Hammering towards QED

Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, Josef Urban

Hammering towards QED

Jasmin C. Blanchette
Inria Nancy & LORIA, France
Cezary Kaliszyk
University of Innsbruck, Austria
Lawrence C. Paulson
University of Cambridge, UK
and
Josef Urban
Radboud University, Nijmegen, the Netherlands

This paper surveys the emerging methods to automate reasoning over large libraries developed with formal proof assistants. We call these methods *hammers*. They give the authors of formal proofs a strong "one-stroke" tool for discharging difficult lemmas without the need for careful and detailed manual programming of proof search.

The main ingredients underlying this approach are efficient automatic theorem provers that can cope with hundreds of axioms, suitable translations of the proof assistant's logic to the logic of the automatic provers, heuristic and learning methods that select relevant facts from large libraries, and methods that reconstruct the automatically found proofs inside the proof assistants.

We outline the history of these methods, explain the main issues and techniques, and show their strength on several large benchmarks. We also discuss the relation of this technology to the QED Manifesto and consider its implications for QED-like efforts.

1. INTRODUCTION

In 1993, the QED Manifesto [1] spelled out in a modern context the great vision of machine-understandable and verified mathematics and science. QED differed from earlier dreams by incorporating the concrete experience of 25 years of machine proof, the decades of research in automatic theorem proving (ATP) and artificial intelligence (AI), and the rise of the Internet and computing power. It was not just a vision, but an attempt at a plan for achieving it.

The main concern of the Manifesto was [1, p. 238] "*to build a computer system that effectively represents all important mathematical knowledge and techniques ... including the use of strict formality in the internal representation of knowledge and the use of mechanical methods to check proofs of the correctness of all entries in the system.*"

A major topic in QED and the related discussions was the role of computer assistance other than just formal verification. The three QED topics and questions

The bottom of the article's title page contains acknowledgment of support, the author(s) address(es), a "permission to copy" statement, and a line containing a copyright symbol (©) and a mysterious number. This is all entered with a `bottomstuff` environment; there must be no blank line after the `\begin{bottomstuff}` command. The permission to copy statement is produced by the `\permission` command.

that are particularly interesting to us were:

Q1 How much are automatic theorem proving and artificial intelligence necessary or useful for constructing QED-like projects?

Q2 How much do the QED-like projects in turn allow the construction and study of interesting ATP and AI systems?

Q3 How do the two previous questions relate to the perhaps most discussed QED topic: the choice of a unified logical framework?

The QED dream has not yet materialized in a form of a unique and universally supported Wikipedia-like system and repository. Nonetheless, large formal proofs and libraries have been developed in the past two decades, along with automation methods. About ten years after QED, systems that automatically reason over large libraries started to emerge, mainly motivated by questions Q1 and Q2, but also contributing some pragmatic answers to Q3. We call such systems *hammers*, after the Sledgehammer tool that has become popular in the Isabelle community and the corresponding tool HOL(y)Hammer for HOL Light.

Hammers are best known today for providing practically useful formalization technology. They give the authors of formal proofs a semi-intelligent brute force tool that can take advantage of very large lemma libraries. In his report on formal proof for the Bourbaki Seminar, Hales writes [66, p. 12]:

> 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.

A full-fledged hammer typically consists of three main components, which are invoked in succession:

(1) The *premise selector* (Section 2) heuristically identifies a fraction of the available theorems as potentially relevant to discharge the current interactive goal.

(2) The *translation module* (Section 3) constructs an ATP problem from the selected premises and the current goal, converting from the proof assistant's rich logic to the ATP's logic, often a variant of many-sorted first-order logic (FOL).

(3) The *proof reconstruction module* (Section 4) processes a proof found by an ATP so that it is accepted by the proof assistant.

Not all hammers or hammer-like systems feature all three modules, and some are equipped with additional ones. Some advisor systems, such as the Mizar Proof Advisor [132], can serve as independent library search tools that help users find useful lemmas, or they can be combined with the translation components to attempt a fully automatic proof [131, 134]. Looking for lemmas in a huge library is a time-consuming chore; one should not underestimate the potential value of such an independent search service, particularly in situations when the ATP and translation methods are still weak or underdeveloped. Proof assistants based on type theory, such as Agda, Coq, and Matita, are not easy to integrate with classical first-order

theorem provers, but could still benefit from automated lemma search. "Recommender systems" and other search-based technologies are of great importance to the information technology industry.

Other systems feature a nearly trivial translation module, which copes with a fragment of the source logic that closely matches the target logic. Finally, proof reconstruction is redundant if the user is ready to trust the hammer's translation module and the underlying provers. However, some hammers use unsound translations; and in the ATP community, more emphasis has been put on performance than on soundness.

There are also systems that are similar to hammers but have a somewhat different purpose. Some versions of MaLARea, the *Ma*chine *L*earner for *A*utomated *Rea*soning [83], focus on solving as many large-theory problems as possible within a global time limit, rather than solving a single problem within a relatively short time. Some of the techniques developed for such tools can be eventually adapted for the hammers and vice versa. Hammer-like linkups have been also used as a method for independent ATP-based checking of the Mizar natural deduction proofs [141] and for ATP-based presentation and explanation of these proofs [144].

There have been many attempts at integrating ATPs with interactive systems. We show the strength of the main systems on several large benchmarks, based on published empirical results (Section 5) as well as on some specific examples (Section 6). Then we discuss some further and future research topics related to hammers (Section 7). While we focus on our own systems, which are possibly the main ones in use today, we also attempt to give a broad historical overview of related approaches (Section 8).

Hammers were not necessarily motivated by the QED manifesto. So far, the hammering has been done towards **qed** (the end-of-proof marker in Isabelle), not QED. However, there is a clear connection between hammers and QED, which will only become stronger as the hammers themselves become stronger. Throughout this paper, we mention how the hammer systems and methods relate to QED and particularly to the three QED points given above.

A Note on Terminology. Hammers pose terminological problems because they operate at the junction of two research areas: automatic theorem proving and interactive theorem proving (ITP). For automatic provers, the de facto standard TPTP (*T*housands of *P*roblems for *T*heorem *P*rovers) [128] supplies the basic terminology. In the interactive world, the situation is more fragmented, with each proof assistant promoting its own jargon. We do not attempt to apply a uniform terminology to such a variety of systems. In fact, it is often useful to keep the ITP and ATP terminologies disjoint to avoid confusion.

In interactive proving, the basic organizational modules are typically *files*—but ACL2 calls them "books," Coq calls them "modules," Isabelle calls them "theories," and Mizar calls them "articles." These files consist of named (or numbered) *theorems*, which may be of various kinds, notably *axioms*, *definitions* (a stylized, conservative form of axioms), as well as what mathematicians would call *lemmas*, *theorems*, and *corollaries*. We already see a clash between theorem as understood by logicians (as a formula derivable in a theory) and by mathematicians (as an important result). We employ "lemma" as a generic term, since lemmas are gen-

erally the most common kind of theorems. For example, a "lemma library" may include definitions, major results (theorems), and corollaries. In Isabelle, *fact* is the generic term for a single lemma or a small collection of related lemmas. In most proof assistants, or interactive theorem provers, the formula that the user is trying to prove is a *goal* or a *subgoal*.

In automatic proving, the largest organizational unit is usually a file that specifies a *problem*: a collection of *axioms* and a *conjecture* passed together to a prover. The TPTP attempts to classify formulas further, by providing "formula roles" for definitions, theorems, hypotheses, etc., but today's provers mostly ignore this meta-information and treat all formulas other than the conjecture in the same way. And since most provers are refutational, even the conjecture is treated uniformly by taking the negation of its universal closure as an axiom and looking for a contradiction.

In the context of hammers, the ITP goal gives rise to an ATP conjecture; a selection of the ITP facts (axioms, definitions, lemmas, theorems, etc.) are specified as axioms to the ATP. However, due to complications in the translation, the ITP facts and the ATP axioms are not necessarily in a one-to-one correspondence. Since the ATP problem can be seen as an implication $\mathit{axioms} \vdash \mathit{conjecture}$, the axioms are sometimes called the *premises*, and the problem of heuristically selecting relevant axioms is called *premise selection*—although the synonyms *axiom selection* and *fact selection*, and the more general phrase *relevance filtering*, are also used.

2. PREMISE SELECTION

By *premise selection*, we mean the task of selecting a part of a large formal library that gives the automatic theorem provers a high chance of proving a given conjecture. Having usable methods for premise selection has been a prerequisite for the development of automated reasoning in large theories (ARLT) [86,129] and for the development of hammers.

By the turn of the millennium, there were some remarkable successes in automatic theorem proving, the most celebrated one being the automatic proof of the Robbins conjecture by William McCune's EQP system in 1996 [91]. ATPs have solved some open equational algebraic problems—for example, in quasigroup and loop theory [109]—finding proofs consisting of thousands of inferences. Linking such strong ATP methods and tools to proof assistants had been a research topic even before 2000, but the large library was usually viewed as an enemy and the problem of premise selection was delegated to the user. This thinking had a substantial impact on the QED question Q1 from Section 1. For example, the Manifesto states:[1]

> 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.

ARLT and hammers have brought with them a new view of large libraries, seeing them as a body of formal knowledge that can save a lot of work when managed effectively. After a decade of ARLT research, we can say that a premise selection method can occasionally beat humans at selecting lemmas from a large library. Section 6 will present some examples of this phenomenon.

[1] `http://www.rbjones.com/rbjpub/logic/qedres04.htm`

This newer, more positive view of the large libraries sees them as a rich collection of concepts and techniques from which machines can learn and reuse in similar contexts. This appears to be more promising than expecting ATP authors to extend their tools with special handling for all domains of mathematics. The large ITP libraries provide hitherto missing data for some of the earlier AI-oriented research in ATP, such as the early learning methods for guiding the internal ATP search developed for E prover by Stephan Schulz and others [54, 119]. Such *internal* methods are however still largely unexplored. Here, we give an overview of premise selection methods that can be thought of as *external* to the core ATP algorithms. The development of such methods in the last decade seems to have provided a convincing answer to QED point Q2: We have seen several interesting AI-like techniques and systems just for addressing the premise selection aspect over QED-like corpora. Nevertheless, the experiments (Section 5) show that current methods still lag behind the intelligence of mathematicians.

The premise selection methods developed so far include *non-learning approaches* based on syntactic heuristics and on heuristics using semantic information, *learning* methods that look at previous proofs, and *combinations* thereof. We explain them in the following subsections. Crucial parameters of all these methods are the characterizations—features, properties (Section 2.4)—of the mathematical formulas on which they operate. A first useful approximation is to take the symbols contained in a particular formula as its characterization. In a sufficiently large mathematical library, the set of symbols can reveal a lot about how much the formulas relate to each other.

Although the main goal of the methods is to select n premises, most also rank them in decreasing order of likely relevance as a side effect. Some ATPs (notably SPASS [24]) can use this metainformation to guide the proof search.

2.1 Non-Learning Methods

The main systems and methods that largely rely on human-constructed heuristics and criteria for premise selection include the following. Usually they start with the goal and add more premises by some iterative process.

MePo. The MePo relevance filter, by Jia *Me*ng and Lawrence *P*aulso*n* [98], keeps track of a set of *relevant symbols*, or *relevant features* (Section 2.4), initially consisting of all the goal's symbols. It performs the following steps iteratively, until n facts have been selected: (i) 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. (ii) 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.

MePo, as implemented in Isabelle/HOL, works on polymorphic higher-order formulas and exploits their rich type information. For example, if the user wants to prove something on lists of integers, it will also pick up lemmas about lists over α, where α is a type variable; but if the user wants to prove something about general lists of α, it will rightly ignore facts about lists of integers. Isabelle-independent variants of MePo have been implemented several times, for example by Yuri Puzis working in Geoff Sutcliffe's group in Miami around 2006 or for some of the ARLT

systems competing in the Large-Theory Batch (LTB) division of CADE's ATP Systems Competition.[2]

SInE. The SInE (*S*UMO *In*ference *E*ngine) algorithm by Kryštof Hoder [68] and its implementation in E by Stephan Schulz [121] are similar to MePo. The idea is to use global frequencies of symbols to define their global *generality* and build a relation, the *D-relation*, linking each symbol s with all formulas ϕ in which s has the lowest global generality among the symbols of ϕ. In common-sense ontologies, such formulas typically *define* the symbols linked to them. Premise selection for a conjecture is then done by recursively following the D-relation, starting with the conjecture's symbols. Various parameters can be used; for example, drastically limiting the recursion depth helps for the Mizar library [139], and preliminary experiments show the same for Isabelle/HOL.

The idea of using inverse frequencies of symbols in the corpus as their weights is analogous to some well-known techniques in information retrieval, such as the widely used IDF inverse document frequency (IDF) weighting [72]. To some extent, SInE can be thus seen as having a feature-preprocessing component that can be generally useful for other methods. The most recent premise selection methods indeed use IDF and its versions for feature weighting by default.

APRILS. The *A*utomated *P*rophesier of *R*elevance *I*ncorporating *L*atent *S*emantics [116] is a signature-based premise selection method that employs latent semantic analysis [52] to define symbol and premise similarity. Latent semantics is a machine-learning method that has been successfully used for example in the Netflix Prize and in web search. Its principle is to automatically derive "semantic" equivalence classes of words (such as "car," "vehicle," and "automobile") from their co-occurrences in documents and to work with these classes instead of the original words. In APRILS, formulas define the symbol co-occurrence, each formula is characterized as a vector over the symbols' equivalence classes, and the premise relevance is its dot product with the conjecture. Latent semantics is also part of Paul Cairns's Alcor system [37] for searching and providing advice over the Mizar library. Latent semantics can be understood as another method that produces suitable features with which other algorithms can work in the same way as with symbols and other features. This has been done in some recent versions of MaLARea and in the experiments over the entire Mizar library [77], using the Gensim toolkit [115].

SRASS. Semantics (in the logical sense) has been used for some time for guiding the ATP inference processes. An older system is John Slaney's SCOTT [124], which constrains Otter inferences by validity in models. A similar idea has been revived around 2004 by Jiří Vyskočil at the Prague ATP seminar: His observation was that mathematicians sometimes reject conjectures quickly on the basis of intuitive models. This motivated Petr Pudlák's semantic axiom selection system for large theories [113], implemented later also by Geoff Sutcliffe as SRASS, the *S*emantic *R*elevance *A*xiom *S*election *S*ystem. The basic idea is to use finite model finders such as MACE [92] and Paradox [42] to find countermodels of the conjecture, and gradually select axioms that exclude such countermodels. The models can differ-

[2] `http://www.cs.miami.edu/~tptp/CASC/J4/Design.html`

entiate between a formula and its negation, which is typically beyond the heuristic symbolic means. This idea has been also further developed in MaLARea [143], adding the probabilistic context with many problems solved simultaneously and many models kept in the pool, and using the models found also as classification features for machine learning.

2.2 Learning-Based Methods

Some systems use learning from previous proofs for guiding premise selection for new conjectures. They define suitable features characterizing conjectures (symbolic, semantic, structural, etc.) and use machine learning on available proofs to learn the function that associates the conjecture features with the relevant premises.

The machine learning algorithms used for ranking generally have a training and a testing (or evaluation) phase. They typically represent their data as points in preselected feature spaces. In the training phase, the algorithms learn (search for) a prediction function that best explains (fits) the training data. This learned prediction function is used in the next phase, for predicting the values on previously unseen testing data. In the setting of a large QED-like library, the algorithms would train on all existing proofs in the library and be used for proposing the most suitable premises for the next goal that a mathematician wants to prove. The speed of the training algorithms thus becomes important: New lemmas and their proofs are usually added by the user in short time intervals (minutes), and the next lemma is often strongly related to its immediate predecessors. Using a hammer that was trained without these immediate predecessors would often result in poor performance. The following learning methods have been experimented with.

