On Evaluating Theorem Provers

Michael Rawson and Giles Reger

The University of Manchester, UK

Abstract

Evaluating automatic theorem provers is a non-trivial, subtle endeavour. There are a number of conflicting issues to consider, be they computational, statistical, economic, or — most difficult of all — *social*. We present a review of such challenges, seen from the perspective of, and with application to, the first-order system Vampire. We further highlight the exemplary achievements of existing thought and practical tools, and sketch some future directions for work in this area. We are not the first to consider this issue, and do not claim to offer the final word, but strongly believe that this is a topic that requires significant and sustained attention from our community.

1 Introduction

Automated theorem provers (ATPs) ingest an input problem consisting of axioms and a conjecture expressed in a certain logic, and attempt to determine if the conjecture is a consequence of the axioms in said logic. Some systems can also detect when this is not the case. Evidence, varying in quality but often in the form of a step-by-step proof, may be output from the system.

It is not straightforward to objectively evaluate ATPs. To begin with, it is not clear which input problems are in some way worthwhile and worth solving, and which are logical detritus to be ignored. Various enumerations of “all possible problems” exist, but here again it is difficult to know which enumeration is most interesting. In practice, ATPs are evaluated with respect to an existing fixed set of problems, often from industrial or mathematical settings. Problem libraries such as Thousands of Problems for Theorem Provers (TPTP) [32] combine several sources of problems into a unified format.

Most users and developers of ATPs agree that more problems solved, faster, is a good objective. For example, it is uncontroversial to say that an ATP using a given algorithm is worse than an ATP miraculously engineered to execute an identical algorithm twice as quickly. ATP users and developers are also impatient, so evaluations are phrased in terms of number of problems solved within a time limit, usually per-problem. Unfortunately, “solving more problems” and “solving problems faster” are usually, but not always, correlated. To further muddy the waters, modern systems are normally deployed as part of a *portfolio*, a collection of different systems or system configurations (“strategies”) all attempting the same problem, either sequentially or with some amount of parallelism. The idea behind the portfolio approach is that it is better on average to give up a doomed strategy after a short time and try again with a different approach, than it is to continue failing to solve the problem with one long strategy run. In this setting the overall performance of a strategy is less important than its contribution to overall portfolio performance.

In the following we discuss typical approaches taken by ourselves and others in the community (§2), some desirable outcomes and problems encountered in their pursuit (§3), and propose some possible solutions (§4) for discussion at ARCADE. We also present some exemplary thinking and useful tools (§5) contributed by others around this area.

2 Typical Approaches

Almost all practical research into ATP systems features some amount of quantitative system evaluation, "experiments". Possible exceptions might include systems that are the only implementation of reasoning in a certain logic or fragment [35], generic over many logics [37], or designed for an entirely new application [7], although even these may still opt for quantitative evaluation. Experiments provide implicit motivation for the great majority of the ATP research ecosystem, but arguably fall far short of the epistemic standards required of other fields, likely due to the staggering number of variables involved and collective acceptance of a trade-off that often sees robustness traded for ease of evaluation.

In simple terms, the typical approach is to select a set of problems, run ATPs over each of these problems, and count how many problems each ATP solved. We expand on this methodology below and describe some common variations.

2.1 Problem Libraries

Experiments most often use some kind of common library of problems, rather than an enumeration of all possible problems in a given class: this choice is usually motivated (if at all) as emphasising ATP performance on "problems of interest", as opposed to "synthetic" problems dismissed as uninteresting[1]. Problem libraries exist on a spectrum of generality, from a relatively-small number of problems extracted from some concrete application [4], to large bodies of problems extracted from a single source library of mathematics [38], to large heterogeneous problem sets[2] such as TPTP [32] and SMT-LIB [3] that are often used for competitions [33, 2]. Libraries vary in the amount of metadata available to ATPs, sometimes necessitating complex heuristics to recover information [24].

Although undoubtedly an exemplary achievement, an absurdly-effective driver of research (what is more annoying than an easily-available set of problems that you cannot solve?), and overall a great boon for ATP progress, problem libraries are not perfectly suited for experiments. Evaluation of systems is not problem libraries' only purpose or virtue, so they will likely never be ideal for this task, but nevertheless we highlight possible evaluation problems.

