Automated Reasoning Service for HOL Light

Cezary Kaliszyk and Josef Urban

[1] University of Innsbruck, Austria
[2] Radboud University, Nijmegen

Abstract. HOL(y)Hammer is an AI/ATP service for formal (computer-understandable) mathematics encoded in the HOL Light system, in particular for the users of the large Flyspeck library. The service uses several automated reasoning systems combined with several premise selection methods trained on previous Flyspeck proofs, to attack a new conjecture that uses the concepts defined in the Flyspeck library. The public online incarnation of the service runs on a 48-CPU server, currently employing in parallel for each task 25 AI/ATP combinations and 4 decision procedures that contribute to its overall performance. The system is also available for local installation by interested users, who can customize it for their own proof development. An Emacs interface allowing parallel asynchronous queries to the service is also provided. The overall structure of the service is outlined, problems that arise are discussed, and an initial account of using the system is given.

1 Introduction and Motivation

HOL Light [10] is one of the best-known interactive theorem proving (ITP) systems. It has been used to prove a number of well-known mathematical theorems[3] and to formalize the proof of the Kepler conjecture targeted by the Flyspeck project [9]. The whole Flyspeck development, together with the required parts of the HOL Light library consists of about 14.000 theorems and 1800 concepts. Motivated by the development of large-theory automated theorem proving [12,18,26,31] and its growing use for ITPs like Isabelle [19] and Mizar [29,30], we have recently implemented translations from HOL Light to ATP (automated theorem proving) formats, developed a number of premise-selection techniques for HOL Light, and experimented with the strongest and most orthogonal combinations of the premise-selection methods and various ATPs. This work, described in [15], has shown that 39% of the 14185 Flyspeck theorems could be proved in a push-button mode (without any high-level advice and user interaction) in 30 seconds of real time on a fourteen-CPU workstation.

The experiments that we did emulated the Flyspeck development (when user always knows all the previous proofs at a given point, and wants to prove the next theorem), however they were all done in an offline mode which is suitable for such experimentally-driven research. The ATP problems were created in large batches using different premise-selection techniques and different ATP encodings

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

(untyped first-order [23], polymorphic typed first-order [4], and typed higher-order [8]), and then attempted with different ATPs (17 in total) and different numbers of the most relevant premises. Analysis of the results interleaved with further improvements of the methods and data have gradually led to the current strongest combination of the AI/ATP methods.

This strongest combination now gives to a HOL Light/Flyspeck user a 39% chance (when using 14 CPUs, each for 30s) that he will not have to search the library for suitable lemmas and figure out the proof of the next toplevel theorem by himself. For smaller (proof-local) lemmas such likelihood should be correspondingly higher. To really provide this strong automated advice to the users, the functions that have been implemented for the experiments need to be combined into a suitable AI/ATP tool. Our eventual goal should be an easy-to-use service, which in its online form offers to formal mathematics (done here in HOL Light, over the Flyspeck-defined concepts) what services like Wolfram Alpha offer for informal/symbolic mathematics. Some expectations (linked to the recent success of the IBM Watson system) are today even higher[4]. Indeed, we believe that developing stronger and stronger AI/ATP tools similar to the one presented here is a necessary prerequisite (providing the crucial semantic understanding/reasoning layer) for building larger Watson-like systems for mathematics that will (eventually) understand (nearly-)natural language and (perhaps reasonably semanticized versions/alternatives of) LaTeX. The more user-friendly and smarter such AI/ATP systems become, the higher also the chance that mathematicians (and exact scientists) will get some nontrivial benefits (apart from the obvious verification/correctness argument, which however so far convinced only a few) from encoding mathematics (and exact science) directly in a computer-understandable form.

This paper describes the first instance of such a HOL Light/Flyspeck-based AI/ATP service. The service – HOL(y)Hammer[5] (HH) – is now available in its strongest form as a public online system, running on a 48-CPU server spawning for each query 25 different AI/ATP combinations and four decision procedures. This functionality is described in Section 2, together with short examples of interaction (Emacs, command-line queries). The service can be also installed locally, and trained on user's private developments. This is described in Section 3. The advantages of the two approaches are briefly compared in Section 4, and Section 5 concludes and discusses future work.

