J Autom Reasoning (2016) 57:219–244
DOI 10.1007/s10817-016-9362-8

A Learning-Based Fact Selector for Isabelle/HOL

Jasmin Christian Blanchette[1,2] **· David Greenaway**[3] **· Cezary Kaliszyk**[4] **·**
Daniel Kühlwein[5] **· Josef Urban**[6]

Received: 31 March 2015 / Accepted: 15 January 2016 / Published online: 3 February 2016

Abstract Sledgehammer integrates automatic theorem provers in the proof assistant Isabelle/HOL. A key component, the fact selector, heuristically ranks the thousands of facts (lemmas, definitions, or axioms) available and selects a subset, based on syntactic similarity to the current proof goal. We introduce MaSh, an alternative that learns from successful proofs. New challenges arose from our "zero click" vision: MaSh integrates seamlessly with the users' workflow, so that they benefit from machine learning without having to install software, set up servers, or guide the learning. MaSh outperforms the old fact selector on large formalizations.

Keywords Relevance filtering · Machine learning · Proof assistants · Automatic theorem provers

1 Introduction

Sledgehammer [1] is a subsystem of the proof assistant Isabelle/HOL [2] that discharges proof goals by harnessing off-the-shelf automatic theorem provers. It heuristically selects

In memoriam Piotr Rudnicki 1951–2012.

✉ Jasmin Christian Blanchette
jasmin.blanchette@inria.fr

✉ Cezary Kaliszyk
cezary.kaliszyk@uibk.ac.at

1 Inria Nancy – Grand-Est & LORIA, Villers-lès-Nancy, France

2 Max-Planck-Institut für Informatik, Saarbrücken, Germany

3 NICTA, University of New South Wales, Sydney, Australia

4 University of Innsbruck, Innsbruck, Austria

5 Radboud University, Nijmegen, The Netherlands

6 Czech Technical University in Prague, Prague, Czech Republic

a number of *facts*—lemmas, definitions, or axioms—from the thousands available in background libraries and the user's formalization, translates the problem to the provers' logics, runs the provers, and reconstructs any machine-found proof in Isabelle (Sect. 2). The tool is popular with both novice and expert users of the proof assistant.

Various aspects of Sledgehammer have been improved since its introduction, notably the use of sound translation schemes [3], the addition of SMT solvers [4], and advances in the provers themselves [5–7, etc.].

One key component that had received little attention until recently is Sledgehammer's fact selector. Meng and Paulson [8] designed a selector, called *Me*ng–*P*auls*o*n (MePo), that iteratively ranks and selects facts similar to the current proof (sub)goal, based on the symbols they contain. Despite its simplicity, this selector greatly increases the success rate of Sledgehammer: Most provers cannot cope with tens of thousands of formulas, and translating so many formulas would also slow down Sledgehammer. Moreover, the translation of Isabelle's higher-order constructs and types is optimized globally for a problem—smaller problems make more optimizations possible, which helps the automatic provers.

Coinciding with the development of Sledgehammer and MePo, a line of research has focused on applying machine learning to large-theory reasoning. Much of this work has been done on the vast Mizar Mathematical Library (MML) [9], either in its original Mizar [10] formulation or in first-order form as the Mizar Problems for Theorem Proving (MPTP) [11]. The MaLARea system [12,13], the CASC LTB [14] competition, and the Mizar@Turing [15] competition have been significant milestones. Comparative studies involving MPTP [16–18] and the Flyspeck project [19,20] in HOL Light [21] have found that fact selectors based on machine learning outperform and complement symbol-based approaches [22].

In the past decade, a number of learning-based selectors have been implemented and have made an impact on the automated reasoning community. In this article, we describe a fact selector that aims to bring the fruits of this research to Isabelle users. This tool, *Ma*chine Learning for *S*ledge*h*ammer (MaSh), offers an alternative to MePo by learning from successful proofs, whether human-written or machine-generated.

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

- *Zero configuration*: The tool should require no installation or configuration steps, even for use with development versions of Isabelle.
- *Zero click*: Existing users of Sledgehammer should benefit from machine learning, both for standard theories and for their custom developments, without having to change their workflow.
- *Zero maintenance*: The tool should not add to the maintenance burden of Isabelle. In particular, it should automatically cope with theory changes, without requiring users to restart servers or update databases.
- *Zero overhead*: Machine learning should incur no overhead to those Isabelle users who do not employ Sledgehammer.

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

MaSh's heart is a pair of machine learning algorithms: sparse naive Bayes and k nearest neighbors (Sect. 3). These algorithms make suggestions based on known proofs. The program maintains a persistent state and supports incremental, nonmonotonic changes. Although MaSh is part of Isabelle, the same approach could be used by other proof assistants, automatic theorem provers, or applications with similar requirements.

The machine learning algorithms form the basis of the MaSh fact selector (Sect. 4). When Sledgehammer is invoked, it exports new facts and their proofs to the machine learner and queries it to obtain facts that are likely to be useful. The main technical difficulty is to perform the learning in a fast and robust way without interfering with other activities of the proof assistant. Power users can enhance the learning by letting automatic provers run for hours on libraries, searching for simpler proofs than those already available.

A strong selector, MeSh, is obtained by combining *Me*Po and Ma*Sh*. We compare the three selectors on large formalizations covering many application areas of Isabelle, including mathematics, programming languages, and term rewriting (Sect. 5). A combination of automatic provers and fact selectors achieves a success rate of over 66 % at reproving the lemmas contained in these formalizations. These empirical results are complemented by Judgment Day, a benchmark suite that has tracked Sledgehammer's development since 2010 (Sect. 6). In addition, we started applying our methods to the huge seL4 microkernel formalization developed by Klein's group at NICTA (Sect. 7). Performance varies depending on the application area and on how much has been learned, but even with little learning MeSh emerges as a leader.

An earlier version of this work was presented at the ITP 2013 conference [23]. Since then, the naive Bayes algorithm has been ported from Python to Standard ML, to improve efficiency and reliability. Like most of Isabelle, Sledgehammer itself is implemented in Standard ML. The k nearest neighbor algorithm has been added as an alternative to naive Bayes. Both algorithms are now combined with inverse document frequency (IDF), a technique that increases precision. The evaluation sections have been updated and extended, notably with the addition of a large formalization of term rewriting to the benchmark suite. The microkernel case study is new. Finally, the description of related work has been updated to keep up with recent research.

2 Sledgehammer and MePo

Whenever Sledgehammer is invoked on a proof goal, MePo selects n facts $\phi_1, \ldots, \phi_n$ from the thousands available, ordering them by decreasing estimated relevance. The selector keeps track of a set of "known" symbols, initially consisting of all the goal's symbols. It performs the following steps iteratively, until n facts have been selected:

1. Compute each fact's score, as roughly given by $k/(k+u)$, where k is the number of known symbols and u the number of unknown symbols occurring in the fact.
2. Select all facts with perfect scores as well as some of the remaining high-scoring facts, and add all their symbols to the set of known symbols.

The implementation refines this approach in several ways. Chained facts (inserted into the proof goal by means of the Isabelle keyword `using`, `from`, `then`, `hence`, or `thus` [24]) take absolute priority; inside a structured Isabelle proof, local facts are preferred to global (top-level) ones; first-order facts are preferred to higher-order ones; rare symbols are weighted more heavily than common ones; and so on.

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

The supported automatic theorem provers include the first-order provers E [25], SPASS [5], and Vampire [26] and the satisfiability modulo theories (SMT) solvers CVC4 [27], veriT [28], and Z3 [29]. The provers are given the first m facts of the selected n facts $\phi_1, \ldots, \phi_n$, for some $m \leq n$. The order of the facts—the estimated relevance—is exploited by some provers to guide the search. Sledgehammer's default time limit is 30 s, but the automatic provers are invoked repeatedly for shorter time periods, with different options and different number of facts m; for example, SPASS is given as few as 50 facts in some time slices and as many as 1000 in others. Excluding some facts restricts the search space, helping the prover find longer proofs within the allotted time, but it also makes fewer proofs possible.

Once a proof is found, Sledgehammer extracts the facts referenced in it and recursively attempts to re-find a simpler proof using a strict subset of these facts, yielding a minimized proof. Then it reconstructs the minimized proof in Isabelle by a suitable proof text—typically a single call to the built-in resolution prover Metis [30], but sometimes a detailed, structured proof [31,32].

Example 1 Given the proof goal

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

MePo selects 1000 facts. The SPASS prover, among others, quickly finds a minimal proof involving the 5th and 17th facts:

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

Example 2 MePo's tendency to miss some useful facts is illustrated by the following proof goal, taken from Paulson's verification of cryptographic protocols [33]:

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

A straightforward proof relies on these four lemmas:

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

MePo ranks the first lemma 3742nd due to the many symbols that do not appear in the goal ($\bigcup$, parts, and initState); with such a high rank, the fact is not passed to any automatic prover, and Sledgehammer fails to find a proof. In contrast, all four lemmas appear among MaSh's first 35 facts and MeSh's first 77 facts, making Sledgehammer succeed.

3 The Machine Learning Engine

The core of MaSh is a pair of algorithms for fact selection with machine learning.[1] The first algorithm is an approximation of naive Bayes adapted to fact selection; the second one is a version of k nearest neighbors. Both are implemented in Standard ML. External learning algorithms, such as those provided by HOLyHammer [22], can be interfaced as well. By default, MaSh simply combines the results of naive Bayes and k nearest neighbors.

