System Description: E.T. 0.1

Cezary Kaliszyk*[1], Stephan Schulz[2], Josef Urban**[3], and Jiří Vyskočil[4]

[1] University of Innsbruck, Austria
[2] DHBW Stuttgart
[3] Radboud University Nijmegen
[4] Czech Technical University in Prague

Abstract. E.T. 0.1 is a meta-system specialized for theorem proving over large first-order theories containing thousands of axioms. Its design is motivated by the recent theorem proving experiments over the Mizar, Flyspeck and Isabelle data-sets. Unlike other approaches, E.T. does not learn from related proofs, but assumes a situation where previous proofs are not available or hard to get. Instead, E.T. uses several layers of complementary methods and tools with different speed and precision that ultimately select small sets of the most promising axioms for a given conjecture. Such filtered problems are then passed to E, running a large number of suitable automatically invented theorem-proving strategies. On the large-theory Mizar problems, E.T. considerably outperforms E, Vampire, and any other prover that does not learn from related proofs. As a general ATP, E.T. improved over the performance of unmodified E in the combined FOF division of CASC 2014 by 6%.

1 Introduction

The latest release of the TPTP benchmark library [22], TPTP 6.1.0, contains 20646 problems for theorem provers. More than a third of these problems have more than 100 axioms, more than 10% (2664) have more than 1000 axioms, and more than 5% (1231) have more than 10000 axioms.

Traditional (pre-1990) automated theorem proving (ATP) did not focus on such large problems. First experience with larger problems came from Quaife's work in the early 1990s [19]. Quaife identified the selection of relevant axioms as a possible way to handle large specifications, but did not offer detailed solutions.

Currently, large ATP problems are coming from ATP-to-ITP (interactive theorem provers) linkups (*hammers* [3]) such as Sledgehammer [16], HolyHammer [12,11] and MizAR [9], and common-sense reasoning [18] (or reasoning with the world's knowledge [25]) problems. Another interesting recent source of larger ATP problems is the work in Tarskian geometry by Beeson and Wos [2], containing in some cases over 300 clauses. We strongly believe that today's age of Big Data will lead to more and more large-theory problems, including problems generated from Wikipedia [6], biology textbook encoding [5], and other science

* Supported by the Austrian Science Fund FWF grant P26201.
** Supported by NWO grant nr. 612.001.208.

domains. Strong and practically usable methods and systems for proving large problems will be crucial for meaningful use of ATP in these new domains.

In the context of ATP/ITP cooperation, a number of methods have been developed recently to attack large problems. In pure ATP systems there has so far been basically just one (highly successful) method for dealing with large problems: the SInE heuristic invented by Hoder, implemented first as a standalone filter [29] and then inside Vampire [15,7] and E [20].

This paper describes E.T., a general large-theory ATP system based on E. It uses a combination of several methods transferred from the recent AITP[5] research. In Section 6 we show that the first version of E.T. already performs very well on large problems from the MPTP2078 benchmark, improving over plain E and Vampire by 98% resp. 22%. When used as a general ATP, E.T. has improved the performance of plain E in the combined FOF division of CASC 2014 by 6%. E.T. is available at `http://mws.cs.ru.nl/~urban/et10`.

2 Overview of E.T.

E.T. is intended for solving ATP problems that have one defining feature: they contain a large fraction of axioms that are not necessary for proving the conjecture. A secondary aspect of such large problems is that they often contain lemmas that can be used to construct alternative proofs. Thus it is useful to have a portfolio of complementary strategies that can select different promising axiom subsets and optimize the proof search over them.

We assume a setting in which a sequence of independent problems is presented to the system. In partcular, we do not assume that many related problems are being solved so that one could use consistent symbol and formula names and previous proofs in learning how to prove the next problems. Neither do we assume that common axioms are pre-loaded and expensively pre-processed once. Instead, the reading and preprocessing is done independently for each query.