2 The Online Service Description

The overall architecture of the system is shown in Figure 1. The service receives a query (a formula to prove, possibly with local assumptions) generated by one of the clients/frontends (Emacs, web interface, HOL session, etc.). If the query produces a parsing (or type-checking) error, an exception is raised, and an error

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

[5] See [33] for an example of future where AIs turn into deities.

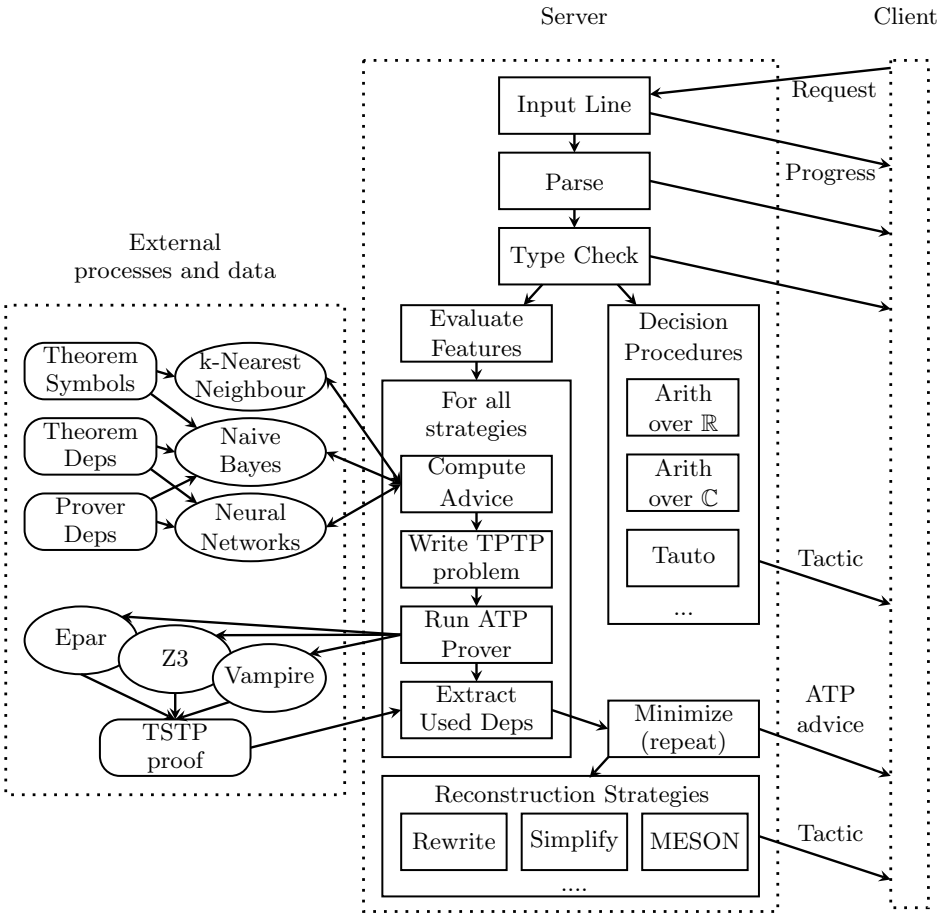

Fig. 1: Online service architecture overview

message is sent as a reply. Otherwise the parsed query is processed in parallel by the (time-limited) AI/ATP combinations and the native HOL Light decision procedures (each managed by its forked HOL Light process, and terminated/killed by the master process if not finished within its global time limit). Each of the AI/ATP processes computes a specific feature representation of the query, and sends such features to a specific instance of a premise advisor trained (using the particular feature representation) on previous proofs. Each of the advisors replies with a specific number of premises, which are then translated to a suitable ATP format, and written to a temporary file on which a specific ATP is run. The successful ATP result is then (pseudo-)minimized, and handed over to the combination of proof-reconstruction procedures. These procedures again run in parallel, and if any of them is successful, the result is sent as a particular tactic

application to the frontend. In case a native HOL Light decision procedure finds a proof, the result (again a particular tactic application) can be immediately sent to the frontend. The following subsections explain this process in more detail.