[1] The source code is distributed as part of Isabelle in `src/HOL/Tools/Sledgehammer/sledgehammer_mash.ML`.

3.1 Basic Concepts

Theorem proving concepts such as facts and proofs are treated by the learning algorithms in an application-agnostic way:

- A *fact* ϕ is represented as a string. There are at most finitely many facts available.
- A *feature* f is also represented as a string. A positive *weight* w is attached to each feature.
- *Visibility* is a partial order $\prec$ on the available facts. A fact ϕ is visible *from* a fact χ if $\phi \prec \chi$. The set $\mathit{par}(\phi)$ of *parents* of a fact ϕ consists of the immediate predecessors of ϕ with respect to $\prec$, i.e., $\{\chi \mid \chi \prec \phi \wedge \nexists\psi.\ \chi \prec \psi \prec \phi\}$.
- A *proof* $\Pi(\phi)$ for ϕ is a set of facts visible from ϕ that can be used to "prove" it (in some application-specific sense). These facts are also called ϕ's *dependencies*.

Visibility captures the notion that facts appearing later in a proof development cannot be used to prove earlier facts. In Isabelle, facts are organized in theories, whose dependencies form a directed acyclic graph. The visibility relation is derived from the theory graph combined with the linear order of facts inside each theory.

Each proof goal or fact ϕ is described abstractly by a finite set of features $F(\phi)$. The features should be mathematically meaningful; for example, they may be the symbols occurring in a logical formula. Machine learning proceeds from the hypothesis that facts with similar features are likely to have similar proofs and uses this to estimate relevance.

3.2 Updates and Selection

MaSh maintains a persistent state on disk, consisting of all the learned information as a list of tuples of the form $(\phi,\ \mathit{par}(\phi),\ F(\phi),\ \Pi(\phi))$. The parents $\mathit{par}(\phi)$ specify how to extend the visibility relation for fact ϕ. The state is duplicated in memory while Isabelle is running. Information about additional facts can be incorporated at any time, by extending the data structure with new tuples.

The algorithms for fact selection take the set of features of the current proof goal and the set of visible facts as arguments and return a predetermined number of suggested facts, ordered by decreasing estimated relevance. The implementation of each algorithm is divided in two parts: a part that is independent of the goal, and whose results can be cached, and a part that must be performed each time MaSh is invoked on a new goal.

3.3 Sparse Naive Bayes

Let γ be a new goal and ϕ a visible fact. The sparse naive Bayes algorithm computes the relevance of ϕ for proving γ as the probability

$$P\big(\phi \text{ is used in } \gamma\text{'s proof}\big)$$

This probability is estimated by characterizing γ with its features $F(\gamma)$ and rewriting the above formula as

$$P\big(\phi \text{ is used in a proof of } \psi \mid \psi \text{ has features } F(\gamma)\big)$$

To compute this, we could use all available features, but for efficiency reasons we restrict the computation to a smaller set of features. More precisely, let the *extended features* $\overline{F}(\phi)$ of a fact ϕ be the features of ϕ and of the facts that were proved using ϕ:[2]

[2] In general, we could extend the features recursively by following the dependency graph, but here we perform only one iteration.

$$\overline{F}(\phi) = F(\phi) \cup \bigcup_{\chi \text{ such that } \phi \in \Pi(\chi)} F(\chi)$$

After limiting the set of all available features to $F(\gamma) \cup \overline{F}(\phi)$, the estimated probability becomes

$$P\big(\phi \text{ is used in a proof of } \psi \mid \text{features in } F(\gamma) \text{ appear in } \psi \text{ and features } \overline{F}(\phi) - F(\gamma) \text{ do not}\big)$$

The features belonging to neither $F(\gamma)$ nor $\overline{F}(\phi)$ are ignored for the estimation. The learning algorithm assumes that the features are independent and applies Bayes's rule to transform the conditional probability (up to a constant factor) to the following product of probabilities:

$$\begin{aligned}&P(\phi \text{ is used in } \psi\text{'s proof})\\ &\cdot \prod_{f \in F(\gamma) \cap \overline{F}(\phi)} P\big(\psi \text{ has feature } f \mid \phi \text{ is used in } \psi\text{'s proof}\big)\\ &\cdot \prod_{f \in F(\gamma) - \overline{F}(\phi)} P\big(\psi \text{ has feature } f \mid \phi \text{ is not used in } \psi\text{'s proof}\big)\\ &\cdot \prod_{f \in \overline{F}(\phi) - F(\gamma)} P\big(\psi \text{ does not have feature } f \mid \phi \text{ is used in } \psi\text{'s proof}\big)\end{aligned}$$

The four probability expressions can be estimated from the known dependencies. To avoid recomputing the same results over and over, the algorithm relies on two tables that are updated whenever new facts are learned:

- $s(\phi, f)$ stores the number of times a fact ϕ occurs as a dependency of a fact described by feature f;
- $t(\phi)$ stores the number of times a fact ϕ occurs as a dependency.

Let K be the total number of known proofs. We have

$$\begin{aligned}P(\phi \text{ is used in a proof of (any) } \psi) &= \frac{t(\phi)}{K}\\ P\big(\psi \text{ has feature } f \mid \phi \text{ is used in } \psi\text{'s proof}\big) &= \frac{s(\phi, f)}{t(\phi)}\\ P\big(\psi \text{ does not have feature } f \mid \phi \text{ is used in } \psi\text{'s proof}\big) &= 1 - \frac{s(\phi, f)}{t(\phi)} \approx 1 - \frac{s(\phi, f) - 1}{t(\phi)}\end{aligned}$$

To avoid a probability of 0, which would make the whole formula collapse, we subtract 1 from $s(\phi, f)$. The latter expression is greater than 0, since the feature is considered only if it is associated with a fact. Finally, $P\big(\psi \text{ has feature } f \mid \phi \text{ is not used in } \psi\text{'s proof}\big)$ is the a priori probability of ϕ being used in a proof. It corresponds to an unlikely event and is estimated by a low, fixed probability (e^{σ_4} in the expression below).

To avoid multiplying small numbers, which may lead to numerical instability due to the limitations of floating-point arithmetic, the final formula takes the logarithm of probabilities. Moreover, the divisor K is shared by all facts, so it can be omitted. Given that the weight for feature f is $w(f)$, this leads to the following expression to estimate the relevance of the fact ϕ for goal γ on a logarithmic scale:

$$\begin{aligned}&\sigma_1 \ln t(\phi) + \sum_{f \in F(\gamma) \cap \overline{F}(\phi)} w(f) \ln \frac{\sigma_2\, s(\phi, f)}{t(\phi)}\\ &+ \sigma_3 \sum_{f \in \overline{F}(\phi) - F(\gamma)} w(f) \ln \left(1 - \frac{s(\phi, f) - 1}{t(\phi)}\right)\\ &+ \sigma_4 \sum_{f \in F(\gamma) - \overline{F}(\phi)} w(f)\end{aligned}$$

The fudge factors σ_1 to σ_4 determine the relative weightings of the formula's four terms. MaSh uses the values $\sigma_1 = 30, \sigma_2 = 5, \sigma_3 = 0.2$, and $\sigma_4 = -18$, which were experimentally determined to produce good results.

3.4 *k* Nearest Neighbors

The k nearest neighbors algorithm implemented in MaSh finds a fixed number k of visible facts considered the most similar to the goal and uses their dependencies to estimate the relevance of all facts. The *nearness* of two facts ϕ, χ is given by

$$n(\phi, \chi) = \sum_{f \in F(\phi) \cap F(\chi)} w(f)^{\tau_1}$$

(Higher values indicate nearer facts.) To find the neighbors of the goal, we first iterate over the goal's features, and for each feature we gather the facts f where $s(\phi, f) > 0$. With appropriate data structures, we can efficiently ignore all facts that have no features in common with the goal.

Let N be the set consisting of the k nearest neighbors (the ones with highest nearness) of the goal. The estimated relevance of each visible fact ϕ for the goal γ is given by

$$\left(\tau_2 \sum_{\chi \in N \mid \phi \in \Pi(\chi)} \frac{n(\chi, \gamma)}{|\Pi(\chi)|} \right) + \begin{cases} n(\phi, \gamma) & \text{if } \phi \in N \\ 0 & \text{otherwise} \end{cases}$$

The above is a slight extension of the standard formula. In the context of fact selection, there are two kinds of information: The dependencies of a fact ϕ are useful for proving ϕ, and ϕ is useful for proving itself. When combining this with the k nearest neighbors algorithm, each neighbor of the goal positively affects all the dependencies of the neighbor (the left summand above), and it positively affects the neighbor itself (the right summand). When combining the two summands, we must take into account that typically a neighbor has many dependencies, hence the $|\Pi(\chi)|$ divisor. MaSh uses $\tau_1 = 6$ and $\tau_2 = 2.7$ as fudge factors.

The relevance estimates can be computed efficiently for all the facts at once, by iterating over the neighbors and updating the relevance for the facts corresponding to the dependencies and the neighbors themselves. If the number of neighbors is small and the neighbors have few or similar dependencies, the estimated relevance might be 0 for most facts. To prevent this, the algorithm starts with $k = 0$ neighbors, and if too few facts have nonzero relevance, it gradually increases the number k until enough facts emerge.