Naive Bayes. Naive Bayes is a statistical learning method based on Bayes's theorem with a strong (naive) independence assumption. Given a conjecture c and a fact f, naive Bayes computes the probability that f is needed to prove c, based on previous uses of f in proving conjectures that are similar to c. The similarity is expressed using the features $F(f)$ of each formula f. The independence assumption says that the occurrence of a feature is not related to the occurrence of any other feature. The predicted relevance of a fact f with the set of features $F(f)$ is estimated by the conditional probability $P(f \text{ is relevant} \mid F(c))$.

The first premise selection experiments [3, 132, 136] used the sparse implementation provided by the SNoW (*S*parse *N*etwork *o*f *W*innows) [38] linguistic-oriented toolkit. Since then, a number of variously customized versions have been developed in C++, OCaml, Python, and Standard ML and have been used in various experiments and systems [73, 75, 77, 83, 84]. One of the main advantages of naive Bayes over more sophisticated learning algorithms is its speed and scalability over many features and training examples. In some of the hammer experiments, naive Bayes has been used with hundreds of thousands of features and examples, still providing good predictions within seconds [77, 84].

k Nearest Neighbors. The k nearest neighbors (k-NN) machine-learning method computes for a conjecture the k nearest (in a given feature distance) previous examples (typically theorems with proofs) and ranks premises by their frequency in these examples. This means that a premise needed for proofs of more of the nearest theorems gets ranked higher. This classical learning method is fast ("lazy" and

trivially incremental), and it can be easily parameterized and often behaves differently from the naive Bayes learner. Most of the k-NN implementations used for premise selection rely on distance-based weighting [55], where the contribution of the k nearest neighbors is weighted by their feature-based similarity to the current conjecture. The overall ranking of the available premises is computed as a function (typically the sum) of the premises' weighted contributions from these k neighbors.

A number of modifications of the basic distance-weighted k-NN have been tried. Examples are various (possibly recursive) schemes for weighting the neighbors' dependencies. One version takes the maximum (instead of the sum) of weights over the given k neighbors. Another version recurses into dependencies of the nearest neighbors, weighting the indirect dependencies by $F^{recursion\_level} \times distance(N)$ (where $F \in (0,1)$), and then again taking the maximum over all such factors. The k-NN method is particularly sensitive to proper weighting of features. The introduction of their IDF-based weighting has made k-NN into one of the strongest today's scalable learning methods for premise selection [79].

Kernel-Based Rankers. Kernel-based algorithms provide a way to use the machinery of linear optimization on nonlinear functions. They allow to specify a nonlinear "kernel" function, which maps data into a high-dimensional feature space, where linear regression optimizations are used. Appropriate kernel functions may generalize the essentially (logarithmically) linear approach taken by naive Bayes, and also the linear-distance approach taken by nearest neighbors. Kernel-based methods are formulated within the regularization framework that provides mechanisms to control the errors on the training set and the complexity ("expressiveness") of the prediction function. Such a setting prevents overfitting of the algorithm and leads to notably better results compared to unregularized methods.

For premise selection, kernel methods were used in the MOR [3] (multi-output ranking) and MOR-CG [86] (multi-output ranking conjugate gradient) algorithms, which differ by the kernel functions used (Gaussian or linear) and the use of the fast approximative conjugate-gradient method in MOR-CG to speed up the most expensive parts of the algorithm. In the training phase, for each premise p, MOR or MOR-CG tries to find a function C_p such that for each conjecture c, $C_p(c) = 1$ if and only if p was used in the proof of c. Given a new conjecture c, the learned prediction function C_p is evaluated on c, and the higher the value $C_p(c)$ the more relevant p is to prove c. These methods perform well on smaller benchmarks [3,86]; however, they still have not been scaled up to the larger corpora that are routinely processed by the hammers.

BiLi. BiLi (*Bi-Li*near) is an experiment by van Laarhoven based on a bilinear model of premise selection, similar to the work of Wei Chu and Seung-Taek Park [40]. Like the kernel-based rankers, BiLi aims to minimize the expected loss. The difference lies in the kind of prediction functions they produce. In MOR and MOR-CG, the prediction functions only take the features of the conjecture into account. In BiLi, the prediction functions use the features of both the conjectures and the premises. The bilinear model learns a weight for each combination of a conjecture feature together with a premise feature. Together, this weighted combination determines whether a premise is relevant to the conjecture. In an early experiment [86], this method was much weaker than the other learning methods,

but improvements may be possible.

2.3 Combinations of Methods

A fascinating issue in large-theory reasoning is the large number of alternative proofs that can be found using different premise selections. This should perhaps not be surprising, because the premises are organized into a large derivation graph, with many relationships among them. The premise selection algorithms described above are based on different ideas of similarity, relevance, and functional approximation spaces and norms. They can be better or worse in capturing different aspects of the premise selection problem, whose optimal solution is generally undecidable.

With machine learning, we could try the combination of different predictors. This corresponds to the common situation when there are a number of experts, each of them equipped with different skills, and they need to be combined in a way that increases their performance as a whole. There has been much machine learning research in this area. *Ensembles* [112] is one of the most popular, while David Sculley [122] deals with the specific task of aggregating rankers.

A combination of two very different premise selection methods—MOR-CG (kernel-based learning) and SInE (non-learning)—turns out to be effective [86]. The final ranking was obtained via weighted linear combination of the predicted individual rankings, testing a limited grid of weights in the interval $(0, 1)$—e.g., $0.5 \times \text{CG} + 0.5 \times \text{SInE}$. Even this simple weighting scheme improved the peak performance (in terms of the number of problems solved) of the strongest single method (MOR-CG) by 10%. In later experiments [77], more advanced combinations using different aggregation techniques resulted in the best results so far.

Such ensemble methods are usually much faster than the standard time slicing, where the results of the various selection methods are combined by running the ATPs independently using the premise sets that were selected by the different selection methods. The time used for premise selection is usually much smaller than the time given to the ATPs that run on the selections. The final hammers do use time slicing over all appropriate parameters (features, rankers, ATPs, etc.). For example, the server-based hammers for Mizar and HOL Light [77, 81] use 14 complementary methods run in parallel on a server with many CPUs.

2.4 Features and Proofs for Premise Selection Methods

Given a conjecture c, how do mathematicians determine that certain previous knowledge will be relevant for proving c? The approach taken in the majority of premise selection methods is to extract from c a number of suitably defined features to use as the input to premise selection for c. The most obvious features for characterizing mathematical statements in large theories are their *symbols* [68, 98, 131]. In addition, the hammers have used the following features:

- types, i.e., type constants, type constructors, and type classes [80];
- term walks of length 2 [84];
- subterms [143];
- validity in a large pool of finite models [143];
- metainformation such as the theory name and presence in various databases [84].

Several normalizations for term and type variables have been tried:

- replacing variables by their (variable-normalized) types [80];
- using de Bruijn indices [143];
- renaming all variables to a unique common variable [143];
- using the original variable names (which can be useful in the presence of consistent naming conventions).

The feature "validity in a pool of finite models" requires finding a diverse set of models, in which the formulas are then evaluated. Even though it may approximate the semantics very well, it is expensive to compute compared with the other features, which can be extracted in linear time. For this reason, the feature has been avoided in the main hammer setups. The following example shows one of the more advanced feature characterization algorithms run on the HOL Light theorem `DISCRETE_IMP_CLOSED`:[3]

$$\forall s \subseteq \mathbb{R}^N \, \forall \epsilon \in \mathbb{R}^+ (\forall x, y \in s \; |y - x| < \epsilon \to y = x) \to s \text{ is closed}$$

whose exact HOL Light wording is as follows:

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

The premise selectors use the following features:

```
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 applying the following steps:

(1) Normalize all type variables to just one, `A`.
(2) Replace all term variables with their normalized type.
(3) Collect all normalized types and their component types recursively.
(4) Collect all normalized atomic formulas and their component terms recursively.
(5) Include all nonlogical terms and type constructors.

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

Feature Weighting and Preprocessing. In the first learning-based methods, various sets of features were simply combined together, relying on the machine learners to deal with them, possibly measuring which combinations provided better or worse performance [143]. This causes little harm with naive Bayes, which seems to be relatively robust in such situations.

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

The situation is different for the k-NN learner, where the default way to measure the similarity of formulas with respect to the conjecture is to compute the overlap of their (sparse) feature vectors. This makes k-NN sensitive to feature frequencies [43]. For example, without additional weighting, the most common symbol has the same weight as the rarest symbol, even though the latter carries much more information.

The most common way to weight Boolean features in text retrieval with respect to their frequency is the inverse document frequency (IDF) scheme [72]. This scheme weights a term (word) t in a collection of documents D using the logarithm of the inverse of the term's frequency in the document collection:

$$\mathrm{IDF}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

For example, a term (symbol) contained only in one document (formula) will have weight $\log |D|$, a term contained in all of them will have weight $\log 1 = 0$, and a term contained in half of the documents will have weight $\log 2 = 1$. Introducing the IDF weighting in k-NN costs only a few lines of code, but it resulted in one of the largest performance increases in scalable premise selection so far. The IDF-weighted k-NN suddenly improved over the best previous scalable method (naive Bayes) over the Flyspeck corpus by 30% [79]. It seems to show again that human-written formal mathematical libraries are not far from human-written linguistic corpora.

From other feature preprocessing methods, only the latent semantic analysis has been tried so far, with some successes [77, 83]. Methods such as random projections [50], latent Dirichlet allocation [27], and the recent Word2Vec model [100] are still waiting to be properly incorporated and tested in the context of large theories.

More and Better Proofs. Anonther interesting problem is getting from the corpora the most useful proof dependencies for learning premise selection. Many of the original ITP dependencies are clearly unnecessary for first-order theorem provers. For example, the definition of the $\wedge$ connective (`AND_DEF`) in Flyspeck is used to prove 14 122 theorems. Another example are proofs performed 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 most of the dependencies are not actually needed.

Some obviously unhelpful dependencies can be filtered by manual heuristics, similar to some of the manual blacklisting procedures used in the more advanced non-learning selectors such as Isabelle's MePo. An automatic exhaustive ITP-based minimization was developed for Mizar [3], pruning the dependencies by a factor of two or three. However, the article refactoring used in this method is fragile and Mizar-specific. The method is computationally expensive (even though some learning-based optimizations can make it faster [5, §6.4]). The measured performance gain from the method when re-proving with ATPs the MPTP2078 problems was only about 4%, and ATPs typically do not use many of the background formulas needed in Mizar [4]. It is thus not surprising that several large experiments showed that learning from minimized ATP proofs is usually preferable to learning from the ITP proofs [80, 86].

Since the ATPs can today re-prove a significant part (40%–50%) of the theorems from their (suboptimal) ITP dependencies, a simpler method of getting good proof

data has been used in the recent experiments over Flyspeck and MML, using several MaLARea-style [136, 143] iterations between proving and learning. The first set of proofs is obtained by running ATPs on the re-proving problems with the (suboptimal) ITP dependencies. Then premise selection is learned on that set and ATPs are run again on various most relevant slices of the re-proving problems. This adds about 6% of proofs for MML. The following passes no longer prune the re-proving problems, but just use the premise selectors trained on the previous passes to suggest the most relevant premises, regardless of the original proof dependencies. Four such passes add about 16% of proofs for MML [77].

These iterations are also expensive to compute for a new development. However, this information can be efficiently reused in the large libraries by employing recursive content-based naming schemes [137] that make the computed data robust against renamings and other manipulations [81]. Another promising set of methods headed in this direction are heuristic (e.g., statistical) methods that attempt to align the concepts and theorems in different libraries [61]. These methods could eventually significantly increase the pool of expensively computed ATP data available for learning, by transferring data between the libraries.

3. TRANSLATION

A central component of most hammer systems is the translation module, which encodes formulas from the proof assistant's logic into the target ATP's logic. Since different ATPs may support different logics, this module is typically parameterized or extended by various backends. For HOL Light and Isabelle/HOL, the proof assistant's formalism is higher-order logic and includes polymorphic types. For Mizar, this is set theory with some higher-order features and dependent soft types (predicates with automations [133] attached) with subtyping. The target logic typically provides at most monomorphic types (if any), even though some ATPs (notably, SPASS) may try to recover some of the more complicated typing mechanisms automatically from the structure of the predicates [149].

3.1 Translation in HOL-Based Systems

Abstractly, the translation module in Sledgehammer and HOLyHammer can be seen as a two-step process:

(1) Eliminate the higher-order features of the formulas to produce a first-order problem.
(2) Encode the type information in the target logic.

In practice, the two steps are intertwined, because this gives rise to opportunities for optimizations.

Translation of Higher-Order Features. The higher-order features to eliminate are listed below, with an example of each.

(1) Higher-order quantification: $\forall x.\, \exists f.\, f\ x = x$;
(2) Higher-order arguments: $\mathsf{map}\ f\ [x, y] = [f\ x, f\ y]$;
(3) Partial function applications: $f = g \longrightarrow f\ x = g\ x$;
(4) λ-abstractions: $(\lambda x\, y.\, y + x) = (\lambda x\, y.\, x + y)$;

(5) Formulas within terms: $\mathsf{p}\ (x = x) \longrightarrow \mathsf{p}\ \mathsf{True}$;
(6) Terms as formulas: $\mathsf{hd}\ [\mathsf{True}]$.

Early Sledgehammer prototypes classified the problem as purely first-order or higher-order and treated the two situations independently. The presence of a single higher-order construct was enough to trigger a heavy, systematic translation of the problem. Eventually, the third author realized that it is possible to make a smooth transition from purely first-order to heavily higher-order problems. To give a flavor of the translation, here are the six examples above after their higher-order features have been eliminated, expressed in a TPTP-like syntax:

(1) $\forall X : \alpha.\ \exists F : (\alpha, \alpha)\ \mathit{fun}.\ \mathsf{app}(F, X) = X$
(2) $\mathsf{map}(F, \mathsf{cons}(X, \mathsf{nil})) = \mathsf{cons}(\mathsf{app}(F, X), \mathsf{nil})$
(3) $F = G \longrightarrow \mathsf{app}(F, X) = \mathsf{app}(G, X)$
(4) $\mathsf{C}(\mathsf{plus}) = \mathsf{plus}$
(5) $\mathsf{p}(\mathsf{equal}(X, X)) \longrightarrow \mathsf{p}(\mathsf{true})$
(6) $\mathsf{boolify}(\mathsf{hd}(\mathsf{cons}(\mathsf{true}, \mathsf{nil})))$

The symbols app, boolify, C, equal, and true are uninterpreted symbols introduced by the translation. They are characterized by auxiliary axioms, such as $\mathsf{equal}(X, X) = \mathsf{true}$ and $\mathsf{app}(\mathsf{app}(\mathsf{C}(F), X), Y) = \mathsf{app}(\mathsf{app}(F, Y), X)$, that can be used by the ATPs to perform some higher-order reasoning.

More specifically, the app function symbol serves as an explicit application operator. It takes a (curried) function and an argument and applies the former on the latter. It permits the application of a variable: If F is a variable, $F(0)$ is illegal in first-order logic, but $\mathsf{app}(F, 0)$ is legal. It also makes it possible to pass a variable number of arguments to a function, as is often necessary if the problem requires partial function applications. The boolify symbol is a predicate that yields true if and only if its Boolean term argument is true. Intuitively, $\mathsf{boolify}(t)$ is the same as $t = \mathsf{true}$, where true is the uninterpreted constant corresponding to the Isabelle/HOL constant True.

The distinguished symbols app and boolify can hugely burden problems if introduced systematically for all arguments and predicates. To reduce clutter, Sledgehammer and HOL$\forall$Hammer compute the minimum arity n needed for each symbol and pass the first n arguments directly, falling back on app for additional arguments. This optimization works well in practice, but it sometimes makes problems unprovable.

Translation of Types. After translating away the higher-order features of a problem, we are left with first-order formulas in which polymorphic types (and, in Sledgehammer's case, type classes) are still present. Various type encoding schemes are possible, but the traditional schemes require burdening the formulas with so much information that the ATPs almost grind to a halt. Until 2011, Sledgehammer implemented a lightweight but unsound translation of types. Since proofs need to be reconstructed anyway, a mildly unsound translation could be used with some success, but this was not entirely satisfactory:

(1) Finite exhaustion rules of the form $X = \mathsf{c}_1 \vee \cdots \vee X = \mathsf{c}_n$ must be left out because they lead to unsound cardinality reasoning in the absence of types [97,

§2.8]. The inability to encode such rules prevents the discovery of proofs by case analysis on finite types.

(2) Spurious proofs are distracting and sometimes conceal sound proofs. The seasoned user eventually learns to recognize facts that lead to unsound reasoning and mark them with a special attribute to remove them from the scope of the relevance filter, but this remains a stumbling block for the novice.

(3) In the longer run, it would be desirable to let the ATPs themselves perform relevance filtering, or even use a sophisticated system based on machine learning, such as MaLARea, where successful proofs guide subsequent ones. However, if unsound encodings are used, such approaches tend to quickly discover and exploit any inconsistencies in the large translated axiom set.

Even in the monomorphic case, it is necessary to encode types, as the equality predicate implemented by ATPs is a polymorphic one. Encoding each type instance of equality as a different predicate together with an axiomatization of each of these predicates directly corresponds to the MESON reconstruction and has been used in the early versions of HOL(y)Hammer. A recent discovery in hammer research is that it is possible to encode polymorphic types in a sound, complete, and efficient manner. Sledgehammer implements a whole family of lightweight encodings that exploit a semantic property called *monotonicity* to safely remove most of the type information that previously cluttered problems [22, 23, 41]. Informally, monotonic types are types whose domain can be extended with new elements while preserving satisfiability. Such types can be merged by synchronizing their cardinalities. Non-monotonic types can be made monotonic by encoding some typing information.

As an example, consider the following Isabelle/HOL formulas:

$$\begin{aligned} \mathsf{Nil} &\neq \mathsf{Cons}\ x\ xs \\ \mathsf{hd}\ (\mathsf{Cons}\ x\ xs) &= x \\ \mathsf{tl}\ (\mathsf{Cons}\ x\ xs) &= xs \end{aligned}$$

The unsound encoding used by earlier versions of Sledgehammer would encode the implicit type variable α as a term variable A, in order to resolve overloading and type classes correctly, but it did not attempt to restrict the range of the universal variables:

$$\begin{aligned} \mathsf{nil}(A) &\neq \mathsf{cons}(A,\ X,\ Xs) \\ \mathsf{hd}(A,\ \mathsf{cons}(A,\ X,\ Xs)) &= X \\ \mathsf{tl}(A,\ \mathsf{cons}(A,\ X,\ Xs)) &= Xs \end{aligned}$$

To effectively prevent ill-typed variable instantiations, one of the two traditional approaches is to add a predicate $\mathsf{type}(X,\ A)$ that encodes the constraint that X has type A, yielding

$$\begin{aligned} \mathsf{type}(X,\ A) \wedge \mathsf{type}(Xs,\ \mathsf{list}(A)) &\longrightarrow \mathsf{nil}(A) \neq \mathsf{cons}(A,\ X,\ Xs) \\ \mathsf{type}(X,\ A) \wedge \mathsf{type}(Xs,\ \mathsf{list}(A)) &\longrightarrow \mathsf{hd}(A,\ \mathsf{cons}(A,\ X,\ Xs)) = X \\ \mathsf{type}(X,\ A) \wedge \mathsf{type}(Xs,\ \mathsf{list}(A)) &\longrightarrow \mathsf{tl}(A,\ \mathsf{cons}(A,\ X,\ Xs)) = Xs \end{aligned}$$

The translation is complemented by typing axioms for all function symbols, which are necessary to discharge the type constraints:

$$\mathsf{type}(\mathsf{nil}(A),\, \mathsf{list}(A))$$
$$\mathsf{type}(X,\, A) \wedge \mathsf{type}(Xs,\, \mathsf{list}(A)) \longrightarrow \mathsf{type}(\mathsf{cons}(A,\, X,\, Xs),\, \mathsf{list}(A))$$
$$\mathsf{type}(Xs,\, \mathsf{list}(A)) \longrightarrow \mathsf{type}(\mathsf{hd}(A,\, Xs),\, A)$$
$$\mathsf{type}(Xs,\, \mathsf{list}(A)) \longrightarrow \mathsf{type}(\mathsf{tl}(A,\, Xs),\, \mathsf{list}(A))$$

Such an approach faithfully encodes the original type system, but it forces the prover to perform inferences to discharge type assumptions, thereby slowing down proof search and lengthening the proofs dramatically. An alternative, based on a type function symbol instead of a predicate, keeps the formulas simpler but burdens the terms instead. The two approaches are roughly equally inefficient in practice.

Exploiting monotonicity of the list type, the above problem can be encoded more simply as

$$\mathsf{nil}(A) \neq \mathsf{cons}(A,\, X,\, Xs)$$
$$\mathsf{type}(X,\, A) \longrightarrow \mathsf{hd}(A,\, \mathsf{cons}(A,\, X,\, Xs)) = X$$
$$\mathsf{tl}(A,\, \mathsf{cons}(A,\, X,\, Xs)) = Xs$$

with the typing axioms

$$\mathsf{type}(X,\, \mathsf{list}(A))$$
$$\mathsf{type}(\mathsf{hd}(A,\, Xs),\, A)$$

More efficient encodings are possible by heuristically (and incompletely) instantiating the type variables in the problem and using the target ATP's support for monomorphic types, if available. Only a few years ago, most provers had no native support for types. This has changed with the rise of SMT (satisfiability modulo theories) solvers. Nowadays, E is perhaps the only remaining high-performance ATP that has no type support.

3.2 Translation of Mizar

Mizar can be seen as first-order logic plus the infinitely many axioms of Tarski–Grothendieck set theory, but it is more conveniently seen as having some higher-order and abstraction mechanisms. For the first large-scale experiments performed over Mizar in 2003 [131], these higher-order features of Mizar were translated in a simplistic and incorrect way. Their frequency was low enough and not likely to significantly confuse the initial statistics about the possible chances of ATPs on the Mizar problems. No such large-scale statistics existed prior to these first experiments, and the expectations were low. Once the first somewhat encouraging ATP statistics were obtained, a proper translation addressing most of the Mizar features (including proofs) was started, involving a relatively large reimplementation of some parts of the Mizar infrastructure [135].

The translation [131, 134, 141] is done in two steps: first by exporting the Mizar internal representation rather faithfully to an extended TPTP-like Prolog syntax, and then translating this extended TPTP syntax into TPTP formulas and problems in various ways. This Prolog module first targeted various batch-style translation

and creation of TPTP problems for experiments and benchmarks from the whole MML, and later the functions for fast translation of single problems were added as part of the online MizAR hammer-like system [140, 142]. The Mizar features addressed by the current translation and ATP problem-creation module include

(1) the Mizar soft type system, including term-dependent types, subtyping, and intersection types;
(2) higher-order language constructs such as abstract (Fraenkel) terms (comprehensions) and the Hilbert choice operator;
(3) second-order theorem schemes such as the Replacement and Induction schemes;
(4) the Mizar abstract structure types;
(5) safe approximation of the Mizar algorithms that use the background knowledge implicitly, especially regarding types;
(6) translation of the Jaśkowski-style natural deduction proofs into extended assumption-based TSTP proofs that can be independently verified.

Translation of Higher-Order Features. Abstract terms, also called Fraenkel terms (comprehensions), are set-theoretical abbreviations for unique objects guaranteed by the Replacement and Comprehension axioms of Zermelo–Fraenkel set theory. Here is an example, in Mizar syntax:

{ N ^ M **where** M, N is Integer : N divides M }

The same example can be written in extended TPTP syntax as follows:

```
all([M : integer, N : integer], divides(N, M), power(N, M))
```

Abstract terms are eliminated by introducing a new functor symbol, corresponding to the abstract term in the following way:

```
![X]: (in(X, all_0_xx) <=>
       ?[N : integer, M : integer]: (X = power(N, M) & divides(N, M))).
```

where `all_0_xx` is the newly introduced Fraenkel functor for the abstract term given above. Fraenkel functors with nonzero arity can arise if their context includes quantified variables. An easy optimization is to extract the abstract terms from the proof context by generalizing the proof-local constants appearing inside terms before creating the definitions of the Fraenkel functors. When this is done for a whole Mizar article, the number of Fraenkel definitions can be reduced significantly, sometimes by a factor of 10 or even 20, and the equality of such functors becomes immediate, rather than having to be established through the Extensionality axiom.

The FOL translation of the second-order schemes such as Replacement is similar to that of the abstract terms: The translation simply detects and possibly generalizes all the first-order instances which are already present in MML. This is sufficient for first-order re-proving of the problems, and with a sufficiently large body of mathematics like MML (and thus sufficiently many first-order scheme instances), it is often sufficient for proving new things, although obviously incomplete in general. A complete method discussed several times when higher-order automatic theorem provers such as LEO-II [16] and Satallax [35] started to show some promise would be to translate such problems also to higher-order syntax (TPTP THF). Other approaches might include heuristic problem-dependent instantiation

of the schemes (possibly involving some probabilistic/learning component), switching to Neumann–Bernays–Gödel set theory as used by Art Quaife [114], or adopting a deep embedding of the Zermelo–Fraenkel constructs in FOL similar to that done by the MetaMath system [95] or by the combinator encodings used in the translators from higher-order logic.

Translation of Types. The following example of matrix multiplication illustrates the use of term-dependent types. Rather than using the human-friendly Mizar syntax, the example is written in the extended TPTP syntax, which roughly corresponds to the internal Mizar representation.

```
![K : natural, A : matrix(K), B : matrix(K)]:
    sort(matrix_multiply(K, A, B), matrix(K)).
```

This formula states that matrix multiplication is well-defined only for two square matrices of dimension K, and its result is also a K-matrix. Semantically, the Mizar types are simply first-order predicates, and the current straightforward translation method translates n-ary type (sort) symbols into $(n + 1)$-ary predicate symbols and relativizes by such predicates (i.e., implication for universal quantification and conjunction for existential). For the example above, the result (in standard TPTP notation) is

```
![K, A, B]: (natural(K) & matrix(K, A) & matrix(K, B)) =>
            matrix(K, matrix_multiply(K, A, B)).
```

An example of more advanced sorted quantification involving the intersection and complements of types `finite` and `graph` is

```
![G : (~ finite & graph), W1 : walk(G), W2 : subwalk(G, W1)]: ...
```

This is the extended TPTP representation of the following Mizar quantification over infinite graphs, walks on them, and their subwalks:

for G **being** infinite Graph, W1 **being** Walk **of** G,
W2 **being** Subwalk **of** W1 **holds** ...

The resulting standard TPTP is as follows:

```
![G, W1, W2]: (~ finite(G) & graph(G) & walk(G, W1)
               & subwalk(G, W1, W2)) => ...
```

The problem creation phase tries to ensure that all necessary typing formulas are added to the problems. Proposed lightweight type-encoding schemes based on inclusion operators [49] may struggle to deal with the implicit associative–commutative behavior of Mizar's intersection types. (In our example, should the inclusion encoding be `~ finite(graph(G))` or `graph(~ finite(G))`?) An incomplete but relatively efficient handling of the Mizar type system was implemented for the MoMM system [133] by extending the E prover's term and indexing structures and prioritizing the predicates originating from Mizar types during clause subsumption. Such ad hoc heuristic techniques might bring more efficiency to ATPs (analogously to the soft-typing mechanisms of SPASS) when dealing with the Mizar type system.

4. PROOF RECONSTRUCTION

Some hammer systems merely exploit the ATPs as *oracles*, meaning that the answer of the ATP is trusted and not rechecked. Oracles bypass the proof assistant's inference kernel and compromise the system's trustworthiness. Moreover, the user must also trust the translation from the interactive prover into the ATP's logic. This is unacceptable to many users of proof assistants, who consider a theorem proved only if it has been certified by the assistant's inference kernel.

The alternative to oracles is to perform *proof reconstruction*, by reducing ATP proofs to kernel inferences. There are three main approaches to achieve this:

(1) *Replay the ATP proof directly, inference by inference, inside the proof assistant.* This is the approach used by PRocH [78], the reconstructor for HOL*y*Hammer, by the Isabelle proof methods *metis* [106] (based on the Metis prover) and *smt* [30] (based on Z3), and by the SMT bridge for HOL4 [30]. Internally, the inferences are simulated using a combination of standard proof methods (such as the simplifier and decision procedures for arithmetic). However, because the proof must be rediscovered each time the user's formalization is processed, this requires the ATP to be available on the user's machine for replaying (or a highly reliable server and Internet connection). A variation of this approach is to store the raw proofs alongside the theory, allowing replay without the ATP; however, the raw proofs must be regenerated whenever the theories change.

(2) *Check the ATP proofs using a verified checker.* This approach is an alternative to direct proof replay, where the proof is checked rather than simulated. The proof checker runs within the proof assistant itself, and yields a proof of the theorem there by *reflection* [32]. This is implemented in the SMT integration in Coq [6] and the Waldmeister integration in Agda [59]. Like the previous approach, it requires the ATP to be available or the use of raw proofs.

(3) *Translate the ATP proofs to the proof assistant's source form.* This was the original approach implemented for Sledgehammer [106]. Various postprocessing can be done on the resulting proofs to make them more palatable [20,21,110,111, 126]. The translation need not preserve all the steps of the ATP proofs. In an extreme form, the ATP proof is completely ignored except for its dependencies (or unsatisfiable core) and replaced by a one-line proof in the proof assistant, typically using a general proof method such as `by` in Mizar, *metis* in Isabelle/ HOL, and MESON in HOL Light.

One of the main difficulties with proof reconstruction is that each prover has its own (often undocumented) proof format and inference rules. The first two approaches are typically much more efficient than the third one, because they largely eliminate search. Success rates can be arbitrarily high with any approach, depending on the level of detail of the ATP proofs; textual proofs (approach 3), in combination with *metis* (approach 1) and other Isabelle tactics, yield reconstruction success rates above 99% in a recent evaluation [21].

In the context of Isabelle, proof reconstruction with *metis* means that the proof must be rediscovered each time the Isabelle theory text is processed; the ATPs are used merely as highly precise relevance filters. This approach sometimes fails for difficult proofs that *metis* cannot find within a reasonable time and is vulnerable to

small changes in the formalization. It also provides no answer to users who would like to understand the proof. But perhaps more importantly, *metis* and similar methods support no theories beyond equality, which is becoming a bottleneck as automatic provers are being extended with procedures for theory reasoning.

The Z3-based *smt* proof method, which implements the first approach listed above, is often a powerful alternative to *metis*, but it depends on the availability of Z3 on the user's machine for proof replay (or the use of stored raw Z3 proofs), which hinders its acceptance among users. In addition, its incomplete quantifier handling means it can fail to find a proof generated by a resolution prover.

An alternative to both *metis* and *smt* is to translate the ATP proofs into structured, multi-line Isabelle proofs. Each step of such a proof is discharged by a suitable proof method; for example, the linear arithmetic steps of an SMT solvers are replayed using Isabelle's decision procedures. Isabelle's textual proof reconstruction module [21] is integrated with many systems (E, LEO-II, Satallax, SPASS, Vampire, veriT, Waldmeister, and Z3). Proofs by contradiction are turned around into direct proofs to make them more readable. The following example was found in an entry of the *Archive of Formal Proofs* [60] about regular algebras. The users wrote the skeleton of the proof, but the *metis* call in the base case and the **proof**–**qed** block in the induction step were generated by Sledgehammer:

lemma *powsum_ub:* $i \le n \Longrightarrow x^i \le x_0^n$
proof (*induct n*)
  case 0 **show** *case*
    by (*metis* (*hide_lams*, *mono_tags*) *0.prems eq_iff le_0_eq power_0 powsum_00*)
next
  case (Suc *n*) **show** *case*
  proof −
    { **assume** *aa1*: $\mathsf{Suc}\ n \neq i$
      have *ff1*: $x_0^{\mathsf{Suc}\ n} \le x_0^{\mathsf{Suc}\ n} \wedge \mathsf{Suc}\ n \neq i$
        using *aa1* **by** *fastforce*
      have *ff2*: $\exists a.\ x_0^n + a \le x_0^{\mathsf{Suc}\ n} \wedge \mathsf{Suc}\ n \neq i$
        using *ff1 powsum2* **by** *auto*
      have $x^i \le x_0^{\mathsf{Suc}\ n}$
        using *ff2*
        by (*metis Suc.hyps Suc.prems add_lub le_SucE less_eq_def*) }
    thus $x^i \le x_0^{\mathsf{Suc}\ n}$
      using *less_eq_def powsum_split_var2* **by** *auto*
  qed
qed

(The notation x_0^n stands for the power sum $x^0 + \cdots + x^n$.)

There is a considerable body of research about processing ATP proofs. Early work focused on translating resolution proofs into natural deduction calculi [101, 108]. Although they are arguably more readable, these calculi operate at the logical level, whereas humans reason mostly at the "assertion level," invoking definitions and lemmas without providing the full logical details. A line of research focused on transforming natural deduction proofs into assertion-level proofs [9,69], culminating

with the systems TRAMP [96] and Otterfier [151]. More related work includes the identification of obvious inferences [51, 117], the successful transformation of the EQP-generated proof of the Robbins conjecture using ILF [47], and the use of TPTP-based tools to present Mizar articles [144].

Perhaps the distinguishing feature of proof reconstruction as performed in modern hammers is that it targets a wide range of ATPs and is rather ad hoc, leveraging existing proof methods whenever possible instead of attempting to closely reconstruct the ATPs' ever evolving, often undocumented calculi.

5. PERFORMANCE DATA

Developing powerful hammers is largely an experimental science. In this section, we present evaluations of the main systems. The systems concerned use different relevance filtering mechanisms, different logics and translations, and different reconstruction mechanisms. Even the formalizations provide different levels of detail. Therefore, the data presented here cannot be used to compare the systems directly.

5.1 Proving Theorems from Their Original Dependencies

The first interesting experiment with ATPs over QED-like corpora is proving the theorems from their exact proof dependencies as recorded by the proof assistants. Such ITP dependencies may be quite redundant and different from the facts an ATP would have used, but they are still useful. As mentioned in Section 2.4, such initial ITP-based proof data can be also significantly improved by running ATPs on the corresponding problems in various ways and by minimizing the number of dependencies by ITP-internal methods. The data such obtained are then useful for learning premise selection (Section 5.3).

Mizar. The first ATP experiment over the whole MML, performed in 2003, consisted in proving from their approximate Mizar dependencies the 27 449 MML (version 3.44.763) theorems that were handled by the first version of the MPTP translator [131]. SPASS 2.1 was used with a 300 s time limit on a cluster of 700 MHz Intel Pentium III CPUs with 200 MB RAM. The evaluation took 70 CPU–days. 11 222 (41%) problems were proved, and about 90% of these were solved within 40 s. The success rate was surprisingly high, but subject to some doubts because of the oversimplifications taken in the first MPTP translation (Section 3.2). For example, SPASS found 625 countersatisfiable problems.

This experiment was repeated with MPTP 0.2 in 2005, with 12 529 MML (version 4.48.930) problems that did not use Mizar internal arithmetics [134]. Lack of handling of the Mizar internal arithmetics was the last (known) potential source of ATP incompleteness in MPTP 0.2. E 0.9 and SPASS 2.1 were used for this experiment, with a 20 s time limit on a cluster of dual Intel Xeons 3.06 GHz with 2 GB RAM each. E was able to prove 4309 of the problems, whereas SPASS proved 3850. Together, they reached 4854 problems (39%). No countersatisfiability was detected.

In 2010, a large evaluation on MML (version 4.100.1011) on 51 424 problems was done with Kryštof Hoder and Andrei Voronkov [139], using their newly overhauled Vampire (whose version numbering confusingly restarted at 0 at the time) and including all of MML, including arithmetic problems. The translation of Mizar

arithmetics has been handled in a heuristic way for some time by MPTP then. The time limit was 30 s, and an eight-core Intel Xeon E5520 2.27 GHz with 8 GB RAM and 8 MB CPU cache was used. No parallelism was used, each problem was always run by one CPU. Table I shows the results. Together, the three ATPs solved 44% of all the MML problems; Vampire alone solved 39%. The MML as a whole seems on average harder than just its nonarithmetic problems considered before, and also later versions of MML typically contain more complicated problems proved by more advanced ITP technology.

Table I: E, SPASS, and Vampire on all MML (version 4.100.1011) problems with ITP dependencies run for 30 s in 2010 (51 424 problems)

ATP	Proved (%)	Countersat.
E 1.1-004	16 191 (31.5%)	4
SPASS 3.7	17 550 (34.1%)	12
Vampire 0.6	20 109 (39.1%)	0
Collectively	22 607 (44.0%)	12

The most recent such evaluation on MML (version 4.181.1147) was done in 2013 [77], using Vampire 3.0 with 300 s on 48-core AMD Opteron 6174 2.2 GHz with 320 GB RAM. Out of all 57 897 top-level theorems, Vampire proved 27 842 (48%). Since these ATP problems are constructed by an overapproximation of what was actually used by Mizar, another set of problems was then constructed by learning premise selection on the 27 842 solutions, and using the trained selectors to reorder the premises in the unsolved problems according to their estimated relevance, and taking the initial slices. E, Vampire, and Z3 were then run for 120 s on these slices, adding 1677 solutions, bringing the total of solved problems to 29 519 (51%). Even though limited to small problems, such methods already rely on more advanced premise selection (Section 5.3).

However, the number of lines that were used to prove these theorems in Mizar is only about 23% of the lines used to prove all Mizar theorems. This shows that the ATPs are much better in proving the top-level theorems that have a short Mizar proof. On the other hand, this metric would improve a lot if also the proof-local lemmas were included in the experiments, and the number of lines corresponding to such lemmas was included in the statistics.

HOL Light. Table II shows the results of running Vampire 2.6, Epar (E 1.6 with automatically developed new large-theory strategies [138]), Z3 4.0, and Paradox 4.0 on 14 185 Flyspeck top-level theorems constructed from their HOL Light proof dependencies [80]. The time limit for Epar, Vampire, and Z3 was 900 s. Paradox was run for 30 s to get some measure of the incompleteness of the translation used for HOL Light (Section 3.1). The hardware was again a 48-core AMD Opteron 6174 server at 2.2 GHz with 320 GB RAM. The systems in Table II are already ordered using the *greedy covering sequence*. This is a sequence that starts with the best system, and each next system in the sequence is the system that greedily adds most solutions to the union of solutions of the previous systems in the sequence.

The joint success rate of the top two systems is 5949 (42%), and all of them prove 6142 problems (43%). The "Unique" column gives the number of theorems proved only by the given system.

Table II: Epar, Vampire, and Z3 re-proving with 900 s and Paradox with 30 s (14 185 problems)

ATP	Proved	Unique	Countersat.	Greedy
Vampire	5641	218	0	5641
Epar	5595	194	0	5949
Z3	4375	193	2	6142
Paradox	5	0	2614	6142
Collectively	6142		2614	

5.2 Proving Large Problems without Premise Selection

How important is the selection of a small number of relevant premises? In the first experiments in 2003, this was investigated: For each proof attempt, SPASS 2.1 was given as premises all the theorems that were proved before. This made the problems much larger than ATPs like SPASS were designed for. Unfortunately, the results were invalidated by imperfections in the first MPTP translations.

In 2010, this experiment was repeated on 1000 randomly selected MML (version 1011) problems, using a sound MPTP translation. Table III shows the results of evaluation of E 1.1 and Vampire 0.6 run with 30 s on Intel E7300 2.66 GHz with 1 GB RAM. While Vampire in this version started to use SInE (Section 2.1) automatically, E did not have SInE built-in yet and was instead combined with Hoder's first standalone filter. The problems had on average 40 898 formulas, and the SInE selection took on average 4 s. The performance of raw E was only 2%, increasing to 14–15% using the strictest version (`-d1`) of the SInE premise selector.

Table III: E and Vampire with SInE (`-d1`) on 1000 large MML problems in 30 s

Vampire+SInE	Vampire+SInE(-d1)	E	E+SInE	E+SInE(-d1)
84	141	21	64	153

5.3 Proving Large Problems with Premise Selection

The results in the previous two subsections have shown that limiting the premises to the most relevant could raise the ATP's chances from 2% to 40%–50% on the top-level ITP lemmas. This has motivated the premise selection research (Section 2), with and without learning. We describe some experiments below. While the non-learning methods just select from a large library of previous facts, the learning methods learn from previous proofs. In both cases, we need the notion of lemma *accessibility*—which other lemmas are available at the point where a given lemma is stated and can be used it its proof? Otherwise, we could easily obtain cyclic

proofs or proofs advised by their future cousins. This notion of accessibility usually already exists over the libraries, which have a partial order of dependencies that can be topologically sorted into a linear *chronological* ordering.

Mizar. For the first experiments in 2003, a naive Bayes learner (from the SNoW toolkit) was incrementally trained and evaluated on the chronologically sorted training examples extracted from the MML. Each training example used the theorem's symbols as its features and the theorem's name and its explicit Mizar proof references as labels. For example, the one-line proof of the theorem `RELAT_1:8`[4] stating that every relation R is equal to the intersection of R with the cartesian product of the domain and range of R:

> **theorem** RELAT_1:8
> **for** R **being** Relation **holds** R ∧ [:(dom R),(rng R):] = R
> **by** RELAT_1:7,XBOOLE_1:28;

would result in the following training example:

```
Relation, /\, [::], dom, rng, =
RELAT_1:8, RELAT_1:7, XBOOLE_1:28
```

This informs the learner that when it sees the symbols on the first line, it should increase the relevance of the labels on the second line. In the incremental training/evaluation mode, given an example, SNoW estimates its labels based on the features and previous examples, before it learns from the actual labels in that example. For each theorem, the 30 highest-ranked labels were given to SPASS as axioms, together with the MPTP-estimated necessary typing formulas. The overall success rate was 14%. While some of these successes were again due to imperfect translation, the ratio of such spurious proofs did not seem to be significant.

In 2005, this was repeated with MPTP 0.2 over the 12 529 MML (version 4.48.930) problems that did not use Mizar internal arithmetics. E 0.9 and SPASS 2.1 were used, with a 20 s time limit on a cluster of dual Intel Xeons 3.06 GHz with 2 GB RAM each. Some 17% of the problems were proved by E, 12% by SPASS, and 19% by either [134].

Remarkably, 329 of these 2408 ATP proofs used fewer Mizar references than the original Mizar proofs, and in some cases the new proofs were much shorter. No spurious proofs were found by a manual inspection. A more thorough analysis of the differences between the ATP and human proofs was done in 2011 [4]; a few examples will be considered in Section 6.1.

The latest evaluation on MML, carried out in 2013, combined a number of new methods for premise selection, for feature extraction and preprocessing, and for improving the proof data used for learning. These methods were first optimized on a smaller random subset of 1930 theorems and then evaluated on all 57 897 top-level theorems. The evaluation used 30 s for each method on a 48-core AMD Opteron 6174 2.2 GHz with 320 GB RAM.

The performance of the 14 most useful methods (when considered jointly) is shown in Table IV. The methods are ordered there from top to bottom by their position in the greedy covering sequence for the whole MML. The table says that

[4] `http://mizar.cs.ualberta.ca/~mptp/8.1.03_5.29.1227/html/relat_1.html#T8`

running these fourteen methods in parallel for 30 s gives a 41% chance of solving an MML theorem without any user interaction. The best method alone—a combination of k-NN and naive Bayes using different features and taking their minimal rank—is more than 50% better than the previously used naive Bayes on symbol-only features.

Table IV: 14 most covering methods on the whole MML, ordered by greedy coverage

Method	Parameters	Premises	ATP	Proved	Greedy
comb	min_2k_20_20	128	Epar	15 789	15 789
lsi	3200ti_8_80	128	Epar	15 561	17 985
comb	qua_2k_k200_33_33	512	Epar	13 907	19 323
knn	is_40	96	Z3	11 650	20 388
nb	idf010	128	Epar	14 004	21 057
knn	is_80	1024	Vampire	12 277	21 561
geo	r_99	64	Vampire	11 578	22 006
comb	geo_2k_50_50	64	Epar	14 335	22 359
comb	geo_2k_60_20	1024	Vampire	12 382	22 652
comb	har_2k_k200_33_33	256	Epar	15 410	22 910
geo	r_90	256	Vampire	13 850	23 107
lsi	3200ti_8_80	128	Vampire	14 783	23 259
comb	geo_2k_50_00	96	Vampire	15 139	23 393
geo	r_90	256	Epar	14 093	23 478

HOL Light. Table V shows similar statistics done on the 14 185 Flyspeck problems in 2012 [80]. The hardware was the same as for Mizar (48-core AMD Opteron 6174 2.2 GHz with 320 GB RAM), but each ATP was run for 300 s. Since k-NN did not use the IDF weighting yet and combined methods were not used either, naive Bayes was by far the strongest method, solving 27% of the problems when used with the most suitable features, premises and proof data. The combination of the best 14 methods solved 39% when each given 300 s, and nearly as much when given only 30 s. This fast decrease of the efficiency of the superposition-based ATPs has been observed many times, motivating the early work on strategy scheduling in Gandalf [130]. It is thus not surprising that the main "meta-method" to improve the efficiency of large-theory reasoning is to develop and combine many different and complementary premise selectors, taking different numbers of the proposed premises, and running complementary ATP strategies on them.

Isabelle/HOL. Sledgehammer has been continuously evaluated over the years. Earlier papers by the third author and his colleagues at Cambridge used a handcrafted collection of problems to evaluate the effect of various translation schemes and for tuning the MePo relevance filter.

In late 2009, Tobias Nipkow and Sascha Böhme [29] performed the first independent evaluation of Sledgehammer. They selected seven Isabelle theories, which they considered to be representative:

Table V: 14 most covering methods on Flyspeck, ordered by greedy coverage

Method	Parameters	Premises	ATP	Proved	Greedy
nb	symst+m10u2+atponly	154	Epar	3810	3810
nb	symst+m10u2+atponly	1024	Epar	3280	4273
nb	symst+m10u2+atponly	92	Vampire	3740	4686
knn40	symst+m10u+atponly	32	Epar	2417	4938
nb	syms0+triv	512	Epar	3239	5148
nb	all000001+old+triv	128	Vampire	2475	5247
nb	symst+m10u+atponly	32	Z3	2191	5332
winnow	symst	128	Epar	1704	5411
knn160	symst+m10u+atponly	512	Z3	1872	5454
nb	v_pref+triv	128	Epar	2814	5492
nb	symst+m10+v_pref+atponly	128	Z3	2257	5528
nb	symst+m10u2+atponly	128	Epar	3799	5553
nb	symst	32	Z3	1408	5571
winnow	symst	32	Z3	711	5580

Arrow	Arrow's impossibility theorem
NS	Needham–Schroeder shared-key protocol
Hoare	Completeness of Hoare logic with procedures
Jinja	Type soundness of a subset of Java
SN	Strong normalization of the typed λ-calculus with de Bruijn indices
FTA	Fundamental theorem of algebra
FFT	Fast Fourier transform

They applied the hammer to each of the 1240 goals that arise in these theories, including to intermediate subgoals. This benchmark suite came to be called *Judgment Day*, after the title of their paper. When running E 1.0, SPASS 3.5, and Vampire 9.0 in parallel for 120 s, followed by Metis with a 30 s time limit, they found that Sledgehammer could solve 48% of the goals.

The experiment was repeated in 2010 to account for the further development of Sledgehammer in Munich [105]. In particular, the tool now communicated with ATPs using full first-order logic instead of clause form, added E-SInE 0.4 to the collection of ATPs, and upgraded to SPASS 3.7 and Vampire 1.0. The results are summarized in Table VI, broken down by theory.

In 2011, SMT solvers were added as backends to Sledgehammer. The corresponding evaluation considers the solvers CVC3 2.2, Yices 1.0.28, and Z3 2.15 in addition to E 1.2, E-SInE 0.4, SPASS 3.7, and Vampire 1.0. The time limit was lowered to 30 s, to reflect Sledgehammer's default. (Earlier versions of the system worked with a 60 s limit, but this was lowered to 30 s when it was observed that the additional time rarely leads to more proofs.) To exercise the SMT solvers' support for arithmetic, two theories were added:

QE	DNF-based quantifier elimination
S2S	Sum of two squares

Table VI: Success rate of Sledgehammer on 1240 goals in 120 s (2010)

	Arrow	*NS*	*Hoare*	*Jinja*	*SN*	*FTA*	*FFT*	∅
E	19%	39%	45%	33%	66%	57%	17%	44%
E-SInE	18%	22%	43%	31%	61%	53%	17%	40%
SPASS	30%	35%	43%	32%	59%	58%	17%	44%
Vampire	36%	40%	50%	35%	63%	60%	17%	47%
Collectively	43%	45%	54%	41%	68%	65%	26%	52%

Table VII summarizes the results. Notice that the SMT solvers were slightly ahead of the traditional, superposition-based ATPs E, SPASS, and Vampire. The lack of support for types (sorts) and arithmetic is one reason for this.

Table VII: Success rate of Sledgehammer on 1591 goals in 30 s (2011)

	Arrow	*FFT*	*FTA*	*Hoare*	*Jinja*	*NS*	*QE*	*S2S*	*SN*	∅
E	24%	15%	60%	42%	31%	31%	25%	39%	60%	40%
SPASS	34%	14%	57%	51%	32%	34%	28%	39%	60%	42%
Vampire	31%	19%	62%	49%	35%	44%	23%	48%	60%	44%
SInE	23%	16%	55%	40%	31%	28%	20%	39%	63%	38%
CVC3	36%	18%	53%	51%	37%	29%	21%	57%	55%	42%
Yices	29%	18%	51%	51%	37%	31%	23%	59%	59%	42%
Z3	48%	18%	62%	54%	47%	42%	25%	58%	62%	49%
Superposition	40%	21%	67%	55%	37%	45%	31%	55%	70%	50%
SMT	50%	23%	66%	65%	48%	42%	27%	66%	63%	52%
All ATPs	55%	28%	73%	67%	48%	51%	41%	73%	72%	59%

On the superposition-based prover front, work of the first author in collaboration with Christoph Weidenbach's group in Saarbrücken resulted in a new version of SPASS that outperformed all other provers on the *Judgment Day* suite [24] at the time. The type encoding issues that affected earlier experiments were mostly resolved by the development of newer, lightweight encodings [22] and by the addition of monomorphic types to SPASS and Vampire.

In 2013, a new relevance filter based on machine learning, called MaSh [84], was introduced as a complement to the non-learning-based MePo. The combination of both filters is called MeSh. An evaluation on *Judgment Day* with E 1.6, SPASS 3.8ds, Vampire 2.6, and Z3 3.2 found a total success rate of 70%, when the four ATPs are run in parallel for up to 60 s.

5.4 ATP-Based Verification of Complete Proofs

The first experiments with using ATPs over Mizar [131, 134] indicated that an overwhelming majority of atomic inference (**by**) steps in the MML can be proved without any further guidance. The success rate was more than 99% of the 18 429 atomic inference steps extracted from 48 initial Mizar articles [134]. Given this

Table VIII: Success rate of Sledgehammer on 1268 goals in 60 s (2013)

	MePo	MaSh	MeSh
E	55%	50%	57%
SPASS	57%	49%	58%
Vampire	55%	50%	56%
Z3	53%	52%	61%
Collectively	66%	63%	70%

capability for verification of atomic inference steps, one can develop techniques extending this to ATP-based verification of entire Mizar proofs, with the potential target of ATP-based verification of the whole MML. This is useful because Mizar is less secure than many other proof assistants, whose architectures rely on a comparatively small inference kernel.

To achieve this, the MPTP translation of formulas has been extended to handle also whole Mizar proofs, producing TPTP derivations similar to those produced by ATPs like Vampire and E [141]. The TPTP format was enriched to encode derivations containing *assumptions* [144]. Correctness conditions for such derivations were designed and implemented in Geoff Sutcliffe's GDV verification tool [127].

Two larger subsets of MML were exported into such extended TPTP derivations and checked with the extended GDV. GDV first does structural verification checks, such as checking that assumptions are correctly propagated and discharged. Then it encodes the required semantic relationship between each inferred formula and its parent formulas into proof obligations, expressed as ATP problems. These problems are given to trusted ATP systems. The 252 MPTP Challenge problems resulted in 6765 ATP problems, of which E and MaLARea automatically proved all but 17. Among these 17 specimens, 14 were solved by E after manual pruning of premises, and the remaining 3 unsolved problems exposed a rare completeness bug in the MPTP ATP problem-creation phase, which could be solved.

It would be generally hard for other proof assistants, such as HOL Light and Isabelle/HOL, to import and verify the translated Mizar proofs. However, it is likely that these systems will be able to check the detailed proof objects created by the ATP systems during this cross-verification, or even use their recent stronger proof-reconstruction techniques (Section 4) for this task. In this way, it seems that the hammers have unintentionally provided technology that can be used to establish links between the ITP systems, giving a partial answer to QED point Q3 mentioned in the introduction. In this case, the logic used as the common denominator is just classical untyped first-order logic.

6. EXAMPLES

The large experiments over entire libraries allow rigorous evaluation of the strength and improvements of the hammer methods, but it is also useful to see some concrete examples of what the methods can and cannot do. In this section, we first give several examples of the more complicated or surprising proofs found with the hammers, before we show their performance on a short challenge problem.

6.1 Remarkable Machine-Generated Proofs

Sometimes machines can beat mathematicians at their own game. An example of this is the proof of the FACE_OF_POLYHEDRON_POLYHEDRON theorem from Flyspeck, found by HOL*y*Hammer. The theorem states that a face of a polyhedron (defined in HOL Light as a finite intersection of half-spaces) is again a polyhedron:

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

The HOL Light proof[5] takes 23 lines and could not be re-played by ATPs from the ITP dependencies, but a much simpler alternative proof was found by the premise selection and the ATPs, by exploiting 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 facts that are obvious to mathematicians: Every polyhedron is convex (POLYHEDRON_IMP_CONVEX), the intersection of two polyhedra is again a polyhedron (POLYHEDRON_INTER), and an affine hull is always a polyhedron (POLYHEDRON_AFFINE_HULL):[6]

```
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)
```

An even more striking example of this phenomenon was found in the recent experiments over MML [77]. The most remarkable hammer-based proof shortening occurred for the theorem REARRAN1:24[7] about rearrangements of finite functions, which has a 534-lines long Mizar proof, while the shortest ATP proof found has only 5 dependencies. This shortening is due to a symmetry between the concepts used in this theorem and the previously proved theorem REARRAN1:17[8], 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 system has managed to express the difference between the two theorems in an operational way, whereas the human authors failed to capture the symmetry so succinctly. In some sense, the AI/ATP system has thus managed to formulate and use a new simple mathematical trick.

Sometimes, the machine-generated proof is much more complicated than the original. The Flyspeck theorem BOUNDED_CLOSURE_EQ[9] states that a set in $\mathbb{R}^n$ is bounded if and only if its closure is bounded. The harder direction of the equivalence

[5] http://mws.cs.ru.nl/~mptp/cgi-bin/browseproofs.cgi?refs=FACE_OF_POLYHEDRON_POLYHEDRON

[6] Even though the proof was found by an ATP, we show the original HOL Light statements, which are more readable than their TPTP translations.

[7] http://mizar.cs.ualberta.ca/~mptp/mml4.181.1147/html/rearran1.html#T24

[8] http://mizar.cs.ualberta.ca/~mptp/mml4.181.1147/html/rearran1.html#T17

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

was already available as theorem BOUNDED_CLOSURE and was used both by the HOL Light and by the ATP proof. The easier direction was in HOL Light proved by theorems CLOSURE_SUBSET and BOUNDED_SUBSET, which state that any set is a subset of its closure and any subset of a bounded set is bounded. The ATP proof 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
```

An example of a difficult ATP proof found by Sledgehammer is the following lemma about intervals being Borel sets, proved in Isabelle as follows:

lemma *greaterThanAtMost_borel*: $\{a <..b\} \in$ sets borel
  unfolding *greaterThanAtMost_def atLeastLessThan_def*
  by (*blast intro*: *borel_open borel_closed open_lessThan*
    open_greaterThan open_greaterThanLessThan
    closed_atMost closed_atLeast closed_atLeastAtMost)+

The Sledgehammer proof requires the following Isabelle theorems:

Set.set_eq_subset, *Borel_Space.greaterThanLessThan_borel*,
Borel_Space.atLeastAtMost_borel, *Set_Interval.ivl_disj_int_one*(3),
Set_Interval.order_class.greaterThanAtMost_empty_iff,
Set_Interval.ord_class.greaterThanAtMost_iff,
Set_Interval.order_class.greaterThanAtMost_empty,
Set_Interval.linorder_class.Ioc_subset_iff,
Set_Interval.linorder_class.Ioc_inj, *Set_Interval.ivl_disj_un*(19),
Sigma_Algebra.sets.empty_sets, *Sigma_Algebra.sets.Un*

An example of a proof that is within the reach of ATPs, but so far outside the reach of the hammers and their premise selectors, is the 30 lines long proof of Lagrange's theorem in Mizar:[10]

theorem GROUP_2:177
for G **being** finite Group
for H **being** Subgroup **of** G **holds** card G = (card H) ∗ (index H)

[10] http://www.tptp.org/MizarTPTP/Articles/group_2.html#T177

SPASS can prove this theorem when given the 25 needed premises. However, already the default MPTP problem-creation algorithm includes another 135 (typing) formulas into the ATP problem, making this version impossible to re-prove, and the premise selectors are unable to make such a fine selection.

Such examples can obviously serve as a material for further focused improvement of the premise selectors, by studying the particular circumstances that allowed the ATP to prove the problem with the exact premises and the issues caused by adding further unnecessary premises. On the other hand, this particular problem was solved only after manual effort by the last author triggered by a challenge from David Stanovský—an algebraist interested in proof automation. A more automated (data-driven), albeit perhaps less spectacular, way used today for obtaining such interesting examples for further study is to run the premise selectors with many more options and slices on a large set of problems, analyzing the most interesting proofs (e.g., the longest, or those using many axioms) thus obtained.

6.2 A Challenge Problem

Various challenge problems have been formulated by people interested in QED and formalization. At the 1995 QED workshop, John McCarthy proposed the classic mutilated chessboard problem [90] as a test showing how far we are from "heavy-duty set theory," i.e., an ITP technology that would understand most of the steps that mathematicians perform in their proofs. Formalizations of the problem followed in Mizar [11], Isabelle/HOL [104], and ProofPower.[11] A similar challenge problem, the irrationality of $\sqrt{2}$, was chosen by Freek Wiedijk [150] to compare the "seventeen provers of the world." In October 2014, during one of the recurring discussion about formalization on the Foundation of Mathematics mailing list, mathematician Lasse Rempe-Gillen suggested a small challenge problem:[12]

> It is hard for me to understand exactly what the [formalization] overhead is, without getting my hands dirty and actually doing it. Here is a very simple statement which I often give to students as a first exercise in iteration, and to practice formal mathematics.
>
> Let f be a real-valued function on the real line, such that $f(x) > x$ for all x. Let x_0 be a real number, and define the sequence x_n recursively by $x_{n+1} := f(x_n)$. Then x_n diverges to infinity.
>
> A standard proof might go along the following steps: 1) By assumption, the sequence is strictly increasing; 2) hence the sequence either diverges to infinity or has a finite limit; 3) by continuity, any finite limit would have to be a fixed point of f, hence the latter cannot occur.
>
> What effort would be required to formalize this kind of statement using current technology?

This problem was quickly formalized in Isabelle/HOL (by Joachim Breitner), Mizar (by Josef Urban), and HOL Light (by Freek Wiedijk), providing a comparison of the language and proof style, libraries, and also of the automation methods.

[11] http://www.lemma-one.com/ProofPower/examples/wrk071.pdf
[12] http://www.cs.nyu.edu/pipermail/fom/2014-October/018243.html

Below we show for the three formalizations how many of the subgoals could be solved by the corresponding hammers. This is not intended as a direct comparison of the hammers, which run with different resource limits and over quite different libraries, but rather to show on a realistic small example how useful the hammers today are. The subgoals that could be solved by the hammers are marked with "*success*" and those where the hammers fail marked with "*failure*." In a few cases, one-line proof reconstruction is not powerful enough, and the hammers provide a detailed subproof. All the three formalizations quickly discovered that the original informal statement by Rempe-Gillen misses a crucial continuity assumption.
Isabelle with Sledgehammer:

```
lemma lasse:
  fixes f :: "real ⇒ real"
  assumes "continuous_on UNIV f"
  assumes "∀x. f x > x"
  shows "(λn. ereal ((f^^n) x₀)) ----> ∞"
proof -
  have "incseq (λn. (f^^n) x₀)"  — failure
  proof
    fix n m :: nat assume "n <= m"
    thus "(f ^^ n) x₀ <= (f ^^ m) x₀"  — failure
    proof (induction rule: dec_induct)
      show "(f ^^ n) x₀ <= (f ^^ n) x₀"  — success
    next
      fix n'
      assume "(f ^^ n) x₀ <= (f ^^ n') x₀"
      have "(f ^^ n') x₀ < f ((f ^^ n') x₀)"  — success
      have "(f ^^ n') x₀ <= (f ^^ Suc n') x₀"  — success
      show "(f ^^ n) x₀ <= (f ^^ Suc n') x₀"  — success
    qed
  qed
  show "(λn. ereal ((f^^n) x₀)) ----> ∞"  — failure
  proof (rule ccontr)
    assume "¬ ?thesis"
    obtain B where "∀N. ∃ n ≥ N. ((f^^n) x₀) ≤ B"  — failure
    have "∀n. ((f^^n) x₀) ≤ B"  — success
    obtain L where "(λn. (f^^n) x₀) ----> L"  — success
    have "(λn. f ((f^^n) x₀)) ----> f L"  — failure
    have "(λn. (f^^Suc n) x₀) ----> f L"  — success
    have "(λn. (f^^n) x₀) ----> f L"  — success
    have "L = f L"  — success
    have "f L > L"  — success
    show False  — success
  qed
qed
```