This setting corresponds to the FOF division of CASC, which uses problems of various origins. It excludes some of the strongest and most obvious learning methods [13] possible in the "Large Theory Batch" scenario, where many problems share a background theory. However, ideas can still be transferred. For example, strategy invention as done by BliStr [28] can be used on sets of problems that do not share symbols and formula names.[6]

Similarly, one can use a number of symbolic and statistical methods successfully used in AITP for extracting useful feature characterizations of the large number of formulas. This has to be done much faster when solving a single large problem, making use of a layered architecture where the layers have different speed/precision trade-offs. Such layered (also called *early-exit*) approaches have been explored in information retrieval and particularly in web-search ranking

[5] We will use AITP as an abbreviation for the ATP/ITP cooperation, hinting also at the AI aspects of that topic.

[6] Although never publicly described, similar methods used are one of the main dark sources of Vampire's success.

systems [4], from which large-theory systems like E.T. can draw useful analogies. The extracted features are then used in E.T. as an input to several premise selection algorithms, such as our custom version of the Meng-Paulson filter (MePo) [17] and a non-learning version of the distance-weighted k-nearest neighbor (k-NN) algorithm [10]. The high-level description of E.T.'s processing chain is as follows:

1. The large input problem (potentially containing millions of axioms) is first reduced to several thousands of axioms using three differently parameterized (and reasonably complementary) non-strict versions of E's fast generalized SInE filter (see Section 4.1). Further processing is done with the union of these three filtered versions. This first reduction takes about 10 s for a problem with 500000 axioms. This speed is achieved by sharing several SInE preprocessing steps between the differently parameterized SInE selection passes.
2. Very long formulas are removed to prevent blow-ups in the following stages.[7]
3. If the original problem is in FOF, the reduced problem is clausified, removing very long clauses to prevent blow-ups in the feature generation phase.
4. Several tools are used to compute features of the formulas (or clauses) in the reduced problems. The current version can use as features symbols, (variable-normalized) shared terms, and all matching terms. See Section 3.
5. Several external premise selectors such as MePo and k-NN (Section 4) use these (probabilistically normalized) features to rank the axioms according to their estimated relevance to the conjecture, and problems with varied numbers of the top-ranking axioms are written.
6. Such problems are then (sequentially)[8] passed to E, which then typically applies much more restrictive SInE filtering to them, followed by a pool of the large-theory ATP strategies (Section 5).

3 Feature Generators

The feature generators are run on the problems reduced by the first fast SInE layer to several thousand formulas or clauses. Apart from using symbols as features, E.T. also enumerates all terms and subterms in the reduced problems' formulas, using E's fast shared term banks. Different variable normalization schemes can be used to increase or decrease the sharing of such features across formulas, providing different term-based similarity metrics.

A recent addition to the pool of such feature generators is a fast implementation of discrimination trees, enumerating for each formula ϕ all terms in all formulas, that are more general than the terms in ϕ. Such features (when suitably probabilistically weighted) provide a better concept of similarity of formulas than any other syntactic features used so far [14]. This feature generator

[7] The current limit for formula/clause size is 5kb. This filters out only a few of formulas from the large corpora of interest, and in practice does not influence completeness.

[8] E.T. runs its strategies sequentially by default. It is also possible to run the strategies, premise selectors, and feature extractors in parallel when more cores are available.

is the reason why the formula and clause sizes need to be kept below certain size (to prevent high quadratic factors), keeping the enumeration of matching terms within seconds for the reduced problem. For the weighting of features, we use the fast IDF scheme that adds practically no overhead while significantly improving the similarity metrics [10].

4 Premise Selectors

Since no previous proofs are available when running E.T., it relies on premise selectors that do not learn from related proofs. In particular, we use a number (currently 28) of differently parameterized E's generalized SInE filters limited to symbolic features in phase #1 and phase #6, and our modified version of the MePo filter using the more expensive features in phase #5. These two methods are briefly described below. Some additional performance is gained (see Table 2) by adding in phase #5 a non-learning version of the distance-weighted k-nearest neighbor (k-NN) algorithm [10], where each formula ϕ only carries the information that it is useful for proving facts with high feature overlap with ϕ.

