [Lov68] have been used for some time in Isabelle [Pau99, Pau97] and HOLs [Har96c] as general first-order automation tactics for discharging goals that are already simple enough. With the arrival of large-theory "hammer" linkups [PB12, KU15, KU13d, KBKU13] between ITPs, state-of-the-art ATPs such as Vampire [KV13] and E [Sch13], and premise selection methods [KvLT+12], such tactics also became used as a relatively cheap method for reconstructing the (minimized) proofs found by the stronger ATPs. In particular, Hurd's Metis has been adopted as the main proof reconstruction tool used by Isabelle's Sledgehammer linkup [PS07a, BN10], while Harrison's version of MESON could reconstruct in 1 second about 80% of the minimized proofs found by E in the first experiments with the HOL(y)Hammer linkup [KU13e].

Since HOL Light already contains a lot of the necessary infrastructure for Prolog-style proof search and its reconstruction, integrating leanCoP into HOL Light in a similar way as MESON should not be too costly, while it could lead to interesting strengthening of HOL Light's first-order automation and proof reconstruction methods. In this paper we describe how this was done, resulting in an OCaml implementation of leanCoP and a general leanCoP first-order tactic in HOL Light. We compare their performance with MESON, Metis and the Prolog version of leanCoP in several scenarios, showing quite significant improvements over MESON and Metis.

## 5.2. leanCoP and Its Calculus

leanCoP is an automated theorem prover for classical first-order logic based on a compact Prolog implementation of the clausal connection (tableaux) calculus [OB03, LS01] with several simple strategies that significantly reduce the search space on many problems. In contrast to saturation-based calculi used in most of the state-of-the-art ATPs (E, Vampire, etc.), connection calculi implement goal-oriented proof search. Their main inference step connects a literal on the current path to a new literal with the same predicate symbol but different polarity. The formal definition (derived from Otten [Ott10]) of the particular *connection calculus* relevant in leanCoP is as follows:

**Definition 5.2.1** *[Connection calculus]* The axiom and rules of the *connection calculus* are given in Figure 5.1. The words of the calculus are tuples "$C, M, Path$" where the clause $C$ is the *open subgoal*, $M$ is a set of clauses in disjunctive normal form (DNF) transformed from *axioms* $\wedge$ *conjecture* with added nullary predicate $\sharp$[3] to all positive clauses[4], and the *active path Path* is a subset of a path through $M$ . In the rules of the

[3]We suppose that $\sharp$ is a new predicate that does not occur anywhere in *axioms* and *conjecture*

[4]Thus by default all positive clauses are used as possible start clauses.

calculus $C, C'$ and $C''$ are clauses, $\sigma$ is a term substitution, and $L_1, L_2$ is a *connection* with $\sigma(L_1) = \sigma(\overline{L_2})$. The rules of the calculus are applied in an analytic (i.e. bottom-up) way. The term substitution $\sigma$ is applied to the whole derivation.

The connection calculus is correct and complete [LS01] in the following sense: A first-order formula $M$ in clausal form is valid iff there is a connection proof for "$\neg\sharp, M, \{\}$", i.e., a derivation for "$\neg\sharp, M, \{\}$" in the connection calculus so that all leaves are axioms. The Prolog predicate `prove/5` implements the axiom, the reduction and the extension rule of the basic connection calculus in leanCoP:

```
% prove(Cla,Path,PathLim,Lem,Set)
prove([Lit|Cla],Path,PathLim,Lem,Set) :-
        % regularity
        (-NegLit=Lit;-Lit=NegLit) ->
        ( % lemmata
          %
          member(NegL,Path),
          unify_with_occurs_check(NegL,NegLit)
        ;
          lit(NegLit,NegL,Cla1,Grnd1),
          unify_with_occurs_check(NegL,NegLit),
          % iterative deepening
          %
          prove(Cla1,[Lit|Path],PathLim,Lem,Set)
        ),
        % restricted backtracking
        prove(Cla,Path,PathLim,Lem,Set).
prove([],_,_,_,_).
```

The tuple "$C, M, Path$" in connection calculus is here represented as follows:

- $C$ representing the open subgoal is a Prolog list `Cla`;
- the active path *Path* is a Prolog list `Path`;
- $M$ is written into Prolog's database before the actual proof search starts in a such way that for every clause $C \in M$ and for every literal $C \in M$ the fact `lit(Indexing_L,L,C1,Grnd)` is stored, where `C1`$=C\backslash\{$`L`$\}$ and `Grnd` is `g` if $C$ is ground, otherwise `Grnd` is `n`. `Indexing_L` is same as `L` modulo all its variables which are fresh (there is no twice or more occurrences in `Indexing_L`) everywhere in `Indexing_L` and it is used for fast finding the right fact in database without affecting the logically correct `L` by standard Prolog unification without occurs check.

**axiom:** $\dfrac{}{\{\}, M, Path}$

**reduction rule:** $\dfrac{C, M, Path \cup \{L_2\}}{C \cup \{L_1\}, M, Path \cup \{L_2\}}$

where there exists a unification substitution $\sigma$ such that $\sigma(L_1) = \sigma(\overline{L_2})$

**extension rule:** $\dfrac{C' \setminus \{L_2\}, M, Path \cup \{L_1\} \qquad C, M, Path}{C \cup \{L_1\}, M, Path}$

where $C'$is a fresh copy of some $C'' \in M$ such that $L_2 \in C'$and $\sigma(L_1) = \sigma(\overline{L_2})$ where $\sigma$ is unification substitution.

Note that the $\sigma$ used in the *reduction* and *extension* rules must be applied on all literals in all derivations except the literals in the set $M$ because these literals are not affected by any substitution $\sigma$.

Figure 5.1.: The basic clause connection calculus used in leanCoP.

- Atoms are represented by Prolog atoms, negation by "−".
- The substitution $\sigma$ is stored implicitly by Prolog.

`PathLim` is the current limit used for iterative deepening, `Lem` is the list of usable (previously derived) lemmas, `Set` a list of options, and `Proof` is the resulting proof. This predicate succeeds (using iterative deepening) iff there is a connection proof for the tuple represented by `Cla`, the DNF representation of the problem stored in Prolog's database using the `lit` predicate, and a `Path` with |`Path`| <`PathLim` where `PathLim` is the maximum size of the active `Path`. The predicate works as follows:

Line 18 implements the axiom, line 4 calculates the complement of the first literal `Lit` in `Cla`, which is used as the principal literal for the next reduction or extension step. The reduction rule is implemented in lines 7, 8 and 17. At line 7 and 8 it is checked whether the active path `Path` contains a literal `NegL` that unifies with the complement `NegLit` of the principal literal `Lit`. In this case the alternative lines after the semicolon are skipped and the proof search for the premise of the reduction rule is invoked in line 17. The extension rule is implemented in lines 10, 11, 14 and 17. Lines 10 and 11 are used to find a clause that contains the complement `NegLit` of the principal literal `Lit`. `Cla1` is the remaining set of literals of the selected clause and the new open subgoal of the left premise. The proof search for the left premise of the extension rule, in which the active path `Path` is extended by the principal literal `Lit`, is invoked in line 14, and if successful, we again continue on line 17.

Compared to standard tableaux or sequent calculi, connection calculi are not confluent[5]. To achieve completeness, an extensive use of backtracking is required. leanCoP

[5]Bad choice of *connection* might end up in dead end

uses two simple incomplete strategies (namely options `scut` and `cut`) for restricting backtracking that significantly reduces the search space [Ott10] without affecting the ability to find proofs in most tested cases (see Section 5.4).

Another major problem in connection calculi is the integration of equality. The paramodulation method that is widely used in saturation-based ATPs is not complete for goal-oriented approach of connection calculi. Therefore equality in leanCoP and similar ATPs is usually managed by adding the axioms of equality (reflexivity, symmetry, transitivity and substitutivity).

To obtain the clausal form, leanCoP uses its own implementation of clausifier introducing definitions (the `def` option), which seems to perform better with the leanCoP's core prover than other standard clausifiers (TPTP2X using the option `-t clausify:tptp`, FLOTTER and E) or direct transformation into clausal form (`nodef` option in leanCoP) [Ott10]. In the following subsections, we summarize several further methods used by leanCoP that improve its performance.

### 5.2.1. Iterative Deepening

Prolog uses a simple incomplete depth-first search strategy to explore the search space. This kind of incompleteness would result in a calculus that hardly proves any formula. In order to obtain a complete proof search in the connection calculus, iterative deepening on the proof depth, i.e. the size of the active path, is performed. It is achieved by inserting the following lines into the code:

```
(12)( Grnd1=g -> true ; length(Path,K),
                         K<PathLim -> true ;
(13) \+ pathlim -> assert(pathlim), fail ),
```

and the prover runs in the following iterative sense starting from `PathLimit`= 1:

```
prove(PathLim,Set) :-
        retract(pathlim) ->
        PathLim1 is PathLim+1,
        prove(PathLim1,Set).
```

When the extension rule is applied and the new clause is not ground, i.e. it does not contain any variable, it is checked whether the size `K` of the active path exceeds the current path limit `PathLim` (line 12). In this case the dynamic predicate `pathlim/0` is written into the Prolog's database (line 13) indicating the need to increase the path limit if the proof search with the current path limit fails. If the proof search fails and the predicate `pathlim` can be found in the database, then `PathLim` is increased by one and the proof search starts again.

### 5.2.2. Regularity Condition Optimization

**Definition 5.2.2** A connection proof is *regular* iff no literal occurs more than once in the active path.

Since the active path corresponds to the set of literals in a branch in the connection tableau representation, a connection tableau proof is regular if in the current branch no literal occurs more than once. The regularity condition is integrated into the connection calculus in Figure 5.1 by imposing the following restriction on the reduction and extension rule: $\forall L' \in C \cup \{L_1\} : \sigma(L') \notin \sigma(Path)$

**Lemma 5.2.3** *A formula $M$ in clausal form described above is valid iff there is a regular connection proof for "$\neg\sharp, M, \{\}$"*

Regularity is correct, since it only imposes a restriction on the applicability of the reduction and extension rules. The completeness proof can be found in [OB03, LS01].
The regularity condition must be checked whenever the reduction, extension or lemma rule is applied. The substitution $\sigma$ is not modified, i.e. the regularity condition is satisfied if the open subgoal does not contain a literal that is syntactically identical with a literal in the active path. This is implemented by inserting the following line into the code:

```
(3)  \+  (member(LitC ,[Lit|Cla]) ,
          member(LitP ,Path) ,
          LitC==LitP) ,
```

The Prolog predicate `\+` *Goal* succeeds only if *Goal* cannot be proven. In line 3 the corresponding *Goal* succeeds if the open subgoal `[Lit|Cla]` contains a literal `LitC` that is syntactically identical (built-in predicate `==/2` in Prolog) with a literal `LitP` in the active path `Path`. The built-in predicate `member/2` is used to enumerate all elements of a list.

### 5.2.3. Lemmata Optimization

The set of lemmata is represented by the list `Lem`. The lemma rule is implemented by inserting the following lines:

```
(5)  (  member(LitL ,Lem) ,  Lit==LitL
(6)  ;
```

In order to apply the lemma rule, the substitution $\sigma$ is not modified, i.e. the lemma rule is only applied if the list of lemmata `Lem` contains a literal `LitL` that is syntactically identical with the literal `Lit`. Furthermore, the literal `Lit` is added to the list `Lem` of lemmata in the (left) premise of the reduction and extension rule by adapting the following line:

```
(15) prove(Cla,Path,PathLim,[Lit|Lem],Set).
```

In the resulting implementation, the lemma rule is applied before the reduction and extension rules.

### 5.2.4. Restricted Backtracking

In Prolog the cut (!) is used to cut off alternative solutions when Prolog tries to prove a goal. The Prolog cut is a built-in predicate, which succeeds immediately when first encountered as a goal. Any attempt to re-satisfy the cut fails for the parent goal, i.e. other alternative choices are discarded that have been made from the point when the parent goal was invoked. Consequently, *restricted backtracking* is achieved by inserting a Prolog cut after the lemma, reduction, or extension rule is applied. It is implemented by inserting the following line into the code:

```
(16) ( member(cut,Set) -> ! ; true ),
```

Restricted backtracking is switched on if the list `Set` contains the option `cut`.
The *restricted start step* is used if the list `Set` includes the option `scut`. In this case only the first matching clause to starting $\neg\sharp$ literal is used.
Restricted backtracking and restricted start step lead to an incomplete proof search. In order to regain completeness, these strategies can be switched off when the search reaches a certain path limit. If the list `Set` contains the option `comp(`*Limit*`)`, where *Limit* is a natural number, the proof search is stopped and started again without using incomplete search strategies.

## 5.3. OCaml Implementation

In this section, we first discuss our implementation[6] of leanCoP in OCaml and its integration in HOL Light: the transformation of the higher-order goal to first order and the proof reconstruction. After that we compare our implementation to Harrison's implementation of MESON.

### 5.3.1. leanCoP in OCaml

Otten's implementation of leanCoP uses the Prolog search, backtracking, and indexing mechanisms to implement the connection tableaux proof search. This is a variation of the general idea of using the "Prolog technology theorem prover" (PTTP) proposed

[6] Available online at `http://cl-informatik.uibk.ac.at/users/cek/cpp15`

by Stickel [Sti88], in which connection tableaux takes a number of advantages from its similarity to Prolog,

In order to implement an equivalent program in a functional programming language, one needs to use either an explicit stack for keeping track of the current proof state (including the trail of variable bindings), or the continuation passing style. We choose to do the former, namely we add an explicit `todo` (stack), `subst` (trail) and `off` (offset in the trail) arguments to the main `prove` function. The stack keeps a list of tuples that are given as arguments to the recursive invocations of `prove` , whose full OCaml declaration (taking the open subgoal as its last argument) then looks as follows:

```
let rec lprove off subst path limit lemmas todo = function
  [] -> begin ... end
  | ((lit1 :: rest_clause) as clause) -> ... ;;
```

The function performs the proof search to the given depth, and if a proof has not been found, it returns the unit. It takes special attention to traverse the tree in the same order as the Prolog version. In particular, when the global option "cut" (restricting backtracking) is off, it performs all the backtrackings explicitly, while if "cut" is on, the parts of backtracking avoided in Prolog are also omitted. When a proof is found, the exception 'Solved' is raised: no further search is performed and the function exits with this exception.

The OCaml version and the Prolog version (simplified and with symbols renamed for clarity of comparison) are displayed together in Fig. 5.2. The algorithm proceeds as follows:

1. If nonempty, decompose the current open subgoal into the first literal `lit` and the rest `cla`.
2. Check for intersection between the current open subgoal and `path`.
3. Compute the negation of `lit`.
4. Check if `lit` is among the lemmas `lem`, if so try to prove `cla`. If `cut` is set, no other options are tried.
5. For each literal on the `path`, if `neglit` unifies with it, try to prove `cla`. If the unification succeeded and `cut` is set, no other options are tried.
6. For each clause in the matrix, try to find a literal that unifies with `neglit`, and then try to prove the rest of the newly created subgoal and the rest of the current open subgoal. If the unification and the first proof succeeded and `cut` is set, no other options are tried.
7. When the current open subgoal is empty, the subproof is finished (the axiom rule).

```
1 let rec prove path lim lem stack = function (lit :: cla) ->     prove([Lit|Cla],Path,PathLim,Lem,Set) :-

2   if not ((exists (fun litp -> exists (substeq litp) path))       \+(member(LitC,[Lit|Cla]),member(LitP,Path),
      (lit :: cla)) then (                                            LitC==LitP),

3   let neglit = negate lit in                                       (-NegLit=Lit;-Lit=NegLit) -> (

4   if not (exists (substeq lit) lem &&                                 member(LitL,Lem), Lit==LitL
       (prove path lim lem stack cla; cut)) then (                      ;

5   if not (fold_left (fun sf plit -> if sf then true else             member(NegL,Path),
      try (unify_lit neglit plit; prove path lim (lit :: lem)           unify_with_occurs_check(NegL,NegLit)
      stack cla; cut) with Unify -> sf) false path) then (              ;

6   let iter_fun (lit2, cla2, ground) = if lim > 0 || ground then      lit(NegLit,NegL,Cla1,Grnd1),
      try let cla1 = unify_rename (snd lit) (lit2, cla2) in             unify_with_occurs_check(NegL,NegLit),
      prove (lit :: path) (lim - 1) lem ((if cut then lim else -1),       ( Grnd1=g -> true ;
      path, lim, lit :: lem, cla) :: stack) cla1 with Unify -> () in        length(Path,K), K<PathLim -> true ;
    try iter iter_fun (try assoc neglit lits with Not_found -> [])          \+ pathlim -> assert(pathlim), fail),
    with Cut n -> if n = lim then () else raise Cut n)))                 prove(Cla1,[Lit|Path],PathLim,Lem,Set)
                                                                       ), ( member(cut,Set) -> ! ; true ),
  | [] -> match stack with
      (ct, path, lim, lem, cla) :: t ->
            prove path lim lem t cla; if ct > 0 raise (Cut ct)         prove(Cla,Path,PathLim,[Lit|Lem],Set).

7     | [] -> raise Solved;;                                         prove([],_,_,_,_,[]).
```

Figure 5.2.: The simplified OCaml and Prolog code side by side. The explicit trail argument and the computation of the resulting proof have been omitted for clarity, and some symbols were renamed to better correspond to each other. White-space and order of clauses has been modified to exemplify corresponding parts of the two implementations. Function `substeq` checks equality under the current context of variable bindings. Note that the last-but-one line in the Prolog code was merged into each of the three cases in the OCaml code. See the function `lprove` in file `leancop.ml` on our web page for the actual (non-simplified) OCaml code.

In Otten's implementation, the behaviour of the program with `cut` set is enabled by the use of the Prolog cut (`!`). Implementing it in OCaml amounts to a different mechanism in each of the three cases. In point 4 in the enumeration above, given that a single lemma has been found, there is no need to check for other lemmas. Therefore a simple `List.exists` call is sufficient to emulate this behaviour in OCaml. No backtracking over other possible occurrences of the lemma is needed here, and it is not necessary to add in this case the literal again into the list of lemmas as is done in the Prolog code (last-but-one line).

In point 5, multiple literals on the path may unify with different substitutions. We therefore use a list fold mechanism which changes the value whenever the unification is successful and `cut` is set. In point 6, we need to change our behaviour in between two successive calls to `prove`. As the first call takes the arguments to the second call on the stack, we additionally add a cut marker on the stack and handle an exception that can be raised by the call on the stack.

Whenever the clause becomes empty, all the tuples in the stack list need to be processed. For each tuple, the first component is the cut marker: if it is set, the `Cut` exception is raised with a depth level. The exception is handled only at the appropriate level. This directly corresponds to the semantics of the cut operator in Prolog [SES$^+$12].

What remains to be implemented is efficient equality checking and unification. Since we want to integrate our mechanism in HOL Light, we reuse the first order logic representation used in the implementation of HOL Light's MESON procedure: the substitutions are implemented as association lists, and applications of substitutions are delayed until an actual equality check or a unification step.

### 5.3.2. leanCoP in HOL Light

In order to enable the OCaml version of leanCoP as a proof tactic and procedure in HOL Light, we first need to transform a HOL goal to a leanCoP problem and when a proof has been found we replay the proof in higher-order logic. In order to transform a problem in higher-order logic to first-order logic without equality, we mostly reuse the steps of the transformation already used by MESON, additionally ensuring that the conjecture is separated from the axioms to preserve leanCoP's goal-directed approach. The transformation starts by assuming the duplicated copies of polymorphic theorems to match the existing assumptions. Next the goal $axioms \rightarrow conjecture$ is transformed to $(axioms \wedge \sharp) \rightarrow (conjecture \wedge \sharp)$ with the help of a special symbol, which we define in HOL as: $\sharp = \top$. Since the conjecture is refuted and the problem is converted to CNF, the only positive occurrence of $\sharp$ is present in a singleton clause, and the negative occurrences of $\sharp$ are present in every clause originating from the conjecture. The CNF clauses directly correspond to the DNF used by the Prolog version of leanCoP. Since no steps distinguish between the positivity of literals, the two can be used interchangeably in the proof procedure. We start the FOL algorithm by trying to prove $\neg\sharp$.

Since the final leanCoP proof may include references to lemmas, the reconstruction cannot be performed the same way as it is done in MESON. There, a tree structure is used for finished proofs. Each subgoal either closes the branch (the literal is a negation of a literal already present on the path) or is a branch extension with a (possibly empty) list of subgoals. In leanCoP, each subgoal can refer to previous subgoals, so the order of the subgoals becomes important. We therefore flatten the tree to a list, which needs to be traversed in a linear order to reconstruct the proof.

We define a type of proof steps, one for each proof step in the calculus. Each application of a lemma step or path step constructs a proof step with an appropriate first-order term. For an application of a tableaux extension step we use Harrison's contrapositive mechanism: we store the reference to the actual transformed HOL theorem whose conclusion is a disjunction together with the number of the disjunct that got resolved[7].

```
type proof = Lem of fol_atom
           | Pat of fol_atom
           | Res of fol_atom * (int * thm);;
```

A list of such proof steps together with a final substitution and an initially empty list of already proved lemmas are the complete input to the proof reconstruction procedure. The reconstruction procedure always looks at the first step on the list. First, a HOL term is constructed from the FOL term with the final substitution applied. This step is straightforward, as it amounts to reversing the mapping of variables and constants applied to transform the HOL CNF to FOL CNF, with new names invented for new variables. Next, we analyze the kind of the step. If the step is a path step, the theorem $tm \vdash tm$ is returned, using the HOL `ASSUME` proof rule. If the step is a lemma step, the theorem whose conclusion is equal to $tm$ is found on the list of lemmas and returned. Finally, if the proof step is an extension step, we first find the disjuncts of the HOL theorem in the proof step apart from the one that got matched. We then fold over this list, at every step calling the reconstruction function recursively with the remaining proof steps and the list of lemmas extended by each of the calls. The result of the fold is the list of theorems $[\vdash tm_1, \vdash tm_2, ..., \vdash tm_n]$ which gets matched with the contrapositive theorem $\vdash tm_1 \wedge \ldots \wedge tm_n \rightarrow tm_g$ using the HOL proof rule `MATCH_MP` to obtain the theorem $\vdash tm_g$. Finally, by matching this theorem to the term $tm$ the theorem $\vdash tm$ is obtained.

As the reconstruction procedure traverses the list, it produces the theorem that corresponds to the first goal, namely $... \vdash \neg\sharp$. By unfolding the definition of $\sharp$, we obtain $... \vdash \bot$ which concludes the refutation proof.

[7]Contrary to the name, the HOL Light type `fol_atom` implements a literal: it is either positive or negative.

### 5.3.3. Comparison to MESON

The simplified OCaml code of the core HOL Light's MESON algorithm as described in [Har09] is as follows:

```
let rec mexpand rules ancestors g cont (env,n,k) =
  if n < 0 then failwith "Too deep" else
    try tryfind (fun a -> cont (unify_literals
                                 env (g,negate a),n,k))
          ancestors
    with Failure _ -> tryfind (fun rule ->
      let (asm,c),k' = renamerule k rule in
      itlist (mexpand rules (g::ancestors)) asm cont
        (unify_literals env (g,c),n - length asm,k'))
      rules;;

let puremeson fm =
  let cls = simpcnf(specialize(pnf fm)) in
  let rules = itlist ((@) ** contrapositives) cls [] in
  deepen (fun n -> mexpand rules [] False (fun x -> x)
                          (undefined,n,0); n)
    0;;
```

The toplevel `puremeson` function proceeds by turning the input formula into a clausal form, making contrapositives (`rules`) from the clauses, and then repeatedly calling the `mexpand` function with these rules using iterative deepening over the number of nodes permitted in the proof tree.

The `mexpand` function takes as its arguments the `rules`, an (initially empty) list of goal `ancestors`, the goal `g` to prove (initially `False`, which was also added to all-negative clauses when creating contrapositives), a continuation function `cont` for solving the rest of the subgoals (initially the identity), and a tuple consisting of the current trail `env`, the number `n` of additional nodes in the proof tree permitted, and a counter `k` for variable renaming.

If the allowed node count is not negative, `mexpand` first tries to unify the current goal with a negated ancestor, followed by calling the current continuation (trying to solve the remaining goals) with the extended trail. If all such unification/continuation attempts fail (i.e., they throw Failure), an extension step is tried with all rules. This means that the head of a (renamed) rule is unified with the goal `g`, the goal is appended to the ancestors and the `mexpand` is called (again using list folding with the subsequently modified trail and continuation) for all the assumptions of the rule, decreasing the allowed node count for the recursive calls.

The full HOL Light version of MESON additionally uses a smarter (divide-and-conquer) policy for the size limit, checks the goal for being already among the ancestors, caches continuations, and uses simple indexing. Below we enumerate some of the most important differences between the leanCoP algorithm and MESON and their implementations in HOL Light. Their practical effect is measured in Section 5.4.

- leanCoP computes and uses lemmas. The literals that correspond to closed branches are stored in a list. Each call to the main `prove` function additionally looks for the first literal in the list of lemmas. This can cost a linear number of equality checks if no parts of the proof are reused, but it saves computations if there are repetitions.
- Both algorithms use iterative deepening; however the depth and termination conditions are computed differently.
- MESON is implemented in the continuation-passing-style, so it can use as an additional optimization caching of the continuations. If any continuations are repeated (at the same depth level), the subproof is not retried. Otten's leanCoP uses a direct Prolog implementation which cannot (without further tricks) do such repetition elimination. The implementation of leanCoP in OCaml behaves the same.
- leanCoP may use the cut after the lemma step, path step or successful branch closing in the extension step. Implementing this behaviour in OCaml exactly requires multiple `Cut` exceptions – one for each depth of the proof.
- The checking for repetitions is done in a coarser way in MESON than in leanCoP, allowing leanCoP to skip some work done by MESON.
- The search is started differently in leanCoP and in MESON. leanCoP starts with a conjecture clause, which likely contributes to its relatively good performance on larger problems.

## 5.4. Experimental Setup and Results

For the experiments we use HOL Light SVN version 199 (September 2014), Metis 2.3, and leanCoP 2.1. Unless noted otherwise, the systems are run on a 48-core server with AMD Opteron 6174 2.2 GHz CPUs, 320 GB RAM, and 0.5 MB L2 cache per CPU. Each problem is always assigned one CPU.

The systems are compared on several benchmarks, corresponding to different modes of use: goals coming from HOL Light itself, the general set of problems from the TPTP library, and the large-theory problems extracted from Mizar [Urb06c]. The first set of problems is important, however since these problems come from HOL Light itself, they are likely naturally biased towards MESON. The Mizar problems come from the two

MPTP-based benchmarks: the MPTP Challenge and MPTP2078 [AHK$^+$14]. These are large-theory problems coming from a different ITP, hence they do not introduce the implicit bias as the HOL Light problems, while coming from a more relevant application domain than the general TPTP problems.

For HOL Light, we evaluate (with 5 second time limit) on two sets of problems. First, we look at 872 MESON-solved HOL Light goals that were made harder by removing splitting. In this scenario the tactic is applied to a subgoal of a proof, which is a bit similar to the Judgement-Day [BN10] evaluation used for Isabelle/Sledgehammer, where the goals are however restricted to the solvable ones. Table 5.1 shows the results. Second, we evaluate on the top-level goals (with their dependencies already minimized) that have been solved with the HOL(y)Hammer system [KU14], i.e., by using the strongest available ATPs. This set is important because tactics such as MESON, Metis and now also leanCoP can be tried as a first cheap method for reconstructing the proofs found by the stronger ATPs. The results are shown in Table 5.2. In both cases, the OCaml implementation of leanCoP performs best, improving the MESON's performance in the first case by about 11%, and improving on Metis on the second set of problems by about 45%.

Table 5.3 shows the results of the evaluation on all 7036 FOF problems coming from TPTP 6.0.0, using 10 second time limit. Here the difference to Metis is not so significant, probably because Metis implements ordered paramodulation, which is useful for many TPTP problems containing equality. The improvement over MESON is about 17%. Table 5.4 and Table 5.5 show the results on the small (heuristically minimized) and large MPTP Challenge problems. The best version of the OCaml implementation of leanCoP improves by 54% on Metis and by 90% on MESON on the small problems, and by 88% on Metis and 100% on MESON on the large problems. Here the goal directedness of leanCoP is probably the main factor.

Finally, to get a comparison also with the best ATPs on a larger ITP-oriented benchmark (using different hardware), we have done a 10s evaluation of several systems on the newer MPTP2078 benchmark (used in the 2012 CASC@Turing competition), see Table 5.6 and Table 5.7. The difference to Metis and MESON on the small problems is still quite significant (40% improvement over MESON), while on the large problems the goal-directedness again shows even more (about 90% improvement). While Vampire's (version 2.6) SInE heuristic [HV11] helps a lot on the larger problems [UHV10], the difference there between E (1.8) and our version of leanCoP is not so great as one could imagine given the several orders of magnitude difference in the size of their implementations.

## 5.5. Conclusion

We have implemented an OCaml version of the leanCoP compact connection prover, and the reconstruction of its proofs inside HOL Light. This proof-reconstruction functionality

| Prover | Theorem (%) | Unique |
|---|---|---|
| mlleancop-cut-comp | 759 (87.04) | 2 |
| mlleancop-nocut | 759 (87.04) | 2 |
| plleancop-cut | 752 (86.23) | 0 |
| plleancop-nc | 751 (86.12) | 0 |
| metis-23 | 708 (81.19) | 26 |
| meson | 683 (78.32) | 4 |
| any | 832 (95.41) | |

Table 5.1.: Core HOL Light MESON calls without splitting (872 goals), 5sec per goal

| Prover | Theorem (%) | Unique |
|---|---|---|
| mlleancop-cut-comp | 1178 (75.70) | 12 |
| mlleancop-nocut | 1162 (74.67) | 0 |
| meson | 1110 (71.33) | 39 |
| plleancop-nc | 1085 (69.73) | 0 |
| plleancop-cut | 1084 (69.66) | 0 |
| metis-23 | 814 (52.31) | 16 |
| any | 1260 (80.97) | |

Table 5.2.: HOL Light dependencies (1556 goals, 5sec)

| Prover | Theorem (%) | Unique |
|---|---|---|
| mlleancop-cut-conj | 1669 (23.72) | 73 |
| plleancop-cut-conj | 1648 (23.42) | 21 |
| plleancop-cut | 1622 (23.05) | 34 |
| mlleancop-cut | 1571 (22.32) | 9 |
| metis-23 | 1562 (22.20) | 261 |
| meson | 1430 (20.32) | 28 |
| plleancop-nocut | 1358 (19.30) | 25 |
| mlleancop-nocut | 1158 (16.45) | 3 |
| any | 2433 (34.57) | |

Table 5.3.: TPTP (7036 goals with at least one conjecture, 10sec)

| Prover | Theorem (%) | Unique |
|---|---|---|
| pllean-cut-conj | 103 (40.87302) | 2 |
| pllean-cut | 99 (39.28571) | 8 |
| mlleancop-cut-conj | 91 (36.11111) | 2 |
| mlleancop-cut | 79 (31.34921) | 0 |
| mlleancop-nocut | 76 (30.15873) | 0 |
| pllean-nc | 62 (24.60317) | 1 |
| metis-23 | 59 (23.41270) | 3 |
| meson-infer | 48 (19.04762) | 0 |
| any | 124 (49.20635) | |

Table 5.4.: Bushy (small) MPTP Challenge problems (252 in total), 10sec)

| Prover | Theorem (%) | Unique |
|---|---|---|
| pllean-cut-conj | 61 (24.20635) | 5 |
| mlleancop-cut-conj | 60 (23.80952) | 9 |
| pllean-cut | 57 (22.61905) | 4 |
| mlleancop-nocut | 47 (18.65079) | 0 |
| mlleancop-cut | 47 (18.65079) | 0 |
| metis-23 | 32 (12.69841) | 3 |
| meson-infer | 30 (11.90476) | 0 |
| pllean-nc | 26 (10.31746) | 0 |
| any | 83 (32.93651) | |

Table 5.5.: Chainy (large) MPTP Challenge problems (252 in total), 10sec)

| Prover | Theorem (%) |
|---|---|
| Vampire | 1198 (57.65) |
| e18 | 1022 (49.18) |
| mlleancop-cut-conj | 613 (29.49) |
| pllean-cut-conj | 597 (28.72) |
| metis-23 | 564 (27.14) |
| mlleancop-cut | 559 (26.90) |
| pllean-cut | 544 (26.17) |
| pllean-comp7 | 539 (25.93) |
| mlleancop-nocut | 521 (25.07) |
| pllean-nc | 454 (21.84) |
| meson-infer | 438 (21.07) |
| any | 1277 (61.45) |

Table 5.6.: Bushy (small) MPTP2078 problems (2078 in total), 10sec)

| Prover | Theorem (%) |
|---|---|
| Vampire | 634 (30.51) |
| e18 | 317 (15.25) |
| mlleancop-cut-conj | 243 (11.69) |
| pllean-cut-conj | 196 (9.43) |
| pllean-cut | 170 (8.18) |
| pllean-comp7 | 159 (7.65) |
| mlleancop-nocut | 150 (7.21) |
| mlleancop-cut | 146 (7.02) |
| meson-infer | 145 (6.97) |
| metis-23 | 138 (6.64) |
| pllean-nc | 126 (6.06) |
| any | 693 (33.34) |

Table 5.7.: Chainy (large) MPTP2078 problems (2078 in total), 10sec)

can be also used to certify in HOL Light an arbitrary TPTP proof produced by leanCoP, thus turning leanCoP into one of the few ATPs whose proofs enjoy LCF-style verification in one of the safest LCF-based systems. The performance of the OCaml version on the benchmarks is comparable to the Prolog version, while it always outperforms Metis and MESON, sometimes very significantly on the relevant ITP-related benchmarks.

We provide a HOL Light interface that is identical to the one offered by `MESON`, namely we provide two tactics and a rule. `LEANCOP_TAC` and `ASM_LEANCOP_TAC` are given a list of helper theorems, and then try to solve the given goal together (or the given goal and assumptions, respectively). The `LEANCOP` rule, given a list of helper theorems acts as a conversion, i.e., given a term statement it tries to prove a theorem whose conclusion is identical to that of the term. The benchmarks show that these are likely the strongest single-step proof-reconstructing first-order tactics available today in any ITP system.

# 6 Re-use of Proof Knowledge

## Publication Details

Cezary Kaliszyk and Josef Urban. Automated Reasoning Service for HOL Light. In Jacques Carette, David Aspinall, Christoph Lange, Petr Sojka, and Wolfgang Windsteiger, editors. Proc. of the 6th *Conference on Intelligent Computer Mathematics (CICM'13)*, volume 7961 of LNCS, pp. 120–135. Springer, 2013. doi: 10.1007/978-3-642-39320-4_8

Cezary Kaliszyk and Josef Urban. HOL(y)Hammer: Online ATP service for HOL Light. *Mathematics in Computer Science*, 9(1):5–22, Springer, 2015. doi: 10.1007/s11786-014-0182-0

## Abstract

HOL(y)Hammer is an online AI/ATP service for formal (computer-understandable) mathematics encoded in the HOL Light system. The service allows its users to upload and automatically process an arbitrary formal development (project) based on HOL Light, and to attack arbitrary conjectures that use the concepts defined in some of the uploaded projects. For that, the service uses several automated reasoning systems combined with several premise selection methods trained on all the project proofs. The projects that are readily available on the server for such query answering include the recent versions of the Flyspeck, Multivariate Analysis and Complex Analysis libraries. The service runs on a 48-CPU server, currently employing in parallel for each task 7 AI/ATP combinations and 4 decision procedures that contribute to its overall performance. The system is also available for local installation by interested users, who can customize it for their own proof development. An Emacs interface allowing parallel asynchronous queries to the service is also provided. The overall structure of the service is outlined, problems that arise and their solutions are discussed, and an initial account of using the system is given.

## 6.1. Introduction and Motivation

HOL Light [Har96a] is one of the best-known interactive theorem proving (ITP) systems.

It has been used to prove a number of well-known mathematical theorems[1] and as a platform for formalizing the proof of the Kepler conjecture targeted by the Flyspeck project [Hal06]. The whole Flyspeck development, together with the required parts of the HOL Light library consisted of about 14000 theorems as of June 2012, growing to about 19000 theorems as of August 2013. Motivated by the development of large-theory automated theorem proving [Urb11a, HV11, PB12, USPV08] and its growing use for ITPs like Isabelle [PS07a] and Mizar [URS13, US10], we have recently implemented translations from HOL Light to ATP (automated theorem proving) formats, developed a number of premise-selection techniques[2] for HOL Light, and experimented with the strongest and most orthogonal combinations of the premise-selection methods and various ATPs. This initial work, described in [KU14], has shown that 39% of the (June 2012) 14185 Flyspeck theorems could be proved in a push-button mode (without any high-level advice and user interaction) in 30 seconds of real time on a fourteen-CPU workstation. More recent work on the AI/ATP methods have raised this performance to 47% [KU13f].

The experiments that we did emulated the Flyspeck development (when the user always knows all the previous proofs[3] at a given point, and wants to prove the next theorem), however they were all done in an offline mode which is suitable for such experimentally-driven research. The ATP problems were created in large batches using different premise-selection techniques and different ATP encodings (untyped first-order [SSCG06], polymorphic typed first-order [BP13], and typed higher-order [GS06]), and then attempted with different ATPs (17 in total) and different numbers of the most relevant premises. Analysis of the results interleaved with further improvements of the methods and data have gradually led to the current strongest combination of the AI/ATP methods.