3.5 Feature Weights

The sparse naive Bayes and k nearest neighbors algorithms are parameterized by a weight function $w(f)$. Weights make it possible to give a higher priority to some features at the expense of others. In practice, it makes sense to assign higher weights to rare features, because a match on a rare feature is more significant than a match on an ubiquitous feature [34]. For example, suppose the user has just introduced a new function f and proved a few lemmas about it. If the next goal involves both f and the empty list nil, it makes sense to give priority to the handful of lemmas about f, which are likely to be very relevant, than to the hundreds of lemmas that refer to nil. Even the memoryless selector MePo used this observation to obtain better results.

The usual way to weight counted features in semantic text retrieval with respect to their frequency is the *inverse document frequency* (IDF) [34,35]. The weight of a feature f

in a set of facts Φ is defined as the logarithm of the inverse of the feature's frequency in Φ:

$$w(f, \Phi) = \ln \frac{|\Phi|}{\big|\{\phi \in \Phi \mid f \in \overline{F}(\phi)\}\big|}$$

This scheme is implemented in MaSh. A feature has weight $\ln |\Phi|$ if it arises in a single proof, $\ln 2$ if it arises in half of the proofs, and $\ln 1 = 0$ if it arises in all the proofs.

4 Integration in Sledgehammer

The abstract machinery described in Sect. 3 is used by Sledgehammer's MaSh fact selector to provide suggestions whenever the user invokes Sledgehammer on a proof goal.

4.1 Learning from and for Isabelle

Facts, features, proofs, and visibility were introduced in Sect. 3.1 as empty shells. The integration with Isabelle fills these concepts with content.

Facts Communication with the learning engine requires a unique name for identifying Isabelle facts. Each global fact in Isabelle carries a stable "name hint" that is identical or very similar to its fully qualified user-visible name (e.g., *List.list.map_2* for *List.list.map*(2)). MaSh uses these hints as names. Local facts in a structured Isabelle proof are disambiguated by appending the fact's statement to its name hint.

Features Machine learning operates not on the formulas directly but on sets of features. The simplest scheme is to encode each symbol occurring in a formula as its own feature. The experience with MePo is that other factors help—for example, the formula's types and type classes or the theory it belongs to. Earlier evaluations on MML and Flyspeck revealed that it is also helpful to preserve parts of the formula's structure, such as subterms [13,22,36].

The scheme adopted for MaSh is inspired by these precursors. For each term in the formula (excluding the outer quantifiers, connectives, and equality), the nontrivial first-order patterns up to a given depth are generated as features. Given a maximum depth of 2, the term $\mathsf{g}\ (\mathsf{h}\ x\ \mathsf{a})$, where x is a variable of type τ, yields the patterns

x	a	$\mathsf{g}\ \_$	$\mathsf{g}\ (\mathsf{h}\ \_\ \_)$
$\mathsf{h}\ \_\ \_$	$\mathsf{h}\ x\ \_$	$\mathsf{h}\ \_\ \mathsf{a}$	$\mathsf{h}\ x\ \mathsf{a}$

These are simplified and encoded into the following features:

τ	a	g	$\mathsf{g}(\mathsf{h})$
h	$\mathsf{h}(\tau)$	$\mathsf{h}(\mathsf{a})$	$\mathsf{h}(\tau, \mathsf{a})$

Variables are replaced by their types, since variable names have no fixed meaning.

The types occurring in a formula (excluding those of propositions and functions) are also considered features and encoded using an analogous scheme to terms. Type variables constrained by type classes give rise to features corresponding to the specified type classes and their superclasses. Finally, two pieces of meta information are encoded as features: the theory to which the fact belongs and whether the fact is local.

Example 3 The lemma

$$\mathsf{transpose}\ (\mathsf{map}\ (\mathsf{map}\ f)\ xss) = \mathsf{map}\ (\mathsf{map}\ f)\ (\mathsf{transpose}\ xss)$$

from the theory of lists has the following features:

map	map(list_list)	fun
map(fun)	map(map, list_list)	list
map(map)	transpose	list_list
map(transpose)	transpose(map)	List
map(map, transpose)	transpose(list_list)	

The last three features correspond to the type α *list*, the type α *list list*, and the theory *List*.

Another heuristic used by MaSh is to consider the features of the chained facts. In Isabelle, the chained facts are effectively premises of the proof goal. When computing the goal's features, MaSh includes the features of the chained facts, but with half the weight they would normally have, since chained facts are not quite as important as the goal itself.

Humans tend to group related lemmas together. MaSh exploits this by considering the features of a few (up to 10) preceding facts, with an even lower weight (0.1) than for chained facts. This is especially beneficial if the goal has very few features, in which case feature-based comparison will be imprecise.

Proofs MaSh predicts which facts are useful for proving the goal at hand by studying successful proofs. There is an obvious source of successful proofs: All the facts in the loaded theories are accompanied by proof terms that store the dependencies [37]. However, not all available facts are equally suitable for learning. Many of them are derived automatically by definitional commands [e.g., for (co)inductive predicates, (co)datatypes, and (co)recursive functions] and proved using custom tactics, and there is not much to learn from those highly technical lemmas. The most interesting lemmas are those stated and proved by humans. Slightly abusing terminology, we call these *Isar proofs*, after Isabelle's proof language.

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

Human-written Isar proofs are abundant, but they are not necessarily the best raw material to learn from. They tend to involve more, different facts than Sledgehammer proofs. Sometimes they rely on induction, which is beyond the scope of first-order provers; but even excluding induction, there is evidence that the provers work better if the proofs used for learning were produced by similar provers [18,38]. A special mode of Sledgehammer runs an automatic prover on all available facts to learn from machine-generated proofs. Users can let it run for hours at a time on their favorite theories. The Isar proof facts are passed to the provers together with a few dozens of MePo-selected facts, to enable finding of alternative proofs. Whenever a prover succeeds, MaSh discards the Isar proof and learns from the new minimized proof. Facts with large Isar proofs are processed first since they are more likely to have significantly shorter alternative proofs.

Visibility The loaded background theories and the user's formalization, including local lemmas, appear to Sledgehammer as a vast collection of facts. Each fact is tagged with its own

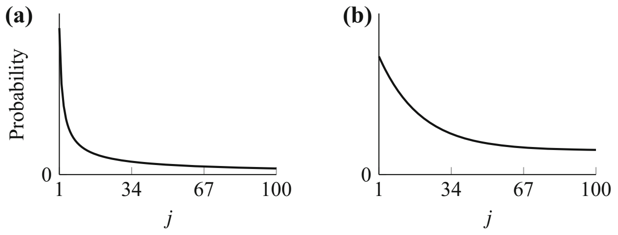

Fig. 1 Estimated probability of the jth fact's appearance in a proof. **a** MaSh. **b** Proximity

abstract theory value, of type theory in Standard ML, that captures the state of affairs when it was introduced. Sledgehammer constructs the visibility graph by using the (very fast) theory extension order $\trianglelefteq$ on the theory type.

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

An alternative concept of visibility, as implemented in HOLYHammer, would be to linearize the partial order to obtain a total order [39]. The proof suggested by the tool can then refer to lemmas that are not currently available, requiring the user to add some directives to import background theories.

4.2 Fact Selectors: MaSh and MeSh

When the user invokes Sledgehammer on a goal, the MaSh-based fact selector computes the goal's features and the visible facts and produces a list with as many suggestions as desired, ordered by decreasing estimated relevance. This process usually takes a few hundred milliseconds on modern hardware, which is reasonable for a proof tool that may run for half a minute overall. In a separate thread, Sledgehammer looks for newly available facts, which may take several seconds depending on how many new facts there are—these will be considered next time Sledgehammer is run. This is consistent with our "zero overhead" design goal: Learning being triggered by Sledgehammer invocations, MaSh does not waste any CPU time or disk space for users who do not invoke the proof tool or who disabled MaSh, relying on MePo instead.

Relying purely on MaSh for fact selection raises an issue: MaSh may not be aware of all the available facts. In particular, it will be oblivious to the very latest facts, introduced after Sledgehammer was invoked for the last time, and these are likely to be crucial for the proof. If only a few facts are unknown, they can be processed quickly before the query is performed (Sect. 4.3). But even then, these new facts will typically appear in few proofs, regardless of how useful they may be.

As a general precaution, the raw MaSh data is enriched with a proximity selector, which sorts the available facts by decreasing proximity in the proof text. Instead of a plain linear combination of ranks, the enriched MaSh selector transforms ranks into probabilities and takes their weighted average, with weight 0.8 for MaSh and 0.2 for proximity. The probabilities are rough approximations based on experiments. Figure 1 shows the curves. For example, the first suggestion given by MaSh is considered about 15 times more likely to appear in a successful proof than the 50th. The curves were chosen based on statistics gathered on large benchmarks of Sledgehammer proofs. These steep curves ensure that if a fact is ranked very high by either MaSh or the proximity selector, it will be ranked very high in the result.

This notion of combining selectors to define new selectors is taken one step further by MeSh, a combination of MePo and MaSh inspired by experiments [18] combining machine learning with the MePo-like SInE selector [40]. Both selectors are weighted 0.5, and both use the probability curve of Fig. 1a. Ideally, the curves and parameters that control the combination of selectors would be learned mechanically rather than hard-coded.