4.1 Generalized SInE in E

E has native support for axiom selection in large theories. It implements a parametrized and efficient version of Hoder's SInE algorithm [7]. SInE is a fixed point algorithm. It starts with an initial set of formulas deemed necessary for the proof (usually including at least the conjecture), and successively adds formulas *related* to formulas already included, until a fixpoint is reached. Relatedness is based on sharing of at least one function- or predicate symbol between already selected clauses/formulas and new candidates. If all symbols are used, this corresponds to a classical relevance relation. However, this typically selects sets of clauses that are much larger than necessary, and only has limited utility.

Hoder correctly conjectured that rare symbols forge a stronger bond than common symbols, as formulas that share rare symbols are more likely to be part of the same microtheory. Using only the rarest symbols in a formula to find related clauses or formulas turned out to be too strict a relation. Thus, to allow the relaxation of this criterion, Hoder used a *benevolence* parameter that allows not just the symbol with the lowest frequency to be used for the relatedness relation, but also symbols wich occur up to a certain factor more often. E adds the *generosity* concept, which always uses the n least frequent symbols.

E allows the following parametrization of its SInE implementation.

- Frequency can be based on counting formulas/clauses containing a symbol, or on counting individual (sub-)terms.
- The initial set of the fixpoint process can consist of just the conjecture, or it can also include formulas defined as additional hypotheses for a particular proof problem by the user via the TPTP formula role.
- Benevolence and generosity can be set.

– While SInE usually runs to a fixpoint, E can terminate the process after a pre-determined number of iterations
– E also allows hard limits on the axiom set size, either in absolute terms, or as a fraction of the original specification.

This generalized SInE algorithm is implemented in E proper, where it is supported by a meta-level automatic parameterization. It also is available as a stand-alone tool that will efficiently apply several different parameterizations, sharing as much of the work as is possible. This includes parsing, frequency counting, and indexing of clauses and formulas by function symbol.

In E.T., SInE is used in two phases:

1. In phase #1 when the following non-strict (manually adjusted) SInE filters are used to make the later more expensive filters reasonably fast[9]:
   ```
   GSinE(CountFormulas, hypos, 3, , , 1500, 1.0)
   GSinE(CountFormulas, hypos, 1.2, , , 1500, 1.0)
   GSinE(CountFormulas, hypos, 30, , , 1500, 1.0)
   ```
   This phase takes about 10 s for problems with 500000 axioms, leaving enough time for the next phases when using 60 s time limit.
2. In phase #6, SInE is used in most of the E strategies that are ultimately run on the problems prepared by the previous filters. The parameters for SInE in these strategies are listed in Table 2. They are designed (jointly with other ATP parameters) automatically by the BliStr loop on suitable samples of large-theory problems (in this case Flyspeck and the 1000 Mizar@Turing training problems). The parameters that can be varied are the benevolence, number of iterations, and the absolute maximum axiom size. The rest of the parameters are fixed to the same values as in the non-strict filters above.

4.2 MePo3

MePo3 is an algorithm for assigning predicted relevance based on the Meng-Paulson relevance filter (MePo) [17] modified in several ways.

The algorithm is implemented as a recursive function which is given as input the set of all axioms A a set of weighted features F together with an increment number p. The initial value of F (F_0) are the features of the conjecture C, i.e., $F_0 = F(C)$, where $F(\phi)$ denotes features of a formula ϕ. The weights of the initial features are set to 1. Each recursive call i ($i > 0$) first computes the cosine distance between the remaining (not yet chosen) axioms in A and the given feature vector F_{i-1}. The axioms are then sorted by this distance, and the p axioms with the smallest distance are included in MePo3's answer. For each axiom ϕ included in the answer, its features $F(\phi)$ weighted by a factor of ϕ's distance to F_{i-1} are added to F_{i-1}, resulting in F_i which is then passed to the next recursive call. The algorithm is inspired by the Meng-Paulson filter, however we have introduced several changes:

[9] Parameters are used in the order given above. Missing parameters use E's built-in default values.