This strongest combination now gives to a HOL Light/Flyspeck user a 47% chance (when using 14 CPUs, each for 30s) that he will not have to search the library for suitable lemmas and figure out the proof of the next toplevel theorem by himself. For smaller (proof-local) lemmas such likelihood should be correspondingly higher. To really provide this strong automated advice to the users, the functions that have been implemented for the experiments need to be combined into a suitable AI/ATP tool. Our eventual goal (from which we are of course still very far) should be an easy-to-use service, which in its online form offers to formal mathematics (done here in HOL Light, over the concepts defined formally in the libraries) what services like Wolfram Alpha offer for informal/symbolic mathematics. Some expectations, linked to the recent success of the IBM Watson system, are today even higher[4]. Indeed, we believe that developing stronger and stronger

[1] `http://www.cs.ru.nl/~freek/100/`

[2] Premise selection [AHK+14, KvLT+12] is the problem of selecting suitable premises (theorems, definitions, lemmas, etc.) from a large formal library for proving a new conjecture over such library.

[3] The Flyspeck processing order is used to define precisely what "previous" means. See [KU14] for details.

[4] See for example Jonathan Borwein's article: `http://theconversation.edu.au/if-i-had-a-blank-cheque-id-turn-ibms-watson-into-a-maths-genius-1213`

AI/ATP tools similar to the one presented here is a necessary prerequisite providing the crucial semantic understanding/reasoning layer for building larger Watson-like systems for mathematics that will (eventually) understand (nearly-)natural language and (perhaps reasonably semanticized versions/alternatives of) LaTeX. The more user-friendly and smarter such AI/ATP systems become, the higher also the chance that mathematicians (and exact scientists) will get some nontrivial benefits[5] from encoding mathematics (and exact science) directly in a computer-understandable form.

This paper describes such an AI/ATP service based on the formal mathematical corpora like Flyspeck developed with HOL Light. The service – HOL(y)Hammer (HH) – is now available as a public online system[6] instantiated for several large HOL Light libraries, running on a 48-CPU server spawning for each query by default 7 different AI/ATP combinations and four decision procedures. We first describe in Section 6.2 the static (i.e., not user-updatable) problem solving functions developed in the first simplified version of the service for the most interesting example of Flyspeck. This initial version of the service allowed the users to experiment with ATP queries over the fixed June 2012 version of Flyspeck for which the AI/ATP components had been gradually developed over several months in the offline experiments described in [KU14]. Section 6.3 then discusses the issues and solutions related to running the service for multiple libraries and their versions at once, allowing the users also to submit a new library to the server or to update an existing library and all its AI/ATP components. Section 6.4 shows examples of interaction with the service, using web, Emacs, and command-line interfaces. The service can be also installed locally, and trained on user's private developments. This is described in Section 6.5. Section 6.6 concludes and discusses future work.[7]

## 6.2. Description of the Problem Solving Functions for Flyspeck

The overall problem solving architecture without the updating functions is shown in Figure 6.1. Since Flyspeck is the largest and most interesting corpus on which this architecture was developed and tested, we use the Flyspeck service as a running example in this whole section. The service receives a query (a conjecture to prove, possibly with local assumptions) generated by one of the clients/frontends (Emacs, web interface, HOL session, etc.). If the query produces a parsing (or type-checking) error, an exception is raised, and an error message is sent as a reply. Otherwise the parsed query is processed in parallel by the (time-limited) AI/ATP combinations and the native HOL Light decision procedures (each managed by its forked HOL Light process, and terminated/killed by the

[5]Formal verification itself is of course a great benefit, but its cost has been so far too high to attract most mathematicians.

[6]`http://colo12-c703.uibk.ac.at/hh/`

[7]This paper is an extended version of [KU13a].

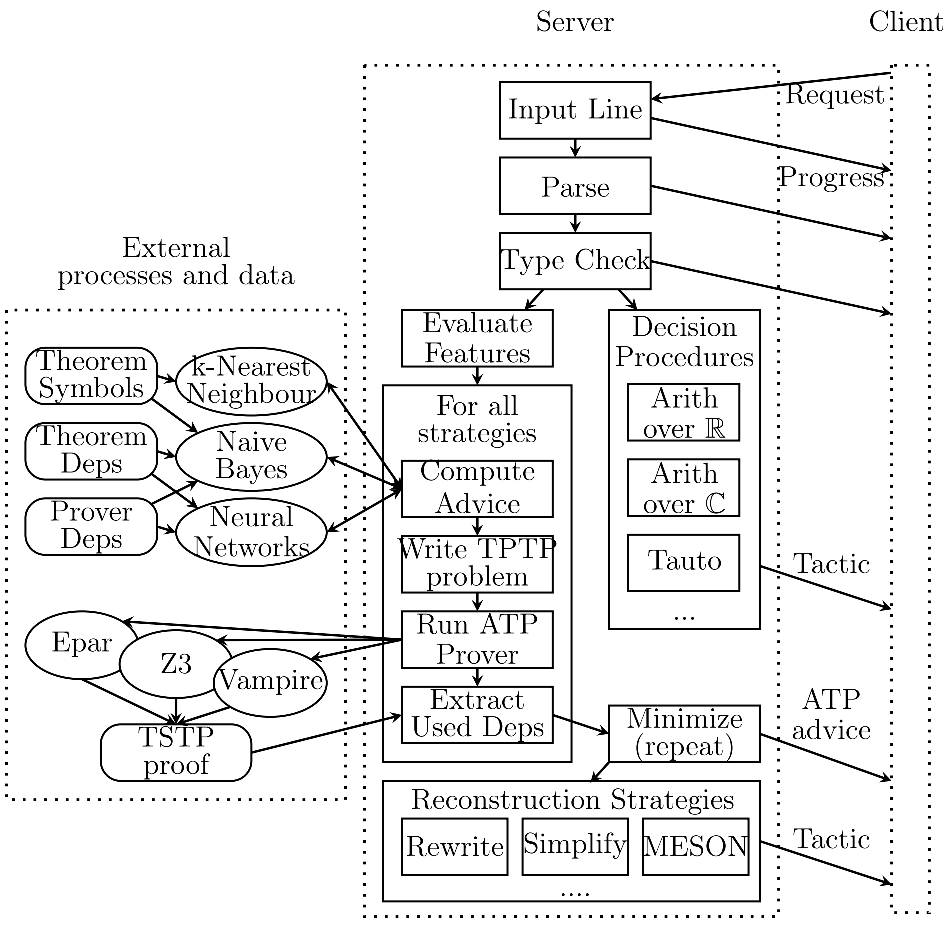


Figure 6.1.: Overview of the problem solving functions

master process if not finished within its global time limit). Each of the AI/ATP processes computes a specific feature representation of the query, and sends such features to a specific instance of a premise advisor trained (using the particular feature representation) on previous proofs. Each of the advisors replies with a specific number of premises, which are then translated to a suitable ATP format, and written to a temporary file on which a specific ATP is run. The successful ATP result is then (pseudo-)minimized, and handed over to the combination of proof-reconstruction procedures. These procedures again run in parallel, and if any of them is successful, the result is sent as a particular tactic application to the frontend. In case a native HOL Light decision procedure finds a proof, the result (again a particular tactic application) can be immediately sent to the frontend. The following subsections explain this process in more detail.

### 6.2.1. Feature Extraction and Premise Selection

Given a (formal) mathematical conjecture, the selection of suitable premises from a large formal library is an interesting AI problem, for which a number of methods have been tried recently [Urb11a, KvLT+12, KU13f]. The strongest methods use machine learning on previous problems, combined in various ways with heuristics like SInE [HV11]. To use the machine learning systems, the previous problems have to be described as training examples in a suitable format, typically as a set of (input) features characterizing a given theorem, and a set of labels (output features) characterizing the proof of the theorem. Devising good feature/label characterizations for this task is again an interesting AI problem (see, e.g. [US10]), however already the most obvious characterizations like the conjecture symbols and the names of the theorems used in the conjecture's proof are useful. This basic scheme can be extended in various ways; see [KU14] for the feature-extraction functions (basically adding various subterm and type-based characteristics) and label-improving methods (e.g., using minimized ATP proofs instead of the original Flyspeck proofs whenever possible) that we have so far used for HOL Light. For example, the currently most useful version of the characterization algorithm would describe the HOL theorem `DISCRETE_IMP_CLOSED`:[8]

```
∀s:real^N→bool e.
      &0 < e ∧ (∀x y. x IN s ∧ y IN s ∧ norm(y − x) < e ⟹ y = x)
      ⟹ closed s
```

by the following set of strings that encode its symbols and normalized types and terms:

```
"real", "num", "fun", "cart", "bool", "vector_sub", "vector_norm",
"real_of_num", "real_lt", "closed", "_0", "NUMERAL", "IN", "=", "&0",
"&0 < Areal", "0", "Areal", "Areal^A", "Areal^A - Areal^A",
"Areal^A IN Areal^A->bool", " Areal^A->bool", "_0",
"closed Areal^A->bool", "norm (Areal^A - Areal^A)",
"norm (Areal^A - Areal^A) < Areal"
```

Here, `real` is a type constant, `IN` is a term constructor, `Areal^A->bool` is a normalized type, `Areal^A` its component type, `norm (Areal^A - Areal^A) < Areal` is a normalized atomic formula, and `Areal^A - Areal^A` is its normalized subterm.

On average, for each of our feature-extraction methods there are in total about 30000 possible conjecture-characterizing features extracted from the theorems in the Flyspeck development. The output features (labels) are in the simplest setting just the names of the Flyspeck theorems[9] extracted from the proofs with a modified (proof record-

[8] `http://mws.cs.ru.nl/~mptp/hol-flyspeck/trunk/Multivariate/topology.html#DISCRETE_IMP_CLOSED`

[9] In practice, the Flyspeck theorems are further preprocessed to provide better learning precision, for example by splitting conjunctions and detecting which of the conjuncts are relevant in which proof. Again, see [KU14] for the details. The number of labels used for the June 2012 Flyspeck version with 14185 theorems is thus 16082.

ing [KK13]) HOL Light kernel. These features and labels are (for each extraction method) serially numbered in a stable way (using hashtables), producing from all Flyspeck proofs the training examples on which the premise selectors are trained. The learning-based premise selection methods currently used are those available in the SNoW [CCRR99] sparse learning toolkit (most prominently sparse naive Bayes), together with a custom implementation [KU13f] of the distance-weighted $k$-nearest neighbor ($k$-NN) learner [Dud76]. Training a particular learning method on all (14185) characterizations extracted from the Flyspeck proofs takes from 1 second for $k$-NN (a lazy learner that essentially just loads all the 14185 proof characterizations) and 6 seconds for naive Bayes using labels from minimized ATP proofs, to 25 seconds for naive Bayes using the labels from the original Flyspeck proofs.[10] The trained premise selectors are then run as daemons (using their server modes) that accept queries in the language of the numerical features over which they have been trained, producing for each query their ranking of all the labels, corresponding to the available Flyspeck theorems.

Given a new conjecture, the first step of each of the forked HOL Light AI/ATP managing process is thus to compute the features of the conjecture according to a particular feature extraction method, compute (using the corresponding hashtable) the numerical representation of the features, and send these numeric features as a query to the corresponding premise-selection daemon. The daemon replies within a fraction of a second with its ranking, the exact speed depending on the learning method and the size of the feature/label sets. This ranking is translated back (using the corresponding table) to the ranking of the HOL Light theorems. Each of the AI/ATP combinations then uses its particular number (optimized so that the methods in the end complement each other as much as possible) of the best-ranked theorems, passing them together with the conjecture to the function that translates such set of HOL Light formulas to a suitable ATP format.

### 6.2.2. Translation to ATP Formats and Running ATPs

As mentioned in Section 6.1, several ATP formalisms are used today by ATP and SMT systems. However the (jointly) most useful proof-producing systems in our experiments turned out to be E [Sch02] version 1.6 (run under the Epar [Urb14] strategy scheduler), Vampire [KV13] 2.6, and Z3 [dMB08] 4.0. All these systems accept the TPTP untyped first-order format (FOF). Even when the input formalism (the HOL logic [Pit93] - polymorphic version of Church's simple type theory) and the output formalism (TPTP FOF) are fixed, there are in general many methods [BBPS13] to translate from the former to the latter, each method providing different tradeoffs between soundness, completeness, ATP efficiency, and the overall (i.e., including HOL proof reconstruction) efficiency. The

[10]The original Flyspeck proofs are often using theorems that are in some sense redundant, resulting in longer proof characterizations (and thus longer learning). This is typically a consequence of using larger building blocks (e.g., decision procedures, drawing in many dependencies) when constructing the ITP proofs.

particular method chosen by us in [KU14] and used currently also for the service is the polymorphic tagged encoding [BBPS13]. To summarize, the higher-order features (such as lambda abstraction, application) of the HOL formulas are first encoded (in a potentially incomplete way) in first-order logic (still using polymorphic types), and then type tags are added in a way that usually guarantees type safety during the first-order proof search.

This translation method is in general not stable on the level of single formulas, i.e., it is not possible to just keep in a global hashtable the translated FOF version for each original HOL formula, as done for example for the Miz$\mathbb{AR}$ ATP service [URS13, KU13d]. This is because a particular optimization (by Meng and Paulson [MP08]) is used for translating higher-order constants, creating for each such constant $c$ a first-order function that has the minimum arity with which $c$ is used in the particular set of HOL formulas that is used to create the ATP (FOF) problem. So once the particular AI/ATP managing process advises its $N$ most-relevant HOL Light theorems for the conjecture, this set of theorems and the conjecture are as a whole passed to the translation function, which for each AI/ATP instance may produce a slightly different FOF encoding on the formula level. The encoding function is still reasonably fast, taking fractions of a second when using hundreds of formulas, and still has the property that both the FOF formula names and the FOF formulas (also those inferred during the ATP proof search) can typically be decoded back into the original HOL names and formulas, allowing later HOL proof reconstruction.

Each AI/ATP instance thus produces its specific temporary file (the FOF ATP problem) and runs its specific ATP system on it with its time limit. The time limit is currently set globally to 30 seconds for each instance, however (as usual in strategy scheduling setups) this could be made instance-specific too, based on further analysis of the time performance of the particular instances. Vampire and Epar already do such scheduling internally: the current version of Epar runs a fixed schedule of 14 strategies, while Vampire runs a problem-dependent schedule using for each problem a varied number of strategies. Assuming one strategy for Z3 and on average eight strategies for Vampire, this now means that using 10-CPU parallelization results in about 100 different proof-data/feature-extraction/learning/premise-slicing/ATP-strategy instantiations tried by the online service within the 30 seconds of the real time allowed for each query. Provided sufficient complementarity of such instantiationsand enough CPUs, this significantly raises the overall power of the service [KU14, KU13f].

### 6.2.3. The AI/ATP Combinations Used

An example of the 25 initially used combinations of the machine learner, proof data, number of top premises used, the feature extraction method, and the ATP system is shown in Table 6.1. The proof data are either just the data from the ATP proofs, or a combination of the ATP proofs with the original HOL proofs. The ATP proofs (ATP0,

..., ATP3) are created by a particular MaLARea-style [USPV08] (i.e., re-using the proofs found in previous iteration for further learning) iteration of the experimenting, possibly preferring either the Vampire or Epar proofs (V_pref, E_pref). The HOL proofs are obtained by slightly different versions of the HOL proof recording. The HOL/ATP combinations typically use the HOL proof only when the ATP proof is not available, see [KU14] for details. The `standard` feature extraction method combines the formula's symbols, standard-normalized subterms and normalized types into its feature vector. The standard normalization here means that each variable name is in each formula replaced by its normalized HOL type. Types are normalized by renaming their polymorphic variables with de Bruijn indices. The `all-vars-same` and `all-vars-diff` methods respectively just rename all formula variables into one common variable, or keep them all different. This obviously influences the concept of similarity used by the machine learners (see [KU14] for more discussion). The 40-NN and 160-NN learners are $k$-nearest-neighbors, run with $k = 40$ and $k = 160$. The particular combination of the AI/ATP is chosen by computing in a greedy fashion the set of methods with the greatest coverage of the solvable Flyspeck problems. This changes often, whenever some of the many components of this AI architecture get improved. For example, after the more recent strengthening of the premise-selection and ATP components described in [KU13f], and the addition of multiple developments and functions for their dynamic update described in Section 6.3, the number of AI/ATP combinations run for a single query was reduced to 7.

### 6.2.4. Use of Decision Procedures

Some goals are hard for ATPs, but are easy for the existing decision procedures already implemented in HOL Light. To make the service more powerful, we also try to directly use some of these HOL Light decision procedures on the given conjecture. A similar effect could be achieved also by mapping some of the HOL Light symbols (typically those encoding arithmetics) to the symbols that are reserved and treated specially by SMT solvers and ATP systems. This is now done for example in Isabelle/Sledgehammer [PB12], with the additional benefit of the combined methods employed by SMTs and ATPs over various well-known theories. Our approach is so far much simpler, which also means that we do not have to ensure that the semantics of such special theories remains the same (e.g., $1/0 = 0$ in HOL Light). The HOL Light decision procedures might often not be powerful enough to prove whole theorems, however for example the `REAL_ARITH`[11] tactic is called on 2678 unique (sub)goals in Flyspeck, making such tools a useful addition to the service.

Each decision procedure is spawned in a separate instance of HOL Light using our parallel infrastructure, and if any returns within the timeout, it is reported to the user. The decision procedures that we found most useful for solving goals are:[12]

[11] `http://www.cl.cam.ac.uk/~jrh13/hol-light/HTML/REAL_ARITH.html`

[12] The reader might wonder why the above mentioned `REAL_ARITH` is not among the tactics used. The

| Learner | Proofs | Premises | Features | ATP |
|---|---|---|---|---|
| Bayes | ATP2 | 0092 | standard | Vampire |
| Bayes | ATP2 | 0128 | standard | Epar |
| Bayes | ATP2 | 0154 | standard | Epar |
| Bayes | ATP2 | 1024 | standard | Epar |
| Bayes | HOL0+ATP0 | 0512 | all-vars-same | Epar |
| Bayes | HOL0+ATP0 | 0128 | all-vars-diff | Vampire |
| Bayes | ATP1 | 0032 | standard | Z3 |
| Bayes | ATP1_V_pref | 0128 | all-vars-diff | Epar |
| Bayes | ATP1_V_pref | 0128 | standard | Z3 |
| Bayes | HOL0+ATP0 | 0032 | standard | Z3 |
| Bayes | HOL0+ATP0 | 0154 | all-vars-same | Epar |
| Bayes | HOL0+ATP0 | 0128 | standard | Epar |
| Bayes | HOL0+ATP0 | 0128 | standard | Vampire |
| Bayes | ATP1_E_pref | 0128 | standard | Z3 |
| Bayes | ATP0_V_pref | 0154 | standard | Vampire |
| 40-NN | ATP1 | 0032 | standard | Epar |
| 160-NN | ATP1 | 0512 | standard | Z3 |
| Bayes | HOL3+ATP3 | 0092 | standard | Vampire |
| Bayes | HOL3+ATP3 | 0128 | standard | Epar |
| Bayes | HOL3+ATP3 | 0154 | standard | Epar |
| Bayes | HOL3+ATP3 | 1024 | standard | Epar |
| Bayes | ATP3 | 0092 | standard | Vampire |
| Bayes | ATP3 | 0128 | standard | Epar |
| Bayes | ATP3 | 0154 | standard | Epar |
| Bayes | ATP3 | 1024 | standard | Epar |

Table 6.1.: The 25 AI/ATP combinations used by the initial Flyspeck service

- `TAUT`[13] — Propositional tautologies.
  `(A ==> B ==> C) ==> (A ==> B) ==> (A ==> C)`

- `INT_ARITH`[14] — Algebra and linear arithmetic over $\mathbb{Z}$ (including $\mathbb{R}$).
  `&2 * &1 = &2 + &0`

reason is that even though `REAL_ARITH` is used a lot in HOL Light formalizations, `INT_ARITH` is simply more powerful. It solves 60% more Flyspeck goals automatically without losing any of those solved by `REAL_ARITH`. As with the AI/ATP instances, the usage of decision procedures is optimized to jointly cover as many problems as possible.

[13] `http://www.cl.cam.ac.uk/~jrh13/hol-light/HTML/TAUT.html`

[14] `http://www.cl.cam.ac.uk/~jrh13/hol-light/HTML/INT_ARITH.html`

- `COMPLEX_FIELD`[15] — Field tactic over ℂ (including multivariate ℝ).
  `(Cx (&1) + Cx(&1)) = Cx(&2)`

Additionally the decision procedure infrastructure can be used to try common tactics that could solve the goal. One that we found especially useful is simplification with arithmetic (`SIMP_TAC[ARITH]`), which solves a number of simple numerical goals that the service users ask the server.

### 6.2.5. Proof Minimization and Reconstruction

When an ATP finds (and reports in its proof) a subset of the advised premises that prove the goal, it is often the case that this set is not minimal. By re-running the prover and other provers with only this set of proof-relevant premises, it is often possible to obtain a proof that uses fewer premises. A common example are redundant equalities that may be used by the ATP for early (but unnecessary) rewriting in the presence of many premises, and avoided when the number of premises is significantly lower (and different ordering is then used, or a completely different strategy or ATP might find a very different proof). This procedure is run recursively, until the number of premises needed for the proof no longer decreases. We call this recursive procedure *pseudo/cross-minimization*, since it is not exhaustive and uses multiple ATPs. Minimizing the number of premises improves the chances of the HOL proof reconstruction, and the speed of (re-)processing large libraries that contain many such reconstruction tactics.[16]

Given the minimized list of advised premises, we try to reconstruct the proof. As mentioned in Section 6.2.1, the advice system may internally use a number of theorem names (now mostly produced by splitting conjunctions) not present in standard HOL Light developments. It is possible to call the reconstruction tactics with the names used internally in the advice system; however this would create proof scripts that are not compatible with the original developments. We could directly address the theorem sub-conjuncts (using, e.g., "`nth (CONJUNCTS thm) n`") however such proof scripts look quite unnatural (even if they are indeed faster to process by HOL Light). Instead, we now prefer to use the whole original theorems (including all conjuncts) in the reconstruction.

Three basic strategies are now tried to reconstruct the proof: `REWRITE`[17] (rewriting), `SIMP`[18] (conditional rewriting) and `MESON` [Har96c] (internal first-order ATP). These three strategies are started in parallel, each with the list of HOL theorems that correspond to the minimized list of ATP premises as explained above. The strongest of these tactics – `MESON` – can in one second reconstruct 79.3% of the minimized ATP proofs. While this is certainly useful, the performance of `MESON` reconstruction drops below 40% as soon as the

[15] `http://www.cl.cam.ac.uk/~jrh13/hol-light/HTML/REAL_FIELD.html`

[16] Premise minimization has been for long time used to improve the quality and refactoring speed of the Mizar articles. It is now also a standard part of Sledgehammer.

[17] `http://www.cl.cam.ac.uk/~jrh13/hol-light/HTML/REWRITE_TAC.html`

[18] `http://www.cl.cam.ac.uk/~jrh13/hol-light/HTML/SIMP_TAC.html`

ATP proof uses at least seven premises. Since the service is getting stronger and stronger, the ratio of MESON-reconstructable proofs is likely to get lower and lower. That is why we have developed also a fine-grained reconstruction method – HH_RECON [KU13e], which uses the quite detailed TPTP proofs produced by Vampire and E. This method however still needs an additional mechanism that maintains the TPTP proof as part of the user development: either dedicated storage, or on-demand ATP-recreation, or translation to a corresponding fine-grained HOL Light proof script. That is why HH_RECON is not yet included by default in the service.

### 6.2.6. Description of the Parallelization Infrastructure

An important aspect of the online service is its parallelization capability. This is needed to efficiently process multiple requests coming in from the clients, and to execute the large number of AI/ATP instances in parallel within a short overall wall-clock time limit. HOL Light uses a number of imperative features of OCaml, such as static lists of constants and axioms, and a number of references (mutable variables). Also a number of procedures that are needed use shared references internally. For example the MESON procedure uses list references for variables. This makes HOL Light not thread safe. Instead of spending lots of time on a thread-safe re-implementation, the service just (in a pragmatic and simple way, similar to the Mizar parallelization [Urb12]) uses separate processes (Unix fork), which is sufficient for our purposes. Given a list of HOL Light tasks that should be performed in parallel and a timeout, the managing process spawns a child process for each of the tasks. It also creates a pipe for communicating with each child process. Progress, failures or completion information are sent over the pipe using OCaml marshalling. This means that it is enough to have running just one managing instance of HOL Light loaded with Flyspeck and with the advising infrastructure. This process forks itself for each client query, and the child then spawns as many AI/ATP, minimization, reconstruction, and decision procedure instances as needed.

### 6.2.7. Use of Caching

Even though the service can asynchronously process a number of parallel requests, it is not immune to overloading by a large number of requests coming in simultaneously. In such cases, each response gets less CPU time and the requests are less likely to succeed within the 30 seconds of wall-clock time. Such overloading is especially common for requests generated automatically. For example the Wiki service that is being built for Flyspeck [TKUG13] may ask many queries practically simultaneously when an article in the wiki is re-factored, but many of such queries will in practice overlap with previously asked queries. Caching is therefore employed by the service to efficiently serve such repeated requests.

Since the parallel architecture uses different processes to serve different requests, a filesystem based cache is used (using file-level locking). For any incoming request the first job done by the forked process handling the request is to check whether an identical request has already been served, and if so, the process just re-sends the previously computed answer. If the request is not found in the cache, a new entry (file) for it is created, and any information sent to the client (apart from the progress information) is also written to the cache entry. This means that all kinds of answers that have been sent to the client can be cached, including information about terms that failed to parse or typecheck, terms solved by ATP only, minimization results and replaying results, including decision procedures. The cache stored in the filesystem has the additional advantage of persistence, and in case of updating the service the cache can be easily invalidated by simply removing the cache entries.

## 6.3. Multiple Projects, Versions, and Their Online Update

The functions described in Section 6.2 allowed the users to experiment with ATP queries over the fixed June 2012 version of Flyspeck. If Flyspeck already contained all of human mathematics in a form that is universally agreed upon, such setting would be sufficient. However, Flyspeck is not the only library developed with HOL Light, and Flyspeck itself has been updated considerably since June 2012 with a number of new definitions, theorems and proofs. In general, there is no final word on how formal mathematics should be done, and even more stable formalization libraries may be updated, refactored, and forked for new experiments.

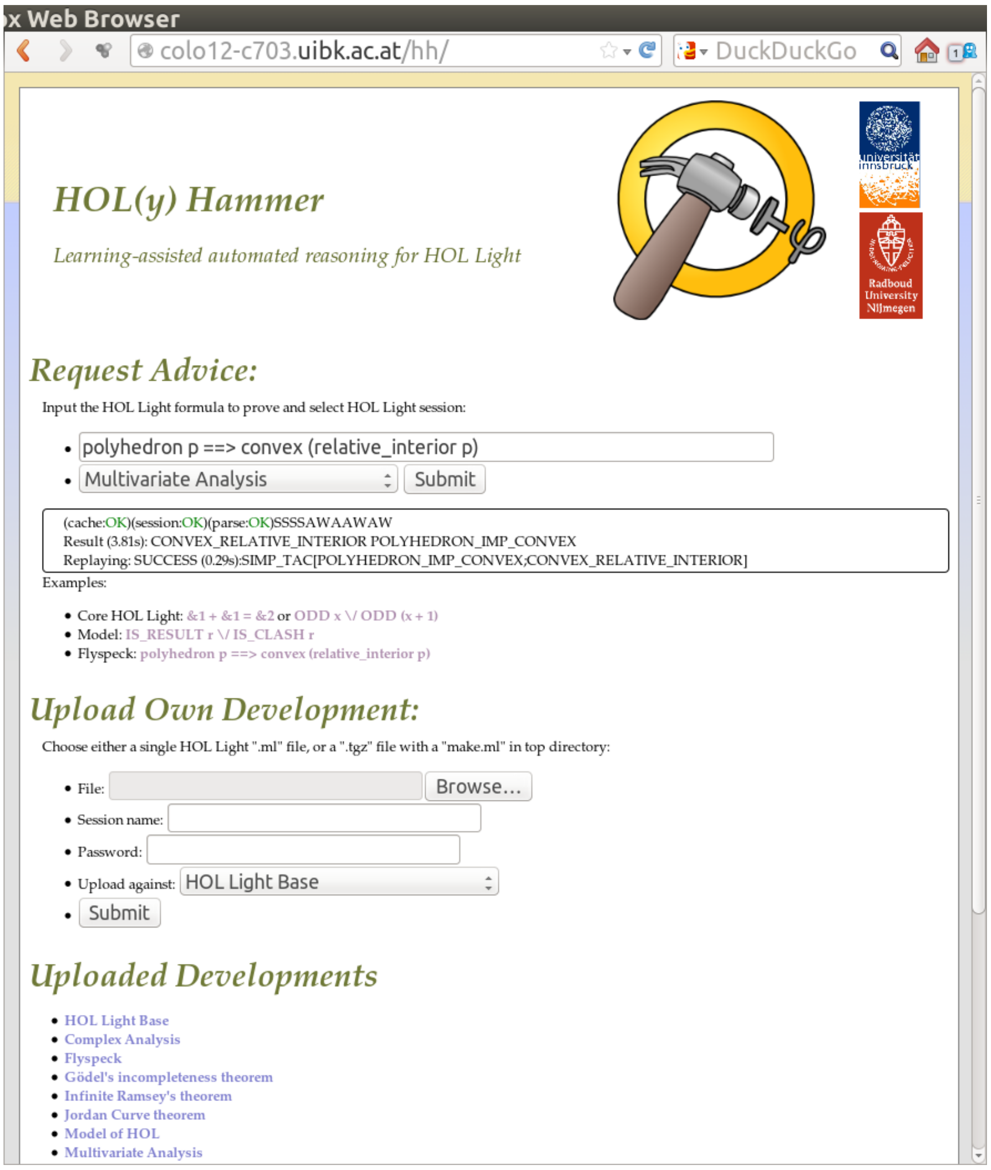
To support this, the current version of HOL(y)Hammer also allows online addition of new projects and updating of existing projects (see Figure 6.2). This leads to a number of issues that are discussed in this section. A particularly interesting and important issue is the transfer and re-use of the expensively obtained problem-solving knowledge between the different projects and their versions.

Another major issue is the speed of loading large projects. *Checkpointing* of OCaml instances is used to save the load time, after HOL Light was bootstrapped. Checkpointing software allows the state of a process to be written to disk, and restore this state from the stored image later. We use DMTCP[19] as our checkpointing software: it does not require kernel modifications, and because of that it is one of the few checkpointing solutions that work on recent Linux versions.

### 6.3.1. Basic Server Infrastructure for Multiple Projects

Instead of just one default project, the server allows multiple projects identified by a unique project name such as “Ramsey”, “Flyspeck” and “Multivariate Analysis”. A new project can be started by an authorized user performing a password-protected upload of

[19]`http://dmtcp.sourceforge.net`



HOL(y) Hammer

Learning-assisted automated reasoning for HOL Light

Request Advice:

Input the HOL Light formula to prove and select HOL Light session:

- polyhedron p ==> convex (relative_interior p)
- Multivariate Analysis | Submit

```
(cache:OK)(session:OK)(parse:OK)SSSSAWAAWAW
Result (3.81s): CONVEX_RELATIVE_INTERIOR POLYHEDRON_IMP_CONVEX
Replaying: SUCCESS (0.29s):SIMP_TAC[POLYHEDRON_IMP_CONVEX;CONVEX_RELATIVE_INTERIOR]
```

Examples:

- Core HOL Light: &1 + &1 = &2 or ODD x \/ ODD (x + 1)
- Model: IS_RESULT r \/ IS_CLASH r
- Flyspeck: polyhedron p ==> convex (relative_interior p)

Upload Own Development:

Choose either a single HOL Light ".ml" file, or a ".tgz" file with a "make.ml" in top directory:

- File: | Browse...
- Session name:
- Password:
- Upload against: HOL Light Base
- Submit

Uploaded Developments

- HOL Light Base
- Complex Analysis
- Flyspeck
- Gödel's incompleteness theorem
- Infinite Ramsey's theorem
- Jordan Curve theorem
- Model of HOL
- Multivariate Analysis

Figure 6.2.: The HOL(y)Hammer web with a query over Multivariate Analysis

the project files via a HTTP POST request. In the same way, an existing project can be updated.[20] The server data specific for each project are kept in its separate directory, which includes the user files, checkpointed images, features and proof dependencies used for learning premise selection, and the heuristically HTML-ized (hyperlinked) version of the user files. An overview of these project-specific data is given in Table 6.2.

| Data | Description |
|---|---|
| User files | User-submitted ML files. These data are additionally Git-managed in this directory. |
| Image1 | Checkpointed HOL Light image preloaded with the user files and the HH functions. |
| Image2 | An analogous image that uses proof recording to extract HOL proof dependencies. |
| Features | Several (currently six) feature characterizations (see Section 6.2.1) of the project's theorems. |
| HOL deps | The theorem dependencies from the original HOL proofs obtained by running the modified proof recording kernel on the user files. |
| ATP deps | The theorem dependencies obtained by running ATPs in various ways and minimizing such proofs. These data may be expensive to obtain, see 6.3.3 for the current re-use mechanisms. |
| Cache | The request cache for the project. |
| Auxiliary | Auxiliary files useful for bookkeeping and debugging. |
| HTML | Heuristically HTML-ized version of the user files, together with index pages for the files and theorems. These files are available for browsing and they are also linked to the Gitweb web interface, which presents the project and file history, allows comparison of different versions, regular expression search over the versions, etc. |

Table 6.2.: The data maintained for each HOL(y)Hammer project.

Apart from the project-specific files, the service also keeps a spare checkpointed core HOL Light image and additional files that typically contain the reusable information from various projects. The core HOL Light image is used for faster creation of images for new projects. A new project can also be cloned from an existing project. In that case, instead of starting with the core HOL Light image, the new project starts with the cloned project's image, and loads the new user files into them. This saves great amount of time when updating large projects like Flyspeck. The server processing of a new or modified project is triggered by the appropriate HTTP POST request. This starts the internal project creator which performs on the server the actions described by Algorithm 1. The

[20]Git-based interface to the projects already exists and will probably also be used for updating the projects with the standard `git-push` command from users' computers. This still requires installation of the Gitolite authentication layer on our server and implementing appropriate Git hooks similar to those developed for the Mizar wiki in [ABMU11].

Algorithm 1: Project creation stages

```
 1: Set up the directory structure for new projects.
 2: Open a copy of the checkpointed core HOL Light image (or another project's cloned image)
    and load it with the user files and the HOL(y)Hammer functions.
 3: Export the typed and variable-normalized statements of named theorems together with their
    MD5 hashes.
 4: Export the various feature characterizations of the theorems.
 5: Checkpoint the new image.
 6: Re-process the user files with a proof-recording kernel that saves the (new) HOL proof de-
    pendencies.
 7: Checkpoint the proof-recording image.
 8: Add further compatible proof dependencies from related projects.
 9: Run ATPs on the problems corresponding to the HOL dependencies, and minimize such
    proof data by running the ATPs further.
10: Run the heuristic HTML-izer and indexer, and push the user files to Git.
```

data sizes and processing times for seven existing projects are summarized in Table 6.3 and Table 6.4.

### 6.3.2. Safety

Since HOL Light is implemented in the OCaml toplevel, allowing users to upload their own development is equivalent to letting them run arbitrary programs on our server.[21] This is inherently insecure. A brief analysis of the related security issues and their possible countermeasures has been done in the context of the WorkingWiki [Wor11] collaborative platform.[22] The easiest practical solution is to allow uploads only by authorized users, i.e., users who are sufficiently trusted to be given shell access. Asking queries to existing projects can still be done by anybody; the query is then just a string processed by a time-limited function that always exits.

We have also briefly considered sandboxing for allowing anonymous user uploads, however it adds a significant overhead to managing the server (HOL(y)Hammer currently runs in user mode), while offering little protection in the case of HOL Light. Combination of chroot jail, an iptables firewall, and disallowing users to write files, has been previously used by us in ProofWeb for multiple proof assistants [Kal07]. This offers a sufficient level of security for a number of proof assistants where the ML access can be disabled, but it is not sufficient for HOL Light. Therefore also in ProofWeb, running HOL Light was restricted to the users that are allowed to use a shell on the server [HKvRW10].

[21] And indeed, the basic infrastructure could be also used as a platform for interacting with any OCaml project.

[22] `http://lalashan.mcmaster.ca/theobio/projects/index.php/WorkingWiki/Security`

| | Core | Ramsey | Model | Gödel | Complex | Multivariate | Flyspeck |
|---|---|---|---|---|---|---|---|
| Proof checking (min) | 3 | 6 | 193 | 166 | 267 | 2716 | 21735 |
| Proof recording (min) | 10 | 14 | 225 | 215 | 578 | 3751 | 52002 |
| Writing data | 26 | 27 | 33 | 47 | 53 | 139 | 758 |
| Writing ATP problems | 38.56 | 45.35 | 51.14 | 73.37 | 72.12 | 139.12 | 650.15 |
| Solving ATP problems | 1582.8 | 1622.4 | 1882.2 | 2173.8 | 2284.8 | 9286.2 | 12034.2 |
| HTML and Git | 4 | 2 | 2 | 3 | 2 | 19 | 61 |
| Image Restart | 1.98 | 2.08 | 2.37 | 2.15 | 3.00 | 3.66 | 6.78 |

Table 6.3.: The processing times for seven HOL(y)Hammer projects in seconds.

| | Core | Ramsey | Model | Gödel | Complex | Multivariate | Flyspeck |
|---|---|---|---|---|---|---|---|
| Normal image size (kB) | 33892 | 40952 | 37584 | 38244 | 55424 | 77292 | 152460 |
| Recording image size (kB) | 50960 | 52692 | 48148 | 46000 | 58368 | 247848 | 365496 |
| Unique theorems | 2482 | 2544 | 2951 | 3408 | 3582 | 6798 | 22336 |
| Unique constants | 234 | 234 | 337 | 367 | 333 | 466 | 1765 |
| Avrg. HOL proof deps. | 12.13 | 12.27 | 11.09 | 14.44 | 17.96 | 12.26 | 21.86 |
| ATP-proved theorems | 1546 | 1578 | 1714 | 1830 | 2042 | 4126 | 8907 |
| Usable ATP proofs | 6094 | 6141 | 6419 | 6644 | 6885 | 11408 | 21733 |
| Avrg. ATP proof deps. | 6.86 | 6.86 | 6.77 | 6.67 | 6.94 | 6.36 | 6.52 |
| Total distinct features | 3735 | 3759 | 4693 | 5755 | 5964 | 11599 | 43858 |
| Avrg. features/formula | 24.81 | 24.61 | 26.05 | 35.61 | 39.05 | 38.15 | 67.61 |

**Usable ATP proofs** Vampire, Epar and Z3 are used, and we keep all the different minimal proofs. This means that the total number of ATP proofs can be higher than the number of theorems.