4.3 Automatic and Manual Control

All MaSh-related activities take place as a result of a Sledgehammer invocation. When Sledgehammer is launched, it checks whether any new facts, unknown to the visibility graph, are available. If there are fewer than 100, it learns from them right away, meaning that it collects their features and dependencies, adds them to the persistent data, and runs the goal-independent part of the machine learning algorithms. Otherwise, it launches a new thread to perform the learning in the background. The first time, it may take about half a minute to learn all the facts in the background theories (assuming about 10,000 facts). Subsequent invocations are much faster.

If one of Sledgehammer's automatic provers succeeds, MaSh immediately learns from the proof. The discharged proof goal may have been only one among many subgoals in an unstructured proof, in which case it has no name. Sledgehammer invents a fresh name for it and stores it as an invisible fact. Although this anonymous goal cannot be used to discharge other goals, MaSh benefits from learning the connection between the formula's features and its proof.

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

4.4 Nonmonotonic Theory Changes

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

- *A fact is deleted.* The fact is kept in MaSh's data structures but is silently ignored by Sledgehammer whenever it is suggested by MaSh.
- *A fact is renamed.* Sledgehammer perceives this as the deletion of a fact and the addition of another fact.
- *A theory is renamed.* Since theory names are encoded in fact names, renaming a theory amounts to renaming all its facts.
- *Two facts are reordered.* The visibility graph loses synchronization with reality. Sledgehammer may end up ignoring a fact suggested by MaSh because the fact is visible according to the graph but invisible according to Isabelle.
- *A fact χ' is introduced between two facts ϕ and χ.* MaSh offers no facility to change the parent of χ, but this is not needed. It is enough to make the new fact χ' a child of ϕ to make it visible to future proof goals: when MaSh is invoked on a goal γ below χ in the theory

text, it will notice that both χ and χ' are maximal nodes in the visibility graph restricted to nodes visible from γ and use both as parents for γ, resulting in a diamond configuration.
- *The fact's formula is modified.* This occurs when users change the statement of a lemma, but also when they rename or relocate a symbol. MaSh does not keep track of such changes and may lose some of its predictive power.
- *The fact's proof is modified.* Again, MaSh does not keep track of such changes and may lose predictive power.

More elaborate schemes for tracking dependencies are possible. However, the benefits are unclear: Presumably, the learning performed on older theories is valuable and should be preserved, despite its deficiencies. This is analogous to teams of humans developing a large formalization: Teammates should not forget everything they know each time a colleague changes the name of some basic lemma. And should users notice a performance degradation after a major refactoring, they can always invoke `unlearn` to restart from scratch. In any event, we want to get more experience with MaSh before investing time and effort in more sophisticated schemes.

5 Evaluation on Large Formalizations

In this section and the next one, we attempt to answer the main questions that existing Sledgehammer users are likely to have: How do MaSh and MeSh compare with MePo? Does machine learning really help? The answer takes the form of two evaluations, performed using an unofficial version of Isabelle slightly older than the 2014 release. Our empirical data are publicly available.[3]

The first evaluation measures the selectors' ability to select meaningful facts from six user formalizations—three from the Isabelle distribution, two from the *Archive of Formal Proofs* [41], and one from a separate online archive:

Auth	Cryptographic protocols	Paulson [33]
IsaFoR	Term rewriting	Thiemann and Sternagel [42]
Jinja	Java-like language	Klein and Nipkow [43]
List	Finite lists	Nipkow [2]
Nominal2	Nominal binder syntax	Urban and Kaliszyk [44]
Probability	Measure and probability theory	Hölzl and Heller [45]

These formalizations are large enough to exercise learning and provide meaningful numbers, while not being so massive as to make experiments impractical. They are also representative of large classes of mathematical and computer science applications. The largest among them, IsaFoR, is a repository of results pertaining to term rewriting, including (non)termination, (non)confluence, completion, and complexity.

For each of the formalizations, the evaluation harness processes the lemmas sequentially according to a linearization of the partial order induced by the theory graph and their location in the theory texts. Each lemma is seen as a proof goal for which facts must be selected. Previously proved lemmas, and the learning performed on their proofs, may be exploited—this includes lemmas from imported background theories. This setup simulates a user who systematically develops a formalization from beginning to end, trying out Sledgehammer on each lemma before engaging in a manual proof.[4]

[3] http://www21.in.tum.de/~blanchet/mash2_data.tgz.

[4] Earlier evaluations of Sledgehammer, starting with Böhme and Nipkow's Judgment Day experiments [46], always operated on individual (sub)goals, guided by the notion that lemmas can be too difficult to be proved

Table 1 Statistics on the evaluated formalizations

Formalization	Number of goals	Avg. dependencies per goal	Avg. features per fact	Total number of facts ('000)	Total number of features ('000)
Auth	739	5.3	42	15	17
IsaFoR	571	7.4	30	65	139
Jinja	749	5.8	33	17	27
List	863	6.7	14	13	15
Nominal2	432	5.9	19	15	19
Probability	1542	7.2	31	24	32
Together	4896	5.6	31	80	166

Table 1 presents statistics on the formalizations. The second and third columns are about the goals corresponding to each formalization's user-entered lemmas; the fourth column is about all facts contained in the formalization, including those generated by definitional commands and other tools; and the last two columns are about the entire formalizations including the libraries on which they build.

The evaluation is twofold. The first part computes how accurately the selectors can re-find the facts referenced in the Isar proofs on which MaSh's learning is based (Sect. 5.1). The second part connects the selectors to automatic provers and measures actual success rates (Sect. 5.2). The first part may seem artificial: After all, real users are interested in any proof that discharges the goal at hand, not a specific known proof. The predictive approach's greatest virtue is that it does not require invoking automatic provers; evaluating the impact of parameters is a matter of seconds instead of hours. MePo itself has been fine-tuned using similar techniques. For MaSh, the approach also helps ascertain whether it is learning the learning materials well.

5.1 Machine Learning Metrics

Three standard metrics—full recall, area under the receiver operating characteristic curve (AUC), and coverage—will be useful, in a generalized form. For a given goal, a fact selector (MePo, MaSh, or MeSh) ranks the N available facts and selects the $n \leq N$ best ranked facts $\phi_1, \ldots, \phi_n$, in decreasing order of estimated relevance, with $\mathit{rank}(\phi_i) = i$ and $\mathit{rank}(\phi) = n + 1$ otherwise. The parameter n is fixed at 1024 in the experiments below. The standard metrics correspond to the $n = N$ case.

Let $\Phi = \{\phi_1, \ldots, \phi_n\}$. The known proof Π serves as a reference point against which the selected facts Φ and their ranks are judged. Ideally, the selected facts should include as many facts from the proof as possible, with as low (i.e., good) ranks as possible.

Definition 1 (*Full Recall*) The *full recall* is the smallest nonnegative number k such that $\Pi \subseteq \{\phi_1, \ldots, \phi_k\}$, or $n + 1$ if no such number exists.

Definition 2 (*AUC*) The *area under the receiver operating characteristic curve* (*AUC*) is defined as

Footnote 4 continued
outright by automatic provers. However, lemmas appear to provide the right level of challenge for modern automation, and they tend to exhibit less redundancy than a sequence of similar subgoals.

Table 2 Average full recall

Formalization	MePo	MaSh		MeSh	
		NB	*k*NN	NB	*k*NN
Auth	647	104	143	**96**	112
IsaFoR	1332	**513**	604	517	570
Jinja	839	244	306	**229**	256
List	1083	**234**	263	259	271
Nominal2	1045	**220**	276	229	264
Probability	1324	434	422	**393**	395

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

Definition 3 (*Coverage*) Given $k \leq n$, the *k-coverage* is defined as

$$\frac{|\{\phi_1, \ldots, \phi_k\} \cap \Pi|}{\min\{k, |\Pi|\}}$$

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

Tables 2, 3, and 4 show the average full recall, the average AUC, and the average rank of necessary dependencies over all goals from the six formalizations. Figure 2 plots the average k-coverage for IsaFoR. Naive Bayes (NB) and k nearest neighbors (kNN) are considered separately.

MaSh clearly outperforms MePo on this kind of benchmarks. Depending on the benchmark, MeSh is sometimes hampered by its MePo component and sometimes helped by it. The data also show important variations between formalizations: IsaFoR and Probability are particularly difficult, possibly because they are larger than the other ones, whereas Auth is comparatively easy.

However, one should be cautious when interpreting these data points. Isar proofs are not necessarily representative of machine-generated proofs. Because several proofs are possible, a failure to re-find the facts referenced in an existing Isar proof does not necessarily amount to a failure to find a proof at all. Since MaSh learns from Isar proofs, it is not surprising that it excels at selecting the facts that arise in that kind of proof. Ultimately, we must run actual provers if we want a fair evaluation of MePo against MaSh.