2.1 Feature Extraction and Premise Selection

Given a (formal) mathematical conjecture, the selection of suitable premises from a large formal library is an interesting AI problem, for which a number of methods have been tried recently [16, 26]. The strongest methods use machine learning on previous problems, combined in various ways with heuristics like SInE [12]. To use the machine learning systems, the previous problems have to be described as training examples in a suitable format, typically as a set of (input) features characterizing a given theorem, and a set of labels (output features) characterizing the proof of the theorem. Devising good feature/label characterizations for this task is again an interesting AI problem (see, e.g. [30]), however already the most obvious characterizations like the conjecture symbols and the names of the theorems used in the conjecture's proof are useful. This basic scheme can be extended in various ways; see [15] for the feature-extraction functions (basically adding various subterm and type-based characteristics) and label-improving methods (e.g., using minimized ATP proofs instead of the original Flyspeck proofs whenever possible) that we have so far used for HOL Light.

On average, for each feature-extraction method there are in total about 30.000 possible conjecture-characterizing features extracted from the theorems in the Flyspeck development. The output features (labels) are in the simplest setting just the names of the 14185 Flyspeck theorems[6] extracted from the proofs with a modified (proof recording [13]) HOL Light kernel. These features and labels are (for each extraction method) serially numbered in a stable way (using hashtables), producing from all Flyspeck proofs the training examples on which the premise selectors are trained. The learning-based premise selection methods currently used are those available in the SNoW [5] sparse learning toolkit (most prominently sparse naive Bayes) together with a custom implementation of the k-nearest neighbor (k-NN) learner. Training a particular learning method on all (14185) characterizations extracted from the Flyspeck proofs takes from 1 second for k-NN (a lazy learner that essentially just loads all the 14185 proof characterizations) and 6 seconds for naive Bayes using labels from minimized ATP proofs, to 25 seconds for naive Bayes using the labels from the original Flyspeck proofs.[7] The trained premise selectors are then run as daemons (using their server modes) that accept queries in the language of the numerical features

[6] In practice, the Flyspeck theorems are further preprocessed to provide better learning precision, for example by splitting conjunctions and detecting which of the conjuncts are relevant in which proof. Again, see [15] for the details. The most recent number of labels used is thus 16082.

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

over which they have been trained, producing for each query their ranking of all the labels (corresponding to the available Flyspeck theorems).

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

2.2 Translation to ATP Formats and Running ATPs

As mentioned in Section 1, several ATP formalisms are used today by ATP and SMT systems. However the (jointly) most useful proof-producing systems in our experiments turned out to be E [22] version 1.6 (run under the Epar [28] strategy scheduler), Vampire [21] 2.6, and Z3 [6] 4.0. All these systems accept the TPTP untyped first-order format (FOF). Even when the input formalism (the HOL logic [20] - polymorphic version of Church's simple type theory) and the output formalism (TPTP FOF) are fixed, there are in general many methods [3] how to translate from the former to the latter, each method providing different tradeoffs between soundness, completeness, ATP efficiency, and the overall (i.e., including HOL proof reconstruction) efficiency. The particular method chosen by us in [15] and used currently also for the service is the polymorphic tagged encoding [3]. To summarize, the higher-order features (such as lambda abstraction, application) of the HOL formulas are first encoded (in a potentially incomplete way) in first-order logic (still using polymorphic types), and then type tags are added in a way that usually guarantees type safety during the first-order proof search.

This translation method is in general not stable on the level of single formulas, i.e., it is not possible to just keep in a global hashtable the translated FOF version for each original HOL formula, as done for example for the MizAℝ ATP service. This is because a particular optimization (by Meng and Paulson [17]) is used for translating higher-order constants, creating for each such constant c a first-order function that has the minimum arity with which c is used in the particular set of HOL formulas that is used to create the ATP (FOF) problem. So once the particular AI/ATP managing process advises its N most-relevant HOL Light theorems for the conjecture, this set of theorems and the conjecture are as a whole passed to the translation function, which for each AI/ATP instance may produce slightly different FOF encoding on the formula level. The encoding function is still reasonably fast (fractions of a second when using hundreds of formulas), and still has the property that both the FOF formula names and the FOF formulas (also those inferred during the ATP proof search) can (typically)

be decoded back into the original HOL names and formulas (allowing later HOL proof reconstruction).