Table 6.4.: The data sizes for seven HOL(y)Hammer projects.

### 6.3.3. Re-use of Knowledge from Related Projects

It has been shown in [KU14] that learning premise selection from minimized ATP proofs is better than learning from the HOL proofs, and also that the two approaches can be productively combined, resulting in further improvement of the overall ATP performance. However, obtaining the data from ATP runs is expensive. For example, just running Vampire, Epar and Z3 on all Flyspeck problems for 30 seconds takes (assuming 70% unsolved problems for each ATP) about 500 CPU hours. Even with 50-fold parallelization, this takes 10 hours of wall-clock time. And this is just the initial ATP pass. In [KU14] we also show that further MaLARea-style learning from such ATP data and re-running of the ATPs with the premises proposed by the learning grows the set of ATP solutions by about 20%. Obviously, such additional passes cost a lot of further CPU time. One option is to sacrifice the ATP data for speed, and only learn from the HOL data, sacrificing the final ATP performance on the queries. However, there is a relatively efficient way how to re-use a lot of the expensive data that were already computed.

Suppose that the user only updates an existing large project by adding a new file. Then it is quite sufficient to (relatively quickly) obtain the minimized ATP proofs of the (ATP-provable) theorems in the file that was added. Such ATP proofs are then added to the existing training data used for the premise selectors. In general, the project can however be modified and updated in a more complicated way, for example by adding/changing some files "in the middle", modifying symbol definitions, theorems, etc. Or it can be a completely new project, that only shares some parts with other projects, restructuring some terminology, theorem names, and proofs. The method that we use to handle such cases efficiently is *recursive content-based encoding* of the theorem and symbol names [Urb11b]. This is the first practical deployment and evaluation of this method, which in HOL(y)Hammer is done as follows:

1. The name of every defined symbol is replaced by the content hash (we use MD5) of its variable-normalized definition containing the full types of the variables. This definition already uses content hashes instead of the previously defined symbols. This means that symbol names are no longer relevant in the whole project, neither white space and variable names.

2. The name of each theorem is also replaced by the content hash of its (analogously normalized) statement.

3. The proof-dependency data extracted in the content encoding from all projects are copied to a special "common" directory.

4. Whenever a project $P$ is started or modified, we find the intersection of the content-encoded names of the project's theorems with such names that already exist in other projects/versions.

5. For each of such “already known” theorems $T$ in $P$, we re-use all its “already known” proofs $D$ that are *compatible* with $P$’s proof graph. This means, that the names of the proof dependencies of $T$ in $D$ must also exist in $P$ (i.e., these theorems have been also proved in $P$, modulo the content-naming), and that these theorems precede $T$ in $P$ in its chronological order (otherwise we might get cyclic training data for $P$).

There are two possible dangers with this approach: collisions in MD5 and dealing with types in the HOL logic. The first issue is theoretical: the chance of unintended MD5 collisions is very low, and if necessary, we can switch to stronger hashes such as SHA-256. The second issue is more real: there is a choice of using content-encoding also for the HOL types, or just using their original names. If original names are used, two differently defined types can get the same name in two different projects, making the theorems about such types incompatible. If content encoding is used, all types with the same definition will get the same content name. However, the HOL logic rejects such semantic equality of the two types already in its parsing layer: two differently named types are always completely different in the HOL logic.[23] We currently use the first method (keeping the original type names), however the second method might be slightly more correct. In both cases, it probably would not be hard to add guards against the possible conflicts. In all cases, these issues only influence the performance of the premise-selection algorithms. The theorem proving (and proof reconstruction) is always done with the original symbols.

### 6.3.4. Analysis of the Knowledge Re-use for Flyspeck Versions

It is interesting to know how much knowledge re-use can be obtained with the content-encoding method. We analyze this in Table 6.5 on the theorems (or rather unique conjuncts) coming from three different Flyspeck SVN versions: 2887, 3006, and 3400. Note that the last version (3400) has not been subjected to several learning/ATP passes. Such passes raised the number of ATP-proved theorems in the earlier versions by about 20%. The table shows that the number of reusable theorems and proofs from the previous version is typically very high. This also means that more expensive AI/ATP computations (e.g., use of higher time limits, MaLARea-style looping, and even BliStr-style strategy evolution [Urb14]) could be in the future added to the tasks done on the server in its idle time, because the results of such computations will typically improve the success rates of all the future versions of such large projects.

A by-product of the content encoding is also information about symbols that are defined multiple times under different names. For the latest version of Flyspeck there are 39 of them, shown in Table 6.6.

[23]The second author could not resist pointing out that this issue disappears in set theory with soft types.

| Version | Unique thms | In prev. (%) | ATP-proved (%) | ATP proofs | Reusable (%) |
|---|---|---|---|---|---|
| 2887 | 13647 | N/A | 7176 (53%) | 20028 | N/A |
| 3006 | 13814 | 13480 (98%) | 7235 (52%) | 20081 | 19977 (99%) |
| 3400 | 18856 | 12866 (93%) | 8914 (47%) | 21780 | 21320 (97%) |

**In previous** Theorems (conjuncts) that exist already in the previous version, and their percentage.

**ATP proof** Vampire, Epar and Z3 are used, and we keep all the different minimal proofs. This means that the total number of ATP proofs can be higher than the number of theorems.

**Re-usable ATP proofs** The proofs from the previous version that are valid also in the current version.

Table 6.5.: The re-use of theorems and ATP proofs between four Flyspeck SVN versions

| | |
|---|---|
| face_path / face_contour | reflect_along / reflection |
| zero6 / dummy6 | UNIV / predT |
| CROSS / *_c | node3_y / rotate3 |
| EMPTY / pred0 | APPEND / cat |
| func / FUN | set_components / set_part_components |
| ONE_ONE / injective | triple_of_real3 / vector_to_pair |
| supp / SUPP | is_no_double_joins / is_no_double_joints |
| dirac_delta / delta_func | unknown / NONLIN |
| o / compose | node2_y / rotate2 |
| I / LET_END / mark_term | |

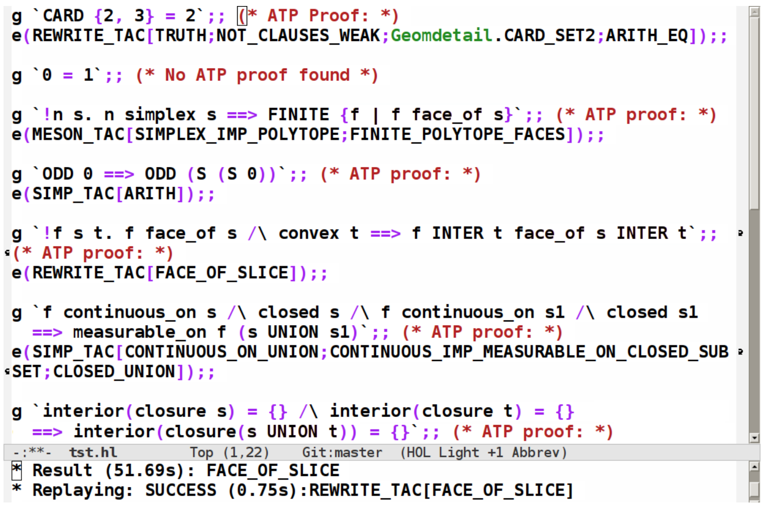
Table 6.6.: 39 symbols with the same content-based definition in Flyspeck SVN 3400

## 6.4. Modes of Interaction with the Service

The standard web interface (Figure 6.2) displays the available projects, links to their documentation, allows queries to the projects, and provides an HTML form for uploading and modifying projects. Requests are processed using asynchronous DOM modification (AJAX): a JavaScript script makes the requests in the background and updates a part of the page that displays the response. Each request is first sent to the external PHP request processor, which communicates with the HOL(y)Hammer server. A prototype of a web editor interacting both with HOL Light and with the online advisor is described in [TKUG13].

Figure 6.3 shows an Emacs session with several HOL Light goals.[24] The online advisor has been asynchronously called on the goals, and just returned the answer for the fifth goal and inserted the corresponding tactic call at an appropriate place in the buffer. The

[24]A longer video of the interaction is at http://mws.cs.ru.nl/~urban/ha1.mp4



```
g `CARD {2, 3} = 2`;; (* ATP Proof: *)
e(REWRITE_TAC[TRUTH;NOT_CLAUSES_WEAK;Geomdetail.CARD_SET2;ARITH_EQ]);;

g `0 = 1`;; (* No ATP proof found *)

g `!n s. n simplex s ==> FINITE {f | f face_of s}`;; (* ATP proof: *)
e(MESON_TAC[SIMPLEX_IMP_POLYTOPE;FINITE_POLYTOPE_FACES]);;

g `ODD 0 ==> ODD (S (S 0))`;; (* ATP proof: *)
e(SIMP_TAC[ARITH]);;

g `!f s t. f face_of s /\ convex t ==> f INTER t face_of s INTER t`;;
(* ATP proof: *)
e(REWRITE_TAC[FACE_OF_SLICE]);;

g `f continuous_on s /\ closed s /\ f continuous_on s1 /\ closed s1
  ==> measurable_on f (s UNION s1)`;; (* ATP proof: *)
e(SIMP_TAC[CONTINUOUS_ON_UNION;CONTINUOUS_IMP_MEASURABLE_ON_CLOSED_SUB
SET;CLOSED_UNION]);;

g `interior(closure s) = {} /\ interior(closure t) = {}
  ==> interior(closure(s UNION t)) = {}`;; (* ATP proof: *)
-:**-  tst.hl         Top (1,22)    Git:master   (HOL Light +1 Abbrev)
* Result (51.69s): FACE_OF_SLICE
* Replaying: SUCCESS (0.75s):REWRITE_TAC[FACE_OF_SLICE]
```

Figure 6.3.: Parallel asynchronous calls of the online advisor from Emacs.

relevant Emacs code (customized for the HOL Light mode distributed with Flyspeck) is available online[25] and also distributed with the local HOL(y)Hammer install. It is a modification of the similar code used for communicating with the MizAR service from Emacs.

The simplest option (useful as a basis for more sophisticated interfaces) is to interact with the service in command line, for example using netcat, as shown for two following two queries. The first query is solved easily by `INT_ARITH`, while the other requires nontrivial premise and proof search.

```
$ echo 'max a b = &1 / &2 * ((a + b) + abs(a - b))'
  | nc colo12-c703.uibk.ac.at 8080
* Replaying: SUCCESS (0.25s): INT_ARITH_TAC
* Loadavg: 48.13 48.76 48.49 52/1151 46604

$ echo '!A B (C:A->bool).((A DIFF B) INTER C=EMPTY) <=> ((A INTER C) SUBSET B)'
  | nc colo12-c703.uibk.ac.at 8080
 * Read OK
 * Theorem! Time: 14.74s Prover: Z Hints: 32 Str:
   allt_notrivsyms_m10u_all_atponly
 * Minimizing, current no: 9
 * Minimizing, current no: 6
 * Result: EMPTY_SUBSET IN_DIFF IN_INTER MEMBER_NOT_EMPTY SUBSET SUBSET_ANTISYM
```

[25] https://raw.github.com/JUrban/hol-advisor/master/hol-advice.el

## 6.5. The Local Service Description

The service can be also downloaded,[26] installed and used locally, for example when a user is working on a private formalization that cannot be included in the public online service.[27] Installing the advisor locally proceeds analogously to the steps described in Algorithm 1. Two passes are done through the user's repository. In the first pass, the names of all the theorems available in the user's repository are exported, together with their features (symbols, terms, types, etc., as explained in Section 6.2.1). In the second pass, the dependencies between the named theorems are computed, again using the modified proof recording HOL Light kernel that records all the processing steps. Given the exported features and dependencies, local advice system(s) (premise selectors) are trained outside HOL Light. Using the fast sparse learning methods described in Section 6.2.1, this again takes seconds, depending on the user hardware and the size of the development. The advisors are then run locally (as independent servers) to serve the requests coming from HOL Light. While the first pass is just a fast additional function that can be run by the user at any time on top of his loaded repository, the second pass now still requires full additional processing of the repository. This could be improved in the future by checkpointing the proof-recording image, as we do in the online server.

The user is provided with a tactic (`HH_ADVICE_TAC`) which runs all the mechanisms described in the Section 6.2 on the current goal locally. This means that the functions relying on external premise selection and ATPs are tried in parallel, together with a number of decision procedures. The ATPs are expected to be installed on the user's machine and (as in the online service) they are run on the goal translated to the TPTP format, together with a limited number of premises optimized separately for each prover. By default Vampire, Eprover and Z3 are now run, using three-fold parallelization.

The local installation in its simple configuration is now only trained using the naive Bayes algorithm on the training data coming from the HOL Light proof dependencies and the features extracted with the standard method. As shown in [KU14], the machine learning advice can be strengthened using ATP dependencies, which can be also optionally plugged into the local mode. Further strengthening can be done with combinations of various methods. This is easy to adjust; for example a user with a 24-CPU workstation can re-use/optimize the parallel combinations from Table 6.1 used by the online service.

### 6.5.1. Online versus Local Systems

The two related existing services are Miz$\mathbb{AR}$ and Sledgehammer. Miz$\mathbb{AR}$ has so far been an online service (accessible via Emacs or web interface), while Sledgehammer has so far required a local install (even though it already calls some ATPs over a network).

[26]`http://cl-informatik.uibk.ac.at/users/cek/hh/`

[27]The online service can already handle private developments that are not shown to the public.

HOL(y)Hammer started as an online service, and the local version has been added recently to answer the demand by some (power)users. The arguments for installing the service locally are mainly the option to use the service offline (possibly using one's own large computing resources), and to keep the development private. As usual, the local install will also require the tools involved to work on all kinds of architectures, which is often an issue, particularly with software that is mostly developed in academia.

As described in Section 6.2, the online service now runs 7 different AI/ATP instances and 4 decision procedures for each query. When counting the individual ATP strategies (which may indeed be very orthogonal in systems like Vampire and E), this translates to about 70 different AI/ATP attempts for each query. If the demands grows, we can already now distribute the load from the current 48-CPU server to 112 CPUs by installing the service on another 64-CPU server. The old resolution-ATP wisdom is that systems rarely prove a result in higher time limits, since the search space grows very fast. A more recent wisdom (most prominently demonstrated by Vampire) however is that using (sufficiently orthogonal) strategy scheduling makes higher time limits much more useful.[28] And even more recent wisdom is that learning in various ways from related successes and failures further improves the systems' chances when given more resources. All this makes a good case for developing strong online computing services that can in short bursts focus a lot of power on the user queries, which are typically related to many previous problems. Also in some sense, the currently used AI/ATP methods are only scratching the surface. For example, further predictive power is obtained in MaLARea [USPV08] by computing thousands of interesting finite models, and using evaluation in them as additional semantic features of the formulas. ATP prototypes like MaLeCoP [UVŠ11] can already benefit from accumulated fine-grained learned AI guidance at every inference step that they make. The service can try to make the best (re-)use of all smaller lemmas that have been proved so far (as in [KU13c, Urb06b]). And as usual in machine learning, the more data are centrally accumulated for such methods, the stronger the methods become. Finally, it is hard to overlook the recent trend of light-weight devices for which the hard computational tasks are computed by large server farms (cloud computing).

## 6.6. Conclusion and Future Work

We believe that HOL(y)Hammer is one of the strongest AI/ATP services currently available. It uses a toolchain of evolving large-theory methods that have been continuously improved as more and more AI/ATP experiments and computations have been recently done, in particular over the Flyspeck corpus. The combinations that jointly provide the greatest theorem-proving coverage are employed to answer the queries with parallelization of practically all of the components. The parallelization factor is probably the highest of all existing ATP services, helping to focus the power of many different AI/ATP

[28]In [KU14], the relative performance of Vampire in 30 and 900 seconds is very different.

methods to answer the queries as quickly as possible. The content-encoding mechanisms allow to re-use a lot of the expensive theorem-proving knowledge computed over earlier projects and versions. And the checkpointing allows reasonably fast update of existing projects.

At this moment, there seems to be no end to better premise selection, better translation methods for ATPs (and SMTs, and more advanced combined systems like MetiTarski [AP10]), better ATP methods (and their AI-based guidance), and better reconstruction methods. Future work also includes broader updating mechanisms, for example using git to not just add, but also delete files from an existing project. A major issue is securing the server for more open (perhaps eventually anonymous) uploads, and maybe also providing encryption/obfuscation mechanisms that guarantee privacy of the non-public developments.[29]

An interesting future direction is the use of the service with its large knowledge base and growing reasoning power as a semantic understanding (connecting) layer for experiments with tools that attempt to extract logical meaning from informal mathematical texts. Mathematics, with its explicit semantics, could in fact pioneer the technology of very deep parsing of scientific natural language writings, and their utilization in making stronger and stronger automated reasoning tools about all kinds of scientific domains.

[29]The re-use performance obtained through content encoding suggests that just name obfuscation done by the client is not going to work as a privacy method.

# 7 External Provers for HOL4

## Publication Details

Thibault Gauthier and Cezary Kaliszyk. Premise Selection and External Provers for HOL4. In Xavier Leroy and Alwen Tiu, editors. *Proc. 4th Conference on Certified Programs and Proofs (CPP'15)*, pp. 49–57, ACM, 2015. doi: `10.1145/2676724.2693173`.

## 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.

## 7.1. Introduction

The HOL4 proof assistant [SN08] 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 [Har96c]), tableau (blast [Pau99]), or resolution (Metis [Hur03]). Although essential to HOL4 developers, the methods are so far not able to compete with the external ATPs [Sch13, KV13] optimized for fast proof search with many axioms present and continuously evaluated on the TPTP library [Sut09] 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 [PB12] for Isabelle/HOL, HOL(y)Hammer [KU14] for HOL Light or Miz$\mathbb{AR}$ for Mizar [KU13d]. 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 [Urb07], which leads to a significant increase in the power of such systems [KBKU13].

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 7.1.1** *(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 [Web11], Z3 [BW10] and Beagle [GKKN14]. 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 reasoning and it can provide easily[1] re-provable problem to Metis.

[1]reconstruction rate is typically above 90%

The HOL4 standard distribution has since long been equipped with proof recording kernels [Won95, KH12]. 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 namespace 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 7.2 we describe the export of the HOL4 and HOL Light data into a common format and the recording of dependencies in HOL4. In Section 7.3, 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 7.4 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 7.5 we conclude and present an outlook on the future work.

## 7.2. Sharing HOL4, HOL Light and HOL(y)Hammer Data

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 [KU14] 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 [SB10] 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 7.2.1** *(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 7.2.2** *(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.

### 7.2.1. 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 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.

### 7.2.2. 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 7.2.3** *(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 7.2.4** *(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.

### 7.2.3. 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 7.2.5** *(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 7.2.6** *(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 7.2.7** *(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 7.2.8** *(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 | 1.8 | 128 |
| Z3 | 4.32 | 32 |
| CVC4 | 1.3 | 128 |
| Spass | 3.5 | 32 |
| IProver | 1.0 | 128 |
| Metis | 2.3 | 32 |

Table 7.1.: ATP provers, their versions and arguments

## 7.3. 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.

### 7.3.1. 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 [PS07a] and Harrison [Har96c]. It is described in detail in [KU14], here we remind only the crucial points. Abstractions are removed by $\beta$-reduction followed by $\lambda$-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 [BBPS13]: 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 7.1. The HOL Light experiments [KU14] 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 [HV11], 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 [Sut14]. For ITP originating problems, especially for E-prover different strategies are often better, so we run it under the alternate scheduler

`Epar` [Urb14].

### 7.3.2. 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 [FGT92].

**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 7.4.5.

**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.

### 7.3.3. 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 [USPV08].
- Variables are replaced by their (variable-normalized) types [KU14].

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.

### 7.3.4. Predictors

In all our experiments we have used the modified k-NN algorithm [KU13f]. This algorithm produces the most precise results in the HOL(y)Hammer experiments for HOL Light [KU14]. 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

## 7.4. 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).

### 7.4.1. 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 7.2 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 [Sch13], we also compare its different scheduling strategies [Urb14]. 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 [BCD+11] 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 [KV13], and Z3 [dMB08]. 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 7.3, we try to reprove conjuncts of these theorems with the different recording methods described in Section 7.2.3. First, we notice that only Z3 benefits from the tracking 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

| 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 7.2.: 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 7.3.: Success rates of reproving (%) on the 13910 conjuncts of the standard library with different dependency tracking mechanism.

did not increase compared to Table 7.2, 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 [KU14], 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 7.4 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`.

### 7.4.2. Different Accessible Sets

In Table 7.5 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 7.3 for easier comparison.

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 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

| | 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 7.4.: 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 7.5.: Percentage (%) of proofs found using different accessible sets: exact dependencies (ED), transitive dependencies (TD), loaded theories (LT), and linear order (LO)

the HOL4 libraries, which makes it harder for automated reasoning techniques.

### 7.4.3. 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.

### 7.4.4. 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 7.4.1** *(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 7.4.2** *(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
```

### 7.4.5. 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 7.6, 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 7.6.: Time (in seconds) taken by each step of the advice loop

## 7.5. 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. Since 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`.

### 7.5.1. 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 [GK14]. 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 [SB10] and polymorphic first-order logic [BP13]. 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 [KU15], 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.

# 8 Machine Learning for Sledgehammer

## Publication Details

Daniel Kühlwein, Jasmin C. Blanchette, Cezary Kaliszyk, and Josef Urban. MaSh: Machine Learning for Sledgehammer. In Proc. of the 4th *International Conference on Interactive Theorem Proving (ITP'13)*, volume 7998 of LNCS, pp. 35–50. Springer, 2013. doi: `10.1007/978-3-642-39634-2_6`

## Abstract

Sledgehammer integrates automatic theorem provers in the proof assistant Isabelle/HOL. A key component, the relevance filter, heuristically ranks the thousands of facts available and selects a subset, based on syntactic similarity to the current goal. We introduce MaSh, an alternative that learns from successful proofs. New challenges arose from our "zero-click" vision: MaSh should integrate seamlessly with the users' workflow, so that they benefit from machine learning without having to install software, set up servers, or guide the learning. The underlying machinery draws on recent research in the context of Mizar and HOL Light, with a number of enhancements. MaSh outperforms the old relevance filter on large formalizations, and a particularly strong filter is obtained by combining the two filters.

*In memoriam Piotr Rudnicki 1951–2012*

## 8.1. Introduction

Sledgehammer [PB12] is a subsystem of the proof assistant Isabelle/HOL [NPW02] that discharges interactive goals by harnessing external automatic theorem provers (ATPs). It heuristically selects a number of relevant facts (axioms, definitions, or lemmas) from the thousands available in background libraries and the user's formalization, translates the problem to the external provers' logics, and reconstructs any machine-found proof in Isabelle (Sect. 8.2). The tool is popular with both novices and experts.

Various aspects of Sledgehammer have been improved since its introduction, notably the addition of SMT solvers [BBP11], the use of sound translation schemes [BBPS13], close cooperation with the first-order superposition prover SPASS [BPWW12], and of course advances in the underlying provers themselves. Together, these enhancements increased the success rate from 48% to 64% on the representative "Judgment Day" benchmark suite [BN10, BPWW12].

One key component that has received little attention is the relevance filter. Meng and Paulson [MP09] designed a filter, MePo, that iteratively ranks and selects facts similar to the current goal, based on the symbols they contain. Despite its simplicity, and despite advances in prover technology [HV11, BPWW12, Sch11], this filter greatly increases the success rate: Most provers cannot cope with tens of thousands of formulas, and translating so many formulas would also put a heavy burden on Sledgehammer. Moreover, the translation of Isabelle's higher-order constructs and types is optimized globally for a problem—smaller problems make more optimizations possible, which helps the automatic provers.

Coinciding with the development of Sledgehammer and MePo, a line of research has focused on applying machine learning to large-theory reasoning. Much of this work has been done on the vast Mizar Mathematical Library (MML) [MML], either in its original Mizar [MR05] formulation or in first-order form as the Mizar Problems for Theorem Proving (MPTP) [Urb06c]. The MaLARea system [Urb07, USPV08] and the competitions CASC LTB and Mizar@Turing [Sut13] have been important milestones. Recently, comparative studies involving MPTP [KvLT$^+$12, AHK$^+$14] and the Flyspeck project in HOL Light [KU13f] have found that fact selectors based on machine learning outperform symbol-based approaches.

Several learning-based advisors have been implemented and have made an impact on the automated reasoning community. In this paper, we describe a tool that aims to bring the fruits of this research to the Isabelle community. This tool, MaSh, offers a memoryful alternative to MePo by learning from successful proofs, whether human-written or machine-generated.

Sledgehammer is a one-click technology—fact selection, translation, and reconstruction are fully automatic. For MaSh, we had four main design goals:

- *Zero-configuration*: The tool should require no installation or configuration steps, even for use with unofficial repository versions of Isabelle.
- *Zero-click*: Existing users of Sledgehammer should benefit from machine learning, both for standard theories and for their custom developments, without having to change their workflow.
- *Zero-maintenance*: The tool should not add to the maintenance burden of Isabelle. In particular, it should not require maintaining a server or a database.
- *Zero-overhead*: Machine learning should incur no overhead to those Isabelle users who do not employ Sledgehammer.

By pursuing these "four zeros," we hope to reach as many users as possible and keep them for as long as possible. These goals have produced many new challenges.

MaSh's heart is a Python program that implements a custom version of a weighted sparse naive Bayes algorithm that is faster than the naive Bayes algorithm implemented in the SNoW [CCRR99] system used in previous studies (Sect. 8.3). The program maintains a persistent state and supports incremental, nonmonotonic updates. Although distributed with Isabelle, it is fully independent and could be used by other proof assistants, automatic theorem provers, or applications with similar requirements.

This Python program is used within a Standard ML module that integrates machine learning with Isabelle (Sect. 8.4). When Sledgehammer is invoked, it exports new facts and their proofs to the machine learner and queries it to obtain relevant facts. The main technical difficulty is to perform the learning in a fast and robust way without interfering with other activities of the proof assistant. Power users can enhance the learning by letting external provers run for hours on libraries, searching for simpler proofs.

A particularly strong filter, MeSh, is obtained by combining MePo and MaSh. The three filters are compared on large formalizations covering the traditional application areas of Isabelle: cryptography, programming languages, and mathematics (Sect. 8.5). These empirical results are complemented by Judgment Day, a benchmark suite that has tracked Sledgehammer's development since 2010. Performance varies greatly depending on the application area and on how much has been learned, but even with little learning MeSh emerges as a strong leader.

## 8.2. Sledgehammer and MePo

Whenever Sledgehammer is invoked on a goal, the MePo (*Me*ng–*P*auls*o*n) filter selects $n$ facts $\varphi_1, \ldots, \varphi_n$ from the thousands available, ordering them by decreasing estimated relevance. The filter keeps track of a set of relevant symbols—i.e., (higher-order) constants and fixed variables—initially consisting of all the goal's symbols. It performs the following steps iteratively, until $n$ facts have been selected:

1. Compute each fact's score, as roughly given by $r/(r+i)$, where $r$ is the number of relevant symbols and $i$ the number of irrelevant symbols occurring in the fact.

2. Select all facts with perfect scores as well as some of the remaining top-scoring facts, and add all their symbols to the set of relevant symbols.

The implementation refines this approach in several ways. Chained facts (inserted into the goal by means of the keywords `using`, `from`, `then`, `hence`, and `thus`) take absolute priority; local facts are preferred to global ones; first-order facts are preferred to higher-order ones; rare symbols are weighted more heavily than common ones; and so on.

MePo tends to perform best on goals that contain some rare symbols; if all the symbols are common, it discriminates poorly among the hundreds of facts that could be relevant. There is also the issue of starvation: The filter, with its iterative expansion of the set of relevant symbols, effectively performs a best-first search in a tree and may therefore ignore some relevant facts close to the tree's root.

The automatic provers are given prefixes $\varphi_1, \ldots, \varphi_m$ of the selected $n$ facts. The order of the facts—the estimated relevance—is exploited by some provers to guide the search. Although Sledgehammer's default time limit is 30 s, the automatic provers are invoked repeatedly for shorter time periods, with different options and different number of facts $m \leq n$; for example, SPASS is given as few as 50 facts in some slices and as many as 1000 in others. Excluding some facts restricts the search space, helping the prover find deeper proofs within the allotted time, but it also makes fewer proofs possible.

The supported ATP systems include the first-order provers E [Sch04], SPASS [BPWW12], and Vampire [RV02, KV13]; the SMT solvers CVC3 [BT07], Yices [DdM06], and Z3 [dMB08]; and the higher-order provers LEO-II [BPTF08] and Satallax [Bro12].

Once a proof is found, Sledgehammer minimizes it by invoking the prover repeatedly with subsets of the facts it refers to. The proof is then reconstructed in Isabelle by a suitable proof text, typically a call to the built-in resolution prover Metis [Hur03].

**Example 8.2.1** *Given the goal*

$$\mathsf{map}\ f\ xs = ys \implies \mathsf{zip}\ (\mathsf{rev}\ xs)\ (\mathsf{rev}\ ys) = \mathsf{rev}\ (\mathsf{zip}\ xs\ ys)$$

*MePo selects 1000 facts: rev_map, rev_rev_ident, ..., add_numeral_special(3). The prover E, among others, quickly finds a minimal proof involving the 4th and 16th facts:*

$$\begin{aligned} \textit{zip\_rev:}\ & \mathsf{length}\ xs = \mathsf{length}\ ys \implies \mathsf{zip}\ (\mathsf{rev}\ xs)\ (\mathsf{rev}\ ys) = \mathsf{rev}\ (\mathsf{zip}\ xs\ ys) \\ \textit{length\_map:}\ & \mathsf{length}\ (\mathsf{map}\ f\ xs) = \mathsf{length}\ xs \end{aligned}$$

**Example 8.2.2** *MePo's tendency to starve out useful facts is illustrated by the following goal, taken from Paulson's verification of cryptographic protocols [Pau98]:*

$$\mathsf{used}\ [] \subseteq \mathsf{used}\ evs$$

*A straightforward proof relies on these four lemmas:*

$$\begin{aligned} \textit{used\_Nil:}\ & \mathsf{used}\ [] = \textstyle\bigcup_B \mathsf{parts}\ (\mathsf{initState}\ B) \\ \textit{initState\_into\_used:}\ & X \in \mathsf{parts}\ (\mathsf{initState}\ B) \implies X \in \mathsf{used}\ evs \\ \textit{subsetI:}\ & (\textstyle\bigwedge x.\ x \in A \Longrightarrow x \in B) \implies A \subseteq B \\ \textit{UN\_iff:}\ & b \in \textstyle\bigcup_{x \in A} B\ x \longleftrightarrow \exists x \in A.\ b \in B\ x \end{aligned}$$

*The first two lemmas are ranked 6807th and 6808th, due to the many initially irrelevant constants ($\bigcup$, $\mathsf{parts}$, $\mathsf{initState}$, and $\in$). In contrast, all four lemmas appear among MaSh's first 45 facts and MeSh's first 77 facts.*

## 8.3. The Machine Learning Engine

MaSh (*Ma*chine Learning for *S*ledge*h*ammer) is a Python program for fact selection with machine learning.[1] Its default learning algorithm is an approximation of naive Bayes adapted to fact selection. MaSh can perform fast model updates, overwrite data points, and predict the relevance of each fact. The program can also use the much slower naive Bayes algorithm implemented by SNoW [CCRR99].

### 8.3.1. Basic Concepts

MaSh manipulates theorem proving concepts such as facts and proofs in an agnostic way, as "abstract nonsense":

- A *fact* $\varphi$ is a string.
- A *feature* $f$ is also a string. A positive *weight* $w$ is attached to each feature.
- *Visibility* is a partial order $\prec$ on facts. A fact $\varphi$ is visible *from* a fact $\chi$ if $\varphi \prec \chi$, and visible *through* the set of facts $\Phi$ if there exists a fact $\chi \in \Phi$ such that $\varphi \preceq \chi$.
- The *parents* of a fact are its (immediate) predecessors with respect to $\prec$.
- A *proof* $\Pi$ for $\varphi$ is a set of facts visible from $\varphi$.

Facts are described abstractly by their feature sets. The features may for example be the symbols occurring in a fact's statement. Machine learning proceeds from the hypothesis that facts with similar features are likely to have similar proofs.

### 8.3.2. Input and Output

MaSh starts by loading the persistent model (if any), executes a list of commands, and saves the resulting model on disk. The commands and their arguments are