5.2 Success Rates of Automatic Theorem Provers

Next comes the "in vivo" part of the evaluation, with actual provers replacing machine learning metrics. The objective is to see how MePo, MaSh, and MeSh compare in a realistic setting. We use a collection of automatic provers for this. As we noted in a similar study [31],

> It is important to bear in mind that the evaluation is not a competition between the provers. Different provers are invoked with different problems and options, and

Table 3 Average AUC (%)

Formalization	MePo	MaSh		MeSh	
		NB	*k*NN	NB	*k*NN
Auth	86.8	98.1	97.3	**98.4**	98.0
IsaFoR	64.0	**93.7**	92.1	92.9	92.1
Jinja	77.6	96.9	95.7	**97.0**	96.5
List	71.6	**96.8**	96.5	96.5	96.4
Nominal2	72.1	**97.0**	96.2	96.9	96.4
Probability	65.4	94.1	**94.8**	**94.8**	**94.8**

Table 4 Average rank of necessary dependencies

Formalization	MePo	MaSh		MeSh	
		NB	*k*NN	NB	*k*NN
Auth	267	41	58	**35**	42
IsaFoR	735	**132**	164	149	165
Jinja	439	66	90	**63**	73
List	578	**67**	75	74	77
Nominal2	571	**63**	80	67	77
Probability	708	123	**109**	**109**	**109**

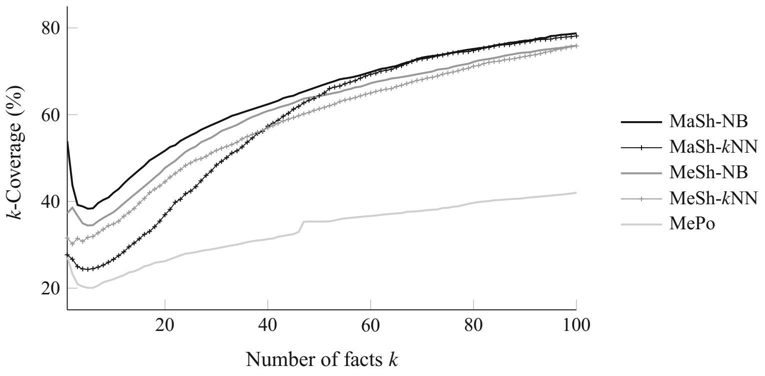

Fig. 2 Average coverage for IsaFoR

although we have tried to optimize the setup for each, we might have missed an important configuration option. Each number must be seen as a lower bound on the potential of the prover.

The experiments were conducted on a 64-bit Linux server equipped with 12-core AMD Opteron 6174 processors running at 2.2 GHz.

Before carrying out more experiments, we first evaluated 15 provers on 1200 randomly selected problems. The results are show in Table 5. Based on this, we selected four provers for the rest of the evaluation: CVC4 1.4, Epar 1.8b (a version of E with strategies computed by BliStr [47]), Vampire 2.6, and Z3 4.3.2.

Table 5 Initial evaluation of provers on 1200 problems from all formalizations and fact selectors

Prover	Proved	Disproved	Uniq.
CVC4 1.4	372	0	**5**
E 1.8	304	26	0
Epar 1.8a	365	**33**	0
Epar 1.8b	381	28	1
Epar 1.8c	381	**33**	3
Epar 1.8d	378	**33**	2
iProver 1	234	15	1
SPASS 3.5	274	0	0
Vampire 1.8	356	9	1
Vampire 2.6	**382**	12	2
Vampire 3.0	367	9	2
Z3 3.2	360	14	1
Z3 4.0	360	14	1
Z3 4.0q	310	14	0
Z3 4.3.2	363	14	2
Any prover	476	49	–

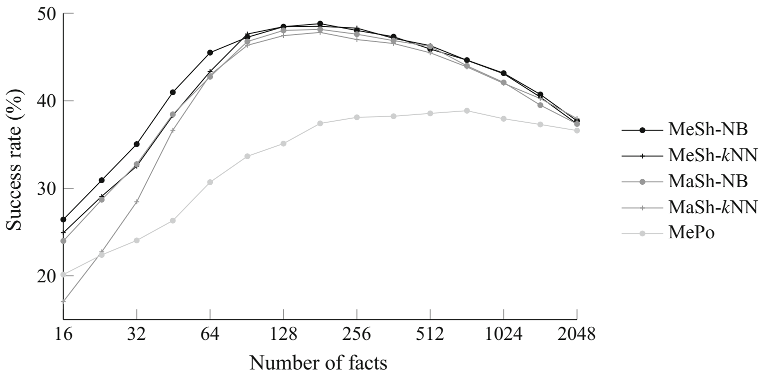

Fig. 3 Success rates per fact selector

For each goal from the formalizations, 15 problems were generated, with 16, 23 ($\approx 2^{4.5}$), 32, …, 1024, 1448 ($\approx 2^{10.5}$), and 2048 facts as axioms. Sledgehammer's translation is parameterized by many options, whose defaults vary from prover to prover and, because of time slicing, even from one prover invocation to another. As a reasonable uniform configuration for the experiments, types are encoded via the so-called polymorphic "featherweight" guard-based encoding (the most efficient complete scheme [3]), and λ-abstractions via λ-lifting (as opposed to the more complete but more explosive SK combinators).

Figure 3 plots the success rates of the four-prover combination on these problems for each fact selector. Two versions of MaSh and MeSh are compared. A problem is considered solved if it is solved within 10 s by any of them, using only one thread.

We observe the following:

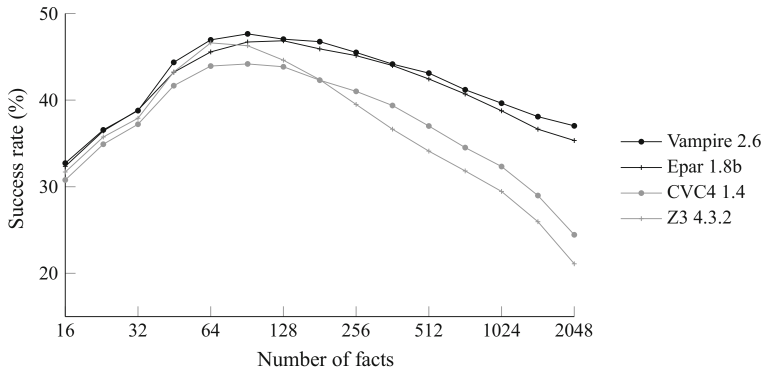

Fig. 4 Success rates per prover

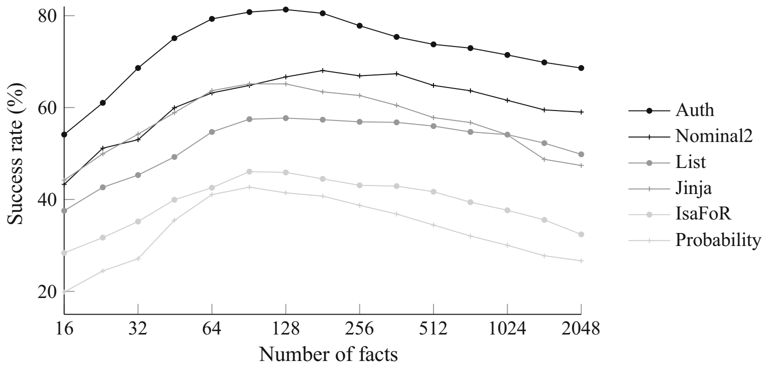

Fig. 5 Success rates per formalization

- MaSh clearly outperforms MePo, especially in the range from 32 to 512 facts. For 64-fact problems, the gap between MaSh and MePo is over 12 percentage points.
- MaSh's peak is both higher than MePo's (48.2 % for MaSh-NB vs. 38.9 % for MePo) and occurs for smaller problems (181 vs. 724 facts), reflecting the intuition that selecting fewer facts more carefully should increase the success rate.
- MeSh is slightly stronger than MaSh. The effect is especially marked for the problems with fewer facts.

Figure 4 presents the same results but focuses on the provers, taking the union of the fact selectors and formalizations. Similarly, Figure 5 focuses on the formalizations, taking the union of the provers and fact selectors. The curves have somewhat different shapes for the individual provers and formalizations, but the general picture remains the same.

Another measure of MaSh and MeSh's power is the total number of goals solved for any number of facts. Tables 6, 7, and 8 explore different combinations of fact selectors, provers, and formalizations. Given a prover and a fact selector, a goal is considered solved if any of the 15 problems generated for different number of facts is solved. With MePo alone, 48.5 % of the goals are solved; adding MaSh and MeSh increases this number to 66.1 %.

Finally, Table 9 presents combinations of provers, fact selectors, and number of facts that should be close to the optimal way to occupy 12 processor cores for 10 s, or 4 cores for 30 s. The combinations are listed as a greedy sequence: Each row is the optimal addition

Table 6 Success rates per fact selector and prover

Fact selector	CVC4 1.4	Epar 1.8b	Vampire 2.6	Z4 4.3.2	Any prover
MaSh-*k*NN	47.0	50.4	51.4	48.0	60.0
MaSh-NB	**47.9**	**51.0**	**52.0**	49.1	**60.3**
MeSh-*k*NN	46.7	48.9	50.8	50.2	59.6
MeSh-NB	46.8	49.0	51.0	**51.3**	60.2
MePo	38.2	40.8	41.3	40.5	48.5

Table 7 Success rates per fact selector and formalization

Fact selector	Auth	IsaFoR	Jinja	List	Nominal2	Probability	Any formalization
MaSh-*k*NN	**83.0**	49.6	66.8	62.9	**71.3**	44.9	60.0
MaSh-NB	82.0	49.7	**68.4**	**64.1**	70.1	44.9	**60.3**
MeSh-*k*NN	80.5	49.0	66.6	62.2	70.4	**45.7**	59.6
MeSh-NB	81.6	**50.8**	67.7	62.5	70.8	45.4	60.2
MePo	72.4	36.3	55.9	52.6	60.9	32.2	48.5
Any selector	84.3	56.4	73.6	68.0	76.4	53.5	66.1

Table 8 Success rates per prover and formalization

Prover	Auth	IsaFoR	Jinja	List	Nominal2	Probability	Any formalization
CVC4 1.4	74.0	42.4	58.5	58.2	66.7	**43.5**	54.9
Epar 1.8b	75.0	46.9	61.3	59.0	69.0	41.1	55.6
Vampire 2.6	79.3	45.9	61.7	59.1	**69.4**	43.2	56.9
Z4 4.3.2	**82.3**	**51.8**	**65.6**	**62.0**	65.5	39.0	**57.5**
Any prover	84.3	56.4	73.6	68.0	76.4	53.5	66.1

to the previous rows, and the last column is the cumulative success rate of all rows up to and including the current row. The 12 combination together have a success rate of 58.9 %, compared with 66.1 % for all 300 possible combinations.

