Kalpa Publications in Computing

Volume 9, 2018, Pages 15–22

LPAR-22 Workshop and Short Paper Proceedings

ProofWatch Meets ENIGMA: First Experiments

Zarathustra Goertzel, Jan Jakubův, and Josef Urban*

Czech Technical University in Prague

Abstract

Watchlist (also hint list) is a technique that allows lemmas from related proofs to guide a saturation-style proof search for a new conjecture. ProofWatch is a recent watchlist-style method that loads many previous proofs inside the ATP, maintains their completion ratios during the proof search and focuses the search by following the most completed proofs. In this work, we start to use the dynamically changing vector of proof completion ratios as additional information about the saturation-style proof state for statistical machine learning methods that evaluate the generated clauses. In particular, we add the proof completion vectors to ENIGMA (efficient learning-based inference guiding machine) and evaluate the new method on the MPTP Challenge benchmark, showing moderate improvement in E's performance over ProofWatch and ENIGMA.

1 Introduction

This work proposes and develops a new learning-based proof guidance – *ENIGMAWatch* – for saturation-style first-order theorem provers. It is based on two previous guiding methods implemented for the E [13] ATP system: ProofWatch [4] and ENIGMA [7, 8]. Both ProofWatch and ENIGMA enable E to use related proofs for guiding the proof search for a new conjecture. ProofWatch is based on the hints (watchlist) mechanism. It uses standard symbolic subsumption to compute the completion ratios of related proofs, and focuses the current proof search towards the most completed ones. ENIGMA uses statistical machine learning from many related proofs to estimate the relevance of the generated clauses for the current conjecture. ENIGMAWatch combines the two approaches by using the completion ratios of the related proofs as an additional characterization of the current proof state, which is used together with the conjecture for ENIGMA-style machine learning of clause relevance.

ENIGMAWatch is implemented for E and evaluated here on the MPTP Challenge[1] [14, 15] benchmark. This is a set of 252 first-order problems extracted from the Mizar Mathematical Library (MML) [5]. This set of lemmas is used in Mizar to prove the Bolzano-Weierstrass theorem.[2]

*Supported by the *AI4REASON* ERC Consolidator grant number 649043, and by the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466 and the European Regional Development Fund.

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

[2] http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/yellow19.html#T36

G. Barthe, K. Korovin, S. Schulz, M. Suda, G. Sutcliffe and M. Veanes (eds.), LPAR-IWIL 2018 (Kalpa Publications in Computing, vol. 9), pp. 15–22

The rest of the paper is organized as follows. Sections 2 and 3 provide the background on ENIGMA and ProofWatch. Section 4 explains how ENIGMA and ProofWatch are combined, and Section 5 evaluates ENIGMAWatch on the MPTP Challenge benchmark.

2 Saturation-based ATP and ENIGMA

Saturation-style first-order theorem provers are based on the *given-clause algorithm*. This algorithm splits the proof state into two subsets of clauses, the initially empty *processed clauses* P and the *unprocessed clauses* U. In each step the algorithm picks one unprocessed clause g (the *given clause*), puts g into P, and performs all possible inferences between g and the clauses in P. The newly generated clauses are, if not trivially discarded, put into U. This process is repeated until U is empty or a proof (contradiction) has been found. The core choice point is the selection of the next given clause.

ENIGMA [7, 8] stands for *E*fficient lear*N*ing-based *I*nference *G*uiding *MA*chine. It steers the selection of the given clauses in saturation-based ATPs like E. ENIGMA is based on the simple but fast *logistic regression* algorithm [2] effectively implemented by the LIBLINEAR open source library [3]. In order to employ logistic regression, first-order clauses need to be translated to fixed-length numeric *feature vectors*. The first version of ENIGMA [7] uses (top-down-)oriented term-tree walks of length 3 as *features*. For example, a unit clause "$P(f(a,b))$" contains only features "(P,f,a)" and "(P,f,b)" (see [7, Sec. 3.2] for details). Features are enumerated and a clause C is translated to the feature vector φ_C whose i-th member counts the number of occurrences of the i-th feature in clause C.