```
learn fact parents features proof
relearn fact proof
query parents features hints
```

The `learn` command teaches MaSh a new fact $\varphi$ and its proof $\Pi$. The parents specify how to extend the visibility relation for $\varphi$, and the features describe $\varphi$. In addition to the supplied proof $\Pi \vdash \varphi$, MaSh learns the trivial proof $\varphi \vdash \varphi$; hence something is learned even if $\Pi = \emptyset$ (which can indicate that no suitable proof is available).

The `relearn` command forgets a fact's proof and learns a new one.

[1]The source code is distributed with Isabelle2013 in the directory `src/HOL/Tools/Sledgehammer/MaSh/src`.

The `query` command ranks all facts visible through the given parents by their predicted relevance with respect to the specified features. The optional hints are facts that guide the search. MaSh temporarily updates the model with the hints as a proof for the current goal before executing the query.

The commands have various preconditions. For example, for `learn`, $\varphi$ must be fresh, the parents must exist, and all facts in $\Pi$ must be visible through the parents.

### 8.3.3. The Learning Algorithm

MaSh's default machine learning algorithm is a weighted version of sparse naive Bayes. It ranks each visible fact $\varphi$ as follows. Consider a `query` command with the features $f_1, \ldots, f_n$ weighted $w_1, \ldots, w_n$, respectively. Let $P$ denote the number of proofs in which $\varphi$ occurs, and $p_j \leq P$ the number of such proofs associated with facts described by $f_j$ (among other features). Let $\pi$ and $\sigma$ be predefined weights for known and unknown features, respectively. The estimated relevance is given by

$$r(\varphi, f_1, \ldots, f_n) \;=\; \ln P \,+ \sum_{j\,:\,p_j \neq 0} w_j \big(\ln\,(\pi p_j) \;-\; \ln P\big) \,+ \sum_{j\,:\,p_j = 0} w_j\,\sigma$$

When a fact is learned, the values for $P$ and $p_j$ are initialized to a predefined weight $\tau$. The models depend only on the values of $P$, $p_j$, $\pi$, $\sigma$, and $\tau$, which are stored in dictionaries for fast access. Computing the relevance is faster than with standard naive Bayes because only the features that describe the current goal need to be considered, as opposed to all features (of which there may be tens of thousands). Experiments have found the values $\pi = 10$, $\sigma = -15$, and $\tau = 20$ suitable.

A crucial technical issue is to represent the visibility relation efficiently as part of the persistent state. Storing all the ancestors for each fact results in huge files that must be loaded and saved, and storing only the parents results in repeated traversals of long parentage chains to obtain all visible facts. MaSh solves this dilemma by complementing parentage with a cache that stores the ancestry of up to 100 recently looked-up facts. The cache not only speeds up the lookup for the cached facts but also helps shortcut the parentage chain traversal for their descendants.

## 8.4. Integration in Sledgehammer

Sledgehammer's MaSh-based relevance filter is implemented in Standard ML, like most of Isabelle.[2] It relies on MaSh to provide suggestions for relevant facts whenever the user invokes Sledgehammer on an interactive goal.

[2]The code is located in Isabelle2013's files `src/HOL/Tools/Sledgehammer/sledgehammer_mash.ML`, `src/HOL/TPTP/mash_export.ML`, and `src/HOL/TPTP/mash_eval.ML`.

### 8.4.1. The Low-Level Learner Interface

Communication with MaSh is encapsulated by four ML functions. The first function resets the persistent state; the last three invoke MaSh with a list of commands:

$$\begin{aligned}&\mathsf{MaSh.unlearn}\ ()\\&\mathsf{MaSh.learn}\ [(\mathit{fact}_1,\ \mathit{parents}_1,\ \mathit{features}_1,\ \mathit{proof}_1),\ \ldots,\\&\qquad\qquad(\mathit{fact}_n,\ \mathit{parents}_n,\ \mathit{features}_n,\ \mathit{proof}_n)]\\&\mathsf{MaSh.relearn}\ [(\mathit{fact}_1,\ \mathit{proof}_1),\ \ldots,\ (\mathit{fact}_n,\ \mathit{proof}_n)]\\&\mathit{suggestions} = \mathsf{MaSh.query}\ \mathit{parents}\ \mathit{features}\ \mathit{hints}\end{aligned}$$

To track what has been learned and avoid violating MaSh's preconditions, Sledgehammer maintains its own persistent state, mirrored in memory. This mainly consists of the visibility graph, a directed acyclic graph whose vertices are the facts known to MaSh and whose edges connect the facts to their parents. (MaSh itself maintains a visibility graph based on `learn` commands.) The state is accessed via three ML functions that use a lock to guard against race conditions in a multithreaded environment [Wen09] and keep the transient and persistent states synchronized.

### 8.4.2. Learning from and for Isabelle

Facts, features, proofs, and visibility were introduced in Sect. 8.3.1 as empty shells. The integration with Isabelle fills these concepts with content.

**Facts.** Communication with MaSh requires a string representation of Isabelle facts. Each theorem in Isabelle carries a stable "name hint" that is identical or very similar to its fully qualified user-visible name (e.g., *List.map.simps_2* vs. *List.map.simps*(2)). Top-level lemmas have unambiguous names. Local facts in a structured Isar proof [Wen07] are disambiguated by appending the fact's statement to its name.

**Features.** Machine learning operates not on the formulas directly but on sets of features. The simplest scheme is to encode each symbol occurring in a formula as its own feature. The experience with MePo is that other factors help—for example, the formula's types and type classes or the theory it belongs to. The MML and Flyspeck evaluations revealed that it is also helpful to preserve parts of the formula's structure, such as subterms [AKU12, KU13f].

Inspired by these precursors, we devised the following scheme. For each term in the formula, excluding the outer quantifiers, connectives, and equality, the features are derived from the nontrivial first-order patterns up to a given depth. Variables are replaced by the wildcard _ (underscore). Given a maximum depth of 2, the term $\mathsf{g}\ (\mathsf{h}\ x\ \mathsf{a})$, where constants $\mathsf{g}$, $\mathsf{h}$, $\mathsf{a}$ originate from theories $T$, $U$, $V$, yields the patterns

T.g(_)   T.g(U.h(_, _))   U.h(_, _)   U.h(_,V.a)   V.a which are simplified and encoded into the features

$$\mathsf{T.g} \qquad \mathsf{T.g(U.h)} \qquad \mathsf{U.h} \qquad \mathsf{U.h(V.a)} \qquad \mathsf{V.a}$$

Types, excluding those of propositions, Booleans, and functions, are encoded using an analogous scheme. Type variables constrained by type classes give rise to features corresponding to the specified type classes and their superclasses. Finally, various pieces of metainformation are encoded as features: the theory to which the fact belongs; the kind of rule (e.g., introduction, simplification); whether the fact is local; whether the formula contains any existential quantifiers or $\lambda$-abstractions.

Guided by experiments similar to those of Sect. 8.5, we attributed the following weights to the feature classes:

| | | | | | |
|---|---|---|---|---|---|
| Fixed variable | 20 | Type | 2 | Presence of $\exists$ | 2 |
| Constant | 16 | Theory | 2 | Presence of $\lambda$ | 2 |
| Localness | 8 | Kind of rule | 2 | Type class | 1 |

**Example 8.4.1** *The lemma*

$$\mathit{transpose}\ (\mathit{map}\ (\mathit{map}\ f)\ xs) = \mathit{map}\ (\mathit{map}\ f)\ (\mathit{transpose}\ xs)$$

*from the List theory has the following features and weights (indicated by subscripts):*

| | |
|---|---|
| $\mathit{List}_{2}$ | $\mathit{List.transpose}_{16}$ |
| $\mathit{List.list}_{2}$ | $\mathit{List.transpose(List.map)}_{16}$ |
| $\mathit{List.map}_{16}$ | $\mathit{List.map(List.transpose)}_{16}$ |
| $\mathit{List.map(List.map)}_{16}$ | $\mathit{List.map(List.map, List.transpose)}_{16}$ |

**Proofs.** MaSh predicts which facts are useful for proving the goal at hand by studying successful proofs. There is an obvious source of successful proofs: All the facts in the loaded theories are accompanied by proof terms that store the dependencies [BN00]. However, not all available facts are equally suitable for learning. Many of them are derived automatically by definitional packages (e.g., for inductive predicates, datatypes, recursive functions) and proved using custom tactics, and there is not much to learn from those highly technical lemmas. The most interesting lemmas are those stated and proved by humans. Slightly abusing terminology, we call these "Isar proofs."

Even for user lemmas, the proof terms are overwhelmed by basic facts about the logic, which are tautologies in their translated form. Fortunately, these tautologies are easy to detect, since they contain only logical symbols (equality, connectives, and quantifiers). The proofs are also polluted by decision procedures; an extreme example is the Presburger arithmetic procedure, which routinely pulls in over 200 dependencies. Proofs involving over 20 facts are considered unsuitable and simply ignored.

Human-written Isar proofs are abundant, but they are not necessarily the best raw material to learn from. They tend to involve more, different facts than Sledgehammer proofs. Sometimes they rely on induction, which is difficult for automatic provers; but even excluding induction, there is evidence that the provers work better if the learned proofs were produced by similar provers [KvLT$^+$12, KU13g].

A special mode of Sledgehammer runs an automatic prover on all available facts to learn from ATP-generated proofs. Users can let it run for hours at at time on their favorite theories. The Isar proof facts are passed to the provers together with a few dozens of MePo-selected facts. Whenever a prover succeeds, MaSh discards the Isar proof and learns the new minimized proof (using MaSh.relearn). Facts with large Isar proofs are processed first since they stand to gain the most from shorter proofs.

**Visibility.** The loaded background theories and the user's formalization, including local lemmas, appear to Sledgehammer as a vast collection of facts. Each fact is tagged with its own abstract theory value, of type theory in ML, that captures the state of affairs when it was introduced. Sledgehammer constructs the visibility graph by using the (very fast) subsumption order $\trianglelefteq$ on theory.

A complication arises because $\trianglelefteq$ lifted to facts is a preorder, whereas the graph must encode a partial order $\preceq$. Antisymmetry is violated when facts are registered together. Despite the simultaneity, one fact's proof may depend on another's; for example, an inductive predicate's definition *p_def* is used to derive introduction and elimination rules *pI* and *pE*, and yet they may share the same theory. Hence, some care is needed when constructing $\preceq$ from $\trianglelefteq$ to ensure that $p\_def \preceq pI$ and $p\_def \preceq pE$.

When performing a query, Sledgehammer needs to compute the current goal's parents. This involves finding the maximal vertices of the visibility graph restricted to the facts available in the current Isabelle proof context. The computation is efficient for graphs with a quasi-linear structure, such as those that arise from Isabelle theories: Typically, only the first fact of a theory will have more than one parent. A similar computation is necessary when teaching MaSh new facts.

### 8.4.3. Relevance Filters: MaSh and MeSh

Sledgehammer's MaSh-based relevance filter computes the current goal's parents and features; then it queries the learner program (using MaSh.query), passing the chained facts as hints. This process usually takes about one second on modern hardware, which is reasonable for a tool that may run for half a minute. The result is a list with as many suggestions as desired, ordered by decreasing estimated relevance.

Relying purely on MaSh for relevance filtering raises an issue: MaSh may not have learned all the available facts. In particular, it will be oblivious to the very latest facts, introduced after Sledgehammer was invoked for the last time, and these are likely to be crucial for the proof. The solution is to enrich the raw MaSh data with a proximity filter, which sorts the available facts by decreasing proximity in the proof text.

Instead of a plain linear combination of ranks, the enriched MaSh filter transforms ranks into probabilities and takes their weighted average, with weight 0.8 for MaSh and 0.2 for proximity. The probabilities are rough approximations based on experiments. Fig. 8.1 shows the mathematical curves; for example, the first suggestion given by MaSh is considered about 15 times more likely to appear in a successful proof than the 50th.

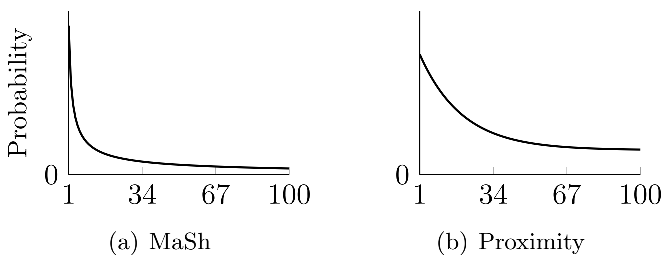


Figure 8.1.: Estimated probability of the $j$th fact's appearance in a proof

This notion of combining filters to define new filters is taken one step further by MeSh, a combination of *Me*Po and Ma*Sh*. Both filters are weighted 0.5, and both use the probability curve of Fig. 8.1(a).

Ideally, the curves and parameters that control the combination of filters would be learned mechanically rather than hard-coded. However, this would complicate and possibly slow down the infrastructure.

### 8.4.4. Automatic and Manual Control

All MaSh-related activities take place as a result of a Sledgehammer invocation. When Sledgehammer is launched, it checks whether any new facts, unknown to the visibility graph, are available. If so, it launches a thread to learn from their Isar proofs and update the graph. The first time, it may take about 10 s to learn all the facts in the background theories (assuming about 10 000 facts). Subsequent invocations are much faster.

If an automatic prover succeeds, the proof is immediately taught to MaSh (using `MaSh.learn`). The discharged (sub)goal may have been only one step in an unstructured proof, in which case it has no name. Sledgehammer invents a fresh, invisible name for it. Although this anonymous goal cannot be used to prove other goals, MaSh benefits from learning the connection between the formula's features and its proof.

For users who feel the need for more control, there is an `unlearn` command that resets MaSh's persistent state (using `MaSh.unlearn`); a `learn_isar` command that learns from the Isar proofs of all available facts; and a `learn_prover` command that invokes an automatic prover on all available facts, replacing the Isar proofs with successful ATP-generated proofs whenever possible.

### 8.4.5. Nonmonotonic Theory Changes

MaSh's model assumes that the set of facts and the visibility graph grow monotonically. One concern that arises when deploying machine learning—as opposed to evaluating its performance on static benchmarks—is that theories evolve nonmonotonically over time. It is left to the architecture around MaSh to recover from such changes. The following scenarios were considered:

- *A fact is deleted.* The fact is kept in MaSh's visibility graph but is silently ignored by Sledgehammer whenever it is suggested by MaSh.
- *A fact is renamed.* Sledgehammer perceives this as the deletion of a fact and the addition of another (identical) fact.
- *A theory is renamed.* Since theory names are encoded in fact names, renaming a theory amounts to renaming all its facts.
- *Two facts are reordered.* The visibility graph loses synchronization with reality. Sledgehammer may need to ignore a suggestion because it appears to be visible according to the graph.
- *A fact is introduced between two facts $\varphi$ and $\chi$.* MaSh offers no facility to change the parent of $\chi$, but this is not needed. By making the new fact a child of $\varphi$, it is considered during the computation of maximal vertices and hence visible.
- *The fact's formula is modified.* This occurs when users change the statement of a lemma, but also when they rename or relocate a symbol. MaSh is not informed of such changes and may lose some of its predictive power.

More elaborate schemes for tracking dependencies are possible. However, the benefits are unclear: Presumably, the learning performed on older theories is valuable and should be preserved, despite its inconsistencies. This is analogous to teams of humans developing a large formalization: Teammates should not forget everything they know each time a colleague changes the capitalization of some basic theory name. And should users notice a performance degradation after a major refactoring, they can always invoke `unlearn` to restart from scratch.

## 8.5. Evaluations

This section attempts to answer the main questions that existing Sledgehammer users are likely to have: How do MaSh and MeSh compare with MePo? Is machine learning really helping? The answer takes the form of two separate evaluations.[3]

[3]Our empirical data are available at `http://www21.in.tum.de/~blanchet/mash_data.tgz`.

### 8.5.1. Evaluation on Large Formalizations

The first evaluation measures the filters' ability to re-prove the lemmas from three formalizations included in the Isabelle distribution and the *Archive of Formal Proofs* [KNP]:

| | | |
|---|---|---|
| *Auth* | Cryptographic protocols [Pau98] | 743 lemmas |
| *Jinja* | Java-like language [KN05] | 733 lemmas |
| *Probability* | Measure and probability theory [HH11] | 1311 lemmas |

These formalizations are large enough to exercise learning and provide meaningful numbers, while not being so massive as to make experiments impractical. They are also representative of large classes of mathematical and computer science applications.

The evaluation is twofold. The first part computes how accurately the filters can predict the known Isar or ATP proofs on which MaSh's learning is based. The second part connects the filters to automatic provers and measures actual success rates.

The first part may seem artificial: After all, real users are interested in any proof that discharges the goal at hand, not a specific known proof. The predictive approach's greatest virtue is that it does not require invoking external provers; evaluating the impact of parameters is a matter of seconds instead of hours. MePo itself has been fine-tuned using similar techniques. For MaSh, the approach also helps ascertain whether it is learning the learning materials well, without noise from the provers. Two (slightly generalized) standard metrics, full recall and AUC, are useful in this context.

For a given goal, a fact filter (MePo, MaSh, or MeSh) ranks the available facts and selects the $n$ best ranked facts $\Phi = \{\varphi_1, \ldots, \varphi_n\}$, with $\mathit{rank}(\varphi_j) = j$ and $\mathit{rank}(\varphi) = n + 1$ for $\varphi \notin \Phi$. The parameter $n$ is fixed at 1024 in the experiments below.

The known proof $\Pi$ serves as a reference point against which the selected facts and their ranks are judged. Ideally, the selected facts should include as many facts from the proof as possible, with as low ranks as possible.

**Definition 8.5.1 (Full Recall)** *The* full recall *is the minimum number $m \in \{0, \ldots, n\}$ such that $\{\varphi_1, \ldots, \varphi_m\} \supseteq \Pi$, or $n + 1$ if no such number exists.*

**Definition 8.5.2 (AUC)** *The* area under the receiver operating characteristic curve *(*AUC*) is given by*

$$\frac{\big|\{(\varphi, \chi) \in \Pi \times (\Phi - \Pi) \mid \mathit{rank}(\varphi) < \mathit{rank}(\chi)\}\big|}{|\Pi| \cdot |\Phi - \Pi|}$$

Full recall tells how many facts must be selected to ensure that all necessary facts are included—ideally as few as possible. The AUC focuses on the ranks: It gives the probability that, given a randomly drawn "good" fact (a fact from the proof) and a randomly drawn "bad" fact (a selected fact that does not appear in the proof), the good fact is ranked before the bad fact. AUC values closer to 1 (100%) are preferable.

| | | MePo | | MaSh | | MeSh | |
|---|---|---|---|---|---|---|---|
| | | Full rec. | AUC | Full rec. | AUC | Full rec. | AUC |
| Isar proofs | *Auth* | 430 | 79.2 | 190 | 93.1 | **142** | **94.9** |
| | *Jinja* | 472 | 73.1 | 307 | 90.3 | **250** | **92.2** |
| | *Probability* | 742 | 57.7 | 384 | 88.0 | **336** | **89.2** |
| ATP proofs | *Auth* | 119 | 93.5 | 198 | 92.0 | **68** | **97.0** |
| | *Jinja* | 163 | 90.4 | 241 | 90.6 | **84** | **96.8** |
| | *Probability* | 428 | 74.4 | 368 | 85.2 | **221** | **91.6** |

Figure 8.2.: Average full recall and average AUC (%) with Isar and ATP proofs

For each of the three formalizations (*Auth*, *Jinja*, and *Probability*), the evaluation harness processes the lemmas according to a linearization (topological sorting) of the partial order induced by the theory graph and their location in the theory texts. Each lemma is seen as a goal for which facts must be selected. Previously proved lemmas, and the learning performed on their proofs, may be exploited—this includes lemmas from imported background theories. This setup is similar to the one used by Kaliszyk and Urban [KU13f] for evaluating their Sledgehammer-like tool for HOL Light. It simulates a user who systematically develops a formalization from beginning to end, trying out Sledgehammer on each lemma before engaging in a manual proof.[4]

Fig. 8.2 shows the average full recall and AUC over all lemmas from the three formalizations. For each formalization, the statistics are available for both Isar and ATP proofs. In the latter case, Vampire was used as the ATP, and goals for which it failed to find a proof are simply ignored. Learning from ATP proofs improves the machine learning metrics, partly because they usually refer to fewer facts than Isar proofs.

There is a reversal of fortune between Isar and ATP proofs: MaSh dominates MePo for the former but performs slightly worse than MePo for the latter on two of the formalizations. The explanation is that the ATP proofs were found with MePo's help. Nonetheless, the combination filter MeSh scores better than MePo on all the benchmarks.

Next comes the "in vivo" part of the evaluation, with actual provers replacing machine learning metrics. For each goal from the formalizations, 13 problems were generated, with 16, 23 ($\approx 2^{4.5}$), 32, …, 724 ($\approx 2^{9.5}$), and 1024 facts. Sledgehammer's translation is parameterized by many options, whose defaults vary from prover to prover and, because of time slicing, even from one prover invocation to another. As a reasonable uniform configuration for the experiments, types are encoded via the so-called polymorphic "featherweight" guard-based encoding (the most efficient complete scheme [BBPS13]),

[4]Earlier evaluations of Sledgehammer always operated on individual (sub)goals, guided by the notion that lemmas can be too difficult to be proved outright by automatic provers. However, lemmas appear to provide the right level of challenge for modern automation, and they tend to exhibit less redundancy than a sequence of similar subgoals.

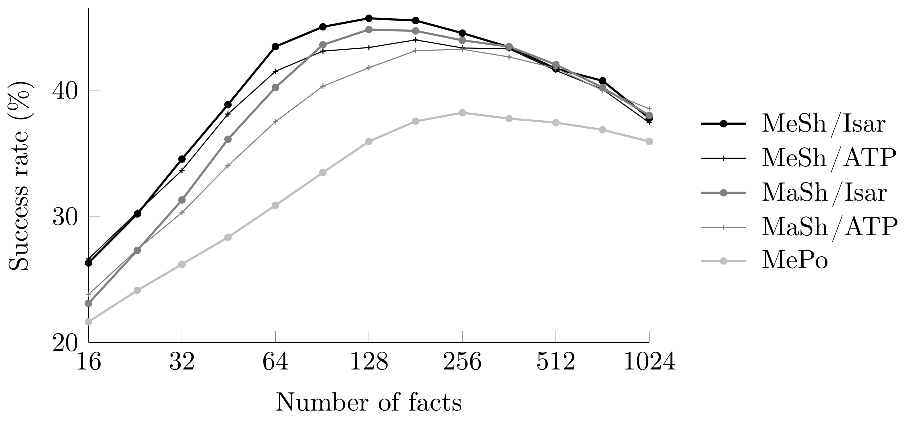


Figure 8.3.: Success rates for a combination of provers on *Auth* + *Jinja* + *Probability*

and $\lambda$-abstractions via $\lambda$-lifting (as opposed to the more explosive SK combinators).

Fig. 8.3 gives the success rates of a combination of three state-of-the-art automatic provers (Epar 1.6 [Urb14], Vampire 2.6, and Z3 3.2) on these problems. Two versions of MaSh and MeSh are compared, with learning on Isar and ATP proofs. A problem is considered solved if it is solved within 10 s by any of them, using only one thread. The experiments were conducted on a 64-bit Linux server equipped with 12-core AMD Opteron 6174 processors running at 2.2 GHz. We observe the following:

- MaSh clearly outperforms MePo, especially in the range from 32 to 256 facts. For 91-fact problems, the gap between MaSh/Isar and MePo is 10 percentage points. (The curves have somewhat different shapes for the individual formalizations, but the general picture is the same.)
- MaSh's peak is both higher than MePo's (44.8% vs. 38.2%) and occurs for smaller problems (128 vs. 256 facts), reflecting the intuition that selecting fewer facts more carefully should increase the success rate.
- MeSh adds a few percentage points to MaSh. The effect is especially marked for the problems with fewer facts.
- Against expectations, learning from ATP proofs has a negative impact. A closer inspection of the raw data revealed that Vampire performs better with ATP (i.e., Vampire) proofs, whereas the other two provers prefer Isar proofs.

Another measure of MaSh and MeSh's power is the total number of goals solved for any number of facts. With MePo alone, 46.3% of the goals are solved; adding MaSh and MeSh increases this figure to 62.7%. Remarkably, for *Probability*—the most difficult formalization by any standard—the corresponding figures are 27.1% vs. 47.2%.

| | MePo | MaSh | MeSh |
|---|---|---|---|
| E | 55.0 | 49.8 | **56.6** |
| SPASS | 57.2 | 49.1 | **57.7** |
| Vampire | 55.3 | 49.7 | **56.0** |
| Z3 | 53.0 | 51.8 | **60.8** |
| Together | 65.6 | 63.0 | **69.8** |

Figure 8.4.: Success rates (%) on Judgment Day goals

### 8.5.2. Judgment Day

The Judgment Day benchmarks [BN10] consist of 1268 interactive goals arising in seven Isabelle theories, covering among them areas as diverse as the fundamental theorem of algebra, the completeness of a Hoare logic, and Jinja's type soundness. The evaluation harness invokes Sledgehammer on each goal. The same hardware is used as in the original Judgment Day study [BN10]: 32-bit Linux servers with Intel Xeon processors running at 3.06 GHz. The time limit is 60 s for proof search, potentially followed by minimization and reconstruction in Isabelle. MaSh is trained on 9852 Isar proofs from the background libraries imported by the seven theories under evaluation.

The comparison comprises E 1.6, SPASS 3.8ds, Vampire 2.6, and Z3 3.2, which Sledgehammer employs by default. Each prover is invoked with its own options and problems, including prover-specific features (e.g., arithmetic for Z3; sorts for SPASS, Vampire, and Z3). Time slicing is enabled. For MeSh, some of the slices use MePo or MaSh directly to promote complementarity.

The results are summarized in Fig. 8.4. Again, MeSh performs very well: The overall 4.2 percentage point gain, from 65.6% to 69.8%, is highly significant. As noted in a similar study, "When analyzing enhancements to automatic provers, it is important to remember what difference a modest-looking gain of a few percentage points can make to users" [BPWW12, §7]. Incidentally, the 65.6% score for MePo reveals progress in the underlying provers compared with the 63.6% figure from one year ago.

The other main observation is that MaSh underperforms, especially in the light of the evaluation of Sect. 8.5.1. There are many plausible explanations. First, Judgment Day consists of smaller theories relying on basic background theories, giving few opportunities for learning. Consider the theory *NS_ Shared* (Needham–Shroeder shared-key protocol), which is part of both evaluations. In the first evaluation, the linear progress through all *Auth* theories means that the learning performed on other, independent protocols (certified email, four versions of Kerberos, and Needham–Shroeder public key) can be exploited. Second, the Sledgehammer setup has been tuned for Judgment Day and MePo over the years (in the hope that improvements on this representative benchmark suite would translate in improvements on users' theories), and conversely MePo's parameters are tuned for Judgment Day.

In future work, we want to investigate MaSh's lethargy on these benchmarks (and MeSh's remarkable performance given the circumstances). The evaluation of Sect. 8.5.1 suggests that there are more percentage points to be gained.

## 8.6. Related Work and Contributions

The main related work is already mentioned in the introduction. Bridges such as Sledgehammer for Isabelle/HOL, Miz$\mathbb{AR}$ [URS13] for Mizar, and HOL(y)Hammer [KU13f] for HOL Light are opening large formal theories to methods that combine ATPs and artificial intelligence (AI) [Urb11a, KvLT$^{+}$12] to help automate interactive proofs. Today such large theories are the main resource for combining semantic and statistical AI methods [UV13].[5]

The main contribution of this work has been to take the emerging machine-learning methods for fact selection and make them incremental, fast, and robust enough so that they run unnoticed on a single-user machine and respond well to common user-interaction scenarios. The advising services for Mizar and HOL Light [Urb06c, Urb06b, KU13f, URS13] (with the partial exception of MoMM [Urb06b]) run only as remote servers trained on the main central library, and their solution to changes in the library is to ignore them or relearn everything from scratch. Other novelties of this work include the use of more proof-related features in the learning (inspired by MePo), experiments combining MePo and MaSh, and the related learning of various parameters of the systems involved.

## 8.7. Conclusion

Relevance filtering is an important practical problem that arises with large-theory reasoning. Sledgehammer's MaSh filter brings the benefits of machine learning methods to Isabelle users: By decreasing the quantity and increasing the quality of facts passed to the automatic provers, it helps them find more, deeper proofs within the allotted time. The core machine learning functionality is implemented in a separate Python program that can be reused by other proof assistants.

Many areas are calling for more engineering and research; we mentioned a few already. Learning data could be shared on a server or supplied with the proof assistant. More advanced algorithms appear too slow for interactive use, but they could be optimized. Learning could be applied to control more aspects of Sledgehammer, such as the prover options or even MePo's parameters. Evaluations over the entire *Archive of Formal Proofs*

[5]It is hard to envisage all possible combinations, but with the recent progress in natural language processing, suitable ATP/AI methods could soon be applied to another major aspect of formalization: the translation from informal prose to formal specification.

might shed more light on MaSh's and MePo's strengths and weaknesses. Machine learning being a gift that keeps on giving, it would be fascinating to instrument a user's installation to monitor performance over several months.

**Acknowledgment.** Tobias Nipkow and Lawrence Paulson have, for years, encouraged us to investigate the integration of machine learning in Sledgehammer; their foresight has made this work possible. Gerwin Klein and Makarius Wenzel provided advice on technical and licensing issues. Tobias Nipkow, Lars Noschinski, Mark Summerfield, Dmitriy Traytel, and the anonymous reviewers suggested many improvements to earlier versions of this paper. The first author is supported by the Nederlandse organisatie voor Wetenschappelijk Onderzoek (NWO) project Learning2Reason. The second author is supported by the Deutsche Forschungsgemeinschaft (DFG) project Hardening the Hammer (grant Ni 491/14-1).

# 9 Lemma Mining

## Publication Details

Cezary Kaliszyk and Josef Urban. Lemma Mining over HOL Light. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors. Proc. of the 19th *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'13)*, volume 8312 of LNCS, pp. 503–517. 2013. doi: `10.1007/978-3-642-45221-5_34`

Cezary Kaliszyk and Josef Urban. Learning-assisted Theorem Proving with Millions of Lemmas. *J. Symbolic Computation*, 69:109–128, 2015. doi: `10.1016/j.jsc.2014.09.032`.

## Abstract

Large formal mathematical libraries consist of millions of atomic inference steps that give rise to a corresponding number of proved statements (lemmas). Analogously to the informal mathematical practice, only a tiny fraction of such statements is named and re-used in later proofs by formal mathematicians. In this work, we suggest and implement criteria defining the estimated usefulness of the HOL Light lemmas for proving further theorems. We use these criteria to mine the large inference graph of the lemmas in the HOL Light and Flyspeck libraries, adding up to millions of the best lemmas to the pool of statements that can be re-used in later proofs. We show that in combination with learning-based relevance filtering, such methods significantly strengthen automated theorem proving of new conjectures over large formal mathematical libraries such as Flyspeck.

## 9.1. Automated Reasoning over Large Mathematical Libraries

In the last decade, large formal mathematical corpora such as the Mizar Mathematical Library [GKN10] (MML), Isabelle/HOL [WPN08] and HOL Light [Har96a] with Flyspeck [Hal06] have been translated to formats that allow easy experiments with external automated theorem provers (ATPs) and AI systems [Urb04, MP08, KU14].

The problem that has immediately emerged is to efficiently perform automated reasoning over such large formal mathematical knowledge bases, providing as much support for authoring computer-understandable mathematics as possible. Reasoning with and over such large ITP (interactive theorem proving) libraries is however not just a new problem, but also a new opportunity, because the libraries already contain a lot of advanced knowledge in the form of concepts, theorems, proofs, and whole theory developments. Such large pre-existing knowledge allows mathematicians to state more advanced conjectures, and experiment on them with the power of existing symbolic reasoning methods. The large amount of mathematical and problem-solving knowledge contained in the libraries can be also subjected to all kinds of knowledge-extraction methods, which can later complement more exhaustive theorem-proving methods by providing domain-specific guidance. Developing the strongest possible symbolic reasoning methods that combine such knowledge extraction and re-use with correct deductive search is an exciting new area of Artificial Intelligence and Symbolic Computation.

Several symbolic AI/ATP methods for reasoning in the context of a large number of related theorems and proofs have been suggested and tried already, including: (i) methods (often external to the core ATP algorithms) that select relevant premises (facts) from the thousands of theorems available in such corpora [MP09, HV11, KvLT+12], (ii) methods for internal guidance of ATP systems when reasoning in the large-theory setting [UVŠ11], (iii) methods that automatically evolve more and more efficient ATP strategies for the clusters of related problems from such corpora [Urb14], and (iv) methods that learn which of such specialized strategies to use for a new problem [KSU13].

In this work, we start to complement the first set of methods – ATP-external premise selection – with *lemma mining* from the large corpora. The main idea of this approach is to enrich the pool of human-defined main (top-level) theorems in the large libraries with the most useful/interesting lemmas extracted from the proofs in these libraries. Such lemmas are then eligible together with (or instead of) the main library theorems as the premises that are given to the ATPs to attack new conjectures formulated over the large libraries.

This high-level idea is straightforward, but there are a number of possible approaches involving a number of issues to be solved, starting with a reasonable definition of a *useful/interesting lemma*, and with making such definitions efficient over corpora that contain millions to billions of candidate lemmas. These issues are discussed in Sections 9.4 and 9.5, after motivating and explaining the overall approach for using lemmas in large theories in Section 9.2 and giving an overview of the recent related work in Section 9.3.

As in any AI discipline dealing with large amount of data, research in the large-theory field is driven by rigorous experimental evaluations of the proposed methods over the existing corpora. For the first experiments with lemma mining we use the HOL Light system, together with its core library and the Flyspeck library. The various evaluation scenarios are defined and discussed in Section 9.6, and the implemented methods are evaluated in Section 9.7. Section 9.8 discusses the various future directions and concludes.[1]

## 9.2. Using Lemmas for Theorem Proving in Large Theories

The main task in the Automated Reasoning in Large Theories (ARLT) domain is to prove new conjectures with the knowledge of a large body of previously proved theorems and their proofs. This setting reasonably corresponds to how large ITP libraries are constructed, and hopefully also emulates how human mathematicians work more faithfully than the classical scenario of a single hard problem consisting of isolated axioms and a conjecture [UV13]. The pool of previously proved theorems ranges from thousands in large-theory ATP benchmarks such as MPTP2078 [AHK$^+$14], to tens of thousands when working with the whole ITP libraries.[2]

The strongest existing ARLT systems combine variously parametrized premise-selection techniques (often based on learning from previous proofs [AHK$^+$14]) with ATP systems and their strategies that are called with varied numbers of the most promising premises. These techniques can go quite far already: when using 14-fold parallelization and 30s wall-clock time, the HOL(y)Hammer system [KU14, KU13a] can today prove 47% of the 14185[3] Flyspeck theorems [KU13f]. This is measured in a scenario[4] in which the Flyspeck theorems are ordered *chronologically* using the loading sequence of the Flyspeck library, and presented in this order to HOL(y)Hammer as conjectures. After each theorem is attempted, its human-designed HOL Light proof is fed to the HOL(y)Hammer's learning components, together with the (often numerous) ATP proofs found by HOL(y)Hammer itself. This means that for each Flyspeck theorem, all human-written HOL Light proofs of all previous theorems are assumed to be known, together with all their ATP proofs found already by HOL(y)Hammer, but nothing is known about the current conjecture and the following parts of the library (they do not exist yet).

So far, systems like HOL(y)Hammer (similar systems include Sledgehammer/MaSh [KBKU13, BBP11], Miz$\mathbb{AR}$ [URS13] [KU13d] and MaLARea [USPV08, KUV14]) have only used the set of *named library theorems* for proving new conjectures and thus also for the premise-selection learning. This is usually a reasonable set of theorems to start with, because the human mathematicians have years of experience with structuring the formal libraries. On the other hand, there is no guarantee that this set is in any sense optimal, both for the human mathematicians and for the ATPs. The following three observations indicate that the set of human-named theorems may be suboptimal:

[1]This paper is an extended version of [KU13c].

[2]23323 theorems are in the HOL Light/Flyspeck library (SVN revision 3437), about 20000 are in the Isabelle/HOL library, and about 50000 theorems are in the Mizar library.

[3]These experiments were done on a earlier version of Flyspeck (SVN revision 2887) than is used here (SVN revision 3437), where the number of theorems is 23323.

[4]A similar scenario has been introduced in 2013 also for the LTB (Large-Theory Batch) division of the CASC competition.

*Proofs of different length:* The human-named theorems may differ considerably in the length of their proofs. The human naming is based on a number of (possibly traditional/esthetical) criteria that may sometimes have little to do with a good structuring of the library.

*Duplicate and weak theorems:* The large collaboratively-build libraries are hard to manually guard against duplications and naming of weak versions of various statements. The experiments with the MoMM system over the Mizar library [Urb06b] and with the recording of the Flyspeck library [KK13] have shown that there are a number of subsumed and duplicated theorems, and that some unnamed strong lemmas are proved over and over again.

*Short alternative proofs:* The experiments with AI-assisted ATP over the Mizar and Flyspeck libraries [AKU12,KU14] have shown that the combined AI/ATP systems may sometimes find alternative proofs that are much shorter and very different from the human proofs, again turning some "hard" named theorems into easy corollaries.

Suboptimal naming may obviously influence the performance of the current large-theory systems. If many important lemmas are omitted by the human naming, the ATPs will have to find them over and over when proving the conjectures that depend on such lemmas. On the other hand, if many similar variants of one theorem are named, the current premise-selection methods might focus too much on those variants, and fail to select the complementary theorems that are also necessary for proving a particular conjecture.[5]

To various extent, this problem might be remedied by the alternative learning/guidance methods (ii) and (iii) mentioned in Section 9.1: Learning of internal ATP guidance using for example Veroff's *hint technique* [Ver96], and learning of suitable ATP strategies using systems like BliStr [Urb14]. But these methods are so far much more experimental in the large-theory setting than premise selection.[6] That is why we propose and explore here the following lemma-mining approach:

1. Considering (efficiently) the detailed graph of all atomic inferences contained in the ITP libraries. Such a graph has millions of nodes for the core HOL Light corpus, and hundreds of millions of nodes for the whole Flyspeck.

2. Defining over such large proof graphs efficient criteria that select a smaller set of the strongest and most orthogonal lemmas from the corpora.

[5]This behavior obviously depends on the premise-selection algorithm. It is likely to occur when the premise selection is mainly based on symbolic similarity of the premises to the conjecture. It is less likely to occur when complementary semantic selection criteria are additionally used as, e.g., in SRASS [SP07] and MaLARea [USPV08].

[6]In particular, several initial experiments done so far with Veroff's hints over the MPTPChallenge and MPTP2078 benchmarks were so far unsuccessful.

3. Using such lemmas together with (or instead of) the human-named theorems for proving new conjectures over the corpora.

## 9.3. Overview of Related Work and Ideas

A number of ways how to measure the quality of lemmas and how to use them for further reasoning have been proposed already, particularly in the context of ATP systems and proofs. Below we summarize recent approaches and tools that initially seemed most relevant to our work.

Lemmas are an essential part of various ATP algorithms. State-of-the-art ATPs such as Vampire [KV13], E [Sch02] and Prover9 [McC10] implement various variants of the ANL loop [WOLB84], resulting in hundreds to billions of lemmas inferred during the prover runs. This gave rise to a number of efficient ATP indexing techniques, redundancy control techniques such as subsumption, and also fast ATP heuristics (based on weight, age, conjecture-similarity, etc.) for choosing the best lemmas for the next inferences. Several ATP methods and tools work with such ATP lemmas. Veroff's *hint technique* [Ver96] extracts the best lemmas from the proofs produced by successful Prover9 runs and uses them for directing the proof search in Prover9 on related problems. A similar lemma-extracting, generalizing and proof-guiding technique (called *E Knowledge Base – EKB*) was implemented by Schulz in E prover as a part of his PhD thesis [Sch00].

Schulz with Denzinger also developed the epcllemma [DS94, DS96] tool that estimates the best lemmas in an arbitrary DAG (directed acyclic graph) of inferences. Unlike the hint-extracting/guiding methods, this tool works not just on the handful of lemmas involved in the final refutational proof, but on the typically very large number of lemmas produced during the (possibly unfinished) ATP runs. The epcllemma's criteria for selecting the next best lemma from the inference DAG are: (i) the size of the lemma's inference subgraph based at the nodes that are either axioms or already chosen (better) lemmas, and (ii) the weight of the lemma. This lemma-selection process may be run recursively, until a stopping criterion (minimal lemma quality, required number of lemmas, etc.) is reached. Our algorithm for selecting HOL Light lemmas (Section 9.5) is quite similar to this.

AGIntRater [PGS06] is a tool that computes various characteristics of the lemmas that are part of the final refutational ATP proof and aggregates them into an overall *interestingness* rating. These characteristics include: obviousness, complexity, intensity, surprisingness, adaptivity, focus, weight, and usefulness, see [PGS06] for details. AGIntRater so far was not directly usable on our data for various reasons (particularly the size of our graph), but we might re-use and try to efficiently implement some of its ideas later.

Pudlák [Pud06] has conducted experiments over several datasets with automated re-use of lemmas from many existing ATP proofs in order to find smaller proofs and also to attack unsolved problems. This is similar to the hints technique, however more automated and closer to our large-theory setting (hints have so far been successfully applied mainly in small algebraic domains). To interreduce the large number of such lemmas with respect to subsumption he used the *CSSCPA* [Sut01] subsumption tool by Schulz and Sutcliffe based on the E prover. MoMM [Urb06b] adds a number of large-theory features to CSSCPA. It was used for (i) fast interreduction of million of lemmas extracted (generalized) from the proofs in the Mizar library, and (ii) as an early ATP-for-ITP hammer-style tool for completing proofs in Mizar with the help of the whole Mizar library. All library lemmas can be loaded, indexed and considered for each query, however the price for this breadth of coverage is that the inference process is limited to subsumption extended with Mizar-style dependent types.

AGIntRater and epcllemma use a lemma's position in the inference graph as one of the lemma's characteristics that contribute to its importance. There are also purely graph-based algorithms that try to estimate a relative importance of nodes in a graph. In particular, research of large graphs became popular with the appearance of the World Wide Web and social networks. Algorithms such as *PageRank* [PBMW98] (eigenvector centrality) have today fast approximative implementations that easily scale to billions of nodes.

## 9.4. The Proof Data

We consider two corpora: the core HOL Light corpus (SVN version 179) and the Flyspeck corpus (SVN version 3437). The core HOL Light corpus contains of 2,239 named theorems, while the Flyspeck corpus consists of 23,323 named theorems. The first prerequisite for implementing and running interesting lemma-finding algorithm is the extraction of the full dependency graph containing all intermediate steps (lemmas), and identification of the named top-level theorems among them.

There are three issues with the named theorems that we initially need to address. First, many theorems in HOL Light are conjunctions. It is often the case that lemmas that deal with the same constant or theory are put in the same theorem, so that they can be passed to tactics and decision procedures as a single argument rather than a list. Second, a single theorem may be given multiple names. This is especially common in case of larger formalizations like Flyspeck. Third, even if theorems are not syntactically equal they may be alpha-equivalent. HOL Light does not natively use de Bruijn indices for representing variables, i.e., two alpha-equivalent versions of the same theorems will be kept in the proof trace if they differ in variable names. Therefore the first operation we perform is to find a unique name for each separate top-level conjunct. The data sizes and processing times of this first phase can be found in Table 9.1.

We next look at all the available intermediate lemmas, each of them corresponding to one of the LCF-style kernel inferences done by HOL Light. The number of these

| | HOL Light (179) | Flyspeck (3437) |
|---|---|---|
| Named theorems | 2,239 | 23,323 |
| Distinct named conjuncts | 2,542 | 24,745 |
| Constant definitions | 234 | 2,106 |
| Type definitions | 18 | 29 |
| Processing time | 2m09s | 327m56s |
| Processing memory | 214MB | 1,645MB |

Table 9.1.: The top-level available data and processing statistics of the analyzed corpora.

lemmas when processing Flyspeck is around 1.7 billion. Here, already performing the above mentioned reduction is hard since the whole graph with the 1.7 billion HOL Light formulas can be considered big data: it fits neither in memory nor on a single hard disk. Therefore we perform the first graph reductions already when recording the proof trace.

To obtain the full inference graph for Flyspeck we run the proof-recording version of HOL Light [KK13] patched to additionally remember all the intermediate lemmas. Obtaining such trace for Flyspeck takes 29 hours of CPU time and 56 GB of RAM on an AMD Opteron 6174 2.2 GHz Because of the memory consumption we initially consider two versions: a) de-duplicating all the intermediate lemmas within a named theorem; we call the graph obtained in this way `TRACE0`, and b) de-duplicating all the lemmas; which we call `TRACE1`. The sizes of the traces are presented in Table 9.2. This time and memory consumption are much lower when working only with the core HOL Light, where a further graph optimization in this step could already be possible.

| | HOL Light graph | | Flyspeck graph | |
|---|---|---|---|---|
| | nodes | edges | nodes | edges |
| kernel inferences | 8,919,976 | 10,331,922 | 1,728,861,441 | 1,953,406,411 |
| `TRACE0` | 2,435,875 | 3,476,767 | 206,699,009 | 302,799,816 |
| `TRACE1` | 2,076,682 | 3,002,990 | 159,102,636 | 233,488,673 |
| tactical inferences | 148,514 | 594,056 | 11,824,052 | 42,296,208 |
| tactical trace | 22,284 | 89,981 | 1,067,107 | 4,268,428 |

Table 9.2.: The sizes of the inference graphs.

There are 1,953,406,411 inference edges between the unique Flyspeck lemmas. During the proof recording we additionally export the information about the symbol weight (size) of each lemma (the weight is used in some of the lemma-quality metrics defined in Section 9.5), and for the small HOL Light traces also the lemma's normalized form that serially numbers bound and free variables and tags them with their types. This information is later used for external postprocessing, together with the information about which theorems where originally named. The initial segment of the Flyspeck proof trace is presented in Fig. 9.1, all the traces are available online.[7]