Mizar with Miz$\mathbb{A}\mathbb{R}$:

```
reserve n for natural number;
reserve r for real number;
reserve m for Element of NAT;

now
  let f be continuous Function of REAL, REAL such that A1: f.r > r ;
  let x be Element of REAL;
  deffunc F(set,set) = f.$2;
  consider s being Real_Sequence such that
    s.0 = x and A4: s.(n+1) = F(n,s.n) from NAT_1:sch 12; :: failure
  s.m < s.(m+1) :: success
  proof
    s.m < f.(s.m) by A1; :: success
    hence s.m < s.(m+1) by A4; :: success
  end;
  then A7: s is increasing by SEQM_3:def 6; :: failure
  now
    assume s is bounded_above;
    then A9: s is convergent by A7; :: success
    A2: dom f = REAL by PARTFUN1:def 2; :: success
    then B9: f is_continuous_in lim s by FCONT_1:def 2; :: success
    A11: rng s c= dom f by RELAT_1:def 19, A2; :: success
    then A12: f /* s is convergent & f . (lim s) = lim (f /* s)
      by A9,B9,FCONT_1:def 1; :: failure
    (f /* s) . n = s.(n+1) :: failure
    proof
      reconsider n as Element of NAT by ORDINAL1:def 12; :: success
      (f /* s) . n = f.(s.n) by A11,FUNCT_2:108 :: failure
      .= s.(n+1) by A4; :: success
      hence thesis; :: success
    end;
    then A13: f /* s = s ^\ 1 by NAT_1:def 3; :: failure
    then f . (lim s) = lim (f /* s) by A12 :: success
          .= lim (s ^\ 1) by A13 :: failure
          .= lim s by A9,SEQ_4:22; :: success
    hence contradiction by A1; :: failure
  end;
  then s is divergent_to+infty by A7,LIMFUNC1:31; :: success
end;
```

HOL Light with HOLyHammer:

```
g `∀x. f contl x) ∧ ∀x. f x > x) ∧ ∀n. x (n + 1) = f (x n)) ⟹
   ∀y. ∃n. x n > y`;;  failure
e (REWRITE_TAC [real_gt]);;  failure
e (STRIP_TAC);;  failure
e (SUBGOAL_THEN `mono x` ASSUME_TAC);;  failure
(* *** Branch 1 *** *)  success
  e (ASM_REWRITE_TAC [MONO_SUC; ADD1; real_ge; REAL_LE_LT]);;  success
(* *** Branch 2 *** *)  failure
  e (ASM_CASES_TAC `convergent x`);;  failure
  (* *** Branch 2.1 *** *)  failure
    e (POP_ASSUM MP_TAC);;  failure
    e (REWRITE_TAC [convergent]);;  failure
    e (STRIP_TAC);;  failure
    e (SUBGOAL_THEN `(λ n. x n) tends_num_real (f:real→real) l`
      (ASSUME_TAC o REWRITE_RULE [ETA_AX]));;  failure
    (* *** Branch 2.1.1 *** *)  failure
      e (ONCE_REWRITE_TAC [SEQ_SUC]);;  failure
      e (ASM_SIMP_TAC [ADD1; CONTL_SEQ]);;  failure
    (* *** Branch 2.1.2 *** *)  success
      e (ASM_MESON_TAC [SEQ_UNIQ; REAL_LT_REFL]);;  success
  (* *** Branch 2.2 *** *)  failure
    e (SUBGOAL_THEN `¬bounded (mr1,(≥)) (x:num→real)` ASSUME_TAC);;  failure
    (* *** Branch 2.2.1 *** *)  success
      e (ASM_MESON_TAC [SEQ_BCONV]);;  success
    (* *** Branch 2.2.2 *** *)  failure
      e (SUBGOAL_THEN `∀y y'. ∃n. ¬(y ≤ (x:num→real) n) ∨ ¬(x n ≤ y')`
        (ASSUME_TAC o REWRITE_RULE [REAL_NOT_LE]));;  failure
      (* *** Branch 2.2.2.1 *** *)  success
        e (ASM_MESON_TAC [SEQ_BOUNDED_2]);;  success
      (* *** Branch 2.2.2.2 *** *)  failure
        e (REWRITE_TAC [FORALL_NOT_THM; EXISTS_NOT_THM]);;  failure
        e (SUBGOAL_THEN `¬ ∃y. ∀n. ¬(y < (x:num→real) n)`
            (fun th → MESON_TAC [th]));;  failure
        e (DISCH_THEN (CHOOSE_THEN ASSUME_TAC));;  failure
        e (SUBGOAL_THEN `∀n. x 0 ≤ x n` ASSUME_TAC);;  failure
        (* *** Branch 2.2.2.2.1 *** *)  failure
          e (INDUCT_TAC);;  failure
          (* *** Branch 2.2.2.2.1.1 *** *)  success
            e (ARITH_TAC);;  success
          (* *** Branch 2.2.2.2.1.2 *** *)  success
            e (ASM_REWRITE_TAC [ADD1]);;  success
            e (REWRITE_TAC [REAL_LE_LT]);;  success
            e (DISJ1_TAC);;  success
            e (MATCH_MP_TAC REAL_LET_TRANS);;  success
            e (EXISTS_TAC `x (n:num):real`);;  success
            e (ASM_REWRITE_TAC []);;  success
        (* *** Branch 2.2.2.2.2 *** *)  failure
          e (FIRST_ASSUM (STRIP_ASSUME_TAC o SPECL [`x 0:real`; `y:real`]));;  failure
          (* *** Branch 2.2.2.2.2.1 *** *)  success
            e (ASM_MESON_TAC [real_le]);;  success
          (* *** Branch 2.2.2.2.2.2 *** *)  success
            e (ASM_MESON_TAC []);;  success
```

7. FURTHER AND FUTURE TOPICS

Hammer research is likely to continue for many years. More can be done both in the hammers themselves and in the underlying provers towards increasing the success rate and spare users some effort.

7.1 Further Issues in Premise Selection

There are many interesting problems related to premise selection that have not been extensively researched yet. One such topic is the level on which premise selection should be done. For humans, the most understandable units are the ITP theorems as stated in the libraries. For ATPs, it already pays off to split larger conjunctions and treat the conjuncts as separate theorems, tracking their dependencies separately if possible. For Mizar, the premise selection benefits from being done after the translation to FOL, where Mizar's rich type system becomes explicit. On the other hand, some translation optimizations in Isabelle and HOL Light are best done only after the premise selection has been performed on the HOL level. To make this topic even merrier, some recent unpublished experiments with the E.T. system [76] have shown that premise selection after clausification can improve or complement the premise selection done on the FOL level. And a very large topic opens when one starts to consider for reuse not just the top-level ITP theorems, but also the millions or billions of smaller lemmas that are proved explicitly or implicitly during the ITP or ATP proof search [82, 133].

There has not been much research yet in optimizing the size of the premise slices in different contexts. Most of the methods use rules of thumb such as doubling the size of the most relevant slice, trying to balance off the ATP search for deep proofs using only a few most relevant axioms with the search for shallow proofs when the premise selection is not guaranteed to be too good, or just for the odd chance of finding an unexpected proof. In principle, one could rely on trained prediction methods again, using their (possibly multiple) confidence thresholds to decide whether a premise should be included. A related problem that also waits for more experiments is a dependent selection of premises, similar to what MePo and SRASS do. Choosing a particular premise should have some influence for choosing the other premises—for example, to avoid two alternative statements of the same lemma. This could lead to interesting ideas related to modeling selection by Markov processes, with SRASS-like ideas concerning the semantics.

Finally, a topic that is also so far unexplored is the interaction of the external selection systems with the internal premise selectors and other large-theory techniques used automatically inside modern ATPs. For example, the recent work on growing strong large-theory SInE strategies and clause selection heuristics for the E prover [79, 138] has significantly improved E's capability to deal with many axioms. This needs to be again taken into account by the premise selectors and globally optimized.

7.2 Internal Guidance in ATPs

The high-level premise selection methods developed so far use existing ATP systems as black boxes, but they do not guide the internal proof search process once the axioms are selected. Without such guidance, combinatorial explosion often prevents finding more difficult proofs automatically, even if the minimal set of premises

needed for the proof is selected with 100% precision. However, ATP-internal proof-guiding methods based on previous proof knowledge are not yet very developed. Given the success of premise selection, there seems to be a lot of potential in transferring some of these methods into the cores of the ATP algorithms. There are several ways to such smarter ATP-internal methods.

The most prominent and successful internal-guiding method is probably the "hints" method [147] by Bob Veroff (a bit differently done also by Stephan Schulz), that directs the proof search towards the lemmas (hints) that were found useful in proofs of related problems. Such methods have so far been used in small theories and often manually tweaked. However, their better automation and scaling to large theories seems feasible. Related to such hint guidance are Stephan Schulz's *Conjecture Symbol Weight* clause selection heuristics in E prover [120], giving lower weights to the symbols contained in the conjecture, thus preferring during the inference steps the clauses that have common symbols with the conjecture. While such approach is to some extent analogous to heuristics like MePo, another possible approach is to completely guide the inference process by the experience learned from a large number of previous proofs. This has been so far tried with the MaLeCoP (*Ma*chine *Le*arning *Co*nnection *P*rover) prototype [145], advising leanCoP's [102] connection tableau by a naive Bayes learner. MaLeCoP has shown an average pruning of the search space by a factor of 20, but its construction poses a number of technical challenges—in particular, the speed of feature extraction, learning, and advising is crucial for such systems to be practically useful.

An internal-guidance approach that works on a higher level is invention of suitable sets of ATP strategies, i.e., parameters that control (sometimes even program) the ATP search. An example of such strategy-inventing system is the Blind Strategymaker [138], which on a large library of different problems co-evolves a population of ATP strategies with a population of solvable training problems, with the ultimate goal of having a reasonably small set of strategies that together solve as many of the problems as possible. The first experiments with such strategy evolution has improved E's performance on the Mizar and Flyspeck problems by 25%. A related topic of research is selection of suitable ATP strategies or portfolios of such strategies for a given problem, based on various characterizations (features) of the problem. Learning methods such as k nearest neighbor, support vector machines, and Gaussian processes have been recently applied to this task [34, 85].

7.3 Upcoming and Future Hammers

Providing hammers for proof assistants based on higher-order logic or on first-order logic and set theory seems relatively straightforward today. The premise selection and translation methods can often be reused, while the remaining ITP-dependent tasks are proof reconstruction, export of proof dependencies, and suitable integration within the proof assistants or their interfaces. This is all well underway for the HOL4 sytem, where an initial evaluation [62] on 9434 theorems gives 49% success in re-proving. Initial experiments have been started also with ACL2 [73] and are expected to start soon with systems based on set theory that have large mathematical libraries such as MetaMath and Isabelle/ZF. With suitable concept-merging methods [61], various interfaces and "meta-hammers" could eventually emerge, providing aggregated search over many libraries, and perhaps also aggregated automatic prov-

ing over them. There is a chance that the QED dream will in the end be achieved in such a bottom-up utility-motivated way, by eventually providing bridges between all systems and libraries that have enough interesting content. To some extent, this vision was already mentioned by William McCune in one of his QED-list postings:[13]

> I hope that there will be many implementations associated with QED, with different types of knowledge bases, different user interfaces, and written in different programming languages. Of course there will have to be (at some level) a common language for formulas, proofs, etc., but that seems like a small issue once the base logic is fixed.

For systems based on type theory, such as Coq, the first issue is defining what kind of advice a hammer can give. So far only a machine learning evaluation has been performed for Coq [75], showing quite similar premise selection performance over the CoRN [46] library as over other systems. It is an interesting question how much the large mathematical developments done with Coq such as the proof of the odd order theorem [64] need to avoid classical logic and rely on advanced features of the Coq logic such as the Curry–Howard isomorphism. Since the original graduate textbooks are not formulated within such framework, it is quite possible that many of the textbook lemmas have a proof that could be found by a hammer.

8. RELATED WORK AND HISTORY

The histories of ATP and ITP are so close that they sometimes lead to terminological confusions. Systems such as NQTHM and ACL2 have been sometimes called automatic theorem provers (and NQTHM indeed started as fully automatic), while relatively large developments have been done interactively with Otter, e.g., by Art Quaife [114] already before the QED manifesto, continued in various ways by Michael Beeson and Lawrence Wos [13, 14] and by Johan Belinfante [15].

Otter and ACL2 have also been linked together in the early Ivy bridge [93], and perhaps the earliest such ATP/ITP bridge was the 1991 integration of the (custom) Faust prover with HOL [87]. The list of such early bridges using both custom ATPs and off-the-shelf external ATPs is quite long, including MESON in HOL Light [67], DISCOUNT, SETHEO, and SPASS in ILF [48], Otter, PROTEIN, SETHEO, SPASS, and ${}_3T^AP$ in KIV [2], CL$^{\text{A}}$M in HOL [125], leanTAP in Isabelle [103], Gandalf and Metis in HOL [70, 71], Bliksem, EQP, LEO, Otter, PROTEIN, SPASS, TPS, and Waldmeister in ΩMEGA [96, 123], and Bliksem in Coq [17].

Most of such early ATP/ITP combinations however used ATPs on user-restricted search space. Exceptions are perhaps the early work of Art Quaife and Johan Belinfante. In 2002, Andrei Voronkov and Alexander Riazanov customized Vampire to answer queries over the whole SUMO ontology [107], and Josef Urban wrote the MoMM (modified E) authoring tool [133] using all MML lemmas for dependently-typed subsumption of Mizar goals. In parallel to the hammers described in this paper, Andrei Paskevich has integrated state-of-the-art ATPs [146] with the SAD/ForTheL system [88], and similar work has been done for the Naproche system [44] including the development of premise selection methods [45]. These two bridges

[13] `http://theory.stanford.edu/~uribe/mail/qed.messages/2.html`

exploit the ATPs as *oracles*, where the ATP proof is trusted and not rechecked. Similar recent hammer-like systems developed for formal (software) verification include Why3 [57], and integration of Zenon and SMTs with TLA+ [39, 99].

Several such attempts have been made to combine proof assistants with SMT solvers, either as oracles or with proof reconstruction. ACL2 and PVS employ UCLID and Yices as oracles [89, 118]. HOL Light integrates CVC Lite and reconstructs its proofs [94]. Isabelle/HOL enjoys two oracle integrations [12, 56] and two tactics with proof reconstruction [28, 30, 58]. HOL4 includes an SMT tactic [28, 30, 148], and an integration of veriT in Coq, based on SMT proof validation, is in progress [6].

A very important set of bridges has been provided by the TPTP framework [128] developed mainly by Geoff Sutcliffe or linking ATP (and experimentally also ITP) systems and tools, and for evaluation of the ATPs. Since 2008, Sutcliffe has organized the Large Theory Batch category of the CASC competition, benchmarking ATPs on the large problems extracted from Mizar, Isabelle, HOL Light, and also from SUMO and Cyc.

9. CONCLUSION AND RELATION TO QED

We hope to have persuaded the readers that the hammer technology works and is getting widespread. The evaluations show that hammers can today prove about 40% of the top-level Mizar and Flyspeck lemmas, and about 70% of the goals in the Isabelle *Judgment Day* benchmark. Some of these libraries contain over 100 000 top-level facts. Methods that take into account the millions of smaller lemmas in the libraries are on the horizon, and it is clear that many other aspects of the technology can be taken further.

Throughout the paper, we have given some answers to the three hammer-related QED question posed in the introduction:

Q1 How much are automatic theorem proving and artificial intelligence necessary or useful for constructing QED-like projects?

Q2 How much do the QED-like projects in turn allow the construction and study of interesting ATP and AI systems?

Q3 How do the two previous questions relate to the perhaps most discussed QED topic: the choice of a unified logical framework?

For Q1, we believe that strong AR and AI methods will turn out to be indispensable for reaching the QED goals. The answer to Q2 is today perhaps obvious: QED-like corpora do provide very useful material for developing interesting AR and AI methods. And a partial answer to Q3 is that having good translations to an efficiently automated logic is of great practical value, and so might be the methods for translating between the libraries.