In order to train an ENIGMA *predictor* $\mathcal{E}$, all the given clauses $\mathcal{C}$ from a set of previous successful proof searches are collected. The given clauses used in the proofs are classified as positive ($\mathcal{C}^+ \subseteq \mathcal{C}$) and the remaining given clauses as negative ($\mathcal{C}^- \subseteq \mathcal{C}$). The clause sets $(\mathcal{C}^+, \mathcal{C}^-)$ are turned into feature vector sets (Φ^+, Φ^-) using a fixed *feature enumeration* π. Then a LIBLINEAR *classifier* w (a *weight vector*) is trained on the classification (Φ^+, Φ^-), classifying each clause as *useful* or *un-useful*. The classifier w and enumeration π produce a predictor $\mathcal{E} = (w, \pi)$ which is used to guide next proof searches in combination with other E heuristics.

The above ENIGMA predictors recommend clauses independently of the conjecture being currently proved. The second version of ENIGMA [8] overcomes this weakness by adding the *conjecture context*. Instead of representing just the clause C using the vector φ_C of length n (where n is the number of different features appearing in the training data), we use a vector (φ_C, φ_G) of length $2n$ where φ_G contains the features of the conjecture G. For a training clause C, G corresponds to the conjecture of the proof search where C was selected as a given clause. When classifying a clause C during a proof search, G corresponds to the conjecture currently being proved. In this way, ENIGMA provides conjecture-specific predictions. The enhanced ENIGMA additionally supports more features, like *horizontal features* and *static features* (see [8, Sec. 2] for more details).

Even with the above conjecture context, ENIGMA predictors still recommend clauses independently on the current state of the proof search. In this work we add the *proof state context*, that is, instead of representing the choice of a clause C by the vector (φ_C, φ_G) we represent the choice of the clause *in the current proof state* by the vector $(\varphi_C, \varphi_G, \varphi_\Pi)$ where φ_Π is a *proof-state vector* which describes the specific proof search state where C was selected. The next sections describe how we construct the proof vectors.

3 ProofWatch

3.1 Standard Watchlist Guidance

The watchlist (hint list) mechanism steers given clause selection via symbolic matching between generated clauses and clauses on a *watchlist* W. This technique has been developed and used extensively by Veroff [16] for the AIM project [9] with Prover9 [12] for obtaining long and advanced proofs of open conjectures. The standard watchlist mechanism as originally implemented in E, Otter [11], and Prover9 [12] uses only one watchlist W. In E, the watchlist mechanism uses a priority function `PreferWatchlist` that gives higher priority to clauses that match the watchlist W.[3] Clauses with higher priority are selected as given before clauses with lower priority[4]. When clauses from previous proofs are put on W, E thus prefers to follow steps from the previous proofs whenever it can.

3.2 ProofWatch

ProofWatch [4, Sec. 5] extends standard watchlist guidance by allowing for multiple watchlists $W_1,\ldots,W_n$, e.g., one corresponding to each related proof used. We say that a generated clause C *matches* a watchlist W if C subsumes a clause $C_W \in W$ (this implies that C logically entails C_W). During a proof search, clauses from some watchlist might get matched more often than clauses from others. The more clauses are matched from a watchlist W_i, the more the current proof search resembles W_i, and hence W_i might be more relevant for this proof search. The idea of ProofWatch is to prioritize clauses that match more relevant watchlists (proofs).

Watchlist relevance is dynamically computed. We define $progress(W)$ to be the count of clauses $C_W \in W$ that have been matched in the proof search thus far. The completion ratio, $\frac{progress(W)}{|W|}$, measures how much of the watchlist W has been matched. The *dynamic relevance* of each generated clause C is defined as the maximum completion ratio over all the watchlists W_i that C matches:[5]

$$relevance(C) = \max_{W \in \{W_i : C \sqsubseteq W_i\}} \left(\frac{progress(W)}{|W|}\right)$$

The higher the dynamic relevance, the higher priority a clause matching that watchlist is given.