```
F13       #1, Definition (size 13): T ⟺ (λA0. A0) = (λA0. A0)
R9        #2, Reflexivity (size 9): (λA0. A0) = (λA0. A0)
R5        #3, Reflexivity (size 5): T ⟺ T
R5        #4, Reflexivity (size 5): (⟺) = (⟺)
C17 4 1   #5, Application(4,1):     (⟺) T = (⟺) ((λA0. A0) = (λA0. A0))
C21 5 3   #6, Application(5,3):     (T ⟺ T) ⟺ (λA0. A0) = (λA0. A0) ⟺ T
E13 6 3   #7, EQ_MP(6,3) (size 13): (λA0. A0) = (λA0. A0) ⟺ T
```

Figure 9.1.: Initial segment of the HOL Light theorem trace commented with the numbers of the steps and the theorems derived by the steps.

## 9.4.1. Initial Post-processing and Optimization

During the proof recording, only exact duplicates are easy to detect. As already explained in the previous Section, HOL Light does not natively use de Bruijn indices for representing variables, so the trace may still contain alpha-equivalent versions of the same theorems. Checking for alpha equivalence during the proof recording would be possible, however is not obvious since in the HOL Light's LCF-style approach alpha conversion itself results in multiple kernel inferences. In order to avoid performing term-level renamings we keep the original proof trace untouched, and implement its further optimizations as external postprocessing of the trace.

In particular, to merge alpha-equivalent lemmas in a proof trace $T$, we just use the above mentioned normalized-variable representation of the lemmas as an input to an external program that produces a new version of the proof trace $T'$. This program goes through the trace $T$ and replaces references to each lemma by a reference to the earliest lemma in $T$ with the same normalized-variable representation. The proofs of the later named alpha variants of the lemmas in $T$ are however still kept in the new trace $T'$, because such proofs are important when computing the usage and dependency statistics over the normalized lemmas. We have done this postprocessing only for the core HOL Light lemmas, because printing out of the variable-normalized version of the 150,142,900 partially de-duplicated Flyspeck lemmas is currently not feasible on our hardware. From the 2,076,682 partially de-duplicated core HOL Light lemmas 1,076,995 are left after this stronger normalization. We call such further post-processed graph `TRACE2`.

It is clear that such post-processing operations can be implemented in various ways. In this case, some original information about the proof graph is lost, while some information

[7] `http://cl-informatik.uibk.ac.at/~cek/lemma_mining/`

(proofs of duplicate lemmas) is still kept, even though it could be also pruned from the graph, producing a differently normalized version.

### 9.4.2. Obtaining Shorter Traces from the Tactic Calls

Considering the HOL kernel proof steps as the atomic steps in construction of intermediate lemmas has (at least) three drawbacks. First, the pure size of the proof traces makes it hard to scale the lemma-mining procedures to big developments like Flyspeck. Second, the multitude of steps that arise when applying simple HOL Light decision procedures overshadows the interesting parts of the proofs. It is not uncommon for a simple operation, like a normalization of a polynomial, to produce tens of thousands of core kernel inferences. Third, some operations (most notably the HOL Light simplifier) produce kernel inferences in the process of proof search. Such inferences are not only uninteresting (as in the previous case), but often useless for the final proof.

In order to overcome the above three issues encountered in the first experiments, we followed by gathering data at the level of the HOL Light *tactic* steps [Har96a]. The execution of each HOL Light tactic produces a new goal state together with a justification function that produces an intermediate lemma. In this approach we consider only the lemmas produced by the justification functions of tactics (instead of considering all kernel steps). The HOL Light tactics work on different levels. The tactics executed by the user and visible in the proof script form the outermost layer. However most of the tactics are implemented as OCaml functions that inspect the goal and execute other (smaller) tactics. If we unfold such internal executions of tactics recursively, the steps performed are of a similar level of detail as in typical natural deduction proofs.

This could give us a trace that is slightly smaller than the typical trace of the kernel inferences; however the size is still of the same order of magnitude. In order to efficiently process large formal developments we decided to look at an intermediate level: only at the tactics that are composed using *tactic combinators* [Har96a].

In order to patch the tactic combinators present in HOL Light and Flyspeck it is enough to patch the three building blocks of tactic combinators: `THEN`, `THENL`, and `by`. Loading Flyspeck with these functions patched takes about 25% more time than the original and requires 6GB of memory to remember all the 20 million new intermediate theorems. This is significantly less than the patched kernel version and the produced graph can be reasonably optimized.

The optimizations performed on the level of named theorems can be done here again: recursively splitting conjunctions and normalizing the quantifiers, as well as the premises we get 2,014,505 distinct conjuncts. After alpha-normalization this leaves a trace with 1,067,107 potential intermediate lemmas. In order to find dependencies between the potential intermediate lemmas we follow the approach by Kaliszyk and Krauss [KK13] which needs a second dependency recording pass over the whole Flyspeck.

The post-processed tactics dependency graph has 4,268,428 edges and only 2,145 nodes have no dependencies. The comparison of all the traces can be seen in Table 9.2. The data is written in the same format as the HOL kernel inference data, so that we can use the same predictors. An excerpt from the tactical trace coming from the proof of MAP_APPEND is presented in Fig. 9.2.

```
X29 3377 3371       #3437, Rewriting with two given theorems, size 29:
         ⊢  ∀l2. MAP f (APPEND [] l2) = APPEND (MAP f []) (MAP f l2)

X66 3378 3372       #3438, Rewriting with two given theorems, size 66:
         ⊢  ∀t. (∀l2. MAP f (APPEND t l2) = APPEND (MAP f t) (MAP f l2))
                          ⟹ (∀l2. MAP f (APPEND (CONS h t) l2) =
                             APPEND (MAP f (CONS h t)) (MAP f l2))

X33 3321 3437 3438  #3439, List induction, size 33:
         ⊢  ∀f l1 l2. MAP f (APPEND l1 l2) = APPEND (MAP f l1) (MAP f l2)
```

Figure 9.2.: An excerpt of the tactical trace showing the dependencies between the goal states in the proof of MAP_APPEND. For simplicity we chose an excerpt that shows the theorems created by the direct application of a tactic that does not call other tactics (LIST_INDUCT_TAC). This means that all the theorems created in this part of the trace directly correspond to goals visible to the proof-assistant user.

### 9.4.3. Other Possible Optimizations

The ATP experiments described below use only the four versions of the proof trace (TRACE0, TRACE1, TRACE2, and the tactical trace) described above, but we have also explored some other normalizations. A particularly interesting optimization from the ATP point of view is the removal of subsumed lemmas. An initial measurement with the (slightly modified) MoMM system done on the clausified first-order versions of about 200,000 core HOL Light lemmas has shown that about 33% of the clauses generated from the lemmas are subsumed. But again, ATP operations like subsumption interact with the level of inferences recorded by the HOL Light kernel in nontrivial ways. It is an interesting task to define exactly how the original proof graph should be transformed with respect to such operations, and how to perform such proof graph transformations efficiently over the whole Flyspeck.

## 9.5. Selecting Good Lemmas

Several approaches to defining the notion of a useful/interesting lemma are mentioned in Section 9.3. There are a number of ideas that can be explored and combined together in various ways, but the more complex methods (such as those used by AGIntRater) are not yet directly usable on the large ITP datasets that we have. So far, we have experimented mainly with the following techniques:

1. A direct OCaml implementation of lemma quality metrics based on the HOL Light proof-recording data structures.
2. Schulz's epcllemma and our modified versions thereof.
3. PageRank, applied in various ways to the proof trace.
4. Graph cutting algorithms with modified weighting function.

### 9.5.1. Direct Computation of Lemma Quality

The advantage of the direct OCaml implementation is that no export to external tools is necessary and all the information collected about the lemmas by the HOL Light proof recording is directly available. The basic factors that we use so far for defining the quality of a lemma $i$ are its: (i) set of direct proof dependencies $d(i)$ given by the proof trace, (ii) number of recursive dependencies $D(i)$, (iii) number of recursive uses $U(i)$, and (iv) number of HOL symbols (HOL weight) $S(i)$. When recursively defining $U(i)$ and $D(i)$ we assume that in general some lemmas may already be named ($k \in Named$) and some lemmas are just axioms ($k \in Axioms$). Note that in HOL Light there are many lemmas that have no dependencies, but formally they are still derived using for example the reflexivity inference rule (i.e., we do not count them among the HOL Light axioms). The recursion when defining $D$ thus stops at axioms, named lemmas, and lemmas with no dependencies. The recursion when defining $U$ stops at named lemmas and unused lemmas. Formally:

**Definition 9.5.1 (Recursive dependencies and uses)**

$$D(i) = \begin{cases} 1 & \textit{if } i \in Named \lor i \in Axioms, \\ \sum_{j \in d(i)} D(j) & \textit{otherwise.} \end{cases}$$

$$U(i) = \begin{cases} 1 & \textit{if } i \in Named, \\ \sum_{i \in d(j)} U(j) & \textit{otherwise.} \end{cases}$$

In particular, this means that

$$D(i) = 0 \iff d(i) = \emptyset \wedge \neg(i \in \mathit{Axioms})$$

and also that

$$U(i) = 0 \iff \forall j \neg(i \in d(j))$$

These basic characteristics are combined into the following lemma quality metrics $Q_1(i)$, $Q_2(i)$, and $Q_3(i)$. $Q_1^r(i)$ is a generalized version of $Q_1(i)$, which we (apart from $Q_1$) test for $r \in \{0, 0.5, 1.5, 2\}$:

**Definition 9.5.2 (Lemma quality)**

$$Q_1(i) = \frac{U(i) * D(i)}{S(i)} \qquad Q_1^r(i) = \frac{U(i)^r * D(i)^{2-r}}{S(i)}$$
$$Q_2(i) = \frac{U(i) * D(i)}{S(i)^2} \qquad Q_3(i) = \frac{U(i) * D(i)}{1.1^{S(i)}}$$

The justification behind these definitions are the following heuristics:

1. The higher $D(i)$ is, the more necessary it is to remember the lemma $i$, because it will be harder to infer with an ATP when needed.
2. The higher $U(i)$ is, the more useful the lemma $i$ is for proving other desired conjectures.
3. The higher $S(i)$ is, the more complicated the lemma $i$ is in comparison to other lemmas. In particular, doubled size may often mean in HOL Light that $i$ is just a conjunction of two other lemmas.[8]

### 9.5.2. Lemma Quality via epcllemma

Lemma quality in epcllemma is defined on clause inferences recorded using E's native PCL protocol. The lemma quality computation also takes into account the lemmas that have been already named, and with minor implementational variations it can be expressed using $D$ and $S$ as follows:

$$EQ_1(i) = \frac{D(i)}{S(i)}$$

[8]The possibility to create conjunctions is quite a significant difference to the clausal setting handled by the existing tools. A longer clause is typically weaker, while longer conjunctions are stronger. A dependence on a longer conjunction should ideally be treated by the evaluating heuristics as a dependence on the multiple conjuncts. Note that for the tactical trace we already split all conjunctions in the trace.

The difference to $Q_1(i)$ is that $U(i)$ is not used, i.e., only the cumulative effort needed to prove the lemma counts, together with its size (this is also very close to $Q_1^r(i)$ with $r = 0$). The main advantage of using epcllemma is its fast and robust implementation using the E code base. This allowed us to load in reasonable time (about one hour) the whole Flyspeck proof trace into epcllemma, taking 67 GB of RAM. Unfortunately, this experiment showed that epcllemma assumes that $D$ is always an integer. This is likely not a problem for epcllemma's typical use, but on the Flyspeck graph this quickly leads to integer overflows and wrong results. To a smaller extent this shows already on the core HOL Light proof graph. A simple way how to prevent the overflows was to modify epcllemma to use instead of $D$ the longest chain of inferences $L$:

$$L(i) = \begin{cases} 1 & \text{if } i \in Named \vee i \in Axioms, \\ max_{j \in d(i)}(1 + L(j)) & \text{otherwise.} \end{cases}$$

This leads to:

$$EQ_2(i) = \frac{L(i)}{S(i)}$$

Apart from this modification, only minor changes were needed to make epcllemma work on the HOL Light data. The proof trace was expressed as a PCL proof (renaming the HOL inferences into E inferences), and TPTP clauses were used instead of the original HOL clauses. We additionally compared two strategies of creating the TPTP clauses. First we applied the MESON [Har96c] translation to the HOL clause, second we tried to create artificial TPTP clauses of the size corresponding to the size of the HOL clause.

## 9.5.3. Lemma Quality via PageRank

PageRank (eigenvector centrality of a graph) is a method that assigns weights to the nodes in an arbitrary directed graph (not just DAG) based on the weights of the neighboring nodes ("incoming links"). In more detail, the weights are computed as the dominant eigenvector of the following set of equations:

$$PR_1(i) = \frac{1-f}{N} + f \sum_{i \in d(j)} \frac{PR_1(j)}{|d(j)|}$$

where $N$ is the total number of nodes and $f$ is a damping factor, typically set to 0.85. The advantage of using PageRank is that there are fast approximative implementations that can process the whole Flyspeck proof graph in about 10 minutes using about 21 GB RAM, and the weights of all nodes are computed simultaneously in this time.

This is however also a disadvantage in comparison to the previous algorithms: PageRank does not take into account the lemmas that have already been selected (named). The closer a lemma $i$ is to an important lemma $j$, the more important $i$ will be. Modifications that use the initial PageRank scores for more advanced clustering exist [ADN+08] and perhaps could be used to mitigate this problem while still keeping the overall processing reasonably fast. Another disadvantage of PageRank is its ignorance of the lemma size, which results in greater weights for the large conjunctions that are used quite often in HOL Light. $PR_2$ tries to counter that:

$$PR_2(i) = \frac{PR_1(i)}{S(i)}$$

$PR_1$ and $PR_2$ are based on the idea that a lemma is important if it is needed to prove many other important lemmas. This can be again turned around: we can define that a lemma is important if it depends on many important lemmas. This is equivalent to computing the reverse PageRank and its size-normalized version:

$$PR_3(i) = \frac{1-f}{N} + f \sum_{i \in u(j)} \frac{PR_3(j)}{|u(j)|} \qquad PR_4(i) = \frac{PR_3(i)}{S(i)}$$

where $u(j)$ are the direct uses of the lemma $j$, i.e., $i \in u(j) \iff j \in d(i)$. The two ideas can again be combined (note that the sum of the PageRanks of all nodes is always 1):

$$PR_5(i) = PR_1(i) + PR_3(i) \qquad PR_6(i) = \frac{PR_1(i) + PR_3(i)}{S(i)}$$

### 9.5.4. Lemma Quality using Graph Cut

The approaches so far tried to define what a “good” lemma is using our intuitions coming from mathematics. Here we will try to estimate the impact that choosing certain lemmas will have on the final dependency graph used for the learning framework.

Choosing a subset of the potential intermediate lemmas can be considered a variant of the graph-theoretic problems of finding a cut with certain properties. We will consider only cuts that respect the chronological order of theorems in the library. Since many of the graph-cut algorithms (for example maximum cut) are NP-complete, we decide to build the cut greedily adding nodes to the cut one by one.

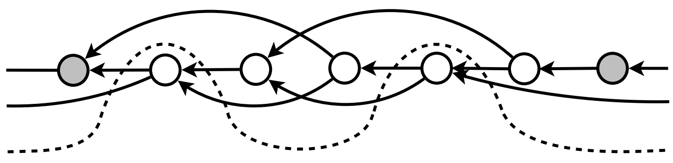


Figure 9.3.: An example cut of the dependency graph that respects the chronological order of the library. The already named theorems are marked in gray.

Given a graph where certain nodes are already named (marked gray in the Figure 9.3) we want to estimate the impact of choosing a new lemma on the evaluation. In the evaluation, we will compute the dependency graph of all the gray nodes together with the newly chosen one. The final graph represents the human dependencies, which means that theorems are ATP-provable using exactly these dependencies. By minimizing the number of edges in this final graph we make the average number of premises in the problems smaller which should make the problems easier to prove. The assumption here, is that training our premise-selection systems on theorems that are easier to prove makes the resulting AI/ATP system stronger.

In order to minimize the number of edges in the final graph we will investigate what is the impact of adding a node $n$ to the cut. We consider all the dependency paths starting which start at $n$. On each path we select the first already named node. All the nodes that have been selected are the dependencies that $n$ will have in the final dependency graph. Lets denote this set as $D(n)$. Similarly we can find all the nodes that will have $n$ as a dependency This can be done in a similar way, taking all the paths the opposite direction again choosing the first gray node on each path. Lets denote the nodes found as $U(n)$. These nodes will have $n$ as a dependency if $n$ is chosen to be in the cut.

**Theorem 9.5.3** *Adding a node $n$ to the cut $c$ will decrease the number of edges in the final graph by $|D(n)| * |U(n)| - |D(n)| - |U(n)|$.*

**Proof** With the cut $c$ the edges in the final graph include all the edges between the nodes in $D(n)$ and $U(n)$. Adding the node $n$ to $c$ these $|D(n)| * |U(n)|$ edges will be replaced by the dependencies from each element of $U(n)$ to $n$ ($|U(n)|$ many of them) and the dependencies from $n$ to all the elements of $D(n)$ ($|D(n)|$ many of them).

The algorithm manipulates sets of nodes rather than numbers, which makes it significantly slower than all the previously described ones. We will test this algorithm only for up to 10,000 lemmas as already finding them takes 11 CPU hours. Similarly to the algorithms in the previous subsections we try to limit the effect of large theorems on the algorithm by considering also the size normalized version:

$$MC_1(i) = |D(i) * U(i)| - |D(i)| - |U(i)| \qquad MC_2(i) = \frac{MC_1(i)}{S(i)}$$

### 9.5.5. Selecting Many Lemmas

From the methods described above, only the various variants of PageRank ($PR_i$) produce the final ranking of all lemmas in one run. Both epcllemma ($EQ_i$) and our custom methods ($Q_i$, $MC_i$) are parametrized by the set of lemmas ($Named$) that have already been named. When the task is to choose a predefined number of the best lemmas, this naturally leads to the recursive lemma-selection Algorithm 2 (used also by epcllemma). This algorithm in each iteration selects the lemma with the highest quality wrt. the

Algorithm 2: Best lemmas

```
Input a lemma-quality metric $Q$, set of lemmas $Lemmas$, an initial set of named lemmas
    $Named_0 \subset Lemmas$, and a required number of lemmas $M$
Output set $Named$ of $M$ best lemmas according to $Q$
 1: $Named \leftarrow Named_0$
 2: $m \leftarrow 0$
 3: while $m < M$ do
 4:     for $i \in Lemmas$ do
 5:         Calculate($Q_{Named}(i)$)
 6:     end for
 7:     $j \leftarrow argmax\{Q_{Named}(i) : i \in Lemmas \setminus Named\}$
 8:     $Named \leftarrow Named \cup \{j\}$
 9:     $m \leftarrow m + 1$
10: end while
11: Return($Named$)
```

current set of named lemmas, and adds this lemma to that set, usually making some recomputing of the lemma quality necessary in the next iteration.

There are two possible choices of the initial set of named lemmas $Named_0$ in Algorithm 2: either the empty set, or the set of all human-named theorems. This choice depends on whether we want to re-organize the library from scratch, or whether we just want to select good lemmas that complement the human-named theorems. Below we experiment with both approaches. Note that this algorithm is currently quite expensive: the fast epcllemma implementation takes 65 seconds to update the lemma qualities over the whole Flyspeck graph after each change of the $Named$ set. This means that with the kernel-based inference trace (`TRACE1`) producing the first 10,000 Flyspeck lemmas takes 180 CPU hours. That is why most of the experiments are limited to the core HOL Light graph and Flyspeck tactical graph where this takes about 1 second and 3 hours respectively.

## 9.6. Evaluation Scenarios and Issues

To assess and develop the lemma-mining methods we define several evaluation scenarios that vary in speed, informativeness and rigor. The simplest and least rigorous is the *expert-evaluation* scenario: We can use our knowledge of the formal corpora to quickly see if the top-ranked lemmas produced by a particular method look plausible.

The *cheating ATP* scenario uses the full proof graph of a corpus to compute the set of the (typically 10,000) best lemmas (*BestLemmas*) for the whole corpus. Then the set of newly named theorems (*NewThms*) is defined as the union of *BestLemmas* with the set of originally named theorems ($OrigThms$): $NewThms := BestLemmas \cup OrigThms$. The derived graph $G_{NewThms}$ of direct dependencies among the elements of $NewThms$ is used for ATP evaluation, which may be done in two ways: with human selection and with AI selection. When using human selection, we try to prove each lemma from its parents in $G_{NewThms}$. When using AI selection, we use the chronological order (see Section 9.2) of $NewThms$ to incrementally train and evaluate the $k$-NN machine learner [KU13f] on the direct dependencies from $G_{NewThms}$. This produces for each new theorem an ATP problem with premises advised by the learner trained on the $G_{NewThms}$ dependencies of the preceding new theorems. This scenario may do a lot of cheating, because when measuring the ATP success on $OrigThms$, a particular theorem $i$ might be proved with the use of lemmas from $NewThms$ that have been stated for the first time only in the original proof of $i$ (we call such lemmas *directly preceding*). In other words, such lemmas did not exist before the original proof of $i$ was started, so they could not possibly be suggested by lemma-quality metrics for proving $i$. Such directly preceding lemmas could also be very close to $i$, and thus equally hard to prove.

The *almost-honest ATP* scenario does not allow the use of the directly preceding new lemmas. The dependencies of each $i \in NewThms$ may consist only of the previous $OrigThms$ and the lemmas that precede them. Directly preceding new lemmas are replaced by their closest $OrigThms$ ancestors. This scenario is still not fully honest, because the lemmas are computed according to their lemma quality measured on the full proof graph. In particular, when proving an early theorem $i$ from $OrigThms$, the newly used parents of $i$ are lemmas whose quality was clear only after taking into account the theorems that were proved later than $i$. These theorems and their proofs however did not exist at the time of proving $i$. Still, we consider this scenario sufficiently honest for most of the ATP evaluations done with the whole core HOL Light dataset and the representative subset of the Flyspeck dataset.

The *fully-honest ATP* scenario removes this last objection, at the price of using considerably more resources for a single evaluation. For each originally named theorem $j$ we limit the proof graph used for computing $BestLemmas$ to the proofs that preceded $j$. Since computing $BestLemmas$ for the whole core HOL Light takes at least three hours for the $Q_i$ and $EQ_i$ methods, the full evaluation on all 1,954 core HOL Light theorems would take about 2,000 CPU hours. That is why we further scale down this evaluation by doing it only for every tenth theorem in core HOL Light.

The *chained-conjecturing ATP* scenario is similar to the cheating scenario, but with limits imposed on the directly preceding lemmas. In $chain_1$*-conjecturing*, any (possibly directly preceding) lemma used to prove a theorem $i$ must itself have an ATP proof using only $OrigThms$. In other words, it is allowed to guess good lemmas that still do not exist, but such lemmas must not be hard to prove from $OrigThms$. Analogously for $chain_2$*-conjecturing* (resp. $chain_N$), where lemmas provable from $chain_1$-lemmas (resp. $chain_{N-1}$) are allowed to be guessed. To some extent, this scenario measures the theoretical ATP improvement obtainable with guessing of good intermediate lemmas.

## 9.7. Experiments

In total, we have performed experiments with 180 different strategies for adding new lemmas based on the kernel inference traces, and with 164 different strategies for adding new lemmas based on the tactical traces. The ATP experiments are done on the same hardware and using the same setup that was used for the earlier evaluations described in [KU14, KU13f]: All ATP systems are run with 30s time limit on a 48-core server with AMD Opteron 6174 2.2 GHz CPUs, 320 GB RAM, and 0.5 MB L2 cache per CPU.

In order to find the exact HOL formulas corresponding to the new lemmas (known only as nodes in a graph) coming from mining the kernel inference traces, we first have to process the formalization again with a patched kernel that takes the lemma numbers as a parameter and exports also the statements of the selected new lemmas. This is no longer necessary for the tactic data, since the formula statements can be stored together with the proof graph during the first run. The slowest part of our setup is computing the formula features needed for the machine learning. For the experiments with the kernel inference lemmas, the features of each final set of selected lemmas ($NewThms$) are computed independently, since we cannot pre-compute the features of all the lemmas in the kernel traces. In case of the Flyspeck tactical trace we can directly compute the features of all of the over 1 million lemmas. Due to their size (the intermediate lemmas are often large implications), it takes 28 hours to extract and normalize [KU14] all the features. The sum of the counts of such features over all these lemmas is 63,433,070, but there are just 383,304 unique features in these lemmas. Even for the extreme case of directly using and predicting premises for all the lemmas from the Flyspeck tactical trace without any preselection, our k-NN predictor can perform all the one million predictions in about 30 hours, taking 0.11s per prediction. Predictions are translated from the HOL logic into FOF problems [KU14] and ATPs are run on them in the usual way to make the evaluations.

In order to compare the new results with the extensive experimental results obtained over the previous versions of HOL Light and Flyspeck used in [KU14], we first detect the set of theorems that are preserved between the different versions. This is done by using the recursive content-based naming of symbols and theorems that we have developed for re-using as much information between different library versions in the HOL(y)Hammer online service [KU15]. In case of HOL Light the complete set of 1954 core HOL Light theorems evaluated in previous evaluations of HOL(y)Hammer has been preserved, only some of the names have been changed. In case of Flyspeck a smaller set of 10,779 theorems is preserved. In order to perform more experiments we further reduced the size of this set by choosing only every sixth theorem and evaluating the performance on the resulting 1796 theorems.

## 9.7.1. Evaluation on Core HOL Light

When using only the original theorems, the success rate of the 14 most complementary AI/ATP methods developed in [KU14] run with 30s time limit each and restricted to the 1954 core HOL Light theorems is 63.1% (1232 theorems) and the union of all those methods solved 65.4% (1278 theorems). In the very optimistic *cheating* scenario (limited only to the $Q_i$ metrics), these numbers go up to 76.5% (1496 theorems) resp. 77.9% (1523 theorems). As mentioned in Section 9.6, many proofs in this scenario may however be too simple because a close directly preceding lemma was used by the lemma-mining/machine-learning/ATP stack. This became easy to see already when using the *almost-honest* scenario, where the 14 best methods (including also $EQ_i$ and $PR_i$) solve together only 66.2% (1293 theorems) and the union of all methods solves 68.9% (1347 theorems). The performance of the various (almost-honest) new lemma-based methods is shown in Table 9.3, together with their comparison and combination with the old experiments.

| Strategy | Theorems (%) | Unique | Theorems |
|---|---|---|---|
| $Q_{1..3}$ (direct quality, sec. 9.5.1) | 62.897 | 68 | 1229 |
| $PR_{1..5}$ (PageRank, sec. 9.5.3) | 58.700 | 17 | 1147 |
| $EQ_{1..2}$ (epcllemma, sec. 9.5.2) | 57.011 | 4 | 1114 |
| $MC_{1..2}$ (graph cut, sec. 9.5.4) | 47.288 | 1 | 924 |
| total | 64.125 | | 1253 |
| only named | 54.452 | 0 | 1064 |
| total | 64.125 | | 1253 |
| HOL(y)Hammer (14 best) | 63.050 | 92 | 1232 |
| combined 14 best | 66.172 | | 1293 |
| total | 68.833 | | 1345 |

Table 9.3.: Comparison of the methods evaluated on the kernel traces on the 1954 HOL Light theorems

The majority of the new solved problems come from the alpha-normalized `TRACE2`, however the non-alpha normalized versions with and without duplicates do contribute as well. When it comes to the number of theorems added, adding more theorems seems to help significantly, see Table 9.4. We do not try to add more than 10,000 theorems for core HOL Light, as this is already much bigger than the size of the library. We will add up to one million theorems when looking at the whole Flyspeck in the next subsection.

For each of the strategies the success rates again depend on the different arguments that the strategy supports. In case of direct lemma computation considering $Q_1$ seems to give the best results, followed by $Q_2$ and $Q_3^{1,1}$; see Table 9.5. This suggest that focusing on either $U$ or $D$ is worse than looking at the combination. For core HOL Light size seems

| Added theorems | Success rate | Unique | Thms |
|---|---|---|---|
| `TRACE2` | 62.078 | 48 | 1213 |
| `TRACE0` | 59.365 | 12 | 1160 |
| `TRACE1` | 58.802 | 17 | 1149 |
| 10,000 | 63.562 | 138 | 1242 |
| 1,000 | 55.374 | 9 | 1082 |

Table 9.4.: Success rate depending on kind of trace used and depending on the number of added theorems

not to be an issue and dividing by size gives us best results. This will change in Flyspeck where the real arithmetic decision procedures produce much bigger intermediate lemmas.

| Lemma quality | Success rate | Unique | Thms |
|---|---|---|---|
| $Q_1$ $(\frac{U(i)*D(i)}{S(i)})$ | 58.751 | 21 | 1148 |
| $Q_2$ $(\frac{U(i)*D(i)}{S(i)^2})$ | 57.932 | 10 | 1132 |
| $Q_3^{1.1}$ $(\frac{U(i)*D(i)}{1.1^{S(i)}})$ | 57.523 | 8 | 1124 |
| $Q_3^{1.25}$ $(\frac{U(i)*D(i)}{1.25^{S(i)}})$ | 53.685 | 2 | 1049 |
| $Q_3^{1.05}$ $(\frac{U(i)*D(i)}{1.05^{S(i)}})$ | 52.866 | 0 | 1033 |
| $Q_2^2$ $(\frac{U(i)}{S(i)})$ | 52.456 | 4 | 1025 |
| $Q_3^{1.025}$ $(\frac{U(i)*D(i)}{1.025^{S(i)}})$ | 49.437 | 0 | 966 |
| $Q_1^2$ $(\frac{U(i)^2}{S(i)})$ | 49.437 | 8 | 966 |
| $Q_1^0$ $(\frac{D(i)^2}{S(i)})$ | 46.469 | 3 | 908 |
| $Q_2^0$ $(\frac{D(i)}{S(i)})$ | 44.882 | 1 | 877 |

Table 9.5.: Success rate of $Q_i$ depending on the quality formula.

In case of epcllemma three main strategies of creating a FOF trace from an inference trace were considered. First, we tried to apply the MESON translation of formulas. On one hand this was most computationally expensive as it involves lambda-lifting and introducing the apply functor, on the other hand it produces first-order formulas whose semantics are closest to those of the higher-order formulas involved. Second, we tried to create arbitrary FOF formulas of the same size as the one of the input HOL formula. Third, we modified the second approach to also initialize epcllemma with the already named theorems. The results can be found in Table 9.6. The size of theorems is much more important than the structure and initialization does not seem to help.

| Added theorems | Success rate | Unique | Thms |
|---|---|---|---|
| Preserve size | 55.732 | 15 | 1089 |
| Preserve size and initialize | 55.322 | 8 | 1081 |
| MESON translation | 47.339 | 11 | 925 |

Table 9.6.: Success rate of epcllemma depending on kinds of formulas given

We next compare the versions of PageRank. The intersection between the first 10,000 lemmas advised by $PR_1$ and $PR_2$ is 79%, which suggests that the lemmas suggested by $PR_1$ are already rather small. For the reverse PageRank it is the opposite: $PR_3$ and $PR_4$ have only 11% intersection. This makes the bigger lemmas suggested by $PR_3$ come out second after the normalized combined $PR_6$ in Table 9.7.

| Added theorems | Success rate | Unique | Thms |
|---|---|---|---|
| $PR_6$ $(\frac{PR_1(i)+PR_3(i)}{S(i)})$ | 53.173 | 22 | 1039 |
| $PR_3$ (reverse $PR_1$) | 52.968 | 13 | 1035 |
| $PR_5$ $(PR_1(i) + PR_3(i))$ | 52.252 | 14 | 1021 |
| $PR_2$ $(\frac{PR_1(i)}{S(i)})$ | 46.008 | 5 | 899 |
| $PR_4$ $(\frac{PR_3(i)}{S(i)})$ | 45.650 | 8 | 892 |
| $PR_1$ | 42.272 | 1 | 826 |

Table 9.7.: Success rate of PageRank depending on kinds of formulas given

The resource-intensive *fully-honest* evaluation is limited to a relatively small subset of the core HOL Light theorems, however it confirms the *almost-honest* results. While the original success rate was 61.7% (less than 14 methods are needed to reach it), the success rate with lemma mining went up to 64.8% (again, less than 14 methods are needed). This means that the non-cheating lemma-mining approaches so far improve the overall performance of the AI/ATP methods over core HOL Light by about 5%. The best method in the *fully-honest* evaluation is $Q_2$ which solves 46.2% of the original problems when using 512 premises, followed by $EQ_2$ (using the longest inference chain instead of $D$), which solves 44.6 problems also with 512 premises. The best PageRank-based method is $PR_2$ (PageRank divided by size), solving 41.4% problems with 128 premises.

An interesting middle-way between the cheating and non-cheating scenarios is the *chained-conjecturing* evaluation, which indicates the possible improvement when guessing good lemmas that are "in the middle" of long proofs. Since this is also quite expensive, only the best lemma-mining method ($Q_2$) was evaluated on the HOL Light TRACE2. $Q_2$ itself solves (altogether, using different numbers of premises) 54.5% (1066) of the problems. This goes up to 61.4% (1200 theorems) when using only $chain_1$*-conjecturing* and to 63.8% (1247 theorems) when allowing also $chain_2$ and $chain_3$*-conjecturing*. These are 12.6% and 17.0% improvements respectively, see Table 9.8.

| Length of chains | Success rate | Unique | Thms |
|---|---|---|---|
| - | 54.5 | 519 | 1066 |
| 1 | 32.0 | 75 | 627 |
| 2 | 12.2 | 30 | 239 |
| 3 | 2.3 | 12 | 46 |
| 4 | 1.1 | 4 | 22 |
| 5 | 0.3 | 4 | 6 |
| 6 | 0.3 | 4 | 6 |
| $> 6$ | 0.1 | 2 | 2 |
| Total | 64.6 | | 1264 |

Table 9.8.: Theorems found with chains of given lengths

## 9.7.2. Evaluation on Flyspeck

For the whole Flyspeck the evaluation is due to the sizes of the data limited to the tactical trace and the almost-honest scenario. Table 9.9 (the Flyspeck counterpart of Table 9.3) presents the performance of the various lemma-based methods on the 1796 selected Flyspeck theorems, together with the comparison and combination with the old experiments. The combination of the 14 best methods tested here solves 37.6% problems, and the combination of all methods solves 40.8% problems. When combined with the most useful old methods developed in [KU14], the performance of the best 14 methods is 44.2%, i.e., we get a 21.4% improvement over the older methods. The sequence of these 14 most-contributing methods is shown in Table 9.10.

| Strategy | Theorems (%) | Unique | Theorems |
|---|---|---|---|
| $PR_{1..5}$ (pagerank, sec. 9.5.3) | 36.860 | 39 | 662 |
| $Q_{1..3}$ (direct quality, sec. 9.5.1) | 35.913 | 31 | 645 |
| $MC_{1..2}$ (graph cut, sec. 9.5.4) | 30.178 | 1 | 542 |
| $EQ_{1..2}$ (epcllemma, sec. 9.5.2) | 29.677 | 0 | 533 |
| all lemmas | 21.047 | 26 | 378 |
| only named | 28.786 | 1 | 517 |
| 14 best | 37.584 | | 675 |
| total | 40.813 | | 733 |
| HOL(y)Hammer (14 best) | 36.414 | 127 | 654 |
| combined 14 best | 44.209 | | 794 |
| total | 47.884 | | 860 |

Table 9.9.: Comparison of the methods evaluated on the tactical trace and the 1796 Flyspeck theorems

| Strategy | Pred. | Feat. | Lemmas | Prem. | ATP | Success | Thms |
|---|---|---|---|---|---|---|---|
| HOL(y)Hammer | NBayes | typed, notriv | ATP-deps | 154 | epar | 24.666 | 443 |
| $MC_2$ | k-NN | typed | 1,000 lemmas | 128 | epar | 31.180 | 560 |
| All lemmas | k-NN | types | all lemmas | 32 | z3 | 34.855 | 626 |
| HOL(y)Hammer | NBayes | types, notriv | ATP-deps | 1024 | epar | 36.693 | 659 |
| $Q_1$ | k-NN | types | 60,000 lemmas | 32 | z3 | 38.474 | 691 |
| HOL(y)Hammer | NBayes | typed, notriv | ATP-deps | 92 | vam | 40.033 | 719 |
| Only Named | k-NN | types | - | 512 | epar | 40.980 | 736 |
| $Q_1^2$ | k-NN | types | 60,000 lemmas | 32 | z3 | 41.759 | 750 |
| HOL(y)Hammer | k-NN160 | types, notriv | ATP deps | 512 | z3 | 42.316 | 760 |
| $Q_1^0$ | k-NN | types | 60,000 lemmas | 32 | z3 | 42.762 | 768 |
| $PR_6$ | k-NN | types | 20,000 lemmas | 512 | epar | 43.207 | 776 |
| HOL(y)Hammer | NBayes | fixed | Human deps | 512 | epar | 43.541 | 782 |
| $PR_1$ | k-NN | types | 20,000 lemmas | 32 | z3 | 43.875 | 788 |
| $PR_4$ | k-NN | types | 20,000 lemmas | 128 | epar | 44.209 | 794 |

Table 9.10.: Combined 14 best covering sequence

There are several issues related to the previous evaluations that need explanation. First, the final 14-method HOL(y)Hammer performance reported in [KU14] was 39%, while here it is only 36.4%. The 39% were measured on the whole older version of Flyspeck, while the 36.4% here is the performance of the old methods limited to the 1796 problems selected from the set of 10,779 theorems that are preserved between the old and the new version of Flyspeck. Additionally, we have recently reported [KU13f] an improvement of the 39% performance to 47% by using better learning methods and better E strategies. However, that preliminary evaluation has been so far done only on a smaller random subset of the old Flyspeck, so we do not yet have the corresponding data for all the 10,779 preserved theorems and their 1796-big subselection used here for the comparison. A very rough extrapolation is that the 47% performance on the smaller subset will drop to 45% on the whole old Flyspeck, which when proportionally decreased by the performance decrease of the old methods (39/36.4) yields 42% performance estimate on the new 1796-big set. Third, we should note that the new lemma-based methods are so far based only on learning from the ITP (human-proof) dependencies, which is for Flyspeck quite inferior to learning on the dependencies extracted from minimized ATP proofs of the problems. Fourth, we do use here the (one) best predictor and the ATP strategies developed in [KU13f], however, we have not so far explored and globally optimized as many parameters (learners, features and their weightings, premise slices, and ATP strategies) as we have done for the older non-lemma methods; such global optimization is future work.

So while the 21.4% improvement over [KU14] is valid, a full-scale evaluation of all the methods on the whole new Flyspeck[9] will likely show a smaller improvement due to the lemma-mining methods. A very conservative estimate is again 5% (44.2%/42%), however a much more realistic is probably 10%, because the effect of learning from ATP proofs is quite significant. Higher lemma-based performance on Flyspeck than on the core HOL Light is quite plausible: the core HOL Light library is much smaller, more stable and optimized, while Flyspeck is a fast-moving project written by several authors, and the library structuring there is more challenging.

As expected the graph cutting method ($MC$) does indeed produce the smallest dependency graph passed to the predictors. For 10,000 added lemmas the average number of edges in the $MC$-produced dependency graph is 37.0, compared with the average over all strategies being 42.9 dependencies per theorem and epcllemma producing graphs with the biggest number: 63.1 dependencies. This however does not yet correspond to high success rates in the evaluation, possibly due to the fact that graph cutting does not so far take into account the number of small steps needed to prove the added lemma. On the other hand, Table 9.10 shows that graph cutting provides the most complementary method, adding about 25% more new solutions to the best method available.

Finally we analyze the influence of the number of added lemmas on the success rate in

[9]Such evaluation could take another month with our current resources.

| Added lemmas | Theorems (%) | Unique | Theorems |
|---|---|---|---|
| 60,000 | 36.804 | 37 | 661 |
| 20,000 | 35.523 | 18 | 638 |
| 10,000 | 33.463 | 3 | 601 |
| 0 | 28.786 | 1 | 517 |
| 5,000 | 27.951 | 0 | 502 |
| 1,000 | 27.895 | 0 | 501 |
| all | 21.047 | 26 | 378 |

Table 9.11.: Influence of the number of added lemmas on the success rate

Table 9.11. As expected adding more lemmas does improve the general performance up to a certain point. The experiments performed with all the lemmas added are already the weakest. However, when it comes to the problems solved only with a certain number of lemmas added, using all the lemmas comes out complementary to the other numbers.

### 9.7.3. Examples

We have briefly looked at some first examples of the problems that can be solved only with the lemma-based methods. So far we have detected two main effects how such new proofs are achieved: (i) the new lemma (or lemmas) is an easy-but-important specialization of a general theorem or theory, either directing the proof search better than its parents or just working better with the other premises, and (ii) no new lemma is needed, but learning on the newly added lemmas improves the predicting systems, which then produce better advice for a previously unsolvable problem. The second effect is however hard to exemplify, since the number of alternative predictions we tried is high, and it usually is not clear why a particular prediction did not succeed. An example in the first category is the theorem