Each AI/ATP instance thus produces its specific temporary file (the FOF ATP problem) and runs its specific ATP system on it with its time limit. The time limit is currently set globally to 30 seconds for each instance, however (as usual in strategy scheduling setups) this could be made instance-specific too, based on further analysis of the time performance of the particular instances. Vampire and Epar already do such scheduling internally: the current version of Epar runs a fixed schedule of 14 strategies, while Vampire runs a problem-dependent schedule of several to dozen of strategies. Assuming one strategy for Z3 and on average eight strategies for Vampire, this means (counting the combinations in Table 1) that for each HOL query there are now 249 different proof-data/feature-extraction/learning/premise-slicing/ATP-strategy instantiations tried by the online service within the 30 seconds of the real time allowed for the query. Provided sufficient complementarity of such instantiations, this significantly raises the overall power of the service.

2.3 The AI/ATP Combinations Used

The 25 currently used combinations of the machine learner, proof data, number of top premises used, the feature extraction method, and the ATP system are shown in Table 1. The proof data are either just the data from the (minimized) ATP proofs (ATP0, ..., ATP3) created by a particular (MaLARea-style [31], i.e., re-using the proofs found in previous iteration for further learning) iteration of the experimenting, possibly preferring either the Vampire or Epar proofs (V_pref, E_pref), or a combination of such data from the ATP proofs with the original HOL proofs, obtained by slightly different versions of the HOL proof recording. Such combination typically uses the HOL proof only when the ATP proof is not available, see [15] for details. The `standard` feature extraction method combines the formula's symbols, standard-normalized subterms and normalized types into its feature vector. The standard normalization here means that each variable name is in each formula replaced by its normalized HOL type. The `all-vars-same` and `all-vars-diff` methods respectively just rename all formula variables into one common variable, or keep them all different. This obviously influences the concept of similarity used by the machine learners (see [15] for more discussion). The 40-NN and 160-NN learners are k-nearest-neighbors, run with $k = 40$ and $k = 160$. The reason for running these 25 particular combinations is that they together (computed in a greedy fashion) currently provide the greatest coverage of the solvable Flyspeck problems. This obviously changes quite often, whenever some of the many components of this AI architecture gets strengthened.

2.4 Decision Procedures

Some goals are hard for ATPs, but are easy for the existing decision procedures already implemented in HOL Light. To make the service more powerful, we also

Table 1: The 25 AI/ATP combinations used by the online service

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

try to directly use some of these HOL Light decision procedures on the given conjecture. A similar effect could be achieved also by mapping some of the HOL Light symbols (typically those encoding arithmetics) to the symbols that are reserved and treated specially by SMT solvers and ATP systems. This is now done for example in Isabelle/Sledgehammer, with the additional benefit of the combined methods employed by SMTs and ATPs over various well-known theories. Our approach is so far much simpler, which also means that we do not have to ensure that the semantics of such special theories remains the same (e.g., $1/0 = 0$ in HOL Light). The HOL Light decision procedures might often not be powerful enough to prove whole theorems, however for example the `REAL_ARITH`[8] tactic is called on 2678 unique (sub)goals in Flyspeck, making such tools a useful addition to the service.

Each decision procedure is spawned in a separate instance of HOL Light using our parallel infrastructure, and if any returns within the timeout, it is reported

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

to the user. The decision procedures that we found most useful for solving goals are:[9]

- `TAUT`[10] — Propositional tautologies.
  `(A ==> B ==> C) ==> (A ==> B) ==> (A ==> C)`
- `INT_ARITH`[11] — Algebra and linear arithmetic over $\mathbb{Z}$ (including $\mathbb{R}$).
  `&2 * &1 = &2 + &0`
- `COMPLEX_FIELD` — Field tactic over $\mathbb{C}$ (including multivariate $\mathbb{R}$[12]).
  `(Cx (&1) + Cx(&1)) = Cx(&2)`

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

2.5 Proof Minimization and Reconstruction

When an ATP finds (and reports in its proof) a subset of the advised premises that prove the goal, it is often the case that this set is not minimal. By re-running the prover and other provers with only this set of proof-relevant premises, it is often possible to obtain a proof that uses less premises. A common example are redundant equalities that may be used by the ATP for early (but unnecessary) rewriting in the presence of many premises, and avoided when the number of premises is significantly lower (and different ordering is then used, or a completely different strategy or ATP might find a very different proof). This (pseudo/cross-minimization) procedure is run recursively, until the number of premises needed for the proof no longer decreases. Minimizing the number of premises improves the chances of the HOL proof reconstruction, and the speed of (re-)processing large libraries that contain many such reconstruction tactics.[13]