- MePo3 computes the distance as the cosine distance of the weighted feature vectors, rather than the proportion of relevant features to irrelevant ones.
- MePo includes in the answer the facts that are nearer to the conjecture than a given factor. This factor is modified in the recursive calls. This did not perform well for FOF problems, so we use an included-number in MePo3,
- MePo has a number of special cases that have been built into the algorithm to optimize for Isabelle/HOL, such as bonuses for elimination rules or facts present in the simplifier. MePo3 only has no such optimizations, instead relies on more advanced feature characterizations.

E.T. 0.1 always uses MePo3 with $p = 100$. The two parameters that are varied are the features used, and the number of best premises selected. The same is true for the distance-weighted k-NN, where in the simple scenario without previous proofs the number of best premises selected is always equal to the number of nearest neighbors k. Both MePo3 and k-NN are implemented efficiently in C++. Since the problems passed to them are already reduced to several thousands axioms, running these premise selectors is usually done within seconds, depending on the number of features used. As in E's SInE, a lot of work is shared between the different instances of k-NN and MePo3.

5 E Strategies and Global Optimization

When phase #5 premise selectors have finished, E is run on the filtered problems, using 36 different strategies (see Table 2). Four of these strategies are taken from E's exisiting portfolio, three are various versions of E's auto mode, which itself selects strategies based on problem characteristics.

The remaining 29 strategies have been designed automatically by BliStr, using the Mizar@Turing training problems and a small random sample of the Flyspeck problems. BliStr finds a small set of strategies that solve as many training problems as possible. This is done in an infinite loop which interleaves (i) fast iterative improvement of the strongest strategies on easy problems, (ii) slow evaluation of the newly invented strategies on all problems, and (iii) subsequent update of the candidate set of strong strategies and of the set of easy problems used for the next iterative improvement. The inclusion of the strategies into the final portfolio was done heuristically, based on their joint (greedy) coverage of the Mizar@Turing and Flyspeck problems.

6 Experimental Analysis

For the main evaluation we use the MPTP2078 benchmark [1], used for the large-theory division (Mizar@Turing) of the 2012 CASC@Turing automated reasoning competition [23]. These are 2078 related large-theory problems (conjectures) from the development of the general topological proof of the Bolzano-Weierstrass theorem extracted from the Mizar library. For each conjecture C we assume that all formulas stated earlier in the development can be used to prove C. This results

ATP	E 1.8 (%)	Vampire 2.6 (%)	E.T. 0.1 (%)	Union (%)
Small problems	1213 (58)	1319 (63)	1357 (65)	1416 (68)
Large problems	580 (28)	940 (45)	1148 (55)	1208 (58)
Large/small ratio	0.48	0.71	0.85	0.85

Table 1. ATPs on the large and small MPTP2078 problems, using 60 s time limit.

in *large* ATP problems that have 1877 axioms on average. For each conjecture C we also know its ITP (Mizar) proof, from which we can (approximately [1]) determine a much smaller set of axioms that are sufficient for an ATP proof after the translation from Mizar to TPTP [26,27]. This gives rise to *small* ATP problems, where the ATP is significantly advised by the human author of the ITP proof. These small problems contain only 31 axioms on average.

Table 1 compares the performance of E 1.8, Vampire 2.6, and E.T. 0.1 on the MPTP2078 problems. All systems are run with 60s time limit on a 32-core server with Intel Xeon E5-2670 2.6GHz CPUs, 128 GB RAM, and 20 MB cache per CPU. Each problem is assigned one CPU. On small problems, the three systems do not differ much. Vampire solves 9% more problems than E, E.T solves 12% more problems than E and 3% more than Vampire. All systems together can solve 68% of the small problems. Differences are larger on large problems, where Vampire solves 62% more problems than E, E.T. solves 98% more problems than E, and E.T. solves 22% more problems than Vampire. An interesting metric is the ratio of the number of large problems solved to the number of small problems solved. For E this ratio is below 0.5, for Vampire it is 0.71, and for E.T. it is 0.85. This suggests that Vampire's large-theory techniques (primarily SInE) are much stronger than those used in the default mode of E, and shows that such techniques in E.T. (i.e., the premise-selection layers) are much more successful than the other systems.