4 ENIGMAWatch: ProofWatch meets ENIGMA

The watchlist completion ratios at each step in E's proof search can be taken as a vectorial representation of the proof state, and used as input to ENIGMA. This is how the *proof-state vector* φ_Π is constructed. The general motivation for this approach is to come up with an *evolving* characterization of the saturation-style proof state, preferably in a vectorial form suitable for machine learning tools. In general, this could be, e.g., a vector of more abstract similarities of the current proof state to other proofs measured in various (possibly approximate) ways. The ProofWatch based proof-state vector is thus the first reasonable implementation of this general idea.

[3] See [4, Sec. 4 and 6] for details.
[4] Numerically the lower the priority, the better. 0 is the best priority.
[5] $C \sqsubseteq W_i$ stands for C subsuming a clause from W_i

In particular, the positive $\mathcal{C}^+$ and negative $\mathcal{C}^-$ given clauses are output along with φ_Π, the proof-state vector at the time of their selection, and used in ENIGMA training.

Table 1 shows a sample proof-state vector based on 32 related proofs[6] for the Mizar theorem **YELLOW 5:36**[7] at the end of the proof search. Note that some related proofs, such as #2, were almost fully matched, while others, such as #7 were mostly not matched in the proof search.

0	0.438	42/96	1	0.727	56/77	2	0.865	45/52	3	0.360	9/25
4	0.750	51/68	5	0.259	7/27	6	0.805	62/77	7	0.302	73/242
8	0.652	15/23	9	0.286	8/28	10	0.259	7/27	11	0.338	24/71
12	0.680	17/25	13	0.509	27/53	14	0.357	10/28	15	0.568	25/44
16	0.703	52/74	17	0.029	8/272	18	0.379	33/87	19	0.424	14/33
20	0.471	16/34	21	0.323	20/62	22	0.333	7/21	23	0.520	26/50
24	0.524	22/42	25	0.523	45/86	26	0.462	6/13	27	0.370	20/54
28	0.411	30/73	29	0.364	20/55	30	0.571	16/28	31	0.357	10/28

Table 1: Example of the proof-state vector for the (serially numbered) 32 proofs loaded to guide the proof of `YELLOW_5:36`. The three columns are the watchlist i, the completion ratio of i, and $progress(W_i)/|W_i|$.

5 Evaluation on the MPTP Challenge Benchmark

5.1 MPTP Challenge

The Mizar Mathematical Library (MML) contains over 1000 articles on diverse topics. The MPTP Challenge was chosen as an initial benchmark because it covers 33 articles, is of manageable size, and focuses on a single problem. Thus the proof-state vector is hypothesized to be meaningful without need for curation. The problems range from easy to hard, and the challenge is still unsolved.[8]

In the *bushy* division used here, each problem's axioms are precisely the ones needed in the human proofs in the MML, thus the premise selection [1] task does not need to be done by the ATP. In 2007, 82% of the 252 bushy problems were solved by 14 ATP systems. Presently we ran Vampire [10] version 4.0, the state of the art theorem prover that performed best on the challenge, for 300 seconds per problem and solved 87% (220/252).

5.2 Results

The experiments are conducted using as the baseline 10 strategies that were previously evolved to perform well as an ensemble on the Mizar problems [6].[9] These strategies outperform E's auto-schedule strategy [6]. All benchmarks are run on the same hardware[10], with the same memory limits, and using E prover version 2.1[11].

[6] The proofs were chosen via k-NN. See [4, Sec. 6.1] for details.

[7] http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/yellow_5#T36

[8] The TPTP version of the Bolzano-Weierstrass theorem and its MML proof was however cross-verified [15].