```
AFFINE_ALT: ⊢  affine s ⟺ (∀x y u. x IN s ∧ y IN s ⟹
(&1 − u) % x + u % y IN s)
```

which E can prove using 15 premises, three of them being new lemmas that are quite “trivial” consequences of more general theorems:

```
NEWDEP309638: ⊢  &1 − a + a = &1
NEWDEP310357: ⊢  − &1 * − &1 = &1
NEWDEP272099_conjunct1: ⊢  ∀m. &m + − &m = &0
```

Another example in the first category is theorem

```
MEASURABLE_ON_NEG: ⊢  ∀f s. measurable_on f s ⟹ measurable_on (\x. −
f x) s
```

whose proof uses a few basic vector facts plus one added lemma:

```
NEWDEP1643063: measurable_on f s ⊢  measurable_on ((%) c o f) s
```

This lemma appeared in the proof of the close theorem

```
MEASURABLE_ON_CMUL: ⊢  ∀c f s. measurable_on f s ⟹
measurable_on (\x. c % f x) s
```

The lemma here is almost the same as the theorem where it was first used, but it likely works better in the FOF encoding because the lambda function is eliminated.

## 9.8. Future Work and Conclusion

We have proposed, implemented and evaluated several approaches that try to efficiently find the best lemmas and re-organize a large corpus of computer-understandable human mathematical ideas, using the millions of logical dependencies between the corpus' atomic elements. We believe that such conceptual re-organization is a very interesting AI topic that is best studied in the context of large, fully semantic corpora such as HOL Light and Flyspeck. The byproduct of this work are the exporting and post-processing techniques resulting in the publicly available proof graphs that can serve as a basis for further research.

The most conservative improvement in the strength of automated reasoning obtained so far over the core HOL Light thanks to lemma mining is about 5%. The improvement in the strength of automated reasoning obtained over Flyspeck problems is 21.4% in comparison to the methods developed in [KU14], however this improvement is not only due to the lemma-mining methods, but also due to some of the learning and strategy improvements introduced in [KU13f]. A further large-scale evaluation using learning from ATP proofs and global parameter optimization is needed to exactly measure the contribution and overall strength of the various AI/ATP methods over the whole Flyspeck corpus.

There are many further directions for this work. The lemma-mining methods can be made faster and more incremental, so that the lemma quality is not completely recomputed after a lemma is named. Fast PageRank-based clustering should be efficiently implemented and possibly combined with the other methods used. ATP-style normalizations such as subsumption need to be correctly merged with the detailed level of inferences used by the HOL Light proof graph. The existing ITP proof-reconstruction methods [SB13,KU13e] will need to be updated to handle not just the top-level theorems, but also the intermediate lemmas. Guessing of good intermediate lemmas for proving harder theorems is an obvious next step, the value of which has already been established to a certain extent in this work.

## Acknowledgments

We would like to thank Stephan Schulz for help with running epcllemma, Yury Puzis and Geoff Sutcliffe for their help with the AGInt tool and Jiří Vyskočil and Petr Pudlák for many discussions about extracting interesting lemmas from proofs.

# 10 Automated Reasoning for Mizar

## Publication Details

Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *J. Automated Reasoning*, in press, 2015. doi: `10.1007/s10817-015-9330-8`.

## Abstract

We develop an AI/ATP system that in 30 seconds of real time on a 14-CPU machine automatically proves 40% of the theorems in the latest official version of the Mizar Mathematical Library (MML). This is a considerable improvement over previous performance of large-theory AI/ATP methods measured on the whole MML. To achieve that, a large suite of AI/ATP methods is employed and further developed. We implement the most useful methods efficiently, to scale them to the 150000 formulas in MML. This reduces the training times over the corpus to 1–3 seconds, allowing a simple practical deployment of the methods in the online automated reasoning service for the Mizar users (Miz$\mathbb{AR}$).

## 10.1. Introduction and Motivation

Since 2003 the Mizar Mathematical Library[1] (MML) has been used as a repository for developing AI/ATP methods for solving formally stated (computer-understandable) conjectures in general large-theory mathematics [Urb06c, Urb04, Urb03]. By *large theories* we mean theories with many concepts, definitions, theorems and lemmas, over which many related conjectures are posed, and where it is not immediately clear which of the previous facts will (not) be useful for a proof of a conjecture. The number and strength of the methods developed has been growing, however the methods were often developed and evaluated on smaller benchmarks such as the MPTP Challenge[2] and MPTP2078 [AHK$^+$14]. Recently, we have tried to develop a strong suite of AI/ATP methods that scale to the whole June 2012 version of the Flyspeck [Hal12, HAB$^+$15] development [KU14,KU13f], containing more than 14000 theorems. The best methods using the accumulated data have been recently deployed in an online ("cloud-based") AI/ATP service for HOL Light [Har96a] formalizations [KU13a, KU15]. When running the 14

[1] `http://www.mizar.org`

[2] `http://www.cs.miami.edu/~tptp/MPTPChallenge/`

strongest methods in parallel, 47% of the Flyspeck theorems can be proved in 30 seconds without any user interaction. To a significant extent, this performance is achieved by learning from the large number of previous proofs various high-level [Urb11a, KvLT$^+$12] and low-level [Urb14, KU13f] guiding methods for state-of-the-art ATP and SMT systems such as Vampire [KV13], E [Sch02] and Z3 [dMB08]. A similar work has been recently undertaken with Isabelle [KBKU13].

We believe that this performance is a milestone on the way to John McCarthy's AI and QED dream of "Heavy Duty Set Theory", i.e., a sufficiently smart AI/ATP/ITP system that can without forcing mathematicians to struggle with various current technologies of explicit "proof programming" automatically understand and check reasoning steps done on the level that is commonly used in mathematical proofs. If such a system is developed, the current barrier preventing computer understanding of common mathematical proofs will to a large extent disappear, and mathematics (and thus all human exact thinking) may enter an era of ubiquitous computer understanding and strong AI assistance.

In this work we employ, further develop, and evaluate a suite of scalable AI/ATP methods on the whole Mizar library, containing nearly 58000 theorems. The main experimental result (Section 10.3) is that the 14 strongest methods run in parallel for 30 seconds prove 40.6% of the 58000 Mizar theorems without any user interaction. If users are also allowed to manually select the relevant premises used then by the ATPs, the performance grows to 56.2%. Our hope is that this performance may significantly lower the barrier to formalizing mathematics in the Mizar system [GKN10], which has a long history of targeting mathematicians with its standard logical foundations, intuitive proof style, and linguistic closeness to mathematical vernacular. The various methods used to achieve this performance are described in Section 10.2. Section 10.3 discusses the experiments and results obtained with the methods, Section 10.4 takes a brief look at the data obtained from the automatically found proofs, Section 10.5 briefly describes the first integration of the methods in the Miz$\mathbb{AR}$ online service [URS13], and Section 10.6 discusses future work and concludes.

## 10.2. Learning Proof Guidance From the MML

The general idea behind the large-theory ATP-for-ITP systems that started to be developed in the last decade is to combine (i) translations between the ATP and ITP formalisms with (ii) high-level premise selection methods [AHK$^+$14] and (iii) state-of-the-art ATP systems which can be further strengthened and tuned in various ways for the large-theory setting. For the translation from the Mizar logic to TPTP we re-use the existing MPTP translation [Urb06c, Urb04]. After several initial experiments with various ATPs and their versions, we have decided to limit the set of ATPs to Vampire 3.0, Z3 4.0, and E 1.8 run using the Epar scheduler and strategies [Urb14]. This combination also worked well for the experiments with Flyspeck. In this work, the main focus is on (ii), i.e., on deploying and improving for MML the suite of scalable high-level premise selection methods which we have recently developed for the whole Flyspeck corpus, containing over 20000 formulas. Here *premise selection* is the task of choosing a subset of all available facts (premises), which is most likely to lead to a successful automated deduction proof of a given conjecture.

The currently strongest premise selection methods for large-theory mathematics are *data-driven* [STC04]: instead of explicit programming of all aspects of knowledge selection, data-driven methods extract (learn) significant parts of such complicated algorithms from the existing large libraries of solutions (proofs). This shift from explicit (*theory-driven*) programming of AI heuristics to learning AI algorithms from data is to a large extent responsible for the recent successes in AI domains such as web search, consumer choice prediction, autonomous car driving, etc. But this also means that extracting good training data from the MML is equally important as the methods that learn premise selection on such data. Interestingly, in large-theory ATP there is a full positive feedback loop [Urb07] between the amount/quality of the data and the strength of the methods: not only more/better data produce stronger methods (which is the standard data-driven argument), but also stronger proving methods produce more/better data in the form of proofs. This is quite a unique property of this very expressive and fully semantic AI domain, born quite recently thanks to the development on large formal mathematical libraries such as the MML [UV13]. The main body of our work thus consists of the following steps:

1. Obtaining from the MML suitable data (proof dependencies) on which premise selection methods can be trained.

2. Developing, training and testing such premise selection methods and their parameters on a small random subset of the MML.

3. Iterating steps (1) and (2), i.e., using the most successful methods to get more proofs, and training further methods on them.

### 10.2.1. Obtaining Proof Dependencies from the MML

There are 57897 Mizar theorems and unnamed toplevel lemmas in the most recent official MML 4.181.1147. This set is canonically (chronologically) ordered by the MML order of articles, and by the (chronological) order of theorems in the articles. This ordering also applies to the about 90000 other Mizar formulas (typically encoding the type system and other automations known to Mizar) used in the problems. Our goal is to prove automatically as many of the 57897 theorems as possible, using at each point all the available formulas and all information about previous theorems and their proofs.

The human-written Mizar proofs contain explicit information about the theorems and definitions used. This information is however incomplete. For re-playing the Mizar proofs with ATPs, a lot of "background" knowledge (typically about typing of terms) needs to be explicitly added. The MPTP system adds such background formulas heuristically in a fixpoint manner, by looking at the set of symbols in the problem and adding the appropriate typing formulas. The average size of an ATP problem constructed in this way by MPTP is 328 formulas, while the average number of the explicit Mizar proof references is only 12. In [AHK$^+$14] we have constructed a computationally expensive method (using the Mizar checker) that reduces the number of the background formulas 2–3 times. However, the measured performance gain from that method when re-proving with ATPs the MPTP2078 problems was only about 4%, and in [AKU12] it was found that the ATPs still typically do not use many of the Mizar-needed background formulas. Since learning from the minimized ATP proofs is typically superior [KvLT$^+$12, KU14], in the current work we decided to skip the expensive Mizar-based proof minimization, and focus on using the ATP proofs of the heuristically constructed re-proving problems coming from the Mizar theorems. This decision was influenced by older preliminary 20-second testing using an Intel Xeon 2.67 GHz server, in which Vampire 1.8 proved 20302 of the MPTP-constructed theorem problems, Epar proved 20324, and together they proved 23141, i.e., 40% of the theorems.

Table 10.1 shows the gradual growth of the set of ATP-computed proof dependencies that we mainly use for learning. The first set is obtained by running Vampire 3.0 for 300 seconds[3] on the MPTP-constructed re-proving problems. The additional 1677 solutions in the second set are obtained by learning premise selection on the first set, and running ATPs for 120 seconds again on various most relevant slices of the re-proving problems (we always include the explicit Mizar references in such pruned ATP problems). The following passes no longer prune the MPTP-constructed re-proving problems. They just use the premise selectors trained on the previous passes to suggest the most relevant premises, regardless of the original Mizar proofs, and re-learn from the newly obtained proofs in several iterations as in the MaLARea system [Urb07, USPV08]. Such iterations are also quite expensive to compute for a new development. However, we have shown in [KU15] that this information can be in large libraries efficiently re-used and does not need to be computed for every version again. As in [KU14], the difference to the original MaLARea iterations is that at each premise-selection point only the chronologically previous proofs are used for learning. This corresponds to the ultimate deployment scenario, when always only the library proofs written so far are known.

In total, these iterations yield 32165 ATP proofs, and with the final evaluation described in Section 10.3.2 this number reaches 32557 theorems. This means that when using either human or AI-based premise selection and their combinations, state-of-the-art ATPs are today able to prove 56.23% of the toplevel MML theorems. This is a very good motivation for developing good premise-selection methods.

[3]The time limit of 300 seconds has worked well in the previous experiments done over Flyspeck [KU14]. Increasing the time limit further does not help significantly and it costs a lot of resources.

| Pass | ATPs | Premise Selection | Theorems (%) | Dependencies |
|---|---|---|---|---|
| 1 | V 300s | MPTP re-proving | 27842 (48%) | 27842 |
| 2 | V, E, Z3 120s | trained on (1), premises limited by (1) | 29519 (51%) | 30024 |
| 3 | V, E 120s | trained on (2), premises unlimited | 30889 (53.4%) | 31720 |
| 4 | V, E 120s | trained on (3), premises unlimited | 31599 (54.6%) | 32976 |
| 5 | V, E 120s | trained on (4), premises unlimited | 32010 (55.3%) | 35870 |
| 6 | V, E 120s | trained on (5), premises unlimited | 32165 (55.6%) | 36122 |

Table 10.1.: Improving the dependency data used for training premise selection

### 10.2.2. Premise Selection Techniques

The premise selection techniques we start with, are the relatively fast scalable methods used for Flyspeck in [KU14]: Naive Bayes (`nb`) and distance-weighted k-nearest neighbor [Dud76] (`knn`). In particular, a family of differently parametrized k-NNs together with the IDF (inverse document frequency) feature weighting scheme [Jon72] have recently provided quite significant performance improvement [KU13f]. This is here extended to naive Bayes (`nb_idf`). We are interested both in the strongest possible methods, and also in methods that can be quite weak, but complement well the stronger methods.

Apart from minor implementational modifications, we characterize each formula with the syntactic features used by MaLARea: symbols, terms and subterms of the formula. In the most successful methods, all variables in such features are renamed to just one variable `A0` (widening the similarity relation), however to a smaller extent, also the features with original variables are useful. Following the recent successful use by MaLARea 0.5 in the 2013 CASC LTB competition [KUV14] we also add a version of distance-weighted k-NN using the Latent Semantic Indexing [DDL$^+$90] (LSI) preprocessing of the feature space done efficiently by the gensim [ŘS10] toolkit. We test the LSI preprocessing with 800, 3200, and 6400 topics (`lsi_800` .. `lsi_6400`), and also versions with and without the TF-IDF feature scaling (e.g., (`lsi_3200ti`)).

The next modification of k-NN are various recursive schemes for weighting the neighbors' dependencies. The `geo_1_F` version stops the dependency recursion at the first level, weighting each dependence of a neighbor $N$ by the factor $F * distance(N)$ (where $F \in (0, 1)$), and taking maximum (instead of sum) of such weights over all neighbors. The `geo_r_F` version does full dependency recursion, weighting the indirect dependencies by $F^{recursionlevel} * distance(N)$, and again taking maximum over all such factors.

In [KvLT$^+$12] a linear combination of the strongest learning method with the SInE [HV11] heuristic produced very good results. This is an instance of *ensemble learning* where multiple base methods are combined into stronger classifiers. We heuristically explore combinations of the various base methods using various weighting schemes. In addition to the linear combination, we try geometric, harmonic, and quadratic average, and also use minimum and maximum of ranks. In particular taking the minimal rank (`comb_min`) and the geometric average (`comb_geo`) of ranks (computed as additions of logarithms) turn out to be quite successful. This can be explained in various ways, for example, taking the geometric average is the correct way of averaging ratios. Since ATPs are very (probably exponentially) sensitive to the number of axioms, treating the particular aggregated rankings as ratios is quite likely fitting to our domain (e.g., the ratio between 50th and 60th premise is 1.2, while the ratio between 10th and 20th premise is 2, whereas the linear distance is the same in the two cases).

We also try several methods of *boosting* [Sch90]: using for training of the next method only those proof dependencies that are badly predicted by the previous methods. While we believe that there are good reasons why this approach should help (e.g., our current methods being quite simple and thus hard to fit to more complicated ideas), so far this has not provided significant improvements.

All the tested methods[4] (apart from LSI (gensim) and SInE (E prover)) are now uniformly implemented in OCaml, which gives significant speedup over the initial Perl implementation on the large number of features, labels and examples used when training over the whole MML (the number of features reaches several hundred thousand). The most useful methods are further implemented in C++, making them about twice as fast as their OCaml version. A particularly useful low-level optimization is the use of partial sorting (based on heapsort) of the scores according to the number of premises demanded from a particular premise-selection method. For example, if only 128 premises are needed, the partial sorting is much more efficient than full sorting of the whole array of 150000 MML formulas.

### 10.2.3. ATPs and Their Low-Level Guidance

No particular development of ATP strategies was done for this work on the whole MML. However, thanks to the recent CASC competitions containing Mizar divisions (Mizar@Turing12, CASC LTB 2013) the recent versions of Vampire and Epar seem to be tuned well for MPTP2078. In particular, a set of strong strategies for E has been automatically developed by BliStr [Urb14] in 2012 on the 1000 Mizar@Turing12 problems, raising the performance on MPTP2078 over E's auto-mode by 25%. A second round of such strategy evolution on these 1000 problems was done for MaLARea 0.5 in CASC LTB 2013, where additionally a number of strong SInE strategies were evolved. Vampire 3.0 is on MML 16% stronger than Vampire 2.6.

[4]For their details see `http://cl-informatik.uibk.ac.at/users/cek/mizAR/legend.txt`

## 10.3. Experiments and Results

### 10.3.1. Random Subset of 1930 Problems

Most of the experimental research was done on a random subset of MML consisting of 1930 theorems (more precisely, every 30th theorem was used). For each of these theorems, the premise selection methods were trained on all the preceding proof data (Section 10.2.1), and chosen numbers (32, 64, 96, 128, 256, 512 and 1024) of the best-ranked premises were given to the ATPs. Most of the experiments during the development of the premise selection methods were done with Vampire 3.0. The final experiments were extended to Epar and Z3. As in [KU14], the systems were run with a 30 second time limit on a 48-core server with AMD Opteron 6174 2.2 GHz CPUs, 320 GB RAM, and 0.5 MB L2 cache per CPU. In real time, each evaluation thus took 2–3 hours for one ATP. In total, 70 different premise-selection methods have been tried on the random sample, see our web page for a detailed listing.[5]

Table 10.2 shows the fourteen methods and their parameters that performed best on the 1930-subset. The Σ-SOTAC (State of the art contribution) is the sum of a system's SOTAC over all problems attempted, where for each problem solved by a system, its SOTAC for the problem is the inverse of the number of systems in our evaluation that solved this problem. This metric shows how useful a particular method is in a collection of other methods (in this case all the 70 methods that have been tried).

Table 10.3 shows the 14 methods that collectively (when computed in a greedy way)[6] cover as many of the 1930 problems as possible. To be as orthogonal as possible, the methods in this set differ a lot in their parameters, the data trained on, and the number of best premises given to the ATP. Their joint performance on this subset is 44%. All the 70 tested methods together solve 968 of the 1930 problems, i.e., 50.155% .

### 10.3.2. Whole MML

When the developed methods on the 1930-subset reached sufficiently high joint performance, we evaluated the most useful 14 methods on the whole MML, again with a 30-second time limit. This took about one week of real time on our server. The performance of these methods is shown in Table 10.4. The methods are ordered there from top to bottom already by their position in the greedy covering sequence for the whole MML. The table says that running these fourteen methods in parallel for 30 seconds gives a 40.6% chance of solving an MML theorem without any user interaction. The best previous result in such fully automated learning/proving over the whole MML was 18%,

[5]`http://cl-informatik.uibk.ac.at/users/cek/mizAR/5yp.html`

[6]Such *greedy (covering) sequence* of methods starts with the best method, and each next method in such sequence is the one that greedily adds most solutions to the union of solutions of the previous methods in the sequence.

| Method | Parameters | Premises | ATP | Theorem (%) | Σ-SOTAC |
|---|---|---|---|---|---|
| comb | min_2k_20_20 | 128 | Epar | 550 (28.50) | 2.41 |
| comb | geo_3k_50_00 | 96 | V | 544 (28.19) | 1.94 |
| lsi | 3200ti_8_80 | 128 | Epar | 537 (27.82) | 2.17 |
| comb | geo_1k_33_33 | 96 | V | 535 (27.72) | 1.60 |
| comb | geo_10k_33_33 | 96 | V | 533 (27.62) | 1.73 |
| comb | geo_3k_20_20 | 96 | V | 533 (27.62) | 1.71 |
| comb | geo_3k_33_33 | 96 | V | 533 (27.62) | 1.63 |
| comb | geo_3k_25_25_25 | 96 | V | 532 (27.57) | 1.62 |
| comb | har_2k_k200_33_33 | 256 | Epar | 532 (27.57) | 2.19 |
| comb | geo_3k_33_33 | 96 | V | 531 (27.51) | 1.69 |
| comb | geo_3k_50_00 | 128 | V | 531 (27.51) | 1.76 |
| comb | geo_1k_33_33 | 128 | V | 529 (27.41) | 1.68 |
| knn | is040 | 128 | V | 528 (27.36) | 1.85 |
| knn | is_40 | 96 | Epar | 528 (27.36) | 2.40 |

**Theorem (%):** Number and percentage of theorems proved by a system. **Σ-SOTAC:** See the explanatory text for this metric.

Table 10.2.: 14 best premise selection methods on the 1930-subset

| Method | Parameters | Deps | Premises | ATP | Sum % | Sum |
|---|---|---|---|---|---|---|
| `comb` | min_2k_20_20 | ATP6 | 128 | Epar | 28.497 | 550 |
| `comb` | qua_k200_3k_33_33 | ATP4 | 512 | V | 32.798 | 633 |
| `comb` | geo_3k_33_33 | ATP3 | 64 | V | 35.959 | 694 |
| `lsi` | 3200ti_8_80 | ATP6 | 128 | Z3 | 37.461 | 723 |
| `geo` | r_99 | ATP6 | 64 | V | 38.653 | 746 |
| `knn` | 200 | ATP1+Mizar | 1024 | V | 39.741 | 767 |
| `nb` | idf010 | ATP6 | 128 | Epar | 40.725 | 786 |
| `comb` | min_20_20 | ATP2 | 128 | V | 41.347 | 798 |
| `comb` | geo_3k_50_00 | ATP3 | 1024 | V | 41.969 | 810 |
| `knn` | is040 | ATP1 | 1024 | Epar | 42.487 | 820 |
| `knn` | is_40 | ATP6 | 96 | Z3 | 43.005 | 830 |
| `geo` | 1_66 | ATP3 | 1024 | V | 43.420 | 838 |
| `lsi` | 6400_8_120 | ATP1 | 64 | V | 43.782 | 845 |
| `geo` | 1_33 | ATP3 | 256 | V | 44.093 | 851 |

Table 10.3.: The top 14 methods in the greedy sequence on the 1930-subset

achieved in [AKU12] by running Vampire 0.6 for 20 seconds (using about twice as fast Intel Xeon machine than our AMD server) on 200 best-ranked premises proposed by the SNoW system using the Naive Bayes learner. Since this was just a single method, a fair comparison is with the best method developed here, which solves 27.3%, i.e. 50% more problems. One of the reasons for this improvement are obviously the better training data developed here by the six MaLARea-style proving/learning passes over the MML.

It should be however noted that much better results than 18% have been achieved on smaller benchmarks such as MPTP2078, where more expensive methods such as kernel-based learning [KvLT$^+$12] could be applied. Comparison with those results is however possible only in a high-level way: we use different MML version here, different versions of the ATPs, coarser slices of the best premises, and we do not limit the premises only to those available in the 33 articles used for MPTP2078. The best result on MPTP2078 reported in [KU13g] was 823 problems (out of 2078) solved with 70 premises, Vampire 0.6 and 5s on an Intel Xeon machine. The best new performance on the 2061 problems corresponding to the 33 MPTP2078 articles in the current MML is 1059 problems solved in 30 seconds by Epar using 128 best premises.[7] To make a bit closer comparison, we test the current best-performing method on the 2061 problems also with the old Vampire 0.6 and 5 seconds on the old Intel machine, solving 726 problems. This is practically the same result as the performance of the best old kernel-based method (combined with SInE) on the MPTP2078 benchmark when using 128 premises. This seems to be an evidence (modulo all the differences named above) that the methods based on fast scalable learning techniques such as k-NN can with enough care catch up with the existing kernel-based techniques. Quite likely, this is however not the last word, and we hope to get further improvements by scaling up and strengthening the kernel-based and related methods.

## 10.4. Proofs

We have briefly compared the shortest ATP proofs found with the corresponding MML proofs. For this, we only consider the 28892 named Mizar theorems for which we have obtained either a human or AI-advised ATP proof. The complexity metric used for a human-written Mizar proof is just the number of proof lines in the Mizar article, while for the ATP proofs we use the number of dependencies.[8] These statistics, sorted by the largest difference between these metrics is available online[9], together with the ATP dependencies used for this comparison[10]. For example the first entry says that the ATP proof of the theorem `REARRAN1:24`[11] has a 534-lines long Mizar proof, while the shortest

[7]The detailed results restricted to the 2061 problems are at `http://cl-informatik.uibk.ac.at/users/cek/mizl/mptp2k.html`.

[8]These choices can obviously be questioned, but as a first comparison they are useful enough.

[9]`http://mizar.cs.ualberta.ca/~mptp/mml4.181.1147/html/00prdiff15.html`

[10]`http://mizar.cs.ualberta.ca/~mptp/mml4.181.1147/html/00atpdeps`

[11]`http://mizar.cs.ualberta.ca/~mptp/mml4.181.1147/html/rearran1.html#T24`

| `Method` | Parameters | Prems. | ATP | Σ-SOTAC | Theorem (%) | Greedy (%) |
|---|---|---|---|---|---|---|
| `comb` | min_2k_20_20 | 128 | Epar | 1728.34 | 15789 (27.3) | 15789 (27.2) |
| `lsi` | 3200ti_8_80 | 128 | Epar | 1753.56 | 15561 (26.9) | 17985 (31.0) |
| `comb` | qua_2k_k200_33_33 | 512 | Epar | 1520.73 | 13907 (24.0) | 19323 (33.4) |
| `knn` | is_40 | 96 | Z3 | 1634.50 | 11650 (20.1) | 20388 (35.2) |
| `nb` | idf010 | 128 | Epar | 1630.77 | 14004 (24.2) | 21057 (36.4) |
| `knn` | is_80 | 1024 | V | 1324.39 | 12277 (21.2) | 21561 (37.2) |
| `geo` | r_99 | 64 | V | 1357.58 | 11578 (20.0) | 22006 (38.0) |
| `comb` | geo_2k_50_50 | 64 | Epar | 1724.43 | 14335 (24.8) | 22359 (38.6) |
| `comb` | geo_2k_60_20 | 1024 | V | 1361.81 | 12382 (21.4) | 22652 (39.1) |
| `comb` | har_2k_k200_33_33 | 256 | Epar | 1714.06 | 15410 (26.6) | 22910 (39.6) |
| `geo` | r_90 | 256 | V | 1445.18 | 13850 (23.9) | 23107 (39.9) |
| `lsi` | 3200ti_8_80 | 128 | V | 1621.11 | 14783 (25.5) | 23259 (40.2) |
| `comb` | geo_2k_50_00 | 96 | V | 1697.10 | 15139 (26.1) | 23393 (40.4) |
| `geo` | r_90 | 256 | Epar | 1415.48 | 14093 (24.3) | 23478 (40.6) |

Table 10.4.: 14 most covering methods on the whole MML, ordered by greedy coverage.

ATP proof found has only 5 dependencies. Indeed, this greatest AI/ATP-found proof shortening is valid, thanks to a symmetry between the concepts used in this theorem and a previously proved theorem `REARRAN1:17`[12] which can be established quite quickly from the concepts' definitions. The Mizar proof instead proceeds by repeating the whole argument from scratch, modifying it at appropriate places to the symmetric concepts. The AI/ATP toolchain has thus managed to succinctly express the difference between the two theorems in a very explicit and operational way, while the human authors probably were on some level also aware of the symmetry, but were not able to capture it so precisely and succinctly. In some sense, the AI/ATP system has thus managed to find, precisely formulate, and productively use a new mathematical trick.

This comparison, showing such most striking shortenings, is also useful for heuristic checking of the correctness of the whole translation/AI/ATP toolchain. By random inspection of a dozen of such shortenings, no suspicious proofs were found, i.e., all the inspected ATP proofs could be replayed in Mizar. On the other hand, some of the ATP proofs can get very long, and may be probably already quite hard to understand without further refactoring and presentation methods.

Finally, the Mizar proof length expressed in terms of the lines of code can also serve as another metric for measuring the performance of the ATP methods. The total number of the Mizar source lines used for the proofs of the 52248 named toplevel theorems is 1297926. The sum of the Mizar proof lines of the 28892 named theorems that were proved automatically (either from human or AI-selected premises) is 300914. This means that on average 23.2% of the proof lines can be "written automatically", if such automation is called on the toplevel named theorems. This is a metric that in some sense complements the 56.23% ratio obtained in Section 10.2.1, showing that the ATPs are much better in proving the Mizar-easy theorems. On the other hand, the 23.2% average would be obviously improved a lot if also the proof-local lemmas were included in the experiments, and the number of lines corresponding to such lemmas was appropriately included in the statistics.

## 10.5. Integration with Miz$\mathbb{AR}$

The new optimized C++ versions of the premise selectors are sufficiently fast to train on the whole MML in 1–3 seconds. This simplifies the integration of the methods in the Miz$\mathbb{AR}$ service. For each query, the premise selectors are always first trained on the whole MML and also on the features and proof dependencies added from the current article.[13] After such training, the premise selectors are presented with the conjecture features, to which they respond by ranking the available theorems according to their relevance for

[12]`http://mizar.cs.ualberta.ca/~mptp/mml4.181.1147/html/rearran1.html#T17`

[13]In the Miz$\mathbb{AR}$ service, the conjecture is always submitted with the whole Mizar article in which the conjecture is stated.

the conjecture. The several premise selection methods with their corresponding ATPs are run in parallel, and if successful, the result is communicated to the user. The main MizAR server (Intel Xeon 2.67 GHz) is considerably faster than the AMD machines used for most of the experiments.

The service thus always updates itself with new data: the conjecture is always a part of a particular Mizar article, which is submitted as a whole to the system. However, in comparison with the recently produced HOL Light service (HOL(y)Hammer), the updating is so far more limited. We do not yet try to get (for better training) the minimized ATP proofs of the article's theorems that precede the current conjecture. One reason is that, unlike in HOL(y)Hammer, the MizAR service allows anonymous uploads of whole articles, but does not yet keep such projects persistent. Adding such persistence should make the computing and minimization of ATP proofs less expensive, because such data can then be quite efficiently cached and re-used (see Section 3 of [KU15]).

Another difference to the HOL Light setting is the very common use of local constants (eigenvariables) in the Jaśkowski-style Mizar proofs. The large-scale experiments (and thus also the training data obtained from them) presented here only deal with the set of toplevel Mizar theorems which do not contain such proof-local constants. This has two different effects when proving the proof-local lemmas that contain such constants:

1. The local assumptions and lemmas about such constants are naturally preferred by the premise selectors (in particular when using weighting schemes such as TF-IDF [KU13f]), because such local constants (which always have a distinct internal name) and the terms containing them are rare. This is good, because such local lemmas are typically quite relevant to the local conjecture.

2. The feature representation of the proof-local lemmas may become quite distant (in the various metrics used by k-NN) from the general theorems that are needed to justify such lemmas, because many terms in the lemmas are instantiated with the local constants. This may be a serious problem, preventing finding the relevant theorems.