6 Judgment Day

The Judgment Day benchmark suite [46] currently consist of 1230 proof goals arising in seven Isabelle theories, covering among them areas as diverse as the fundamental theorem of algebra, the completeness of a Hoare logic, and Jinja's type soundness. The evaluation harness invokes Sledgehammer on each goal. The hardware setup consists of Linux servers equipped with Intel Core2 Duo CPUs running at 2.40 GHz. The time limit is 30 s for proof search. In case of success, the search is followed by brute-force minimization and reconstruction in Isabelle. MaSh is trained on nearly 14,000 Isar proofs from the background libraries imported by the seven theories under evaluation.

Table 9 Greedy sequence of 12 combinations of fact selectors, provers, and number of facts

Prover	Number of facts	Success rate (%)
(a) MePo only		
Epar 1.8b	256	32.6
Z3 4.3.2	724	36.9
Vampire 2.6	2048	39.5
Z3 4.3.2	91	41.7
Vampire 2.6	64	42.9
CVC4 1.4	256	43.6
CVC4 1.4	64	44.2
Z3 4.3.2	2048	44.7
Epar 1.8b	724	45.1
Z3 4.3.2	181	45.5
Z3 4.3.2	16	45.8
Vampire 2.6	91	46.0

Fact selector	Prover	Number of facts	Success rate (%)
(b) All fact selectors			
MeSh-*k*NN	Vampire 2.6	128	40.1
MaSh-NB	Z4 4.3.2	91	47.4
MaSh-NB	Epar 1.8b	256	50.5
MeSh-NB	Z4 4.3.2	64	52.7
MaSh-*k*NN	CVC4 1.4	181	54.1
MeSh-*k*NN	Z4 4.3.2	256	55.4
MaSh-NB	Vampire 2.6	512	56.2
MaSh-NB	Epar 1.8b	45	57.0
MaSh-*k*NN	Z4 4.3.2	45	57.6
MeSh-*k*NN	CVC4 1.4	91	58.1
MaSh-NB	Epar 1.8b	23	58.5
MePo	Z4 4.3.2	64	58.9

The comparison comprises the main superposition-based provers and SMT solvers integrated with Sledgehammer: CVC4 1.5 prerelease (revision 7b72e4d), E 1.8, SPASS 3.8ds, Vampire 3.0, veriT smtcomp2014 postrelease (0a723b4), and Z3 4.3.2 prerelease (revision a10c318). Each prover is invoked with its own options and problems, including prover-specific features (e.g., arithmetic for CVC4, veriT, and Z3). Time slicing is enabled: The 30 s slot is split into several slices, each corresponding to somewhat different problems and prover options. For MeSh, some of the slices use MePo or MaSh directly to promote complementarity.

The results are summarized in Table 10. As expected, MeSh performs very well: Running all six provers in parallel for 30 s solves 14.7 % more goals with MeSh than with MePo (919 vs. 801), corresponding to 9.6 percentage points. In particular, CVC4 yields truly remarkable results.[5] The overall success rate, for all six provers and all three fact selectors in parallel, is 76.7 %.

[5] CVC4's developers have been reporting high success rates on Sledgehammer-generated benchmarks before [6], but this is the first time that we independently corroborate those results.

Table 10 Number of successful Sledgehammer invocations per prover on 1230 Judgment Day goals

Prover	MePo	MaSh	MeSh	Any selector
CVC4 1.5pre	679	749	**783**	830
E 1.8	622	601	**665**	726
SPASS 3.8ds	678	684	**739**	789
Vampire 3.0	703	698	**740**	789
veriT 2014post	543	556	**590**	655
Z3 4.3.2pre	638	668	**703**	788
Any prover	801	885	**919**	943

The other main observation is that MaSh somewhat underperforms, especially in the light of the evaluation of Sect. 5. The overall numbers look reasonable, but it is hard to explain why MaSh is beaten by MePo for E and Vampire and does not exactly shine for SPASS. One hypothesis is that MaSh might have a tendency to select harmful facts—superfluous facts that would trigger some explosive misbehavior in the superposition-based provers, hampering proof search. Moreover, the Sledgehammer setup has been tuned for Judgment Day and MePo over the years (in the hope that improvements on this representative benchmark suite would translate in improvements on users' theories), and conversely MePo's parameters are tuned for Judgment Day. Finally, MaSh's weakness might simply reflect the goal-based nature of the benchmarks: Individual goals in a detailed proof tend to rely more heavily on local facts and symbols, about which little has been learned. Nonetheless, quite some progress has been made since we first introduced MaSh at ITP 2013 [23, Section 5.2].

7 Case Study: Microkernel Verification

The verification of the seL4 operating system microkernel by Klein's group at NICTA [48], at several hundreds of thousand lines of Isabelle text, is surely the largest project ever undertaken in Isabelle. Following the first release of MaSh, the engineers in the project were interested in applying it to their ever growing formal development.

Dealing with actual users inevitably raises real issues:

- The seL4 formalization is a very large proof, which causes scalability issues.
- The formalization is typically at least one version of Isabelle behind, corresponding to eight months of development on the proof assistant.
- At the time (in 2013), the proof effort was a commercial venture and hence could not be freely shared with the MaSh developers.

We started a collaboration with NICTA to look into this. The second author, Greenaway, joined the four MaSh developers to help debug and tune it further. He had access to the proprietary seL4 work. In addition, his AutoCorres tool [49], which is used for verifying seL4, has been open source for some years. Hence, it was possible for his coauthors to do some experiments with these theories.

Initially, scalability was an even larger problem than we had expected. With the entire seL4 proof loaded in memory, amounting to almost 59,000 facts, it took about 50 s for a single invocation of MePo, 120 s for MaSh, and 130 s for MeSh on a standard workstation (Intel Core i5-3470 at 3.2 GHz), before the automatic provers could even be started. Although most users normally do not work that deep in the proof, these timings were completely unacceptable. Since Sledgehammer runs for 30 s by default (which roughly corresponds to most users' patience), at most a few seconds should be used for fact selection.

When profiling the tool, we discovered several inefficient algorithms. Fortunately, they could either be optimized or bypassed. The following list of improvements gives a flavor of the changes we did:

- The most expensive piece of code was the formula duplication check. Despite the use of appropriate functional data structures, it scaled poorly took about 30 s irrespective of which fact selector was used. Removing duplicates is desirable but not essential, so we no longer do it for huge background theories ($\geq$50,000 facts).
- Meng and Paulson [8] realized that complex formulas, with many nested function applications or many λ-abstractions, rarely arise in proofs. Omitting them increases the success rate slightly. This check is now skipped with little loss for very large background theories ($\geq$25,000 facts).
- MePo and even some provers [5] exploit metainformation about the formulas—for example, whether a formula is a simplification rule in Isabelle. Collecting this information is expensive and yields comparatively small benefits. This is now avoided if a certain threshold is reached ($\geq$10,000 simplification rules).
- The formulas' term structure was traversed several times to look for certain internal constructs, to blacklist obviously useless formulas. This code could easily be rewritten to require only one traversal.
- MaSh wasted much time extracting the dependencies from a few huge proof terms. The solution was to introduce a threshold on the size of proof terms considered by machine learning.
- MePo iterates several times over all visible facts. A score is associated with each fact and updated at each iteration. Ignoring facts with very low scores after a few (5) iterations speeds up the algorithm without changing its results much.

Thanks to these and other similar changes, MePo and MaSh came out much more usable with a fully loaded seL4—for example, MaSh took only 12.5 s afterward—and also faster for more pedestrian scenarios. The NICTA team had their own locally patched version of Isabelle to work around various issues. Thus, we could backport all changes to Sledgehammer and MaSh and distribute the changes rapidly via that channel, bypassing the proof assistant's eight-month-or-so release cycle.

For most of MaSh's existence, the machine learning engine was implemented in an external Python program. Whenever Sledgehammer needed to select fact, it launched the Python program, which first loaded all its persistent data. For huge background theories, a lot of time was wasted loading and storing data. We eventually implemented a local server mode to reduce the overhead, but this introduced many reliability issues that affected the NICTA users. Moreover, in a context where users keep switching between different Isabelle versions, race conditions and data corruptions were frequent occurrences. This was ultimately solved by porting the machine learning engine to Standard ML and integrating it directly in Sledgehammer.

Although their formalization was proprietary, the NICTA users were willing to enable Sledgehammer's "spy" mode. With this mode enabled, each invocation of Sledgehammer is logged along with information about the proofs found. The spy mode was activated between September 2013 and June 2014 on 23 users' machines. The exact dates vary from user to user. Some basic data could be gathered about Sledgehammer usage and about the old Python-based MaSh implementation.