[9] We care about the problems proven by the union of 10 strategies than the performance of any individual strategy.

[10] Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz with 256G RAM.

[11] Our version of E can be found at https://github.com/ai4reason/eprover/tree/devel.

We conduct three benchmarks to see how many more MPTP Challenge problems each method enables E to solve. We first run the baseline strategies, then ProofWatch[12] and ENIGMA using those results. For ENIGMAWatch, a second run of the baseline strategies while recording the proof-state vector φ_{Π} is needed before training the ENIGMA models.

The first two benchmarks run E for 1s and 30s per problem and strategy. As ProofWatch and ENIGMA can slow E down, we run one benchmark using abstract time instead of CPU time. Abstract time is measured by given-clause loops. $T15+C40000$ means that each problem is run until 40000 given clauses are processed or 15s passes.[13]

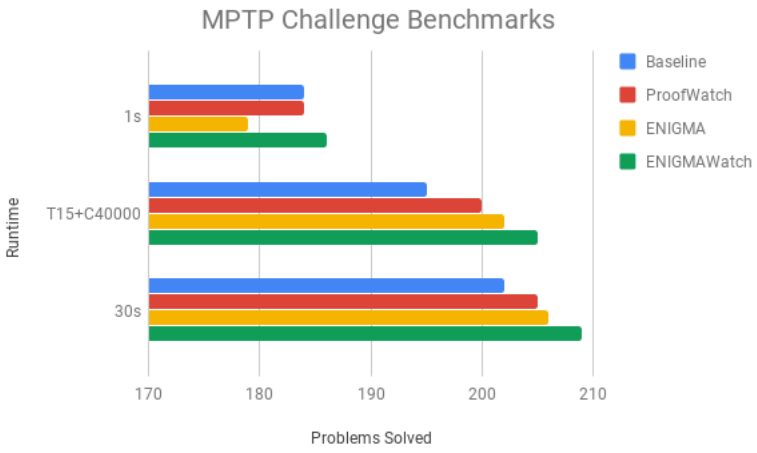

Figure 1: The absolute number of problems solved by each method. In 1s the baseline strategies prove 184 and ENIGMAWatch proves 186. With $T15 + C40000$ the baselines prove 195 and ENIGMAWatch proves 205. In 30s the baselines prove 202 and ENIGMAWatch proves 209.

ENIGMAWatch performs best in all of the benchmarks in Figure 1; however the difference in problems proved is small. As anticipated, the difference is most distinct when using abstract time rather than CPU time.

A performance metric in addition to the number of problems proven is how many given-clause loops E takes to find the proof.[14] This allows given-clause selection strategies to be compared on problems solved.

Figure 2 shows the average number of processed clauses used to find the proof on the $T15 + C40000$ benchmark. ProofWatch cuts the abstract proof-search time by about 75%, while ENIGMA and ENIGMAWatch do significantly better. The ENIGMA-based methods likely have superior efficiency because they provide guidance for each generated clause, whereas ProofWatch only provides guidance when a clause subsumes a watchlist clause. This shows that with proper guidance, E can find proofs much faster. In scenarios where many similar proofs have to be done, this seems useful. The MPTP Challenge is too small to use a standard train/test split[15], so we so far only measure the number of problems additionally proved and

[12] ProofWatch is run with a static watchlist.

[13] The baseline strategies often process this many clauses in $5 - 10$s.

[14] Which is best measured by looking at non-trivial processed clauses, as E has heuristics for labeling clauses trivial, and checks to see if they are subsumed by another processed clause.

[15] One could do leave-one-out testing to test this on this 252 problem dataset.

the proof shortening in terms of the abstract time.

Figure 3 examines how much more efficient ENIGMAWatch is on each problem by taking the average of the ratios:

$$\frac{\text{clauses by ENIGMAWatch on problem p}}{\text{clauses by Baseline on problem p}}$$

The same trend is present as with average clauses, but the outliers seem to stand out less. It's interesting that with one strategy, *mzr06*, ENIGMA uses more processed clauses per problem than the baseline. However *mzr06* only proved 17 problems, so ENIGMA did not have much training data at its disposal.