Given the minimized list of advised premises, we try to reconstruct the proof. As mentioned in Section 2.1, the advice system may internally use a number of theorem names (now mostly produced by splitting conjunctions) not present in standard HOL Light developments. It is possible to call the reconstruction tactics with the names used internally in the advice system; however this would create proof scripts that are not compatible with the original developments. We could directly address the theorem sub-conjuncts (using, e.g., "`nth (CONJUNCTS thm)`

[9] The reader might wonder why the above mentioned `REAL_ARITH` is not among the tactics used. The reason is that even though `REAL_ARITH` is used a lot in HOL Light formalizations, `INT_ARITH` is simply more powerful. It solves 60% more Flyspeck goals automatically without losing any of those solved by `REAL_ARITH`. As with the AI/ATP instances, the usage of decision procedures is optimized to jointly cover as many problems as possible.

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

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

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

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

n") however such proof scripts look quite unnatural (even if they are indeed faster to process by HOL Light). Instead, we now prefer to use the whole original theorems (including all conjuncts) in the reconstruction.

Three basic strategies are now tried to reconstruct the proof: `REWRITE`[14] (rewriting), `SIMP`[15] (conditional rewriting) and `MESON` [11] (internal first-order ATP). These three strategies are started in parallel, each with the list of HOL theorems that correspond to the minimized list of ATP premises as explained above. The strongest of these tactics – `MESON` – can in one second reconstruct 79.3% of the minimized ATP proofs. While this is certainly useful, the performance of `MESON` reconstruction drops below 40% as soon as the ATP proof uses at least seven premises. Since the service is getting stronger and stronger, the ratio of `MESON`-reconstructable proofs is likely to get lower and lower. That is why we have developed also a fine-grained reconstruction method – `HH_RECON` [14], which uses the quite detailed TPTP proofs produced by Vampire and E. This method however still needs an additional mechanism that maintains the TPTP proof as part of the user development: either dedicated storage, or on-demand ATP-recreation, or translation to a corresponding fine-grained HOL Light proof script. That is why `HH_RECON` is not yet included by default in the service.

2.6 Parallel Infrastructure

An important aspect of the online service is its parallelization capability. This is needed to efficiently process multiple requests coming in from the clients, and to execute the large number of AI/ATP instances in parallel within a short overall wall-clock time limit. HOL Light uses a number of imperative features of OCaml, such as static lists of constants and axioms, and a number of references (mutable variables). Also a number of procedures that are needed use shared references internally. For example the `MESON` procedure uses list references for variables. This makes HOL Light not thread safe. Instead of spending lots of time on a thread-safe re-implementation, the service just (in a pragmatic and simple way, similar to the Mizar parallelization [27]) uses separate processes (Unix fork). Given a list of HOL Light tasks that should be performed in parallel and a timeout, the managing process spawns a child process for each of the tasks. It also creates a pipe for communicating with each child process. Progress, failures or completion information are sent over the pipe using OCaml marshalling. This means that it is enough to have running just one managing instance of HOL Light loaded with Flyspeck and with the advising infrastructure. This process forks itself for each client query, and the child then spawns as many AI/ATP, minimization, reconstruction, and decision procedure instances as needed.

2.7 Cache

Even though the service can asynchronously process a number of parallel requests, it is not immune to overloading by a large number of requests coming in

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

simultaneously. In such cases, each response gets less CPU time and the requests are less likely to succeed within the 30 seconds of wall-clock time. Such overloading is especially common for requests generated automatically. For example the Wiki service that is being built for Flyspeck [24] may ask many queries practically simultaneously when an article in the wiki is re-factored, but many of such queries will in practice overlap with previously asked queries. Caching is therefore employed by the service to efficiently serve such repeated requests.

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

2.8 Interaction with the Service

Fig. 2: Parallel asynchronous calls of the online advisor from Emacs.