Table 11 Sledgehammer usage at NICTA as revealed by the spy mode

User	Fact selector	Number of distinct goals	Number of successes	Success rate (%)	Avg. number of background facts	Avg. number of facts in proofs
Janise	MeSh	1002	354	35	18,207	19
Colby	MePo	584	192	33	19,932	4
Damien	MeSh	411	200	49	16,504	9
Kia	MePo	332	116	35	13,024	2
Shiela	MePo	207	53	26	10,152	14
Emil	MePo	169	53	31	19,037	6
Mozella	MeSh	112	63	56	18,033	16
Ammie	MeSh	97	9	9	18,427	5
Azalee	MeSh	46	12	26	23,510	41
Shanelle	MePo	40	11	28	18,992	6
Soo	MePo	35	23	66	12,043	5
Lonnie	MeSh	33	10	30	21,499	4
Travis	MePo	20	15	75	11,526	4
Lillie	MeSh	20	7	35	20,950	7
Dallas	MePo	16	6	38	14,731	48
Alyssa	MePo	13	3	23	44,480	3
Joi	MePo	10	4	40	12,758	4
Nelida	MeSh	8	2	25	56,114	10
Angel	MePo	8	7	88	7942	7
Yetta	MePo	3	1	33	10,54	3
Shona	MePo	2	0	0	9989	–
Myra	MePo	2	0	0	37,920	–
Kylee	MePo	1	1	100	10,662	1
All users	MePo	1455	486	33	16,514	6
	MeSh	1712	655	38	18,172	15
	Either	3166	1141	36	17,422	12

Table 11 summarizes the results; the users' names were changed to preserve anonymity. The overall success rate (after reconstruction) is 38 % for MaSh users, 33 % for MePo users, and 36 % collectively. This is encouraging but inconclusive because of the small number of users, and of possible biases introduced by the choice of selector by a user. It should be no surprise that these results are lower than for Judgment Day: seL4 problems are likely more difficult, and on an evolving theory useful lemmas tend to be missing—some goals might even be unprovable. On average, users invoked Sledgehammer twice for each goal they tried it on. The spy logs suggest that they often tried the tool on a goal, then added one lemma or changed their specification, then tried to discharge the goal again, possibly reiterating the last two steps.

If there is too little data to determine with certainty whether MaSh (or rather, the old version in Python with naive Bayes as its sole algorithm and with its bugs) helped, at least we see how often NICTA users invoke Sledgehammer and how often it succeeds. We also see that the number of background facts considered for fact selection is typically much lower than the worst case of about 59,000, when all of seL4 is loaded. The high number of facts in some proofs is due to needless dependencies in the output of the E-SInE [40] prover, which used to be part of the standard Sledgehammer setup.

We learned several lessons from the experiment:

- Scalability is an issue for large formalizations, but most of the time the NICTA users do not work as deeply in the formalization as we had initially feared.
- Judging from the spy data, the main reason why NICTA benefits only moderately from Sledgehammer (compared with other Isabelle projects) seems to be that many users have not yet integrated it in their workflow. The success rate is similar across users, so those who invoke Sledgehammer the most are also those who find the most proofs with it.
- The earlier separation of MaSh into a Python part and a Standard ML part, while making the core engine easily reusable (e.g., by other proof assistants), was a continual source of worries. Robustness and performance were achieved through a more integrated design.

For a project like seL4, one could imagine having a shared server, instead of performing the learning on each machine. HOLyHammer, for HOL Light, provides such a facility [50]. The "four zeros" mentioned in the introduction were chosen with casual Isabelle users in mind, but some teams are ready to spend more time and effort setting up their environment if they know it will bring significant gains.

Now that seL4 formalization is open source, it would be an obvious choice for evaluating large-theory reasoning in the style of Sect. 5. Unfortunately, it does not track Isabelle's development as closely as IsaFoR, making it technically difficult to conduct experiments with the latest version of Isabelle and MaSh.

8 Related Work and Contributions

The main related work is already mentioned in the introduction. Bridges such as Sledgehammer for Isabelle/HOL, Miz𝔸ℝ [51] for Mizar, and HOLyHammer [22] for HOL Light are opening large formal theories to methods that combine automatic theorem provers and artificial intelligence [18,52,53] to help automate interactive proofs. Today such large theories are the main resource for combining semantic and statistical AI methods [54,55].[6]

The main contribution of this work has been to add the emerging machine learning methods for fact selection to Sledgehammer and make them incremental, fast, and robust enough so that they run unnoticed on a single-user machine and respond well to common user-interaction scenarios. The advising services for Mizar and HOL Light [17,22,51,56] (with the partial exception of MoMM [56]) run primarily as remote servers, whereas Sledgehammer does most of its work on the user's machine. Other novelties of this work include the use of more proof-related features in the learning (inspired by MePo), experiments combining MePo and MaSh, and the related learning of various parameters of the systems involved. We have evaluated the methods on several proof developments, and scaled them to very large ones such as the seL4 formalization. The overall success rate of 66.1 % on the lemmas from several large Isabelle formalizations, and 76.7 % on the goals from Judgment Day, should be good news for Isabelle users.

[6] It is hard to envisage all possible combinations, but with the recent progress in natural language processing, suitable combination methods could soon be applied to another major aspect of formalization: the translation from informal prose to formal specification.

9 Conclusion

Fact selection is an important practical problem that arises with large-theory reasoning. Sledgehammer's MaSh selector brings the benefits of machine learning to Isabelle users: By decreasing the quantity and increasing the quality of facts passed to the automatic provers, it helps them find more, deeper proofs within the allotted time. Starting with the 2014 edition of Isabelle, MaSh is enabled by default and delivers on its promises: zero configuration, zero click, zero maintenance, and zero overhead. The core learning functionality is implemented as a pair of general-purpose algorithms that can be reused by other proof assistants.

Many areas are calling for more engineering and research; we mentioned a few already. Learning data could be shared on a server or supplied with the proof assistant. More advanced algorithms appear too slow for interactive use, but they could be optimized. Learning could be applied to control more aspects of Sledgehammer, such as the prover options or even MePo's parameters. Evaluations over the entire *Archive of Formal Proofs*, or on the seL4 formalization, might shed more light on MaSh's and MePo's strengths and weaknesses.

Acknowledgments Tobias Nipkow and Lawrence Paulson have, for years, encouraged us to investigate the integration of machine learning in Sledgehammer; their foresight has made this work possible. Gerwin Klein and Makarius Wenzel provided advice on technical and licensing issues. Andrew Reynolds helped us tune the command-line options passed to CVC4. Tobias Nipkow, Lars Noschinski, Mark Summerfield, Dmitriy Traytel, and the anonymous reviewers suggested many improvements to earlier versions of this paper. Blanchette was partially supported by the Deutsche Forschungsgemeinschaft (DFG) project Hardening the Hammer (Grant NI 491/14-1). Kaliszyk was supported by the Austrian Science Fund (FWF): P26201. Kühlwein was supported by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO) project Learning to Reason (Grant 612.001.010). Urban was supported by the NWO project Knowledge-Based Automated Reasoning (Grant 612.001.208) and the European Research Council (ERC) grant AI4REASON.

References

1. Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Schulz, S., Ternovska, E. (eds.) IWIL-2010, Volume 2 of EPiC, pp. 1–11. EasyChair (2012)
2. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, Volume 2283 of LNCS. Springer, Berlin (2002)
3. Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. In: Piterman, N., Smolka, S. (eds.) TACAS 2013, Volume 7795 of LNCS, pp. 493–507. Springer, Berlin (2013)
4. Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending Sledgehammer with SMT solvers. J. Autom. Reason. **51**(1), 109–128 (2013)
5. Blanchette, J.C., Popescu, A., Wand, D., Weidenbach, C.: More SPASS with Isabelle-Superposition with hard sorts and configurable simplification. In: Beringer, L., Felty, A. (eds.) ITP 2012, Volume 7406 of LNCS, pp. 345–360. Springer, Berlin (2012)
6. Reynolds, A., Tinelli, C., de Moura, L.: Finding conflicting instances of quantified formulas in SMT. In: Claessen, K., Kuncak, V. (eds.) FMCAD 2014, pp. 195–202. IEEE (2014)
7. Voronkov, A.: AVATAR: the architecture for first-order theorem provers. In: Biere, A., Bloem, R. (eds.) CAV 2014, Volume 8559 of LNCS, pp. 696–710. Springer, Berlin (2014)
8. Meng, J., Paulson, L.C.: Lightweight relevance filtering for machine-generated resolution problems. J. Appl. Logic **7**(1), 41–57 (2009)
9. The Mizar Mathematical Library. http://mizar.org/
10. Grabowski, A., Korniłowicz, A., Naumowicz, A.: Mizar in a nutshell. J. Formaliz. Reason. **3**(2), 153–245 (2010)
11. Urban, J.: MPTP 0.2: design, implementation, and initial experiments. J. Autom. Reason. **37**(1–2), 21–43 (2006)