Table 2 sheds more light on how E.T. achieves its performance on large problems. It lists the first 30 strategies (of 49 total) as tried sequentially by E.T., together with their success on the small and large problems. On the small problems, 79% is solved already by the first two strategies that use only non-strict (or none) SInE filtering. On the large problems, these two strategies solve however only 34% of the problems, while the next two restrictive strategies solve 40% of the problems (they are given only the problems unsolved by the first two strategies). The third strategy does only two SInE iterations and takes only 60 best axioms, and the fourth strategy combines MePo (taking 128 best premises) with similarly restrictive SInE.

The second independent evaluation is the FOF division of CASC-J7 [21]. The results of the E-based ATPs and Vampire are shown in Table 3. The overall improvement of E.T. (using E version 1.8) over E (newer version 1.9) is 6% (18 problems more), and on problems with equality this is 9%. There is no improvement on the problems without equality. This is likely an artifact of E.T.'s strategy invention being done on Flyspeck and Mizar problems, which almost always use equality. While Vampire solves 11% more problems than E.T., its

nr.	small	large	selector	premises	B_{SInE}	R_{SInE}	L_{SInE}	ATP strategy (name)
1	933	331	ful		2.0		500	b57035dec1c1e73fa888146ae569c7cc8f0
2	140	55	ful					G-E-.208.C18.F1.SE.CS.SP.PS.S0Y
3	28	262	ful		1.1	2	60	eba37f91665fc364eeb63558058658ee9a1
4		198	mepo3-nrm	128	2.0	2	100	88760aa43d575e84b7030b8a6188f74ba5f
5		43	mepo3-nrm	400				G-E-.208.B07.F1.SE.CS.SP.PS.S0Y
6	40	14	knn-nrm	8				G-E-.208.B07.F1.SE.CS.SP.PS.S0Y
7		30	mepo3-std	64	1.5	3	40	1b33b681d9260087e24d422ea286498f4a4
8		24	mepo3-std	512	1.5	3	40	1b33b681d9260087e24d422ea286498f4a4
9		10	mepo3-nrm	128	1.1	1	60	2af8141978cb6a38e97452761cdbd9e1007
10		29	mepo3-std	64				G-E-.208.C18.F1.SE.CS.SP.PS.S0Y
11		18	mepo3-std	512	1.2	2	20000	my8simple.sine13
12	13	6	knn-nrm	20	1.5	4	100	cfee9ff42189552c6557cda7d36f20820c8
13		7	mepo3-std	512	2.0		500	b57035dec1c1e73fa888146ae569c7cc8f0
14		11	mepo3-std	512	1.1	2	60	eba37f91665fc364eeb63558058658ee9a1
15		1	mepo3-std	512				G-E-.208.B07.F1.SE.CS.SP.PS.S0Y
16		2	knn-nrm	96	1.5	4	100	cfee9ff42189552c6557cda7d36f20820c8
17		8	mepo3-nrm	400	6.0	2	20000	92168ebc2ef464a6f2d6a311a4fa90219fd
18		10	knn-nrm	256	2.0	2	100	37be21ea059a2fcb865621e373a97f33a9d
19		5	knn-nrm	64	5.0	2	60	c284f1f10aedfccc65cdb7d9b1210ef814c
20	13	3	knn-nrm	8				G-E-.200.B02.F1.SE.CS.SP.PI.S0S
21	31	5	knn-nrm	20				G-E-.200.B02.F1.SE.CS.SP.PI.S0S
22	46		ful					X-.sauto.schedule
23		8	ful		1.1	1	60	c7bb78cc4c665670e6b866a847165cb4bf9
24			ful		6.0	2	20000	92168ebc2ef464a6f2d6a311a4fa90219fd
25	18	15	cnf		6.0	1	20000	a3154f3180cc47331f1b05c36960c32e480
26	11	3	ful		1.5	4	100	cfee9ff42189552c6557cda7d36f20820c8
27	7	15	cnf		1.2	2	20000	X-.auto.sine03
28	2	1	ful		1.5	4	10	7cec1e0745ab65767d5d930d1f61b255ba3
29	5	6	ful		6.0	1	80	a74b37f2d8b7e35be554fc999f671188cf4
30	32	1	cnf		5.0	6	80	2af8b399ea0b8c22c6fc1b13069ad80214f