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

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

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

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

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

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

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

Figure 2 shows an Emacs session with several HOL Light goals.[16] The online advisor has been asynchronously called on the goals, and just returned the answer

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

for the fifth goal and inserted the corresponding tactic call at an appropriate place in the buffer. The relevant Emacs code (customized for the HOL Light mode distributed with Flyspeck) is available online[17] and also distributed with the local HOL(y)Hammer install. It is a modification of the similar code used for communicating with the MizAR service from Emacs.

An experimental web editor interacting both with HOL Light and with the online advisor is described in [24]. The simplest option (useful as a basis for more sophisticated interfaces) is to interact with the service in command line, for example using netcat, as shown for two following two queries. The first query is solved easily by `INT_ARITH`, while the other requires nontrivial premise and proof search. Table 2 gives an overview of the service use so far (the queries came from 67 unique IP addresses).

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

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

Table 2: Statistics of the queries to the online service (Jan 24 - Mar 11 2013)

Total (Unique)	Parsed	Typechecked	Solved	ATP-solved	Reconstructed	Dec. Proc. solved
482	445	382	228	108	86	142

3 The Local Service Description

The service can be also downloaded,[18] installed and used locally, for example when a user is working on a private formalization that cannot be included in the public online service.[19]

Installing the advisor locally now requires two passes through the user's repository. In the first pass, a special module of the advisor stores the names of all the theorems available in the user's repository, together with their features (symbols, terms, types, etc., as explained in Section 2.1). In the second pass,

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

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

[19] The online service could eventually also accommodate private clones, using for example the techniques proposed for the Mizar Wiki in [2].

the dependencies between the named theorems are computed, again using the modified proof recording HOL Light kernel that records all the processing steps. Given the exported features and dependencies, local advice system(s) (premise selectors) are trained outside HOL Light. Using the fast sparse learning methods described in Section 2.1, this again takes seconds, depending on the user hardware and the size of the development. The advisors are then run locally (as independent servers) to serve the requests coming from HOL Light. While the first pass is just a fast additional function that can be run by the user at any time on top of his loaded repository, the second pass now still requires full additional processing of the repository. This could be improved in the future by running the proof-recording kernel as a default, as it is done for example in Isabelle.

The user is provided with a tactic (`HH_ADVICE_TAC`) which runs all the mechanisms described in the Section 2 on the current goal locally. This means that the functions relying on external premise selection and ATPs are tried in parallel, together with a number of decision procedures. The ATPs are expected to be installed on the user's machine and (as in the online service) they are run on the goal translated to the TPTP format, together with a limited number of premises optimized separately for each prover. By default Vampire, Eprover and Z3 are now run, using three-fold parallelization.

The local installation in its simple configuration is now only trained using the naive Bayes algorithm on the training data coming from the HOL Light proof dependencies and the features extracted with the standard method. As shown in [15], the machine learning advice can be strengthened using ATP dependencies, which can be also optionally plugged into the local mode. Further strengthening can be done with combinations of various methods. This is easy to adjust; for example a user with a 24-CPU workstation can re-use/optimize the parallel combinations from Table 1 used by the online service.

4 Comparison of the Online and Local Service

The two related existing services are MizAR and Sledgehammer. MizAR has so far been an online service (accessible via Emacs or web interface), while Sledgehammer has so far required a local install (even though it already calls some ATPs over a network). HOL(y)Hammer started as an online service, and the local version has been added recently to answer the demand by some (power)users.

As described in Section 2, the online service now runs 25 different AI/ATP instances and 4 decision procedures for each query. When counting the individual ATP strategies (which may indeed be very orthogonal in systems like Vampire and E), this translates to 249 different AI/ATP attempts for each query. If the demands grows, we can already now distribute the load from the current 48-CPU server to 112 CPUs by installing the service on another 64-CPU server. The old resolution-ATP wisdom is that systems rarely prove a result in higher time limits, since the search space grows very fast. A more recent wisdom (most prominently demonstrated by Vampire) however is that using (sufficiently orthogonal) strategy scheduling makes higher time limits much more useful.[20] And even more

[20] In [15], the relative performance of Vampire in 30 and 900 seconds is very different.