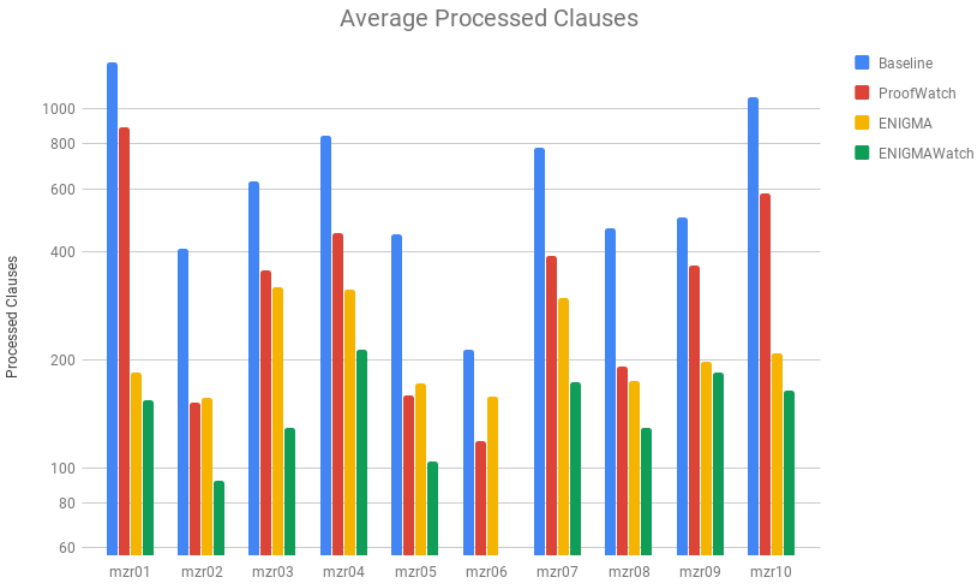

Figure 2: The average processed clauses used to find proofs on the $T15 + C40000$ benchmark.

6 MPTP2078: The Next Frontier

The MPTP2078 benchmark is similar to the MPTP Challenge however all theorems from the 33 Mizar articles are included, growing the number of problems to 2078. In previous work [4], we discovered that ProofWatch becomes slow when there are $10,000$ clauses on the watchlist. The inference speed is good enough with up to 128 proofs on the watchlist.[16] The baseline ensemble proves 1461 problems. Thus to use ENIGMAWatch, a small subset of the proofs available must be chosen as the proof-state vector.

We have tried to use k-medoids based on ENIGMA-style features of the problems and their initial clause sets for $k \in \{4, 8, 16, 32, 64\}$ to select the proof-state vectors; however this is not yet effective. The most effective watchlist curation method in ProofWatch is to use k-NN based on

[16] Thus the small MPTP Challenge dataset already uses watchlists near the limits of ProofWatch's capabilities.

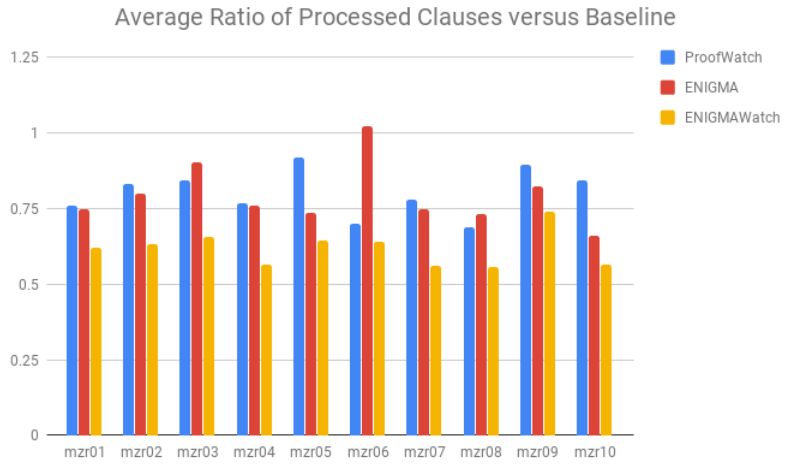

Figure 3: The average ratio of each method over the Baseline, which is a constant bar at 1.