nr, small, large: order of the strategy; performance on small/large problems
premises: number of best-ranked premises used by the strategy (for MePo and k-NN)
SInE: B_{SInE} – Benevolence; R_{SInE} – Iteration limit; L_{SInE} – Absolute axiom limit.
Selectors: ful – reduced FOF (after phase #1); cnf – reduced FOF clausified (after phase #3); (mepo3|knn)-std – MePo3 or k-NN using symbols and shared terms with numbered variables (de Bruijn indeces); (mepo3|knn)-nrm – MePo3 or k-NN using symbols and matching terms with all variables renamed to one.
Strategies: The names of BliStr strategies are usually content-based hashes. The names of the original E strategies mirror their main parameters.

Table 2. The first 30 E.T. strategies run sequentially on the large and small MPTP2078 problems (60 s total time). E.T. exits immediatelly when a strategy finds a proof, therefore the success rates of the strategies are not directly comparable.

ATP	Vampire 2.6	E.T. 0.1	E 1.9	VanHElsing 1.0
FOF with Equality	234/250	224/250	205/250	199/250
FOF without Equality	141/150	115/150	116/150	111/150
FOF total	375/400	339/400	321/400	310/400

Table 3. Vampire and E-based ATPs on the CASC-J7 FOF division.

margin over E.T. on the equational problems is reduced to only 4%. Quite likely, Vampire's advantage on the problems without equality comes from splitting improvements and integration of SAT-solving [8,30].

7 Conclusion and Future Work

E.T. 0.1 shows very good performance on large problems, while being competitive on the problems from the standard FOF category of CASC. The performance is achieved without relying on slow preprocessing phases and learning from related proofs, however this requires a layered architecture with several filtering phases with different speed/precision trade-offs, and very efficient implementation of the core algorithms, using a lot of sharing and indexing data-structures. The other important aspects of E.T.'s performance are (i) relatively sophisticated features that provide good characterization of formulas, allowing more precise high-level approximation of the search problem, (ii) three non-learning state-of-the-art premise selection methods that complement each other, and (iii) a number of complementary automatically designed large-theory search strategies.

Future work may include addition of further non-learning selection methods such as the model-based selection used in SRASS [24], re-use of the strongest lemmas between the strategies, and, e.g., integration of the more expressive features into E's SInE and into E's clause-evaluation heuristics. Some of the techniques developed for E.T. could be also transferred back to learning systems like MaLARea and the AITP hammers.

References

1. J. Alama, T. Heskes, D. Kühlwein, E. Tsivtsivadze, and J. Urban. Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning*, 52(2):191–213, 2014.
2. M. Beeson and L. Wos. OTTER proofs in Tarskian geometry. In S. Demri, D. Kapur, and C. Weidenbach, editors, *IJCAR*, volume 8562 of *LNAI*, pages 495–510, 2014.
3. J. C. Blanchette, C. Kaliszyk, L. C. Paulson, and J. Urban. Hammering towards QED. Submitted, `http://www4.in.tum.de/~blanchet/h4qed.pdf`, 2015.
4. B. B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. Early exit optimizations for additive machine learned ranking systems. In B. D. Davison, T. Suel, N. Craswell, and B. Liu, editors, *WSDM*, pages 411–420. ACM, 2010.
5. V. K. Chaudhri, D. Elenius, A. Goldenkranz, A. Gong, M. E. Martone, W. Webb, and N. Yorke-Smith. Comparative analysis of knowledge representation and reasoning requirements across a range of life sciences textbooks. *J. Biomedical Semantics*, 5:51, 2014.
6. U. Furbach, I. Glöckner, and B. Pelzer. An application of automated reasoning in natural language question answering. *AI Commun.*, 23(2-3):241–265, 2010.
7. K. Hoder and A. Voronkov. Sine qua non for large theory reasoning. In N. Bjørner and V. Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *LNCS*, pages 299–314. Springer, 2011.