10. ACKNOWLEDGMENTS

We would like to acknowledge the following colleagues, with whom we have collaborated on hammer-related research, including the authors of the underlying provers that have made this work possible in the first place: Mark Adams, Jesse

Alama, Grzegorz Bancerek, Christoph Benzmüller, Nikolaj Bjørner, Sascha Böhme, Chad Brown, Czesław Byliński, Ingo Dahn, David Desharbe, Mathias Fleury, Pascal Fontaine, Thibault Gauthier, Herman Geuvers, Tom Hales, John Harrison, Tom Heskes, Thomas Hillenbrand, Kryštof Hoder, Joe Hurd, Sebastiaan Joosten, Daniel Kühlwein, Twan van Laarhoven, Lionel Mamane, Bill McCune, Jia Meng, Leonardo de Moura, Tobias Nipkow, Jens Otten, Andrei Paskevich, Karol Pąk, Andrei Popescu, Petr Pudlák, Dan Roth, Piotr Rudnicki, Stephan Schulz, Nicholas Smallbone, Steffen Smolka, Albert Steckermeier, Nik Sultana, Kong Woei Susanto, Geoff Sutcliffe, Petr Štěpánek, Evgeni Tsivtsivadze, Andrei Voronkov, Jiří Vyskočil, Daniel Wand, Tjark Weber, Christoph Weidenbach, Makarius Wenzel, and Christoph Wernhard. We are also grateful to the anonymous reviewers for their many suggestions and comments.

Blanchette's Sledgehammer research was supported by the Deutsche Forschungsgemeinschaft projects Quis Custodiet (grants NI 491/11-1 and NI 491/11-2) and Hardening the Hammer (grant NI 491/14-1). Kaliszyk is supported by the Austrian Science Fund (FWF) grant P26201. Sledgehammer was originally supported by the UK's Engineering and Physical Sciences Research Council (grant GR/S57198/01). Urban's work was supported by the Marie-Curie Outgoing International Fellowship project AUTOKNOMATH (grant MOIF-CT-2005-21875) and by the Netherlands Organisation for Scientific Research (NWO) project Knowledge-based Automated Reasoning (grant 612.001.208).

References

[1] The QED Manifesto. In Bundy [36], pages 238–251.

[2] Wolfgang Ahrendt, Bernhard Beckert, Reiner Hähnle, Wolfram Menzel, Wolfgang Reif, Gerhard Schellhorn, and Peter H. Schmitt. Integrating automated and interactive theorem proving. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction—A Basis for Applications*, volume II: Systems and Implementation Techniques, pages 97–116. Kluwer, 1998.

[3] Jesse Alama, Tom Heskes, Daniel Kühlwein, Evgeni Tsivtsivadze, and Josef Urban. Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reason.*, 52(2):191–213, 2014.

[4] Jesse Alama, Daniel Kühlwein, and Josef Urban. Automated and human proofs in general mathematics: An initial comparison. In Bjørner and Voronkov [19], pages 37–45.

[5] Jesse Alama, Lionel Mamane, and Josef Urban. Dependencies in formal mathematics: Applications and extraction for Coq and Mizar. In Johan Jeuring, John A. Campbell, Jacques Carette, Gabriel Dos Reis, Petr Sojka, Makarius Wenzel, and Volker Sorge, editors, *Intelligent Computer Mathematics (AISC 2012), Calculemus 2012, DML 2012, MKM 2012, Systems and Projects*, volume 7362 of *LNCS*, pages 1–16. Springer, 2012.

[6] Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. A modular integration of SAT/SMT solvers to Coq through proof witnesses. In Jouannaud and Shao [74], pages 135–150.

[7] Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors. *International Joint Conference on Automated Reasoning (IJCAR 2008)*, volume 5195 of *LNCS*. Springer, 2008.

[8] Andrea Asperti, Grzegorz Bancerek, and Andrzej Trybulec, editors. *Mathematical Knowledge Management (MKM 2004)*, volume 3119 of *LNCS*. Springer, 2004.

[9] Serge Autexier, Christoph Benzmüller, Armin Fiedler, Helmut Horacek, and Quoc Bao Vo. Assertion-level proof representation with under-specification. *Electr. Notes Theor. Comput. Sci.*, 93:5–23, 2004.

[10] Serge Autexier, Jacques Calmet, David Delahaye, Patrick D. F. Ion, Laurence Rideau, Renaud Rioboo, and Alan P. Sexton, editors. *Intelligent Computer Mathematics (AISC 2010), Calculemus 2010, MKM 2010*, volume 6167 of *LNCS*. Springer, 2010.

[11] Grzegorz Bancerek. The mutilated chessboard problem—checked by Mizar. In Boyer and Trybulec [33], pages 43–46.

[12] Damián Barsotti, Leonor Prensa Nieto, and Alwen Tiu. Verification of clock synchronization algorithms: Experiments on a combination of deductive tools. *Formal Asp. Comput.*, 19(3):321–341, 2007.

[13] Michael Beeson. Mathematical induction in Otter-Lambda. *J. Autom. Reason.*, 36(4):311–344, 2006.

[14] Michael Beeson and Larry Wos. OTTER proofs in Tarskian geometry. In Demri et al. [53], pages 495–510.

[15] Johan G. F. Belinfante. On computer-assisted proofs in ordinal number theory. *J. Autom. Reason.*, 22(2):341–378, 1999.

[16] Christoph Benzmüller, Lawrence C. Paulson, Frank Theiss, and Arnaud Fietzke. LEO-II—A cooperative automatic theorem prover for classical higher-order logic (system description). In Armando et al. [7], pages 162–170.

[17] Marc Bezem, Dimitri Hendriks, and Hans de Nivelle. Automatic proof construction in type theory using resolution. *J. Autom. Reason.*, 29(3-4):253–275, 2002.

[18] Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors. *Conference on Automated Deduction (CADE-23)*, volume 6803 of *LNCS*. Springer, 2011.

[19] Nikolaj Bjørner and Andrei Voronkov, editors. *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-18)*, volume 7180 of *LNCS*. Springer, 2012.

[20] Jasmin Christian Blanchette. Redirecting proofs by contradiction. In Blanchette and Urban [25], pages 11–26.

[21] Jasmin Christian Blanchette, Sascha Böhme, Mathias Fleury, Steffen Juilf Smolka, and Albert Steckermeier. Semi-intelligible Isar proofs from machine-generated proofs. To appear in *J. Autom. Reason.*

[22] Jasmin Christian Blanchette, Sascha Böhme, Andrei Popescu, and Nicholas Smallbone. Encoding monomorphic and polymorphic types. In Nir Piterman and Scott A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2013)*, volume 7795 of *LNCS*, pages 493–507. Springer, 2013.

[23] Jasmin Christian Blanchette and Alexander Krauss. Monotonicity inference for higher-order formulas. *J. Autom. Reason.*, 47(4):369–398, 2011.

[24] Jasmin Christian Blanchette, Andrei Popescu, Daniel Wand, and Christoph Weidenbach. More SPASS with Isabelle—Superposition with hard sorts and configurable simplification. In Lennart Beringer and Amy P. Felty, editors, *Interactive Theorem Proving (ITP 2012)*, volume 7406 of *LNCS*, pages 345–360. Springer, 2012.

[25] Jasmin Christian Blanchette and Josef Urban, editors. *Proof Exchange for Theorem Proving (PxTP 2013)*, volume 14 of *EPiC Series*. EasyChair, 2013.

[26] Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors. *Interactive Theorem Proving (ITP 2013)*, volume 7998 of *LNCS*. Springer, 2013.

[27] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003.

[28] Sascha Böhme, Anthony C. J. Fox, Thomas Sewell, and Tjark Weber. Reconstruction of Z3's bit-vector proofs in HOL4 and Isabelle/HOL. In Jouannaud and Shao [74], pages 183–198.

[29] Sascha Böhme and Tobias Nipkow. Sledgehammer: Judgement Day. In Giesl and Hähnle [63], pages 107–121.

[30] Sascha Böhme and Tjark Weber. Fast LCF-style proof reconstruction for Z3. In Matt Kaufmann and Lawrence C. Paulson, editors, *Interactive Theorem Proving (ITP 2010)*, volume 6172 of *LNCS*, pages 179–194. Springer, 2010.

[31] Maria Paola Bonacina, editor. *Conference on Automated Deduction (CADE-24)*, volume 7898 of *LNCS*. Springer, 2013.

[32] Samuel Boutin. Using reflection to build efficient and certified decision procedures. In Martín Abadi and Takayasu Ito, editors, *Theoretical Aspects of Computer Software (TACS '97)*, volume 1281 of *LNCS*, pages 515–529. Springer, 1997.

[33] Robert Boyer and Andrzej Trybulec, editors. *QED Workshop II*. `http://mizar.org/qed/`, 1995.

[34] James P. Bridge, Sean B. Holden, and Lawrence C. Paulson. Machine learning for first-order theorem proving—Learning to select a good heuristic. *J. Autom. Reason.*, 53(2):141–172, 2014.

[35] Chad E. Brown. Satallax: An automatic higher-order prover. In Gramlich et al. [65], pages 111–117.

[36] Alan Bundy, editor. *Conference on Automated Deduction (CADE-12)*, volume 814 of *LNCS*. Springer, 1994.

[37] Paul A. Cairns. Informalising formal mathematics: Searching the Mizar library with latent semantics. In Asperti et al. [8], pages 58–72.

[38] Andy Carlson, Chad Cumby, Jeff Rosen, and Dan Roth. The SNoW learning architecture. Technical Report UIUCDCS-R-99-2101, UIUC C.S. Dept., 1999.

[39] Kaustuv Chaudhuri, Damien Doligez, Leslie Lamport, and Stephan Merz. A TLA$^+$ proof system. In Piotr Rudnicki, Geoff Sutcliffe, Boris Konev, Renate A. Schmidt, and Stephan Schulz, editors, *LPAR 2008 Workshops*, volume 418 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[40] Wei Chu and Seung-Taek Park. Personalized recommendation on dynamic content using predictive bilinear models. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors, *World Wide Web (WWW 2009)*, pages 691–700. ACM, 2009.

[41] Koen Claessen, Ann Lillieström, and Nicholas Smallbone. Sort it out with monotonicity: Translating between many-sorted and unsorted first-order logic. In Bjørner and Sofronie-Stokkermans [18], pages 207–221.

[42] Koen Claessen and Niklas Sörensson. New techniques that improve MACE-style finite model finding. In Peter Baumgartner and Chris Fermueller, editors, *Model Computation—Principles, Algorithms, Applications*, 2003.

[43] Fabrice Colas and Pavel Brazdil. Comparison of svm and some older classification algorithms in text classification tasks. In Max Bramer, editor, *Artificial Intelligence in Theory and Practice*, volume 217 of *IFIP International Federation for Information Processing*, pages 169–178. Springer, 2006.

[44] Marcos Cramer, Bernhard Fisseni, Peter Koepke, Daniel Kühlwein, Bernhard Schröder, and Jip Veldman. The Naproche project: Controlled natural language proof checking of mathematical texts. In Norbert E. Fuchs, editor, *Controlled Natural Language (CNL 2009)*, volume 5972 of *LNCS*, pages 170–186. Springer, 2009.

[45] Marcos Cramer, Peter Koepke, Daniel Kühlwein, and Bernhard Schröder. Premise selection in the Naproche system. In Giesl and Hähnle [63], pages 434–440.

[46] Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. C-CoRN, the Constructive Coq Repository at Nijmegen. In Asperti et al. [8], pages 88–103.

[47] Bernd I. Dahn. Robbins algebras are Boolean: A revision of McCune's computer-generated solution of Robbins problem. *J. Algebra*, 208(2):526–532, 1998.

[48] Bernd I. Dahn, Jürgen Gehne, Th. Honigmann, and Andreas Wolf. Integration of automated and interactive theorem proving in ILP. In William McCune, editor, *Conference on Automated Deduction (CADE-14)*, volume 1249 of *LNCS*, pages 57–60. Springer, 1997.

[49] Ingo Dahn. Interpretation of a Mizar-like logic in first-order logic. In Ricardo Caferra and Gernot Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *LNCS*, pages 137–151. Springer, 2000.

[50] Sanjoy Dasgupta. Experiments with random projection. In Craig Boutilier and Moisés Goldszmidt, editors, *Uncertainty in Artificial Intelligence (UAI '00)*, pages 143–151. Morgan Kaufmann, 2000.

[51] Martin Davis. Obvious logical inferences. In Patrick J. Hayes, editor, *International Joint Conference on Artificial Intelligence (IJCAI '81)*, pages 530–531. William Kaufmann, 1981.

[52] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *J. Assoc. Inf. Sci. Technol.*, 41(6):391–407, 1990.

[53] Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors. *International Joint Conference on Automated Reasoning (IJCAR 2014)*, volume 8562 of *LNCS*. Springer, 2014.

[54] Jörg Denzinger, Matthias Fuchs, Christoph Goller, and Stephan Schulz. Learning from previous proof experience. Technical Report AR99-4, C.S. Dept., Technische Universität München, 1999.

[55] Sahibsingh A. Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Trans. Syst. Man Cybern.*, SMC-6(4):325–327, 1976.

[56] Levent Erkök and John Matthews. Using Yices as an automated solver in Isabelle/HOL. In John Rushby and Natarajan Shankar, editors, *Automated Formal Methods (AFM '08)*, pages 3–13, 2008.

[57] Jean-Christophe Filliâtre and Andrei Paskevich. Why3—Where programs meet provers. In Matthias Felleisen and Philippa Gardner, editors, *European Symposium on Programming (ESOP 2013)*, volume 7792 of *LNCS*, pages 125–128. Springer, 2013.

[58] Pascal Fontaine, Jean-Yves Marion, Stephan Merz, Leonor Prensa Nieto, and Alwen Tiu. Expressiveness + automation + soundness: Towards combining SMT solvers and interactive proof assistants. In Holger Hermanns and Jens Palsberg, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2006)*, volume 3920 of *LNCS*, pages 167–181. Springer, 2006.

[59] Simon Foster and Georg Struth. Integrating an automated theorem prover into agda. In Mihaela Bobaru, Klaus Havelund, GerardJ. Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods (NFM 2011)*, volume 6617 of *LNCS*, pages 116–130. Springer, 2011.

[60] Simon Foster and Georg Struth. Regular algebras. *Archive of Formal Proofs*, 2014. `http://afp.sf.net/entries/Regular_Algebras.shtml`.

[61] Thibault Gauthier and Cezary Kaliszyk. Matching concepts across HOL libraries. In Stephen M. Watt, James H. Davenport, Alan P. Sexton, Petr Sojka, and Josef Urban, editors, *Conference on Intelligent Computer Mathematics (CICM 2014)*, volume 8543 of *LNCS*, pages 267–281. Springer, 2014.

[62] Thibault Gauthier and Cezary Kaliszyk. Premise selection and external provers for HOL4. In Xavier Leroy and Alwen Tiu, editors, *Certified Programs and Proofs (CPP 2015)*, pages 49–57. ACM, 2015.

[63] Jürgen Giesl and Reiner Hähnle, editors. *International Joint Conference on Automated Reasoning (IJCAR 2010)*, volume 6173 of *LNCS*. Springer, 2010.

[64] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A machine-checked proof of the Odd Order Theorem. In Blazy et al. [26], pages 163–179.

[65] Bernhard Gramlich, Dale Miller, and Uli Sattler, editors. *International Joint Conference on Automated Reasoning (IJCAR 2012)*, volume 7364 of *LNCS*. Springer, 2012.

[66] Thomas C. Hales. Developments in formal proofs. *CoRR*, abs/1408.6474, 2014. Séminaire Bourbaki, 66ème année, 2013–2014, n° 1086.

[67] John Harrison. Optimizing proof search in model elimination. In Michael A. McRobbie and John K. Slaney, editors, *Conference on Automated Deduction (CADE-13)*, volume 1104 of *LNCS*, pages 313–327. Springer, 1996.

[68] Kryštof Hoder and Andrei Voronkov. Sine qua non for large theory reasoning. In Bjørner and Sofronie-Stokkermans [18], pages 299–314.