We use a simple method to counter (2): We generalize all local constants in such proof-local conjectures to variables. The term features of such generalized versions of the lemmas are used together with the standard term features. This is clearly just a first step, the general task of getting features that indicate for example the (lack of) the instantiation relationship between two formulas is quite interesting, and various syntactic and semantic methods are possible [KUV15]. Further experiments and evaluation of such issues, as well as of the user-perceived strengthening of the MizAR service are left as future work.

## 10.6. Conclusion, Future Work, and Thanks

The main result of this work is the 40.6% success rate in proving the toplevel Mizar theorems fully automatically. This has been achieved by several iterations of implementing better premise selection methods, using them to obtain better training data, and using such data to further improve the performance of the learning methods. The methods were implemented very efficiently, allowing their easy deployment in the Miz$\mathbb{AR}$ service. We believe that such strong AI/ATP systems are very useful tools that make formal mathematics much more accessible, and their gradual strengthening is today one of the most promising paths towards the eventual adoption of computer-assisted mathematics (and exact science) by mainstream mathematicians (and exact scientists).

The main body of future work is thus further strengthening of the various parts (e.g., features/labels and the whole learning setup, machine-learning techniques, ATPs) of the AI/ATP methods. Also, more advanced proof reconstruction such as [KU13e, SB13] is still missing for Mizar. With longer and longer ATP proofs, human-friendly transformations and presentations of such proofs are becoming more and more important tasks that will quite likely also benefit from learning the "human-friendliness" from large repositories of human-oriented proofs such as the MML.

Thus it seems that the forty years of incessant and stubborn designing and building of the human-oriented formal mathematical language and large library by the Mizar team, and in particular by the recently deceased Mizar gurus Andrzej Trybulec and Piotr Rudnicki, have already resulted in one of the most interesting AI corpora currently available to mankind. It will be quite hard for the historians to properly enumerate all their inventions that led to the current state of the art. We would like to thank Andrzej and Piotr for this lifelong Opus Magnum, for their infatuating dreams, their wide and never-ending interest in science (and science fiction), and for their great sense of fun combined with high doses of self-criticism, down-to-earth common sense, caution and modesty, that made them into such great scientists, hackers, teachers, debaters, critics, and friends.

# 11 Semantic Features for Machine Learning

## Publication Details

Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Efficient Semantic Features for Automated Reasoning over Large Theories. Accepted for the *Proc. International Joint Conference on Artificial Intelligence*, to appear, IJCAI/AAAI, 2015.

## Abstract

Large formal mathematical knowledge bases encode considerable parts of advanced mathematics and exact science, allowing deep semantic computer assistance and verification of complicated theories down to the atomic logical rules. An essential part of automated reasoning over such large theories are methods learning selection of relevant knowledge from the thousands of proofs in the corpora. Such methods in turn rely on efficiently computable features characterizing the highly structured and inter-related mathematical statements.

In this work we (i) propose novel semantic features characterizing the statements in such large semantic knowledge bases, (ii) propose and carry out their efficient implementation using deductive-AI data-structures such as substitution trees and discrimination nets, and (iii) show that they significantly improve the strength of existing knowledge selection methods and automated reasoning methods over the large formal knowledge bases. In particular, on a standard large-theory benchmark we improve the average predicted rank of a mathematical statement needed for a proof by 22% in comparison with state of the art. This allows us to prove 8% more theorems in comparison with state of the art.

## 11.1. Introduction: Reasoning in Large Theories

In the conclusion of his seminal paper on AI [Tur50], Turing suggests two alternative ways how to eventually build learning (AI) machines: (i) focusing on an abstract activity like chess, and (ii) focusing on learning through physical senses. Both these paths have been followed with many successes, but so far without producing AI competitive in the most advanced application of human intelligence: scientific thinking. The approach we follow is to try to learn that from large bodies of computer-understandable scientific reasoning.

In the last decade, large corpora of complex mathematical (and scientific) knowledge and reasoning have been encoded in a fully computer-understandable form. In such encodings, the mathematical knowledge consisting of definitions, theorems, proofs and theories is explained in complete detail, allowing the computers to fully understand the semantics of such complicated objects and to verify correctness of the long reasoning chains with respect to the formal inference rules of the chosen logical framework (set theory, type theory, etc.).

Recent highlights of this development include the formal encoding and verification of two graduate textbooks leading to the proof of the Odd Order theorem ("every finite group of odd order is solvable") [GAA$^+$13], the formal verification of the 300-page book leading the proof of the Kepler conjecture [Hal12], and verification of the seL4 operating system microkernel [KAE$^+$10].

This means that larger and larger parts of mathematics and mathematical thinking can now be analyzed, explored, assisted, and further developed by computers in ways that are impossible in domains where complete semantics is missing. The computers can not only use inductive AI methods (such as learning) to extract ideas from the large corpora, but they can also combine them with deductive AI tools such as automated theorem provers (ATPs) to attempt formal proofs of new ideas, thus further growing the body of verified scientific knowledge (which can then again be further learned from, and so on ad infinitum). In fact, this has started to happen recently. Large formal corpora built in expressive logics of interactive theorem provers (ITPs) such as Isabelle [NK14], Mizar [GKN10] and HOL Light [Har96a] have been translated to first-order logic, and ATPs such as Vampire [KV13], E [Sch02] and Z3 [dMB08] are used to prove more and more complicated lemmas in the large theories.

Since existing ATP calculi perform poorly when given thousands to millions of facts, a crucial component that makes such ATP assistance practical are heuristic and learning AI methods that select a small number of most relevant facts for proving a given lemma [KvLT$^+$12]. This means that we want to characterize all statements in the knowledge bases by *mathematically relevant features* that will to a large extent allow to precompute the most promising combinations of formulas for proving a given conjecture. In some sense, we are thus trying to make a high-level approximation of the proof-search problem, and to restrict the fragile local decisions taken by the underlying ATP search by such high-level knowledge about what makes sense globally and what is likely a blind alley.[1] The question is how to design efficient features that will make such global approximative methods as good as possible. This is the subject of this paper.

[1]Obviously, when developed, such guiding methods can be also tried directly inside the ATP calculi. See for example the hints method [Ver96], a similar work done for E [Sch00], and the MaLeCoP system [UVŠ11].

### 11.1.1. Contributions

**1. Semantic features for characterizing mathematical statements.** We propose matching, abstraction and unification features and their combinations as a suitable means for characterizing statements in large mathematical corpora written in expressive logical frameworks (Section 11.4).
**2. Fast semantic feature-extraction mechanisms.** The crucial idea making the use of semantic features feasible is that such features often correspond to the nodes of fast deductive-AI data-structures such as substitution trees and discrimination nets. We implement and optimize such feature extraction mechanisms and demonstrate that they scale very well even on the largest formal corpora, achieving extraction times below 100 seconds for over hundred thousand formulas (Sections 11.5 and 11.6).
**3. Improved Premise-Selection Performance.** We evaluate the performance of the semantic features when selecting suitable premises for proofs and compare them to the old features using standard machine-learning metrics such as Recall, Precision, AUC, etc. The newly proposed features improve the average predicted rank of a mathematical statement needed for a proof by 22% in comparison with the best old features.
**4. Improved Theorem-Proving Performance.** We compare the overall performance of the whole feature-characterization/learning/theorem-proving stack for the new and old features and their combinations. The improved machine-learning performance translates to 8% more theorems proved automatically over the standard MPTP2078 large-theory benchmark, getting close to the ATP performance obtained by using human-selected facts (Section 11.7).

## 11.2. The Large Theory Setting: Premise Selection and Learning from Proofs

The object of our interest is a large mathematical corpus, understood as a set $\Gamma$ of formally stated theorems, each with zero or more proofs.[2] Examples of such corpora are the Mizar Mathematical Library (MML),[3] the Isabelle Archive of Formal Proofs (AFP),[4] and the Flyspeck (Formal Proof of the Kepler Conjecture) development[5] done in HOL Light. Such large corpora contain tens to hundreds of thousands of proved statements. To be able to do many ATP experiments,[6] smaller benchmarks have been defined as meaningful subsets of the large corpora. Here we rely on the MPTP2078

[2]We treat axioms and definitions as theorems with empty proof. In general, a theorem can have several alternative proofs.
[3]`http://mizar.org/`
[4]`http://afp.sourceforge.net/`
[5]`https://code.google.com/p/flyspeck/`
[6]A large number of ATP experiments is expensive. The ATPs are usually run with time limits between 1 second and 300 seconds.

benchmark [AHK+14] used in the 2012 CASC@Turing ATP competition [Sut13] of the Alan Turing Centenary Conference.[7]

In this setting, the task that we are interested in is to automatically prove a new conjecture given all the available theorems. Because the performance of existing ATP methods degrades considerably [UHV10, HV11] when given large numbers of redundant axioms, our research problem is to estimate the facts that are most likely to be useful in the final proof. Following [AHK+14] we define:

**Definition 11.2.1 (*Premise selection problem*)** *Given an ATP $A$, a corpus $\Gamma$ and a conjecture $c$, predict those facts from $\Gamma$ that are likely to be useful when $A$ searches for a proof of $c$.*

The currently strongest premise selection methods use machine learning on the proofs of theorems in the corpus, such as naive Bayes, distance-weighted $k$-nearest neighbor, kernel methods, and basic ensemble methods [KvLT+12, KBKU13, KU13f, KU13c, KU14]. It is possible that a particular fact is useful during a proof search without being used in the final formal proof object. Such cases are however rare and hard to detect efficiently. Therefore the relation of *being useful during the proof search* is usually approximated by *being used in the final proof.* Additionally, the learning setting is usually simplified by choosing at most one ("best") proof for each theorem $c$ [KU13g], and representing the proof of $c$ as a set of theorems $P(c)$ used in the proof. The query and update speed is important: in the ITP setting there are a number of alternative (usually less automated) theorem-proving techniques that can be used if full automation is weak or slow. Likewise, the learning systems should quickly digest and adapt to new theorems and proofs. The overall large-theory setting is shown in Figure 11.1.

Assuming a method $F$ for extracting mathematically relevant features characterizing the theorems, this setting leads to the following *multi-label learning task*: Each proved theorem $c \in \Gamma$ produces a training example consisting of $F(c)$ and $P(c)$, i.e., given the features $F(c)$ we want the trained predictor to recommend the labels $P(c)$ . The learning methods mentioned above are typically used as rankers: given the features of a new conjecture $c$, the highest ranked (using heuristically determined thresholds) labels for $F(c)$ are given to an ATP, which then attempts a proof. To maximize the ATP performance, usually a portfolio of several most complementary learning methods, feature characterizations and ranking thresholds is used, and the (typically exponentially behaving) ATP is run in a strategy-scheduling mode [Tam97], i.e., with several shorter time limits over such complementary predictions rather than using the whole time limit for the most promising method.

[7] `http://www.turing100.manchester.ac.uk/`

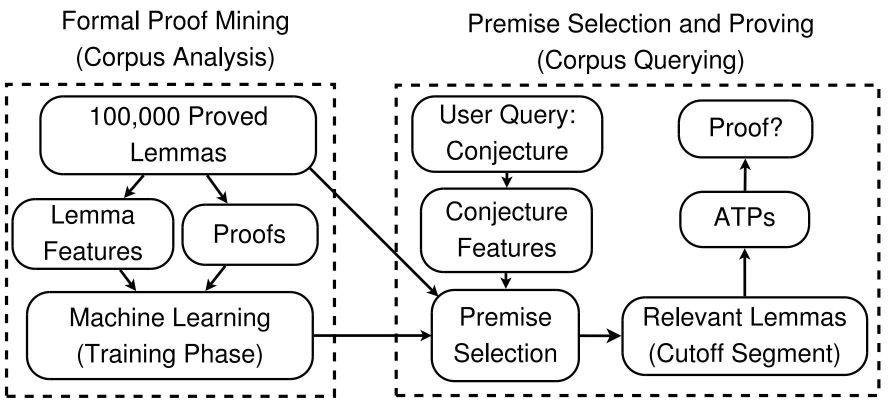


Figure 11.1.: Theorem proving over large formal corpora. The corpus contains many lemmas that can be used to prove a new conjecture (user query). The corpus comes with many formal proofs that can be mined, i.e., used to learn which lemmas (premises) are most relevant for proving particular conjectures (queries). The strength of the learning methods depends on designing *mathematically relevant features* faithfully characterizing the statements. When a new conjecture is attempted, the learners trained on the corpus rank the available lemmas according to their estimated relevance for the conjecture, and pass a small number (cutoff segment) of the best-ranked lemmas to ATP systems, which then attempt a proof.

## 11.3. Previously Introduced Features

The most commonly used features for characterizing mathematical statements in large theories are just their *symbols* [HV11, MP09]. In addition to that, large-theory ATP systems like HOL(y)Hammer [KU14], Sledgehammer [KBKU13] and MaLARea [USPV08] have so far used features that represent:

- Types, i.e., type constants, type constructors, and type classes [KU14]
- Term walks of length 2 [KBKU13]
- Subterms [USPV08]
- Validity in a pool of finite models [USPV08]
- Meta-information such as the theory name, as well as presence in the various databases [KBKU13]

The normalizations for term and type variables that have been tried so far include:

- Replacing variables by their (variable-normalized) types [KU14]
- Using de Bruijn indices [USPV08]
- Renaming all variables to a unique common variable [USPV08]
- Using the original variable names (this is useful when the same variable names are used for similar purposes)

Except from validity in finite models, all these features can be extracted in a linear time and are thus easy to use. Since their distribution may be quite uneven, normalizing them by methods like TF-IDF[8] is relatively important before handing them over to fast-but-simple learners such as distance-weighted $k$-NN (used here for evaluation). Since the MPTP2078 is untyped and already comes with variables renamed to de Bruijn indices, we do not use the type enhancements and original variable names here. Validity in a large pool of finite models requires finding a diverse set of models in which the formulas are then evaluated. Even though such features may approximate the semantics really well, they are in general much more expensive to compute than the rest, and for that reason we also avoid them here.

## 11.4. New Semantic Features for Reasoning

A formal proof is usually defined as a sequence (resp. DAG) of formulas where each of them is either an axiom or is derived from earlier formulas (resp. DAG parents) by an application of an inference rule. In various ITP and ATP systems such inference rules differ, however a crucial and very frequent idiom of (not just) mathematical reasoning is the rule of *Universal Instantiation*, allowing to instantiate a general statement in a concrete context.

**Scenario 1.** To give the first simple example, the commutativity of addition ($CA$): $X + Y = Y + X$ may be applied to prove that ($L1$): $1 + 2 = 2 + 1$ , by matching 1 with $X$ and 2 with $Y$. Assume that somebody already proved $1 + 2 = 2 + 1$ by referring to $X + Y = Y + X$, and a new conjecture to prove is ($L2$): $3 + 7 = 7 + 3$. When using only the features introduced previously (see Section 11.3), the feature overlap between these two instances will be the same as with ($L3$): $5 + 8 = 4 + 9$ (which might have been proved very differently). In other words, none of the features captures the fact that $L1$ and $L2$ are closer to each other in the instantiation lattice than to $L3$, and thus have a higher chance of sharing a common proof pattern.

A complete way how to remedy this would be to use all (variable-normalized) generalizations of a term as its features. However, the lattice of all generalizations of a term is typically exponentially large wrt. the size of the term (see Figure 11.2). Fortunately, in this simple scenario, we can replace such full enumeration of generalizations of a given term $t$ with a much smaller set: the set $Gen_\Gamma(t)$ of all terms in our corpus $\Gamma$ that generalize $t$. Indeed, since $CA$ was used to prove $L1$, it must be in the corpus $\Gamma$. Therefore $CA$ itself is in $Gen_\Gamma(L1)$ and also in $Gen_\Gamma(L2)$, introducing a new common *matching feature* for $L1$ and $L2$, but not for $L3$. Extraction of all such matching features for large corpora can be efficiently implemented using ATP indexing datastructures called *discrimination trees* ($DT_\Gamma$), which we briefly discuss in Section 11.5.

**Scenario 2.** Now imagine a more complicated (but still quite common) case: $L1$ is in the corpus, but has a more complicated proof, and $CA$ is not in the corpus (yet). For

[8]Readers might wonder about *latent semantics* [DDL+90]. So far this does not raise the performance significantly.

example, 1 and 2 are defined as von Neuman ordinals, where addition is the standard ordinal addition which is not commutative in general. Since $CA$ is not in $\Gamma$, it is neither in its discrimination tree, and $L1$ and $L2$ will not get the common matching feature corresponding to $CA$. But it is still quite likely that the proof of $L1$ is relevant also for proving $L2$: if we are lucky, the proof is sufficiently general and actually derives the commutativity of ordinal addition for finite ordinals behind the scenes. This means that quite often the common matching feature is still interesting, even if not explicitly present in $\Gamma$.

One might again first try to use brute force and attempt to generate all common generalizations for all pairs of terms in $\Gamma$. Again, see Figure 11.2 for an example when such lattice is exponentially large. It is however no longer possible to rely just on the terms that already are in $\Gamma$ as in the previous case, and some method for generating some common matching features seems needed in this case. We have proposed and implemented two kinds of heuristic solutions. The first inserts for each term $t$ a small number of more general terms into $DT_\Gamma$, typically by generalizing the term only linearly-many times, see Section 11.5 for details. Some of such generalization methods actually corresponds to the internal nodes of $DT_\Gamma$. The second solution collects such (differently optimized) generalization nodes explicitly, using another kind of efficient ATP indexing datastructure called *substitution tree* ($ST_\Gamma$), described in more detail in Section 11.6 (see Figure 11.4). This means that just by building $ST_\Gamma$, we naturally obtain for each term $t$ in $\Gamma$ a set of (some) generalizing features of $t$: the set $ST_\Gamma^{Anc}(t)$ of the ancestors of $t$ in $ST_\Gamma$. Even the substitution tree however cannot (reasonably) guarantee that every two terms in $\Gamma$ have there their least general generalization (lgg). Such requirement would again result in exponential size for examples such as Figure 11.2.

**Scenario 3.** Finally, what substitution trees can guarantee (and are usually used for in ATP) is efficient retrieval of all unifying terms. The easiest semantic motivation for collecting such features comes from the resolution rule, which is the basis of many ATP procedures. Given formulas $p(X, a)$ and $p(a, X) \implies False$, the resolution rule derives $False$ by unifying $p(X, a)$ and $p(a, X)$. Note that the two unifying literals do not match in any direction, however they will always have a nontrivial lgg.

## 11.5. Discrimination Trees for Matching and Generalization Features

Selection of candidate clause-heads that unify, match, or subsume a given goal is a central operation in automated deduction. In order to perform such selection efficiently, all major theorem provers use term indexing techniques [RV01]. Discrimination trees, as first implemented by [Gre86], index terms in a trie, which keeps single path-strings at each of the indexed terms (Figure 11.3 taken from [RV01]). A discrimination tree can be constructed efficiently, by inserting each term in the traversal preorder. Since discrimi-

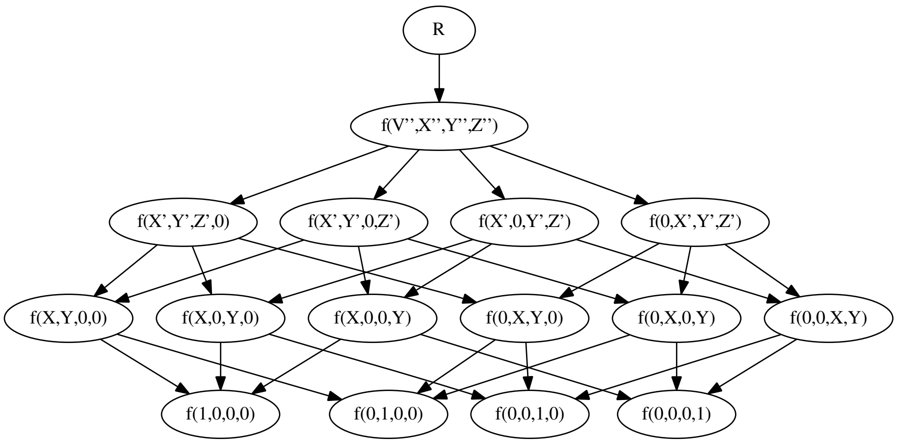


Figure 11.2.: The lattice of least general generalization of four specific ground terms. To calculate the number $F$ of such least general generalizations for some $n$, we have to count all nodes in every layer using the following equation (counting from the bottom layer to the top): $F(n) = \binom{n}{2} + \binom{n}{3} + \cdots + \binom{n}{n-1} + \binom{n}{n} = \sum_{k=2}^{n} \binom{n}{k}$ from which we get by the Binomial theorem: $F(n) = 2^n - n - 1$. So many generalization cannot be effectively extracted and used as features for more than a hundred of such terms.

nation trees are based on path indexing, retrieval of generalizations is straightforward, but retrieval of unifiable terms is more involved.

To address Scenario 1, we create a discrimination net with all the available terms in the corpus inserted in the net. Efficient implementation is needed, since the largest mathematical corpora contain millions of terms. The result of a lookup then corresponds to first order matching. This means that all terms in the corpus that are more general than the queried one are returned. In order to extend this to generalizations that are shared between terms, but not yet in the net (Scenario 2), we do heuristic generalizations. This consists of selecting a subterm of the term and replacing it by a variable. We consider several strategies for selecting the subterms to generalize:

- repeated right-most inner-most
- repeated left-most inner-most
- all positions
- combinations of above (quadratic in the size of the term), including combination of the previous one with itself.

To avoid redundant generation of generalizations of terms that are already in the net, the generation of subterms and their right-most inner-most generalizations is done together. We first iterate over the term top-down, and for each subterm level we try to insert its generalizations iteratively. If a particular generalization is already present in the net, we

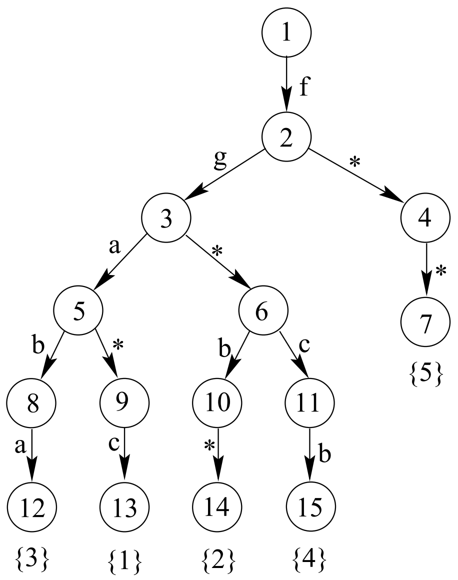


Figure 11.3.: Discrimination Tree containing the terms $f(g(a,*),c)$, $f(g(*,b),*)$, $f(g(a,b),a)$, $f(g(*,c),b)$, and $f(*,*)$.

do not compute any further generalizations, but instead proceed to the next subterm. This optimization brings the sum of feature extraction times for all formulas below 100s for the largest formal mathematical corpus available (MML1147) containing millions of terms, see Section 11.7 (Table 11.2).

## 11.6. Substitution Trees for Unification and Generalization Features

Substitution trees [Gra95] are a term indexing technique which is based on unifiability checking, rather than simple equality tests. In order to do so, substitution trees keep substitutions in the nodes. This allows for substitution trees to be smaller than other indexing techniques. Since the traversal order is not fixed, substitution trees need to compute the common generalizations of terms during insertion.

Our implementation is using the so called *linear substitution trees* in the same way as described in [Gra96, RV01]. The retrieval from a substitution tree may require more backtracking steps than in other indexing techniques, but the retrieval of unifiable terms is straightforward: it amounts to following all the paths that contain substitutions compatible with the given term. This is simpler than for example implementing unification on discrimination trees [HV09].

To get interesting generalizations as features of a term $t$ that is already inserted in the

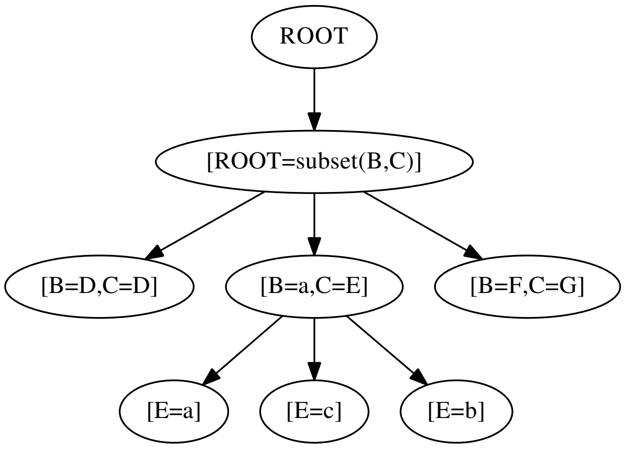


Figure 11.4.: A substitution tree of the terms: subset(A,B), subset(a,b), subset(a,c), subset(C,C), subset(a,a). For term subset(a,c) then there is a path containing the following generalized terms (from the root to the leaf): ROOT, subset(B,C), subset(a,E), subset(a,c).

tree, we extract a path from the root to the leaf. Each node on such path represents generalization of the term $t$ (Figure 11.4).

## 11.7. Experimental Analysis

### 11.7.1. Benchmark Data and Evaluation Scenario

The MPTP2078 benchmark consists of 2078 related large-theory problems (conjectures) extracted from the Mizar library. These problems contain 4494 unique formulas used as conjectures and axioms. The formulas and problems have a natural linear ordering derived from their (chronological) order of appearance in the Mizar library. As usual in such large-theory evaluations [KU14], we emulate the scenario of using AI/ATP assistance in interactive theorem proving: For each conjecture $C$ we assume that all formulas stated earlier in the development can be used to prove $C$. This scenario results in *large* ATP problems that have 1877 axioms on average. Such problems are typically difficult to solve without techniques for pre-selection of the most relevant axioms (cf. Table 11.4).

For each conjecture $C$ we also use its ITP (human-written) proof to extract *only the premises needed for the ITP proof* of $C$, i.e., $P(C)$.[9] This proof information is used in three ways: (i) to train the machine learners on all previous proofs for each conjecture $C$, (ii) to compare the premises predicted by such trained machine learners with the actual

[9]In general, we can also use ATP proofs of the previous theorems alongside with their ITP proofs for the learning, however we will not complicate the evaluation setting here.

| Name | Description |
|---|---|
| sym | Constant and function symbols |
| $\text{trm}_0$ | Subterms, all variables unified |
| $\text{trm}_\alpha$ | Subterms, de Bruijn normalized |
| $\text{mat}_\varnothing$ | Matching terms, no generalizations |
| $\text{mat}_r$ | Repeated gener. of rightmost innermost constant |
| $\text{mat}_l$ | Repeated gener. of leftmost innermost constant |
| $\text{mat}_1$ | Gener. of each application argument |
| $\text{mat}_2$ | Gener. of each application argument pair |
| $\text{mat}_\cup$ | Union of all above generalizations |
| pat | Walks in the term graph |
| abs | Substitution tree nodes |
| uni | All unifying terms |

Table 11.1.: Summary of all the features used.

proof premises $P(C)$, and (iii) to construct the *small* ATP problem for $C$, which (unlike the *large* version) contains as axioms only the (few) premises $P(C)$ – and in this way, the ATP is very significantly advised by the human author of the ITP proof.

## 11.7.2. Speed and Machine Learning Performance

The features that we evaluate are summarized in Table 11.1. We limit the evaluation to two fast learning methods: distance-weighted $k$-NN and naive Bayes, however the latter performs significantly worse. Table 11.2 shows the numbers of features obtained on MPTP2078, the feature-extraction and learning speeds on such MPTP2078 features. To see how the feature extraction process scales, we have also tested the methods on the whole MML library (version 1147) containing 146500 formulas. Note that practically all the extraction methods scale well to this corpus (we did not run UNI, due to its size), typically taking less than 100 seconds to process the whole corpus. This means that computing the features of a new conjecture which is being proved over the (already memory-loaded) corpus will take only miliseconds. Such times are negligible in comparison with standard ATP times (seconds).

Table 11.3 shows the standard machine learning ($k$-NN) evaluation, comparing the actual (needed) ITP proof premises of each conjecture $C$ (i.e., $P(C)$) with the predictions of the machine learners trained on all ITP proofs preceding $C$. The average rank of a needed premise on MPTP2078 decreases from 53.03 with the best old feature method

| Method | Speed (sec) | | Number of features | | Learning and prediction | |
|---|---|---|---|---|---|---|
| | MPTP2078 | MML1147 | total | unique | knn | naive Bayes |
| sym | 0.25 | 10.52 | 30996 | 2603 | 0.96 | 11.80 |
| $\text{trm}_\alpha$ | 0.11 | 12.04 | 42685 | 10633 | 0.96 | 24.55 |
| $\text{trm}_0$ | 0.13 | 13.31 | 35446 | 6621 | 1.01 | 16.70 |
| $\text{mat}_\varnothing$ | 0.71 | 38.45 | 57565 | 7334 | 1.49 | 24.06 |
| $\text{mat}_r$ | 1.09 | 71.21 | 78594 | 20455 | 1.51 | 39.01 |
| $\text{mat}_l$ | 1.22 | 113.19 | 75868 | 17592 | 1.50 | 37.47 |
| $\text{mat}_1$ | 1.16 | 98.32 | 82052 | 23635 | 1.55 | 41.13 |
| $\text{mat}_2$ | 5.32 | 4035.34 | 158936 | 80053 | 1.65 | 96.41 |
| $\text{mat}_\cup$ | 6.31 | 4062.83 | 180825 | 95178 | 1.71 | 112.66 |
| pat | 0.34 | 64.65 | 118838 | 16226 | 2.19 | 52.56 |
| abs | 11 | 10800 | 56691 | 6360 | 1.67 | 23.40 |
| uni | 25 | N/A | 1543161 | 6462 | 21.33 | 516.24 |

Table 11.2.: Feature numbers and learning speed on MPTP2078, and extraction speed on MPTP2078 and MML1147 (in seconds)

$\text{trm}_0$ (formulas characterized by all their subterms with all variables renamed to just one) to 43.43 with the best new method $\text{mat}_\varnothing$ (using the set of all matching terms as features, without any generalizations). This is a large improvement of the standard machine-learning prediction: thanks to better features, the ranking of needed premises is improved by 22.1%. The best combination of features ($_{\text{sym}|\text{trm}_0|\text{mat}_\varnothing|\text{abs}}$) improves this to 41.97. This is a big difference for the ATP search, evaluated next.

### 11.7.3. Evaluation of the Theorem Proving Performance

First we measure the performance of unaided ATPs running for 60 seconds on the large and small versions of the problems, see Table 11.4. While Vampire outperforms E, particularly on the large problems, we choose E for the complete evaluation which includes premise selection, because the machine learning advice will not interact with the SInE selection heuristics [HV11] used by Vampire very frequently on the large problems.

The complete evaluation then proceeds as follows. For each of the best new and old premise-selection methods we create six versions of ATP problems, by taking the top-ranking segments of 8, 16, 32, 64, 128, and 256 predicted premises, and we run E on such problems for 10 seconds. Each problem thus takes again at most 60 seconds altogether, allowing us to compare the results also with the 60-second unaided ATP runs on the

| Method | 100Cover | Prec | Recall | AUC | Rank |
|---|---|---|---|---|---|
| $\text{mat}_{\varnothing}$ | 0.918 | 17.711 | 234.12 | 0.9561 | 43.43 |
| $\text{mat}_1$ | 0.918 | 17.711 | 234.69 | 0.9557 | 43.83 |
| $\text{mat}_l$ | 0.918 | 17.702 | 235.04 | 0.9555 | 43.91 |
| $\text{mat}_r$ | 0.917 | 17.7 | 234.31 | 0.9557 | 43.84 |
| $\text{mat}_2$ | 0.917 | 17.708 | 235.37 | 0.9554 | 44.06 |
| abs | 0.917 | 17.686 | 237.89 | 0.9542 | 44.11 |
| pat | 0.916 | 17.64 | 235.2 | 0.9557 | 44.13 |
| $\text{mat}_{\cup}$ | 0.916 | 17.672 | 236.31 | 0.9551 | 44.4 |
| $\text{trm}_0$ | 0.903 | 17.425 | 281.46 | 0.9447 | 53.03 |
| uni | 0.891 | 16.822 | 257.1 | 0.9465 | 51.83 |
| sym | 0.884 | 17.137 | 326.67 | 0.9325 | 63.21 |
| $\text{trm}_{\alpha}$ | 0.861 | 16.801 | 378.9 | 0.9156 | 75.52 |
| $\text{sym}\vert\text{trm}_0\vert\text{mat}_{\varnothing}\vert\text{abs}$ | 0.922 | 17.737 | 227.7 | 0.9587 | 41.97 |

Table 11.3.: Machine Learning evaluation: 100Cover = average ratio of the proof premises present in the first 100 advised premises; Prec = precision; Recall = average number of premises needed to recall the whole training set; AUC = area under ROC curve; Rank = average rank of a proof premise. The methods are sorted by their 100Cover.

large and small problems. Table 11.5 then compares the ATP performance (measured as a union of the six slices), also adding the best-performing combination of the old and new features. The best new method $\text{mat}_{\varnothing}$ solves 80 problems (8%) more than the best old method $\text{trm}_0$, and the best-performing combination solves 103 (10%) problems more. This is getting close to the performance of E on the human-advised premises (1210 in Table 11.4). The behaviour of the ATP with respect to the number of premises used is depicted in Figure 11.5, showing even higher improvements.

| ATP | E 1.8 | | Vampire 2.6 | | Z3 |
|---|---|---|---|---|---|
| Problem set | large | small | large | small | small |
| Proved | 573 | 1210 | 907 | 1319 | 1065 |

Table 11.4.: Unaided ATPs on the large and small problems.

| Method | Proved (%) | Theorems |
|---|---|---|
| $\mathrm{mat}_{\varnothing}$ | 54.379 | 1130 |
| $\mathrm{mat}_r$ | 54.331 | 1129 |
| $\mathrm{mat}_l$ | 54.283 | 1128 |
| pat | 54.235 | 1127 |
| $\mathrm{mat}_{\cup}$ | 53.994 | 1122 |
| $\mathrm{mat}_1$ | 53.994 | 1122 |
| $\mathrm{mat}_2$ | 53.898 | 1120 |
| abs | 53.802 | 1118 |
| $\mathrm{trm}_0$ | 50.529 | 1050 |
| uni | 50.241 | 1044 |
| sym | 48.027 | 998 |
| $\mathrm{trm}_{\alpha}$ | 43.888 | 912 |
| $\mathrm{sym}\vert\mathrm{trm}_0\vert\mathrm{mat}_{\varnothing}\vert\mathrm{abs}$ | 55.486 | 1153 |

Table 11.5.: E 1.8 prover on the large problems filtered by learning-based premise selection using different features. The methods are sorted by the number of theorems proved.