8. K. Hoder and A. Voronkov. The 481 ways to split a clause and deal with propositional variables. In M. P. Bonacina, editor, *CADE*, volume 7898 of *LNCS*, pages 450–464. Springer, 2013.
9. C. Kaliszyk and J. Urban. MizAR 40 for Mizar 40. *CoRR*, abs/1310.2805, 2013.
10. C. Kaliszyk and J. Urban. Stronger automation for Flyspeck by feature weighting and strategy evolution. In J. C. Blanchette and J. Urban, editors, *PxTP 2013*, volume 14 of *EPiC Series*, pages 87–95. EasyChair, 2013.
11. C. Kaliszyk and J. Urban. Learning-assisted automated reasoning with Flyspeck. *J. Autom. Reasoning*, 53(2):173–213, 2014.
12. C. Kaliszyk and J. Urban. HOL(y)Hammer: Online ATP service for HOL Light. *Mathematics in Computer Science*, 9(1):5–22, 2015.
13. C. Kaliszyk, J. Urban, and J. Vyskočil. Machine learner for automated reasoning 0.4 and 0.5. *CoRR*, abs/1402.2359, 2014. PAAR'14, to appear.
14. C. Kaliszyk, J. Urban, and J. Vyskočil. Efficient semantic features for automated reasoning over large theories. In *IJCAI*, 2015, to appear.
15. L. Kovács and A. Voronkov. First-order theorem proving and Vampire. In N. Sharygina and H. Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.
16. D. Kühlwein, J. C. Blanchette, C. Kaliszyk, and J. Urban. MaSh: Machine learning for Sledgehammer. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *ITP*, volume 7998 of *LNCS*, pages 35–50. Springer, 2013.
17. J. Meng and L. C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. *J. Applied Logic*, 7(1):41–57, 2009.
18. A. Pease and S. Schulz. Knowledge engineering for large ontologies with Sigma KEE 3.0. In S. Demri, D. Kapur, and C. Weidenbach, editors, *IJCAR*, LNAI, pages 519–525, 2014.
19. A. Quaife. *Automated Development of Fundamental Mathematical Theories*. Kluwer Academic Publishers, 1992.
20. S. Schulz. System description: E 1.8. In K. L. McMillan, A. Middeldorp, and A. Voronkov, editors, *LPAR*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.
21. G. Sutcliffe. Proceedings of the 7th IJCAR ATP system competition. `http://www.cs.miami.edu/~tptp/CASC/J7/Proceedings.pdf`.
22. G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.
23. G. Sutcliffe. The 6th IJCAR automated theorem proving system competition - CASC-J6. *AI Commun.*, 26(2):211–223, 2013.
24. G. Sutcliffe and Y. Puzis. SRASS - a semantic relevance axiom selection system. In F. Pfenning, editor, *CADE*, volume 4603 of *LNCS*, pages 295–310, 2007.
25. G. Sutcliffe, M. Suda, A. Teyssandier, N. Dellis, and G. de Melo. Progress towards effective automated reasoning with world knowledge. In H. W. Guesgen and R. C. Murray, editors, *FLAIRS*. AAAI Press, 2010.
26. J. Urban. MPTP - Motivation, Implementation, First Experiments. *Journal of Automated Reasoning*, 33(3-4):319–339, 2004.
27. J. Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reasoning*, 37(1-2):21–43, 2006.
28. J. Urban. BliStr: The Blind Strategymaker. *CoRR*, abs/1301.2683, 2013.
29. J. Urban, K. Hoder, and A. Voronkov. Evaluation of automated theorem proving on the Mizar Mathematical Library. In *ICMS*, pages 155–166, 2010.
30. A. Voronkov. AVATAR: the architecture for first-order theorem provers. In A. Biere and R. Bloem, editors, *CAV*, volume 8559 of *LNCS*, pages 696–710. Springer, 2014.