recent wisdom is that learning in various ways from related successes and failures further improves the systems' chances when given more resources.[21] All this makes a strong case for developing powerful online computing services that can in short bursts focus its great power on the user queries, which are typically related to many previous problems. Also in some sense, the currently used AI/ATP methods are only scratching the surface. For example, further predictive power is obtained in MaLARea by computing thousands of interesting finite models, and using evaluation in them as additional semantic features of the formulas. ATP prototypes like MaLeCoP [32] can already benefit from accumulated fine-grained learned AI guidance at every inference step that they make. The service can try to make the best (re-)use of all smaller lemmas that have been proved so far (as in [25]). And as usual in machine learning, the more data are centrally accumulated for such methods, the stronger the methods become. Finally, it is hard to overlook the recent trend of light-weight devices for which the hard computational tasks are computed by large server farms (cloud computing).

The arguments for installing the service locally are mainly the option to use the service offline, and so far also the fact that the online service does not yet accept and learn on (possibly private) user developments. The latter is just a matter of additional implementation work. For example the MizAR service already now keeps a number of (incompatible) MML versions over which the query can be formulated, and techniques have been recently developed for the Mizar wiki that provide very fast and space-efficient cloning of large libraries and private additions over them managed by the server. As usual, the local install will also require the tools involved to work on all kinds of architectures, which is often an issue, particularly with software that is mostly developed in academia.

5 Conclusion and Future Work

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

At this moment, there seems to be no end to better premise selection, better translation methods for ATPs (and SMTs, and more advanced combined systems like MetiTarski [1]), better ATP methods (and their AI-based guidance), and better reconstruction methods. Useful work can be also done by making the online service accept private user developments and clones that currently have to rely only on the local installation. An interesting future direction is the use of the service with its large knowledge base and growing reasoning power as a semantic

[21] See, e.g., the performance graph for the MaLARea 0.4 system in the recent Mizar@Turing12 competition: `http://www.tptp.org/CASC/J6/TuringWWWFiles/ResultsPlots.html#MRTProblems`

understanding (connecting) layer for experiments with tools that attempt to extract logical meaning from informal mathematical texts. Mathematics, with its explicit semantics, could in fact pioneer the technology of very deep parsing of scientific natural language writings, and their utilization in making stronger and stronger automated reasoning tools about all kinds of scientific domains.

References

1. Behzad Akbarpour and Lawrence C. Paulson. MetiTarski: An automatic theorem prover for real-valued special functions. *J. Autom. Reasoning*, 44(3):175–205, 2010.
2. Jesse Alama, Kasper Brink, Lionel Mamane, and Josef Urban. Large formal wikis: Issues and solutions. In James H. Davenport, William M. Farmer, Josef Urban, and Florian Rabe, editors, *Calculemus/MKM*, volume 6824 of *LNCS*, pages 133–148. Springer, 2011.
3. Jasmin Christian Blanchette, Sascha Böhme, Andrei Popescu, and Nicholas Smallbone. Encoding Monomorphic and Polymorphic Types. In *TACAS*, 2013. To appear, `http://www21.in.tum.de/~blanchet/enc_types_paper.pdf`.
4. Jasmin Christian Blanchette and Andrei Paskevich. TFF1: The TPTP typed first-order form with rank-1 polymorphism. Available online at `http://www21.in.tum.de/~blanchet/tff1spec.pdf`.
5. Andy Carlson, Chad Cumby, Jeff Rosen, and Dan Roth. The SNoW Learning Architecture. Technical Report UIUCDCS-R-99-2101, UIUC Computer Science Department, 5 1999.
6. Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.
7. Ulrich Furbach and Natarajan Shankar, editors. *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *LNCS*. Springer, 2006.
8. Allen Van Gelder and Geoff Sutcliffe. Extending the TPTP language to higher-order logic with automated parser generation. In Furbach and Shankar [7], pages 156–161.
9. Thomas C. Hales. Introduction to the Flyspeck project. In Thierry Coquand, Henri Lombardi, and Marie-Françoise Roy, editors, *Mathematics, Algorithms, Proofs*, volume 05021 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.
10. John Harrison. HOL Light: A tutorial introduction. In Mandayam K. Srivas and Albert John Camilleri, editors, *FMCAD*, volume 1166 of *LNCS*, pages 265–269. Springer, 1996.
11. John Harrison. Optimizing Proof Search in Model Elimination. In M. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th International Conference on Automated Deduction*, number 1104 in Lecture Notes in Artificial Intelligence, pages 313–327. Springer-Verlag, 1996.
12. Krystof Hoder and Andrei Voronkov. Sine qua non for large theory reasoning. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *LNCS*, pages 299–314. Springer, 2011.
13. Cezary Kaliszyk and Alexander Krauss. Scalable LCF-style proof translation. Submitted to ITP 2013, `http://cl-informatik.uibk.ac.at/~cek/import.pdf`.
14. Cezary Kaliszyk and Josef Urban. PRocH: Proof reconstruction for HOL Light. Accepted for CADE 2013, `http://mws.cs.ru.nl/~urban/proofs.pdf`.

15. Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *CoRR*, abs/1211.7012, 2012.
16. Daniel Kühlwein, Twan van Laarhoven, Evgeni Tsivtsivadze, Josef Urban, and Tom Heskes. Overview and evaluation of premise selection techniques for large theory mathematics. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR*, volume 7364 of *LNCS*, pages 378–392. Springer, 2012.
17. Jia Meng and Lawrence C. Paulson. Translating higher-order clauses to first-order clauses. *J. Autom. Reasoning*, 40(1):35–60, 2008.
18. Lawrence C. Paulson and Jasmin Blanchette. Three years of experience with Sledgehammer, a practical link between automated and interactive theorem provers. In *8th IWIL*, 2010. Invited talk.
19. Lawrence C. Paulson and Kong Woei Susanto. Source-level proof reconstruction for interactive theorem proving. In Klaus Schneider and Jens Brandt, editors, *TPHOLs*, volume 4732 of *LNCS*, pages 232–245. Springer, 2007.
20. Andrew Pitts. The HOL logic. In M. J. C. Gordon and T. F. Melham, editors, *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993.
21. Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *AI Commun.*, 15(2-3):91–110, 2002.
22. Stephan Schulz. E - A Brainiac Theorem Prover. *AI Commun.*, 15(2-3):111–126, 2002.
23. Geoff Sutcliffe, Stephan Schulz, Koen Claessen, and Allen Van Gelder. Using the TPTP language for writing derivations and finite interpretations. In Furbach and Shankar [7], pages 67–81.
24. Carst Tankink, Cezary Kaliszyk, Josef Urban, and Herman Geuvers. Formal mathematics on display: A wiki for Flyspeck. Submitted to CICM 2013.
25. Josef Urban. MoMM - fast interreduction and retrieval in large libraries of formalized mathematics. *International Journal on Artificial Intelligence Tools*, 15(1):109–130, 2006.
26. Josef Urban. An Overview of Methods for Large-Theory Automated Theorem Proving (Invited Paper). In Peter Höfner, Annabelle McIver, and Georg Struth, editors, *ATE Workshop*, volume 760 of *CEUR Workshop Proceedings*, pages 3–8. CEUR-WS.org, 2011.
27. Josef Urban. Parallelizing Mizar. *CoRR*, abs/1206.0141, 2012.
28. Josef Urban. BliStr: The Blind Strategymaker. *CoRR*, abs/1301.2683, 2013.
29. Josef Urban, Piotr Rudnicki, and Geoff Sutcliffe. ATP and presentation service for Mizar formalizations. *J. Autom. Reasoning*, 50:229–241, 2013.
30. Josef Urban and Geoff Sutcliffe. Automated reasoning and presentation support for formalizing mathematics in Mizar. In Serge Autexier, Jacques Calmet, David Delahaye, Patrick D. F. Ion, Laurence Rideau, Renaud Rioboo, and Alan P. Sexton, editors, *AISC/MKM/Calculemus*, volume 6167 of *LNCS*, pages 132–146. Springer, 2010.
31. Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jiří Vyskočil. MaLARea SG1 - Machine Learner for Automated Reasoning with Semantic Guidance. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 441–456. Springer, 2008.
32. Josef Urban, Jiří Vyskočil, and Petr Štěpánek. MaLeCoP: Machine learning connection prover. In Kai Brünnler and George Metcalfe, editors, *TABLEAUX*, volume 6793 of *LNCS*, pages 263–277. Springer, 2011.
33. Vernor Vinge. *A Fire Upon the Deep*. Tor Books, 1992.