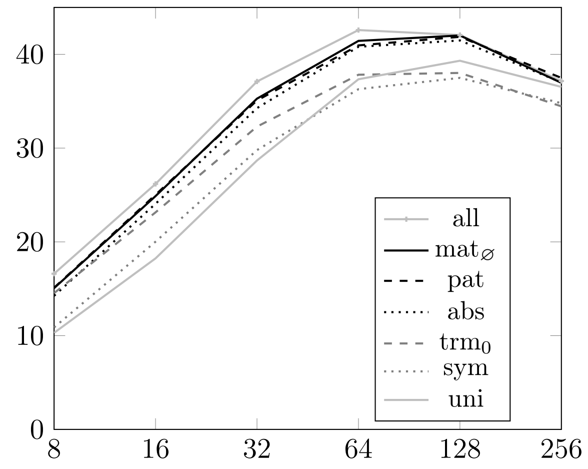


Figure 11.5.: ATP (E 1.8) performance (in % of all large problems) for selected premise numbers and features.

## 11.8. Conclusion

We have shown that even when dealing with the largest formal corpora, one can have efficient features that encode important semantic relations and thus make the selection of relevant knowledge much more precise. The newly introduced semantic features significantly improve selection of relevant knowledge from large formal mathematical corpora.

This improvement is 22% in terms of the average predicted rank, while the combination of the new and old features helps to increase the intelligence of the advising algorithms to a level that is nearly equal to that of the human formalizers when comparing the final ATP performance. In particular, the best new method $\mathrm{mat}_{\varnothing}$ proves 8% more theorems automatically than the best old method. The cost of manually producing these proofs is very high: The Flyspeck project took about 25 person-years and the Mizar library about 100–150 person-years. So the reported improvements just for these two corpora translate to 2 and 8–12 person-years respectively.

There is a large number of directions for future work, including employing such semantic features also for internal guidance in systems like MaLeCoP and in resolution/superposition ATPs, re-using their efficient term-indexing datastructures.

## Acknowledgments

Kaliszyk was supported by the Austrian Science Fund (FWF): P26201, Urban was supported by NWO grant *Knowledge-based Automated Reasoning* and Vyskočil was supported by the Grant Agency of the Czech Republic under the Project GACR P103/12/1994.

# Bibliography

[ABH$^+$98] W. Ahrendt, B. Beckert, R. Hähnle, W. Menzel, W. R. f, G. Schellhorn, and P. H. Schmitt. Integrating automated and interactive theorem proving. In W. Bibel and P. H. Schmitt, editors, *Automated Deduction—A Basis for Applications*, volume II: Systems and Implementation Techniques, pages 97–116. Kluwer, 1998.

[ABMU11] J. Alama, K. Brink, L. Mamane, and J. Urban. Large formal wikis: Issues and solutions. In J. H. Davenport, W. M. Farmer, J. Urban, and F. Rabe, editors, *Calculemus/MKM*, volume 6824 of *LNCS*, pages 133–148. Springer, 2011.

[Ada10] M. Adams. Introducing HOL Zero - (extended abstract). In K. Fukuda, J. van der Hoeven, M. Joswig, and N. Takayama, editors, *ICMS*, volume 6327 of *LNCS*, pages 142–143. Springer, 2010.

[ADN$^+$08] K. Avrachenkov, V. Dobrynin, D. Nemirovsky, S. K. Pham, and E. Smirnova. Pagerank based clustering of hypertext document collections. In S.-H. Myaeng, D. W. Oard, F. Sebastiani, T.-S. Chua, and M.-K. Leong, editors, *SIGIR*, pages 873–874. ACM, 2008.

[AHK$^+$14] J. Alama, T. Heskes, D. Kühlwein, E. Tsivtsivadze, and J. Urban. Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning*, 52(2):191–213, 2014.

[AKU12] J. Alama, D. Kühlwein, and J. Urban. Automated and Human Proofs in General Mathematics: An Initial Comparison. In N. Bjørner and A. Voronkov, editors, *LPAR 2012*, pages 37–45, 2012.

[Ano94] The QED Manifesto. In A. Bundy, editor, *CADE*, volume 814 of *LNCS*, pages 238–251. Springer, 1994.

[AP10] B. Akbarpour and L. C. Paulson. MetiTarski: An automatic theorem prover for real-valued special functions. *J. Autom. Reasoning*, 44(3):175–205, 2010.

[BBN11] J. C. Blanchette, L. Bulwahn, and T. Nipkow. Automatic proof and disproof in Isabelle/HOL. In C. Tinelli and V. Sofronie-Stokkermans, editors, *FroCoS*, volume 6989 of *LNCS*, pages 12–27. Springer, 2011.

[BBP11] J. C. Blanchette, S. Böhme, and L. C. Paulson. Extending Sledgehammer with SMT solvers. In N. Bjørner and V. Sofronie-Stokkermans, editors, *CADE 2011*, pages 116–130, 2011.

[BBPS13] J. C. Blanchette, S. Böhme, A. Popescu, and N. Smallbone. Encoding monomorphic and polymorphic types. In N. Piterman and S. Smolka, editors, *TACAS 2013*, volume 7795 of *LNCS*, pages 493–507. Springer, 2013.

[BCD$^+$11] 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.

[Bee01] M. Beeson. Automatic derivation of the irrationality of e. *J. Symb. Comput.*, 32(4):333–349, 2001.

[Bel99] J. G. F. Belinfante. On computer-assisted proofs in ordinal number theory. *J. Autom. Reasoning*, 22(2):341–378, 1999.

[BHdN02] M. Bezem, D. Hendriks, and H. de Nivelle. Automatic proof construction in type theory using resolution. *J. Autom. Reasoning*, 29(3-4):253–275, 2002.

[Bla12] J. C. Blanchette. *Automatic Proofs and Refutations for Higher-Order Logic.* PhD thesis, Fakultät für Informatik, Technische Universität München, 2012. `http://www21.in.tum.de/~blanchet/phdthesis.pdf`.

[BM84] R. S. Boyer and J. S. Moore. Program verification. Technical report, DTIC Document, 1984.

[BN00] S. Berghofer and T. Nipkow. Proof terms for simply typed higher order logic. In M. Aagaard and J. Harrison, editors, *TPHOLs 2000*, volume 1869 of *LNCS*, pages 38–52. Springer, 2000.

[BN10] S. Böhme and T. Nipkow. Sledgehammer: Judgement Day. In J. Giesl and R. Hähnle, editors, *IJCAR 2010*, volume 6173 of *LNCS*, pages 107–121. Springer, 2010.

[BP95] B. Beckert and J. Posegga. leanTAP: Lean tableau-based deduction. *J. Autom. Reasoning*, 15(3):339–358, 1995.

[BP13] J. C. Blanchette and A. Paskevich. TFF1: The TPTP typed first-order form with rank-1 polymorphism. In M. P. Bonacina, editor, *CADE 2013*, pages 414–420, 2013.

[BPTF08] C. Benzmüller, L. C. Paulson, F. Theiss, and A. Fietzke. LEO-II—A cooperative automatic theorem prover for higher-order logic. In A. Armando, P. Baumgartner, and G. Dowek, editors, *IJCAR 2008*, volume 5195 of *LNCS*, pages 162–170. Springer, 2008.

[BPWW12] J. C. Blanchette, A. Popescu, D. Wand, and C. Weidenbach. More SPASS with Isabelle—Superposition with hard sorts and configurable simplification. In L. Beringer and A. Felty, editors, *ITP 2012*, volume 7406 of *LNCS*, pages 345–360. Springer, 2012.

[BR02] G. Bancerek and P. Rudnicki. A compendium of continuous lattices in MIZAR. *J. Autom. Reasoning*, 29(3-4):189–224, 2002.

[Bro12] C. E. Brown. Satallax: An automatic higher-order prover. In B. Gramlich, D. Miller, and U. Sattler, editors, *IJCAR 2012*, pages 111–117, 2012.

[BT07] C. Barrett and C. Tinelli. CVC3. In W. Damm and H. Hermanns, editors, *CAV 2007*, volume 4590 of *LNCS*, pages 298–302. Springer, 2007.

[BW10] 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 *LNCS*, pages 179–194. Springer, 2010.

[CCRR99] A. Carlson, C. Cumby, J. Rosen, and D. Roth. The SNoW Learning Architecture. Technical Report UIUCDCS-R-99-2101, UIUC Computer Science Department, May 1999.

[CDLM08] K. Chaudhuri, D. Doligez, L. Lamport, and S. Merz. A tla+ proof system. In P. Rudnicki, G. Sutcliffe, B. Konev, R. A. Schmidt, and S. Schulz, editors, *LPAR 2008 Workshops*, volume 418 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[Chu40] A. Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5:56–68, 1940.

[CKKS10] M. Cramer, P. Koepke, D. Kühlwein, and B. Schröder. Premise selection in the Naproche system. In J. Giesl and R. Hähnle, editors, *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings*, pages 434–440, 2010.

[Dah98] I. Dahn. Interpretation of a Mizar-like logic in first-order logic. In R. Caferra and G. Salzer, editors, *FTP (LNCS Selection)*, volume 1761 of *LNCS*, pages 137–151. Springer, 1998.

[DDL$^+$90] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by Latent Semantic Analysis. *JASIS*, 41(6):391–407, 1990.

[DdM06] B. Dutertre and L. de Moura. The Yices SMT solver. `http://yices.csl.sri.com/tool-paper.pdf`, 2006.

[DGHW97] B. Dahn, J. Gehne, T. Honigmann, and A. Wolf. Integration of automated and interactive theorem proving in ILF. In W. McCune, editor, *CADE-14*, volume 1249 of *LNCS*, pages 57–60. Springer, 1997.

[dMB08] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *TACAS 2008*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[DS94] J. Denzinger and S. Schulz. Analysis and Representation of Equational Proofs Generated by a Distributed Completion Based Proof System. Seki-Report SR-94-05, Universität Kaiserslautern, 1994.

[DS96] J. Denzinger and S. Schulz. Recording and Analysing Knowledge-Based Distributed Deduction Processes. *J. Symbolic Computation*, 21(4/5):523–541, 1996.

[Dud76] S. A. Dudani. The distance-weighted k-nearest-neighbor rule. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-6(4):325–327, 1976.

[DW97] I. Dahn and C. Wernhard. First order proof problems extracted from an article in the Mizar Mathematical Library. In M. P. Bonacina and U. Furbach, editors, *Int. Workshop on First-Order Theorem Proving (FTP'97)*, RISC-Linz Report Series No. 97-50, pages 58–62. Johannes Kepler Universität, Linz (Austria), 1997.

[FGT92] W. M. Farmer, J. D. Guttman, and F. J. Thayer. Little theories. In D. Kapur, editor, *Automated Deduction - CADE-11, 11th International Conference on Automated Deduction, Saratoga Springs, NY, USA, June 15-18, 1992, Proceedings*, volume 607 of *LNCS*, pages 567–581. Springer, 1992.

[FM07] J.-C. Filliâtre and C. Marché. The Why/Krakatoa/Caduceus platform for deductive program verification. In W. Damm and H. Hermanns, editors, *CAV*, volume 4590 of *LNCS*, pages 173–177. Springer, 2007.

[FP13] J. Filliâtre and A. Paskevich. Why3—Where programs meet provers. In M. Felleisen and P. Gardner, editors, *European Symposium on Programming (ESOP 2013)*, volume 7792 of *Lecture Notes in Computer Science*, pages 125–128. Springer, 2013.

[GAA$^+$13] G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. L. Roux, A. Mahboubi, R. O'Connor, S. O. Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi, and L. Théry. A machine-checked proof of the Odd Order Theorem. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *ITP*, volume 7998 of *LNCS*, pages 163–179. Springer, 2013.

[GF92] M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, 1992.

[GK14] 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.

[GKKN14] T. Gauthier, C. Kaliszyk, C. Keller, and M. Norrish. Beagle as a HOL4 external ATP method. In L. de Moura, B. Konev, and S. Schulz, editors, *Proc. of the 4th Workshop on Practical Aspects of Automated Reasoning (PAAR'14)*, EPiC, 2014. to appear.

[GKN10] A. Grabowski, A. Korniłowicz, and A. Naumowicz. Mizar in a nutshell. *J. Formalized Reasoning*, 3(2):153–245, 2010.

[Gon07] G. Gonthier. The four colour theorem: Engineering of a formal proof. In D. Kapur, editor, *Computer Mathematics, 8th Asian Symposium, ASCM 2007, Singapore, December 15-17, 2007. Revised and Invited Papers*, volume 5081 of *LNCS*, page 333. Springer, 2007.

[Gra95] P. Graf. Substitution tree indexing. In *RTA*, volume 914 of *LNCS*, pages 117–131, 1995.

[Gra96] P. Graf. *Term Indexing*, volume 1053 of *LNCS*. Springer, 1996.

[Gre86] S. Greenbaum. *Input transformations and resolution implementation techniques for theorem-proving in first-order logic*. PhD thesis, University of Illinois at Urbana-Champaign, 1986.

[GS06] A. V. Gelder and G. Sutcliffe. Extending the TPTP language to higher-order logic with automated parser generation. In U. Furbach and N. Shankar, editors, *IJCAR*, volume 4130 of *LNCS*, pages 156–161. Springer, 2006.

[HAB$^+$15] T. C. Hales, M. Adams, G. Bauer, D. T. Dang, J. Harrison, T. L. Hoang, C. Kaliszyk, V. Magron, S. McLaughlin, T. T. Nguyen, T. Q. Nguyen, T. Nipkow, S. Obua, J. Pleso, J. Rute, A. Solovyev, A. H. T. Ta, T. N.

Tran, D. T. Trieu, J. Urban, K. K. Vu, and R. Zumkeller. A formal proof of the Kepler conjecture. *CoRR*, abs/1501.02155, 2015.

[Hal06] T. C. Hales. Introduction to the Flyspeck project. In T. Coquand, H. Lombardi, and M.-F. Roy, editors, *Mathematics, Algorithms, Proofs*, number 05021 in Dagstuhl Seminar Proceedings, pages 1–11, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[Hal12] T. Hales. *Dense Sphere Packings: A Blueprint for Formal Proofs*, volume 400 of *London Mathematical Society Lecture Note Series.* Cambridge University Press, 2012.

[Hal14a] T. C. Hales. Developments in formal proofs. *CoRR*, abs/1408.6474, 2014.

[Hal14b] T. C. Hales. Mathematics in the age of the Turing machine. In R. Downey, editor, *Turing's Legacy*, number 42 in Lecture Notes in Logic, pages 253–298. Cambridge University Press, 2014.

[Har96a] J. Harrison. HOL Light: A tutorial introduction. In M. K. Srivas and A. J. Camilleri, editors, *FMCAD*, volume 1166 of *LNCS*, pages 265–269. Springer, 1996.

[Har96b] J. Harrison. A Mizar mode for HOL. In J. von Wright, J. Grundy, and J. Harrison, editors, *TPHOLs*, volume 1125 of *LNCS*, pages 203–220. Springer, 1996.

[Har96c] 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, 1996.

[Har09] J. Harrison. *Handbook of Practical Logic and Automated Reasoning.* Cambridge University Press, 2009.

[HH11] J. Hölzl and A. Heller. Three chapters of measure theory in Isabelle/HOL. In M. van Eekelen, H. Geuvers, J. Schmaltz, and F. Wiedijk, editors, *ITP 2011*, volume 6898 of *LNCS*, pages 135–151. Springer, 2011.

[HHM$^+$10] T. C. Hales, J. Harrison, S. McLaughlin, T. Nipkow, S. Obua, and R. Zumkeller. A revision of the proof of the Kepler Conjecture. *Discrete & Computational Geometry*, 44(1):1–34, 2010.

[Hin69] R. Hindley. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.

[HKvRW10] M. Hendriks, C. Kaliszyk, F. van Raamsdonk, and F. Wiedijk. Teaching logic using a state-of-the-art proof assistant. *Acta Didactica Napocensia*, 3(2):35–48, June 2010.

[HKW96] R. Hähnle, M. Kerber, and C. Weidenbach. Common syntax of the DFGSchwerpunktprogramm deduction. Technical Report TR 10/96, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany, 1996.

[Hur99] J. Hurd. Integrating Gandalf and HOL. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *TPHOLs*, volume 1690 of *LNCS*, pages 311–322. Springer, 1999.

[Hur02] J. Hurd. An LCF-style interface between HOL and first-order logic. In A. Voronkov, editor, *CADE*, volume 2392 of *LNCS*, pages 134–138. Springer, 2002.

[Hur03] J. Hurd. First-order proof tactics in higher-order logic theorem provers. In M. Archer, B. Di Vito, and C. Muñoz, editors, *Design and Application of Strategies/Tactics in Higher Order Logics*, NASA Tech. Reports, pages 56–68, 2003.

[HV09] K. Hoder and A. Voronkov. Comparing unification algorithms in first-order theorem proving. In B. Mertsching, M. Hund, and M. Z. Aziz, editors, *KI 2009: Advances in Artificial Intelligence*, volume 5803 of *LNCS*, pages 435–443. Springer, 2009.

[HV11] K. Hoder and A. Voronkov. Sine qua non for large theory reasoning. In N. Bjørner and V. Sofronie-Stokkermans, editors, *CADE 2011*, pages 299–314, 2011.

[Jon72] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *J. Documentation*, 28:11–21, 1972.

[KAE$^+$10] G. Klein, J. Andronick, K. Elphinstone, G. Heiser, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: formal verification of an operating-system kernel. *Commun. ACM*, 53(6):107–115, 2010.

[Kal07] C. Kaliszyk. Web interfaces for proof assistants. In S. Autexier and C. Benzmüller, editors, *Proc. of the Workshop on User Interfaces for Theorem Provers (UITP'06)*, volume 174[2] of *ENTCS*, pages 49–61, 2007.

[Kau92] M. Kaufmann. An extension of the boyer-moore theorem prover to support first-order quantification. *J. Autom. Reasoning*, 9(3):355–372, 1992.

[KBKU13] 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, *ITP 2013*, volume 7998 of *LNCS*, pages 35–50. Springer, 2013.

[Kep11] J. Kepler. *Strena seu de nive sexangula.* Frankfurt: Gottfried. Tampach, 1611.

[KH12] R. Kumar and J. Hurd. Standalone tactics using OpenTheory. In L. Beringer and A. P. Felty, editors, *ITP 2012*, pages 405–411, 2012.

[KHG13] E. Komendantskaya, J. Heras, and G. Grov. Machine learning in proof general: Interfacing interfaces. In C. Kaliszyk and C. Lüth, editors, *Proceedings 10th International Workshop On User Interfaces for Theorem Provers, UITP 2012, Bremen, Germany, July 11th, 2012.*, volume 118 of *EPTCS*, pages 15–41, 2013.

[KK13] C. Kaliszyk and A. Krauss. Scalable LCF-style proof translation. 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 51–66. Springer, 2013.

[KKS91] R. Kumar, T. Kropf, and K. Schneider. Integrating a first-order automatic prover in the HOL environment. In M. Archer, J. J. Joyce, K. N. Levitt, and P. J. Windley, editors, *Proceedings of the 1991 International Workshop on the HOL theorem proving system and its Applications*, pages 170–176, University of California at Davis, Davis CA, USA, 1991. IEEE Computer Society Press.

[KN05] G. Klein and T. Nipkow. Jinja is not Java. In G. Klein, T. Nipkow, and L. Paulson, editors, *Archive of Formal Proofs.* `http://afp.sf.net/entries/Jinja.shtml`, 2005.

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

[KSU13] D. Kühlwein, S. Schulz, and J. Urban. E-MaLeS 1.1. In M. P. Bonacina, editor, *CADE 2013*, pages 407–413, 2013.

[KU13a] C. Kaliszyk and J. Urban. Automated reasoning service for HOL Light. In J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger, editors, *MKM/Calculemus/DML*, volume 7961 of *LNCS*, pages 120–135. Springer, 2013.

[KU13b] C. Kaliszyk and J. Urban. Initial experiments with external provers and premise selection on HOL Light corpora. In P. Fontaine, R. A. Schmidt, and S. Schulz, editors, *PAAR-2012*, volume 21 of *EPiC Series*, pages 72–81. EasyChair, 2013.

[KU13c] C. Kaliszyk and J. Urban. Lemma mining over HOL Light. In K. L. McMillan, A. Middeldorp, and A. Voronkov, editors, *LPAR 2013*, pages 503–517, 2013.

[KU13d] C. Kaliszyk and J. Urban. MizAR 40 for Mizar 40. *CoRR*, abs/1310.2805, 2013.

[KU13e] C. Kaliszyk and J. Urban. PRocH: Proof reconstruction for HOL Light. In M. P. Bonacina, editor, *CADE 2013*, pages 267–274, 2013.

[KU13f] 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.

[KU13g] D. Kuehlwein and J. Urban. Learning from multiple proofs: First experiments. In P. Fontaine, R. A. Schmidt, and S. Schulz, editors, *PAAR-2012*, volume 21 of *EPiC Series*, pages 82–94. EasyChair, 2013.

[KU14] C. Kaliszyk and J. Urban. Learning-assisted automated reasoning with Flyspeck. *J. Autom. Reasoning*, 53(2):173–213, 2014.

[KU15] C. Kaliszyk and J. Urban. HOL(y)Hammer: Online ATP service for HOL Light. *Mathematics in Computer Science*, 9(1):5–22, 2015.

[KUV14] C. Kaliszyk, J. Urban, and J. Vyskočil. Machine learner for automated reasoning 0.4 and 0.5. *CoRR*, abs/1402.2359, 2014. Accepted to PAAR'14.

[KUV15] C. Kaliszyk, J. Urban, and J. Vyskočil. Efficient semantic features for automated reasoning over large theories. In *International Joint Conference on Artificial Intelligence*. IJCAI/AAAI, 2015. to appear.

[KV13] L. Kovács and A. Voronkov. First-order theorem proving and Vampire. In N. Sharygina and H. Veith, editors, *CAV 2013*, pages 1–35, 2013.

[KvLT$^+$12] D. Kühlwein, T. van Laarhoven, E. Tsivtsivadze, J. Urban, and T. Heskes. Overview and evaluation of premise selection techniques for large theory mathematics. In B. Gramlich, D. Miller, and U. Sattler, editors, *IJCAR 2012*, pages 378–392, 2012.

[Ler07] X. Leroy. Formal verification of an optimizing compiler. In F. Baader, editor, *RTA*, volume 4533 of *LNCS*, page 1. Springer, 2007.

[Lov68] D. W. Loveland. Mechanical theorem proving by model elimination. *J. the ACM*, 15(2):236–251, April 1968.

[Lov78] D. W. Loveland. *Automated Theorem Proving: A Logical Basis.* North-Holland, Amsterdam, 1978.

[LS01] R. Letz and G. Stenz. Model elimination and connection tableau procedures. In Robinson and Voronkov [RV01], pages 2015–2114.

[McA89] D. A. McAllester. *Ontic: a knowledge representation system for mathematics.* MIT Press, Cambridge, MA, USA, 1989.

[McC97] W. McCune. Solution of the Robbins problem. *J. Autom. Reasoning*, 19(3):263–276, 1997.

[McC10] W. McCune. Prover9 and Mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.

[Mei00] A. Meier. System description: Tramp: Transformation of machine-found proofs into nd-proofs at the assertion level. In D. A. McAllester, editor, *CADE*, volume 1831 of *LNCS*, pages 460–464. Springer, 2000.

[MM00] W. McCune and O. S. Matlin. Ivy: A Preprocessor and Proof Checker for First-Order Logic. In M. Kaufmann, P. Manolios, and J. Strother Moore, editors, *Computer-Aided Reasoning: ACL2 Case Studies*, number 4 in Advances in Formal Methods, pages 265–282. Kluwer Academic Publishers, 2000.

[MML] The Mizar Mathematical Library. `http://mizar.org/`.

[MP08] J. Meng and L. C. Paulson. Translating higher-order clauses to first-order clauses. *J. Autom. Reasoning*, 40(1):35–60, 2008.

[MP09] J. Meng and L. C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. *J. Applied Logic*, 7(1):41–57, 2009.

[MQP06] J. Meng, C. Quigley, and L. C. Paulson. Erratum to "automation for interactive proof: First prototype" [inform. and comput. 204(2006) 1575-1596]. *Inf. Comput.*, 204(12):1852, 2006.

[MR05] R. Matuszewski and P. Rudnicki. Mizar: the first 30 years. *Mechanized Mathematics and Its Applications*, 4:3–24, 2005.

[MV12] S. Merz and H. Vanzetto. Automatic verification of TLA + proof obligations with SMT solvers. In N. Bjørner and A. Voronkov, editors, *Logic* *for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings*, volume 7180 of *LNCS*, pages 289–303, 2012.

[MW97] W. McCune and L. Wos. Otter: The CADE-13 Competition Incarnations. *J. Autom. Reasoning*, 18(2):211–220, 1997.

[NK14] T. Nipkow and G. Klein. *Concrete Semantics - With Isabelle/HOL.* Springer, 2014.

[NPW02] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[OB03] J. Otten and W. Bibel. leanCoP: lean connection-based theorem proving. *J. Symb. Comput.*, 36(1-2):139–161, 2003.

[OS06] S. Obua and S. Skalberg. Importing HOL into Isabelle/HOL. In U. Furbach and N. Shankar, editors, *IJCAR 2006*, pages 298–302, 2006.

[Ott10] J. Otten. Restricting backtracking in connection calculi. *AI Commun.*, 23(2-3):159–182, 2010.

[Pau97] L. C. Paulson. Generic automatic proof tools. In R. Veroff and G. W. Pieper, editors, *Automated Reasoning and Its Applications: Essays in Honor of Larry Wos*, pages 23–47. MIT Press, Cambridge, MA, USA, 1997.

[Pau98] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Comput. Secur.*, 6(1-2):85–128, 1998.

[Pau99] L. C. Paulson. A generic tableau prover and its integration with Isabelle. *J. UCS*, 5(3):73–87, 1999.

[PB12] L. C. Paulson and J. C. Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In G. Sutcliffe, S. Schulz, and E. Ternovska, editors, *IWIL-2010*, volume 2 of *EPiC*, pages 1–11. EasyChair, 2012.

[PBMW98] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.

[PGS06] Y. Puzis, Y. Gao, and G. Sutcliffe. Automated generation of interesting theorems. In G. Sutcliffe and R. Goebel, editors, *FLAIRS Conference*, pages 49–54. AAAI Press, 2006.

[Pit93] A. Pitts. The HOL logic. In M. J. C. Gordon and T. F. Melham, editors, *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[Poi13] H. Poincaré. *The foundations of science: Science and hypothesis, The value of science, Science and method*. The Science Press, New York, 1913.

[PS07a] 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.

[PS07b] A. Pease and G. Sutcliffe. First order reasoning on a large ontology. In G. Sutcliffe, J. Urban, and S. Schulz, editors, *ESARLT 2007*, 2007.

[PS10] J. D. Phillips and D. Stanovský. Automated theorem proving in quasigroup and loop theory. *AI Commun.*, 23(2-3):267–283, 2010.

[Pud06] P. Pudlák. Search for faster and shorter proofs using machine generated lemmas. In G. Sutcliffe, R. Schmidt, and S. Schulz, editors, *Proceedings of the FLoC'06 Workshop on Empirically Successful Computerized Reasoning, 3rd International Joint Conference on Automated Reasoning*, volume 192 of *CEUR Workshop Proceedings*, pages 34–52, 2006.

[PW06] V. Prevosto and U. Waldmann. SPASS+T. In G. Sutcliffe, R. Schmidt, and S. Schulz, editors, *ESCoR 2006*, volume 192 of *CEUR*, pages 18–33, 2006.

[Pąk13] K. Pąk. Methods of lemma extraction in natural deduction proofs. *J. Autom. Reasoning*, 50:217–228, 2013.

[Qua92] A. Quaife. *Automated Development of Fundamental Mathematical Theories*. Kluwer Academic Publishers, 1992.

[RPG05] D. Ramachandran, R. P., and K. Goolsbey. First-orderized ResearchCyc: Expressiveness and Efficiency in a Common Sense Knowledge Base. In S. P., editor, *Proceedings of the Workshop on Contexts and Ontologies: Theory, Practice and Applications*, 2005.

[ŘS10] R. Řehůřek and P. Sojka. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

[RT99] P. Rudnicki and A. Trybulec. On equivalents of well-foundedness. *J. Autom. Reasoning*, 23(3-4):197–234, 1999.

[RT03] P. Rudnicki and A. Trybulec. On the integrity of a repository of formalized mathematics. In A. Asperti, B. Buchberger, and J. H. Davenport, editors, *MKM*, volume 2594 of *LNCS*, pages 162–174. Springer, 2003.

[RV01] J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

[RV02] A. Riazanov and A. Voronkov. The design and implementation of Vampire. *AI Commun.*, 15(2-3):91–110, 2002.

[SB10] G. Sutcliffe and C. Benzmüller. Automated reasoning in higher-order logic using the TPTP THF infrastructure. *J. Formalized Reasoning*, 3(1):1–27, 2010.

[SB13] S. J. Smolka and J. C. Blanchette. Robust, semi-intelligible Isabelle proofs from ATP proofs. In J. C. Blanchette and J. Urban, editors, *PxTP 2013*, volume 14 of *EPiC Series*, pages 117–132. EasyChair, 2013.

[SBF$^+$03] J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, I. Normann, and M. Pollet. Proof development with OMEGA: The irrationality of $\sqrt{2}$. In F. Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, volume 28 of *Applied Logic*, pages 271–314. Springer, 2003.

[Sch90] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[Sch00] S. Schulz. *Learning search control knowledge for equational deduction*, volume 230 of *DISKI*. Infix Akademische Verlagsgesellschaft, 2000.

[Sch02] S. Schulz. E - A brainiac theorem prover. *AI Commun.*, 15(2-3):111–126, 2002.

[Sch04] S. Schulz. System description: E 0.81. In D. Basin and M. Rusinowitch, editors, *IJCAR 2004*, volume 3097 of *LNCS*, pages 223–228. Springer, 2004.

[Sch11] S. Schulz. First-order deduction for large knowledge bases. Presentation at Deduction at Scale 2011 Seminar, Schloss Ringberg, 2011.

[Sch13] S. Schulz. System description: E 1.8. In K. L. McMillan, A. Middeldorp, and A. Voronkov, editors, *LPAR 2013*, pages 735–743, 2013.

[SES$^+$12] T. Ströder, F. Emmes, P. Schneider-Kamp, J. Giesl, and C. Fuhs. A linear operational semantics for termination and complexity analysis of ISO Prolog. In G. Vidal, editor, *LOPSTR*, volume 7225 of *LNCS*, pages 237–252, 2012.

[SGBB98] K. Slind, M. J. C. Gordon, R. J. Boulton, and A. Bundy. System description: An interface between CLAM and HOL. In C. Kirchner and H. Kirchner, editors, *CADE*, volume 1421 of *LNCS*, pages 134–138. Springer, 1998.

[SN08] 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 *LNCS*, pages 28–32. Springer, 2008.

[SP07] G. Sutcliffe and Y. Puzis. SRASS - a semantic relevance axiom selection system. In F. Pfenning, editor, *CADE*, volume 4603 of *LNCS*, pages 295–310. Springer, 2007.

[SSCB12] G. Sutcliffe, S. Schulz, K. Claessen, and P. Baumgartner. The TPTP typed first-order form with arithmetic. In N. Bjørner and A. Voronkov, editors, *LPAR 2012*, pages 406–419, 2012.

[SSCG06] G. Sutcliffe, S. Schulz, K. Claessen, and A. V. Gelder. Using the TPTP language for writing derivations and finite interpretations. In U. Furbach and N. Shankar, editors, *IJCAR 2006*, pages 67–81, 2006.

[STC04] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, New York, NY, USA, 2004.

[Sti88] M. E. Stickel. A Prolog Technology Theorem Prover: Implementation by an extended Prolog compiler. *J. Autom. Reasoning*, 4(4):353–380, 1988.

[Sut01] G. Sutcliffe. The Design and Implementation of a Compositional Competition-Cooperation Parallel ATP System. In H. de Nivelle and S. Schulz, editors, *Proceedings of the 2nd International Workshop on the Implementation of Logics*, number MPI-I-2001-2-006 in Max-Planck-Institut für Informatik, Research Report, pages 92–102, 2001.

[Sut06] G. Sutcliffe. Semantic derivation verification: Techniques and implementation. *International Journal on Artificial Intelligence Tools*, 15(6):1053–1070, 2006.

[Sut08] G. Sutcliffe. The CADE-21 automated theorem proving system competition. *AI Commun.*, 21(1):71–81, 2008.

[Sut09] G. Sutcliffe. The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[Sut10] G. Sutcliffe. The TPTP world - infrastructure for automated reasoning. In E. M. Clarke and A. Voronkov, editors, *LPAR (Dakar)*, volume 6355 of *LNCS*, pages 1–12. Springer, 2010.

[Sut13] G. Sutcliffe. The 6th IJCAR automated theorem proving system competition - CASC-J6. *AI Commun.*, 26(2):211–223, 2013.

[Sut14] G. Sutcliffe. The CADE-24 automated theorem proving system competition - CASC-24. *AI Commun.*, 27(4):405–416, 2014.

[Tam97] T. Tammet. Gandalf. *Journal of Automated Reasoning*, 18:199–204, 1997.

[TKUG13] C. Tankink, C. Kaliszyk, J. Urban, and H. Geuvers. Formal mathematics on display: A wiki for Flyspeck. In J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger, editors, *MKM 2013*, pages 152–167, 2013.

[Tur50] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[UHV10] J. Urban, K. Hoder, and A. Voronkov. Evaluation of automated theorem proving on the Mizar Mathematical Library. In *ICMS*, pages 155–166, 2010.

[Urb03] J. Urban. Translating Mizar for first order theorem provers. In *MKM*, volume 2594 of *LNCS*, pages 203–215. Springer, 2003.

[Urb04] J. Urban. MPTP - Motivation, Implementation, First Experiments. *J. Autom. Reasoning*, 33(3-4):319–339, 2004.

[Urb06a] J. Urban. MizarMode - an integrated proof assistance tool for the Mizar way of formalizing mathematics. *J. Applied Logic*, 4(4):414 – 427, 2006.

[Urb06b] J. Urban. MoMM - fast interreduction and retrieval in large libraries of formalized mathematics. *Int. J. on Artificial Intelligence Tools*, 15(1):109–130, 2006.

[Urb06c] J. Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reasoning*, 37(1-2):21–43, 2006.

[Urb07] J. Urban. MaLARea: a metasystem for automated reasoning in large theories. In G. Sutcliffe, J. Urban, and S. Schulz, editors, *ESARLT 2007*, 2007.

[Urb08] J. Urban. Automated reasoning for mizar: Artificial intelligence through knowledge exchange. In P. Rudnicki, G. Sutcliffe, B. Konev, R. A. Schmidt, and S. Schulz, editors, *Proceedings of the LPAR 2008 Workshops, Knowledge Exchange: Automated Provers and Proof Assistants, and the 7th International Workshop on the Implementation of Logics, Doha, Qatar, November 22, 2008*, volume 418 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[Urb11a] J. Urban. An Overview of Methods for Large-Theory Automated Theorem Proving (Invited Paper). In P. Höfner, A. McIver, and G. Struth, editors, *ATE Workshop*, volume 760 of *CEUR Workshop Proceedings*, pages 3–8. CEUR-WS.org, 2011.

[Urb11b] J. Urban. Content-based encoding of mathematical and code libraries. In C. Lange and J. Urban, editors, *Proceedings of the ITP 2011 Workshop on Mathematical Wikis (MathWikis)*, number 767 in CEUR Workshop Proceedings, pages 49–53, Aachen, 2011.

[Urb12] J. Urban. Parallelizing Mizar. *CoRR*, abs/1206.0141, 2012.

[Urb14] J. Urban. BliStr: The Blind Strategymaker. *CoRR*, abs/1301.2683, 2014. Accepted to PAAR'14.

[URS13] J. Urban, P. Rudnicki, and G. Sutcliffe. ATP and presentation service for Mizar formalizations. *J. Autom. Reasoning*, 50:229–241, 2013.

[US10] J. Urban and G. Sutcliffe. Automated reasoning and presentation support for formalizing mathematics in Mizar. In S. Autexier, J. Calmet, D. Delahaye, P. D. F. Ion, L. Rideau, R. Rioboo, and A. P. Sexton, editors, *AISC/MKM/Calculemus*, volume 6167 of *LNCS*, pages 132–146. Springer, 2010.

[USPV08] 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 2008*, volume 5195 of *LNCS*, pages 441–456. Springer, 2008.

[UV13] J. Urban and J. Vyskočil. Theorem proving in large formal mathematics as an emerging AI field. In M. P. Bonacina and M. E. Stickel, editors, *Automated Reasoning and Mathematics: Essays in Memory of William McCune*, volume 7788 of *LNAI*, pages 240–257. Springer, 2013.

[UVŠ11] J. Urban, J. Vyskočil, and P. Štěpánek. MaLeCoP: Machine learning connection prover. In K. Brünnler and G. Metcalfe, editors, *TABLEAUX*, volume 6793 of *LNCS*, pages 263–277. Springer, 2011.

[Ver96] R. Veroff. Using hints to increase the effectiveness of an automated reasoning program: Case studies. *J. Autom. Reasoning*, 16(3):223–239, 1996.

[VSU10] J. Vyskočil, D. Stanovský, and J. Urban. Automated Proof Compression by Invention of New Definitions. In E. M. Clarke and A. Voronkov, editors, *LPAR (Dakar)*, volume 6355 of *LNCS*, pages 447–462. Springer, 2010.

[WAB+99] C. Weidenbach, B. Afshordel, U. Brahm, C. Cohrs, T. Engel, E. Keen, C. Theobalt, and D. Topić. System description: SPASS version 1.0.0. In *Automated Deduction - CADE-16*, volume 1632 of *LNCS*, pages 378–382. Springer Berlin Heidelberg, 1999.

[Wal00] L. Wall. *Programming Perl.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 3rd edition, 2000.

[WDF+09] C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski. SPASS Version 3.5. In R. A. Schmidt, editor, *CADE*, volume 5663 of *LNCS*, pages 140–145. Springer, 2009.

[Web11] T. Weber. SMT solvers: new oracles for the HOL theorem prover. *International Journal on Software Tools for Technology Transfer*, 13(5):419–429, 2011.

[Wen07] M. Wenzel. Isabelle/Isar—A generic framework for human-readable proof documents. In R. Matuszewski and A. Zalewska, editors, *From Insight to Proof—Festschrift in Honour of Andrzej Trybulec*, volume 10(23) of *Studies in Logic, Grammar, and Rhetoric*. Uniwersytet w Białymstoku, 2007.

[Wen09] M. Wenzel. Parallel proof checking in Isabelle/Isar. In G. Dos Reis and L. Théry, editors, *PLMMS 2009*, pages 13–29. ACM Digital Library, 2009.

[Wie99] F. Wiedijk. Mizar: An impression. `http://www.cs.kun.nl/~freek/mizar/mizarintro.ps.gz`, 1999.

[Wie01] F. Wiedijk. Estimating the cost of a standard library for a mathematical proof checker. `http://www.cs.ru.nl/~freek/notes/mathstdlib2.pdf`, 2001.

[Wie12] F. Wiedijk. A synthesis of the procedural and declarative styles of interactive theorem proving. *Logical Methods in Computer Science*, 8(1), 2012.

[WOLB84] L. Wos, R. Overbeek, E. L. Lusk, and J. Boyle. *Automated Reasoning: Introduction and Applications.* Prentice-Hall, 1984.

[Won95] 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.

[Wor11] L. Worden. WorkingWiki: a MediaWiki-based platform for collaborative research. In C. Lange and J. Urban, editors, *ITP Workshop on Mathematical Wikis (MathWikis)*, number 767 in CEUR Workshop Proceedings, pages 63–73, Aachen, 2011.

[WPN08] M. Wenzel, L. C. Paulson, and T. Nipkow. The Isabelle framework. In A. Mohamed, Munoz, and Tahar, editors, *TPHOLs 2008*, volume 5170 of *LNCS*, pages 33–38. Springer, 2008.