[69] Xiaorong Huang. Translating machine-generated resolution proofs into ND-proofs at the assertion level. In Norman Y. Foo and Randy Goebel, editors, *Pacific Rim International Conference on Artificial Intelligence (PRICAI '96)*, volume 1114 of *LNCS*, pages 399–410. Springer, 1996.

[70] Joe Hurd. Integrating Gandalf and HOL. In Yves Bertot, Gilles Dowek, André Hirschowitz, C. Paulin, and Laurent Théry, editors, *Theorem Proving in Higher Order Logics (TPHOLs '99)*, volume 1690 of *LNCS*, pages 311–321. Springer, 1999.

[71] Joe Hurd. First-order proof tactics in higher-order logic theorem provers. In Myla Archer, Ben Di Vito, and César Muñoz, editors, *Design and Application of Strategies/Tactics in Higher Order Logics (STRATA 2003)*, number NASA/CP-2003-212448 in NASA Technical Reports, pages 56–68, 2003.

[72] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *J. Doc.*, 28:11–21, 1972.

[73] Sebastiaan Joosten, Cezary Kaliszyk, and Josef Urban. Initial experiments with TPTP-style automated theorem provers on ACL2 problems. In Freek Verbeek and Julien Schmaltz, editors, *ACL2 Theorem Prover and its Applications (ACL2 2014)*, volume 152 of *Electronic Proceedings in Theoretical Computer Science*, pages 77–85, 2014.

[74] Jean-Pierre Jouannaud and Zhong Shao, editors. *Certified Programs and Proofs (CPP 2011)*, volume 7086 of *LNCS*. Springer, 2011.

[75] Cezary Kaliszyk, Lionel Mamane, and Josef Urban. Machine learning of Coq proof guidance: First experiments. In Temur Kutsia and Andrei Voronkov, editors, *Symbolic Computation in Software Science (SCSS 2014)*, volume 30 of *EPiC Series*, pages 27–34. EasyChair, 2014.

[76] Cezary Kaliszyk, Stephan Schulz, Josef Urban, and Jiří Vyskočil. System description: E.T. 0.1. In Amy Felty and Aart Middeldorp, editors, *Conference on Automated Deduction (CADE-25)*, volume 9195 of *LNCS*. Springer, 2015.

[77] Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *CoRR*, abs/1310.2805, 2013. To appear in *J. Autom. Reason.*

[78] Cezary Kaliszyk and Josef Urban. PRocH: Proof reconstruction for HOL Light. In Bonacina [31], pages 267–274.

[79] Cezary Kaliszyk and Josef Urban. Stronger automation for Flyspeck by feature weighting and strategy evolution. In Blanchette and Urban [25], pages 87–95.

[80] Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *J. Autom. Reason.*, 53(2):173–213, 2014.

[81] Cezary Kaliszyk and Josef Urban. HOL(y)Hammer: Online ATP service for HOL Light. *Math. Comput. Sci.*, 9(1):5–22, 2015.

[82] Cezary Kaliszyk and Josef Urban. Learning-assisted theorem proving with millions of lemmas. *J. Symb. Comput.*, 69:109–128, 2015.

[83] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Machine Learner for Automated Reasoning 0.4 and 0.5. *CoRR*, abs/1402.2359, 2014. To appear in *Practical Aspects of Automated Reasoning (PAAR-2014)*.

[84] Daniel Kühlwein, Jasmin Christian Blanchette, Cezary Kaliszyk, and Josef Urban. MaSh: Machine learning for Sledgehammer. In Blazy et al. [26], pages 35–50.

[85] Daniel Kühlwein, Stephan Schulz, and Josef Urban. E-MaLeS 1.1. In Bonacina [31], pages 407–413.

[86] Daniel Kühlwein, Twan van Laarhoven, Evgeni Tsivtsivadze, Josef Urban, and Tom Heskes. Overview and evaluation of premise selection techniques for large theory mathematics. In Gramlich et al. [65], pages 378–392.

[87] Ramaya Kumar, Thomas Kropf, and Klaus Schneider. Integrating a first-order automatic prover in the HOL environment. In Myla Archer, Jeffrey J. Joyce, Karl N. Levitt, and Phillip J. Windley, editors, *HOL Theorem Proving System and Its Applications (HOL 1991)*, pages 170–176. IEEE, 1991.

[88] Alexander V. Lyaletski and Konstantin Verchinine. Evidence algorithm and system for automated deduction: A retrospective view. In Autexier et al. [10], pages 411–426.

[89] Panagiotis Manolios and Sudarshan K. Srinivasan. Verification of executable pipelined machines with bit-level interfaces. In *International Conference on Computer-Aided Design (ICCAD 2005)*, pages 855–862. IEEE, 2005.

[90] John McCarthy. The mutilated checkerboard in set theory. In Boyer and Trybulec [33], pages 25–26.

[91] William McCune. Solution of the Robbins problem. *J. Autom. Reason.*, 19(3):263–276, 1997.

[92] William McCune. Mace4 Reference Manual and Guide. Technical Report ANL/MCS-TM-264, Argonne National Laboratory, 2003.

[93] William McCune and Olga Shumsky Matlin. Ivy: A preprocessor and proof checker for first-order logic. In Matt Kaufmann, Panagiotis Manolios, and J Strother Moore, editors, *Computer-Aided Reasoning: ACL2 Case Studies*, number 4 in Advances in Formal Methods, pages 265–282. Kluwer, 2000.

[94] Sean McLaughlin, Clark Barrett, and Yeting Ge. Cooperating theorem provers: A case study combining HOL-Light and CVC Lite. *Electr. Notes Theor. Comput. Sci.*, 144(2):43–51, 2006.

[95] Norman Megill. *Metamath: A Computer Language for Pure Mathematics*. Lulu Press, 1997.

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

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

[98] Jia Meng and Lawrence C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. *J. Appl. Log.*, 7(1):41–57, 2009.

[99] Stephan Merz and Hernán Vanzetto. Automatic verification of TLA$^+$ proof obligations with SMT solvers. In Bjørner and Voronkov [19], pages 289–303.

[100] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[101] Dale Miller and Amy P. Felty. An integration of resolution and natural deduction theorem proving. In Tom Kehler, editor, *National Conference on Artificial Intelligence (AAAI-86)*, volume I. Science, pages 198–202. Morgan Kaufmann, 1986.

[102] Jens Otten and Wolfgang Bibel. leanCoP: Lean connection-based theorem proving. *J. Symb. Comput.*, 36(1-2):139–161, 2003.

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

[104] Lawrence C. Paulson. A simple formalization and proof for the mutilated chess board. *Log. J. IGPL*, 9(3):499–509, 2001.

[105] Lawrence C. Paulson and Jasmin Christian Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska, editors, *International Workshop on the Implementation of Logics (IWIL 2010)*, volume 2 of *EPiC Series*, pages 1–11. EasyChair, 2012.

[106] Lawrence C. Paulson and Kong Woei Susanto. Source-level proof reconstruction for interactive theorem proving. In Klaus Schneider and Jens Brandt, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2007)*, volume 4732 of *LNCS*, pages 232–245. Springer, 2007.

[107] Adam Pease and Geoff Sutcliffe. First order reasoning on a large ontology. In Sutcliffe et al. [129], pages 61–70.

[108] Frank Pfenning. Analytic and non-analytic proofs. In Robert E. Shostak, editor, *Conference on Automated Deduction (CADE-7)*, volume 170 of *LNCS*, pages 394–413. Springer, 1984.

[109] J. D. Phillips and David Stanovský. Automated theorem proving in quasigroup and loop theory. *AI Commun.*, 23(2-3):267–283, 2010.

[110] Karol Pąk. The methods of improving and reorganizing natural deduction proofs. In *Mathematical User Interfaces (MathUI10)*, 2010.

[111] Karol Pąk. Improving legibility of natural deduction proofs is not trivial. *Log. Meth. Comput. Sci.*, 10(3), 2014.

[112] Robi Polikar. Ensemble based systems in decision making. *IEEE Circ. Syst. Mag.*, 6(3):21–45, 2006.

[113] Petr Pudlák. Semantic selection of premisses for automated theorem proving. In Sutcliffe et al. [129], pages 27–44.

[114] Art Quaife. *Automated Development of Fundamental Mathematical Theories*. Kluwer, 1992.

[115] Radim Řehůřek and Petr Sojka. Software framework for topic modelling with large corpora. In René Witte, Hamish Cunningham, Jon Patrick, Elena Beisswanger, Ekaterina Buyko, Udo Hahn, Karin Verspoor, and Anni R. Coden, editors, *New Challenges for NLP Frameworks 2010*, pages 45–50. ELRA, 2010.

[116] Alex Roederer, Yury Puzis, and Geoff Sutcliffe. Divvy: An ATP meta-system based on axiom relevance ordering. In Renate A. Schmidt, editor, *Conference on Automated Deduction (CADE-22)*, volume 5663 of *LNCS*, pages 157–162. Springer, 2009.

[117] Piotr Rudnicki. Obvious inferences. *J. Autom. Reason.*, 3(4):383–393, 1987.

[118] John M. Rushby. Tutorial: Automated formal methods with PVS, SAL, and Yices. In Dang Van Hung and Paritosh Pandya, editors, *Software Engineering and Formal Methods (SEFM 2006)*, page 262. IEEE, 2006.

[119] Stephan Schulz. *Learning Search Control Knowledge for Equational Deduction*, volume 230 of *DISKI*. Infix, 2000.

[120] Stephan Schulz. E—A brainiac theorem prover. *AI Commun.*, 15(2-3):111–126, 2002.

[121] Stephan Schulz. First-order deduction for large knowledge bases. In *Deduction at Scale 2011 Seminar, Ringberg Castle*, 2011.

[122] D. Sculley. Rank aggregation for similar items. In *SIAM International Conference on Data Mining (SDM07)*, pages 587–592. SIAM, 2007.

[123] Jörg Siekmann, Christoph Benzmüller, Armin Fiedler, Andreas Meier, Immanuel Normann, and Martin Pollet. Proof development with ΩMEGA: The irrationality of $\sqrt{2}$. In Fairouz Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, volume 28 of *Applied Logic*, pages 271–314. Springer, 2003.

[124] John K. Slaney, Ewing L. Lusk, and William McCune. SCOTT: Semantically constrained Otter—System description. In Bundy [36], pages 764–768.

[125] Konrad Slind, Michael J. C. Gordon, Richard J. Boulton, and Alan Bundy. System description: An interface between CLAM and HOL. In Claude Kirchner and Hélène Kirchner, editors, *Conference on Automated Deduction (CADE-15)*, volume 1421 of *LNCS*, pages 134–138. Springer, 1998.

[126] Steffen Juilf Smolka and Jasmin Christian Blanchette. Robust, semi-intelligible Isabelle proofs from ATP proofs. In Blanchette and Urban [25], pages 117–132.

[127] Geoff Sutcliffe. Semantic derivation verification: Techniques and implementation. *Int. J. Artif. Intell. Tools*, 15(6):1053–1070, 2006.

[128] Geoff Sutcliffe. The TPTP World—Infrastructure for automated reasoning. In Edmund M. Clarke and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-16)*, volume 6355 of *LNCS*, pages 1–12. Springer, 2010.

[129] Geoff Sutcliffe, Josef Urban, and Stephan Schulz, editors. *Empirically Successful Automated Reasoning in Large Theories (ESARLT 2007)*, volume 257 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[130] Tanel Tammet. Gandalf. *J. Autom. Reason.*, 18:199–204, 1997.

[131] Josef Urban. MPTP—Motivation, implementation, first experiments. *J. Autom. Reason.*, 33(3-4):319–339, 2004.

[132] Josef Urban. MizarMode—An integrated proof assistance tool for the Mizar way of formalizing mathematics. *J. Appl. Log.*, 4(4):414–427, 2006.

[133] Josef Urban. MoMM—Fast interreduction and retrieval in large libraries of formalized mathematics. *Int. J. Artif. Intell. Tools*, 15(1):109–130, 2006.

[134] Josef Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reason.*, 37(1-2):21–43, 2006.

[135] Josef Urban. XML-izing Mizar: Making semantic processing and presentation of MML easy. In Michael Kohlhase, editor, *Mathematical Knowledge Management (MKM 2005)*, volume 3863 of *LNCS*, pages 346–360. Springer, 2006.

[136] Josef Urban. MaLARea: A metasystem for automated reasoning in large theories. In Sutcliffe et al. [129], pages 45–58.

[137] Josef Urban. Content-based encoding of mathematical and code libraries. In Christoph Lange and Josef Urban, editors, *Proceedings of the ITP 2011 Workshop on Mathematical Wikis (MathWikis)*, number 767 in CEUR Workshop Proceedings, pages 49–53, Aachen, 2011. CEUR-WS.org.

[138] Josef Urban. BliStr: The Blind Strategymaker. *CoRR*, abs/1301.2683, 2014. To appear in *Practical Aspects of Automated Reasoning (PAAR-2014)*.

[139] Josef Urban, Kryštof Hoder, and Andrei Voronkov. Evaluation of automated theorem proving on the Mizar Mathematical Library. In Komei Fukuda, Joris van der Hoeven, Michael Joswig, and Nobuki Takayama, editors, *International Congress on Mathematical Software (ICMS 2010)*, volume 6327 of *LNCS*, pages 155–166. Springer, 2010.

[140] Josef Urban, Piotr Rudnicki, and Geoff Sutcliffe. ATP and presentation service for Mizar formalizations. *J. Autom. Reason.*, 50:229–241, 2013.

[141] Josef Urban and Geoff Sutcliffe. ATP-based cross-verification of Mizar proofs: Method, systems, and first experiments. *Math. Comput. Sci.*, 2(2):231–251, 2008.

[142] Josef Urban and Geoff Sutcliffe. Automated reasoning and presentation support for formalizing mathematics in Mizar. In Autexier et al. [10], pages 132–146.

[143] Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jiří Vyskočil. MaLARea SG1—Machine learner for automated reasoning with semantic guidance. In Armando et al. [7], pages 441–456.

[144] Josef Urban, Geoff Sutcliffe, Steven Trac, and Yury Puzis. Combining Mizar and TPTP semantic presentation and verification tools. *Studies in Logic, Grammar and Rhetoric*, 18(31):121–136, 2009.

[145] Josef Urban, Jiří Vyskočil, and Petr Štěpánek. MaLeCoP: Machine learning connection prover. In Kai Brünnler and George Metcalfe, editors, *Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2011)*, volume 6793 of *LNCS*, pages 263–277. Springer, 2011.

[146] Konstantin Verchinine, Alexander V. Lyaletski, and Andrei Paskevich. System for automated deduction (SAD): A tool for proof verification. In Frank Pfenning, editor, *Conference on Automated Deduction (CADE-21)*, volume 4603 of *LNCS*, pages 398–403. Springer, 2007.

[147] Robert Veroff. Using hints to increase the effectiveness of an automated reasoning program: Case studies. *J. Autom. Reason.*, 16(3):223–239, 1996.

[148] Tjark Weber. SMT solvers: New oracles for the HOL theorem prover. *J. Softw. Tools Technol. Transfer*, 13(5):419–429, 2011.

[149] Christoph Weidenbach. Combining superposition, sorts and splitting. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, pages 1965–2013. Elsevier and MIT Press, 2001.

[150] Freek Wiedijk, editor. *The Seventeen Provers of the World*, volume 3600 of *LNCS*. Springer, 2006.

[151] Jürgen Zimmer, Andreas Meier, Geoff Sutcliffe, and Yuan Zhan. Integrated proof transformation services. In Christoph Benzmüller and Wolfgang Windsteiger, editors, *Computer-Supported Mathematical Theory Development*, number 04-14 in RISC Report Series, pages 33–48. RISC Institute, University of Linz, 2004.