12. Urban, J.: MaLARea: a metasystem for automated reasoning in large theories. In: Sutcliffe, G., Urban, J., Schulz, S. (eds.) ESARLT 2007, Volume 257 of CEUR Workshop Proceedings. CEUR-WS.org (2007)
13. Urban, J., Sutcliffe, G., Pudlák, P., Vyskočil, J.: MaLARea SG1-Machine learner for automated reasoning with semantic guidance. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008, Volume 5195 of LNCS, pp. 441–456. Springer, Berlin (2008)
14. Sutcliffe, G.: The 4th IJCAR automated theorem proving system competition-CASC-J4. AI Commun. **22**(1), 59–72 (2009)
15. Sutcliffe, G.: The 6th IJCAR automated theorem proving system competition-CASC-J6. AI Commun. **26**(2), 211–223 (2013)
16. Alama, J., Heskes, T., Kühlwein, D., Tsivtsivadze, E., Urban, J.: Premise selection for mathematics by corpus analysis and kernel methods. J. Autom. Reason. **52**(2), 191–213 (2014)
17. Kaliszyk, C., Urban, J.: MizAR 40 for Mizar 40. J. Autom. Reason. **55**(3), 245–256 (2015)
18. Kühlwein, D., van Laarhoven, T., Tsivtsivadze, E., Urban, J., Heskes, T.: Overview and evaluation of premise selection techniques for large theory mathematics. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012, Volume 7364 of LNCS, pp. 378–392. Springer, Berlin (2012)
19. Hales, T.C.: Introduction to the Flyspeck project. In: Coquand, T., Lombardi, H., Roy, M.-F. (eds.) Mathematics, Algorithms, Proofs, number 05021 in Dagstuhl Seminar Proceedings, pp. 1–11. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2006)
20. Hales, T., Adams, M., Bauer, G., Dang, D.T., Harrison, J., Hoang, L.T., Kaliszyk, C., Magron, V., McLaughlin, S., Nguyen, T.T., Nguyen, Q.T., Nipkow, T., Obua, S., Pleso, J., Rute, J., Solovyev, A., Ta, T.H.A., Tran, N.T., Trieu, T.D., Urban, J., Vu, K.K., Zumkeller, R.: A formal proof of the Kepler conjecture. CoRR, abs/1501.02155 (2015)
21. Harrison, J.: HOL light: a tutorial introduction. In: Srivas, M.K., Camilleri, A.J. (eds.) FMCAD '96, Volume 1166 of LNCS, pp. 265–269. Springer, Berlin (1996)
22. Kaliszyk, C., Urban, J.: Learning-assisted automated reasoning with Flyspeck. J. Autom. Reason. **53**(2), 173–213 (2014)
23. Kühlwein, D., Blanchette, J.C., Kaliszyk, C., Urban, J.: MaSh: machine learning for sledgehammer. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) ITP 2013, Volume 7998 of LNCS, pp. 35–50. Springer, Berlin (2013)
24. Wenzel, M.: Isabelle/Isar—a generic framework for human-readable proof documents. In: Matuszewski, R., Zalewska, A. (eds.) From Insight to Proof-Festschrift in Honour of Andrzej Trybulec, Volume 10(23) of Studies in Logic, Grammar, and Rhetoric. Uniwersytet w Białymstoku (2007)
25. Schulz, S.: System description: E 1.8. In: McMillan, K.L., Middeldorp, A., Voronkov, A. (eds.) LPAR-19, Volume 8312 of LNCS, pp. 735–743. Springer, Berlin (2013)
26. Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: Sharygina, N., Veith, H. (eds.) CAV 2013, Volume 8044 of LNCS, pp. 1–35. Springer, Berlin (2013)
27. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011, Volume 6806 of LNCS, pp. 171–177. Springer, Berlin (2011)
28. Bouton, T., de Oliveira, D.C.B., Déharbe, D., Fontaine, P.: veriT: an open, trustable and efficient SMT-solver. In: Schmidt, R.A. (ed.) CADE-22, Volume 5663 of LNCS, pp. 151–156. Springer, Berlin (2009)
29. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008, Volume 4963 of LNCS, pp. 337–340. Springer, Berlin (2008)
30. Hurd, J.: First-order proof tactics in higher-order logic theorem provers. In: Archer, M., Di Vito, B., Muñoz, C. (eds.) Design and Application of Strategies/Tactics in Higher Order Logics, NASA Technical Reports, pp. 56–68 (2003)
31. Blanchette, J.C., Böhme, S., Fleury, M., Smolka, S.J., Steckermeier, A.: Semi-intelligible Isar proofs from machine-generated proofs. J. Autom. Reason. (2015). doi:10.1007/s10817-015-9335-3
32. Paulson, L.C., Susanto, K.W.: Source-level proof reconstruction for interactive theorem proving. In: Schneider, K., Brandt, J. (eds.) TPHOLs 2007, Volume 4732 of LNCS, pp. 232–245. Springer, Berlin (2007)
33. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. J. Comput. Secur. **6**(1–2), 85–128 (1998)
34. Kaliszyk, C., Urban, J.: Stronger automation for Flyspeck by feature weighting and strategy evolution. In: Blanchette, J.C., Urban, J. (eds.) PxTP 2013, volume 14 of EPiC, pp. 87–95. EasyChair (2013)
35. Spärck Jones, K.: A statistical interpretation of term specificity and its application in retrieval. J. Doc. **28**, 11–21 (1972)
36. Alama, J., Kühlwein, D., Urban, J.: Automated and human proofs in general mathematics: an initial comparison. In: Bjørner, N., Voronkov, A. (eds.) LPAR-18, Volume 7180 of LNCS, pp. 37–45. Springer, Berlin (2012)

37. Berghofer, S., Nipkow, T.: Proof terms for simply typed higher order logic. In: Aagaard, M., Harrison, J. (eds.) TPHOLs 2000, Volume 1869 of LNCS, pp. 38–52. Springer, Berlin (2000)
38. Kühlwein, D., Urban, J.: Learning from multiple proofs: first experiments. In: Fontaine, P., Schmidt, R.A., Schulz, S. (eds.) PAAR-2012, Volume 21 of EPiC, pp. 82–94. EasyChair (2013)
39. Gauthier, T., Kaliszyk, C.: Premise selection and external provers for HOL4. In: Leroy, X., Tiu, A. (eds.) CPP 2015, pp. 49–57. ACM (2015)
40. Hoder, K., Voronkov, A.: Sine qua non for large theory reasoning. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE-23, Volume 6803 of LNCS, pp. 299–314. Springer, Berlin (2011)
41. Klein, G., Nipkow, T., Paulson, L. (eds.) Archive of Formal Proofs. http://afp.sf.net/
42. Thiemann, R., Sternagel, C.: Certification of termination proofs using CeTA. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009, Volume 5674 of LNCS, pp. 452–468. Springer, Berlin (2009)
43. Klein, G., Nipkow, T.: Jinja is not Java. In: Klein, G., Nipkow, T., Paulson, L. (eds.) Archive of Formal Proofs. http://afp.sf.net/entries/Jinja.shtml (2005)
44. Urban, C., Kaliszyk, C.: General bindings and alpha-equivalence in Nominal Isabelle. Log. Methods Comput. Sci. **8**(2:14), 1–35 (2012)
45. Hölzl, J., Heller, A.: Three chapters of measure theory in Isabelle/HOL. In: van Eekelen, M., Geuvers, H., Schmaltz, J., Wiedijk, F. (eds.) ITP 2011, Volume 6898 of LNCS, pp. 135–151. Springer, Berlin (2011)
46. Böhme, S., Nipkow, T.: Sledgehammer: judgement day. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010, Volume 6173 of LNCS, pp. 107–121. Springer, Berlin (2010)
47. Urban, J.: BliStr: The Blind Strategymaker. Presented at PAAR-2014. CoRR, abs/1301.2683, (2014)
48. Klein, G., Andronick, J., Elphinstone, K., Heiser, G., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., Winwood, S.: seL4: formal verification of an operating-system kernel. Commun. ACM **53**(6), 107–115 (2010)
49. Greenaway, D., Andronick, J., Klein, G.: Bridging the gap: automatic verified abstraction of C. In: Beringer, L., Felty, A. (eds.) ITP 2012, Volume 7406 of LNCS, pp. 99–115. Springer, Berlin (2012)
50. Kaliszyk, C., Urban, J.: HOL(y)Hammer: online ATP service for HOL light. Math. Comput. Sci. **9**(1), 5–22 (2015)
51. Urban, J., Rudnicki, P., Sutcliffe, G.: ATP and presentation service for Mizar formalizations. J. Autom. Reason. **50**(2), 229–241 (2013)
52. Denzinger, J., Fuchs, M., Goller, C., Schulz, S.: Learning from previous proof experience. Technical Report AR99-4, Institut für Informatik, Technische Universität München (1999)
53. Urban, J.: An overview of methods for large-theory automated theorem proving. In: Höfner, P., McIver, A., Struth, G. (eds.) ATE-2011, Volume 760 of CEUR Workshop Proceedings, pp. 3–8. CEUR-WS.org (2011)
54. Heras, J., Komendantskaya, E., Johansson, M., Maclean, E.: Proof-pattern recognition and lemma discovery in ACL2. In: McMillan, K.L., Middeldorp, A., Voronkov, A. (eds.) LPAR-19, Volume 8312 of LNCS, pp. 389–406. Springer, Berlin (2013)
55. Urban, J., Vyskočil, J.: Theorem proving in large formal mathematics as an emerging AI field. In: Bonacina, M.P., Stickel, M.E. (eds.) Automated Reasoning and Mathematics-Essays in Memory of William McCune, Volume 7788 of LNCS, pp. 240–257. Springer, Berlin (2013)
56. Urban, J.: MoMM—fast interreduction and retrieval in large libraries of formalized mathematics. Int. J. AI Tools **15**(1), 109–130 (2006)