ENIGMA-style features to suggest proofs for each problem [4]. ENIGMAWatch needs to have a consistent proof-state vector, so one idea is to take the union of k-NN suggested proofs and set the unused proofs' watchlists to zero (the empty clause) on a problem-specific basis. Another option is to have faster algorithms for matching (based on better indexing), for approximate matching or in general for estimating how much a clause belongs to a related proof. The latter ones could again be based on learning such approximate concepts from a large body of proofs.

7 Conclusion

The first experiment with ENIGMAWatch on the MPTP Challenge is encouraging. The performance is better than both ProofWatch and ENIGMA, especially with regard to the number of processed clauses needed to find a proof. This indicates that combining symbolic and statistical machine learning in this way can be fruitful.

However additional work is needed to find out how to best leverage the potential of the ENIGMAWatch method. ProofWatch works best when the watchlists are targeted on the specific problem. ENIGMA, using logistic regression, works best when given lots of data to learn from. Reconciling the two and applying ENIGMAWatch to larger datasets presents interesting research challenges.

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] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM.

[3] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, June 2008.

[4] Z. Goertzel, J. Jakubuv, S. Schulz, and J. Urban. ProofWatch: Watchlist guidance for large theories in E. In J. Avigad and A. Mahboubi, editors, *Interactive Theorem Proving - 9th International Conference, ITP 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10895 of *Lecture Notes in Computer Science*, pages 270–288. Springer, 2018.

[5] A. Grabowski, A. Korniłowicz, and A. Naumowicz. Mizar in a nutshell. *J. Formalized Reasoning*, 3(2):153–245, 2010.

[6] J. Jakubuv and J. Urban. BliStrTune: hierarchical invention of theorem proving strategies. In Y. Bertot and V. Vafeiadis, editors, *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, pages 43–52. ACM, 2017.

[7] J. Jakubuv and J. Urban. ENIGMA: efficient learning-based inference guiding machine. In H. Geuvers, M. England, O. Hasan, F. Rabe, and O. Teschke, editors, *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings*, volume 10383 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 2017.

[8] J. Jakubuv and J. Urban. Enhancing ENIGMA given clause guidance. In F. Rabe, W. M. Farmer, G. O. Passmore, and A. Youssef, editors, *Intelligent Computer Mathematics - 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings*, volume 11006 of *Lecture Notes in Computer Science*, pages 118–124. Springer, 2018.

[9] M. K. Kinyon, R. Veroff, and P. Vojtechovský. Loops with abelian inner mapping groups: An application of automated deduction. In M. P. Bonacina and M. E. Stickel, editors, *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, volume 7788 of *LNCS*, pages 151–164. Springer, 2013.

[10] 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.

[11] W. McCune and L. Wos. Otter: The CADE-13 Competition Incarnations. *Journal of Automated Reasoning*, 18(2):211–220, 1997. Special Issue on the CADE 13 ATP System Competition.

[12] W. W. McCune. Prover9 and Mace4. http://www.cs.unm.edu/~mccune/prover9/, 2005–2010. (acccessed 2016-03-29).

[13] 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.

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

[15] J. Urban and G. Sutcliffe. ATP cross-verification of the Mizar MPTP Challenge problems. In N. Dershowitz and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 546–560, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[16] R. Veroff. Using hints to increase the effectiveness of an automated reasoning program: Case studies. *Journal of Automated Reasoning*, 16(3):223–239, 1996.