Large numbers of so-called "trivial" problems (solved very quickly by most systems) in problem libraries can skew results, a recurring theme in the SMT-LIB world [8, 22, 42] but also present elsewhere. Such problems are often denigrated, but serve many useful purposes, from simple historical value to testing new systems to exposing weaknesses in established strong systems. We would happily argue that these problems should be retained in libraries and evaluation methods adapted to accommodate their presence.

Diversity of problems (and problem tasks/types, e.g. comparing satisfiable and unsatisfiable SMT instances [22]) is another issue [12], but here it's not even clear what a desirable direction is. Very diverse problem sets emphasize strong general-purpose techniques at the expense of specialised reasoning techniques, but very specific sets achieve the opposite, and furthermore penalise systems with the wrong specialisation. It may be interesting to note that for many years SMT-COMP [42] used a division per logic, aiming to celebrate success within a single SMT-LIB logic[3] at the expense of a large number of divisions, but in 2021 the organisers have decided to start combining logics — divisions become more general, at the risk of failing to reward solvers specialising in a particular logic.

[1] not necessarily *easy*: even small constructed problems give modern systems trouble [19, 43]

[2] although these often have substructure, such as the domains in TPTP, or the problem families in SMT-LIB

[3] a combination of pre-defined theories such as linear integer arithmetic or uninterpreted functions

2.2 Running Systems

Even invoking a single ATP/problem pair in a reproducible fashion is not straightforward. Powerful heuristics such as Vampire's *limited resource strategy* [27] can dramatically change behaviour on repeated runs, and even supposedly-deterministic system configurations can be sensitive to the environment in which they run: hard disk, CPU, operating system state, etc. Small experiments may be more conveniently run on researchers' own desktop machines, which certainly have many sources of experimental "noise", such as other programs also demanding machine resources. Larger experiments are usually run on resources such as StarExec [31], which may reduce but does not completely remove this noise [8], although personal experience indicates such systems can introduce other issues such as scheduling non-determinism.

Choice of resource limits (most pertinently wall-clock time) is another interesting experimental parameter usually chosen arbitrarily: it is perfectly possible for different systems or configurations to be strong at different time limits, and time limits themselves range from a few seconds to as long as necessary for research mathematics [17], although it has often been noted that the majority of problems are solved within the first few seconds. Empirically, there is a "Peter Principle Point" where linear increase in resources makes little difference [36]. Both CASC and SMT-COMP have had an array of different wall-clock times. In 2019 SMT-COMP had an industrial challenge track with a 12 hour time limit per problem. In the same year SMT-COMP also introduced *24-second*[4] cross-division award. Similarly, in 2017 CASC ran a new track using a 30 second time limit, much shorter than its normal 5 minutes. This reflects varied use cases: in hardware verification long time limits (12 hours) are not unheard of, whilst software verification tools and proof assistants often employ very short time limits.

2.3 Comparison Methods

Experiments often seek to determine the behaviour of some technique with respect to a baseline, or similarly several different techniques (or parameter values, "options" henceforth) with respect to each other. This may be influenced by the question being asked e.g. the subtle difference between "which of these options should I keep in my solver" and "which of these options should be the default value".

The simplest approach is to run both the baseline and the new options identically and compare results — this remains valid and is used in some contexts — but is complicated by the presence of portfolio modes and systems with many existing options, such as Vampire. It may well happen that some new option is very useful in combination with some other system options, but not with others. Equally, in the presence of a large (and increasing) number of existing options, a full evaluation with all combinations of options is not computationally feasible. Vampire's developers already investigated this particular aspect of evaluation [25], discussing the following four methods:

"The cube" Fix a baseline ATP configuration (or at least aggressively subsample the existing option and problem space), then try all combinations of the new options under test.

"Randomly sample the super-cube" Randomly and repeatedly selecting ATP configurations and problems with new options can give some impressions about how useful the new options are, and which existing options they combine well with. However, it is not clear how to interpret the results, or how best to choose resource limits.

[4] Anecdotally (the second author was part of the conversation) this time limit came from an industrial use of SMT solvers that we were aware of.

“Compare with the Past” Run a portfolio previously used in competition with the new options. Relatively cheap, and if results are improved this is a clear victory. If not (and this is quite likely, given how portfolios are constructed) then this is not so clear.

“Building different Futures” Not tried in the investigation, but building a new portfolio with the same method used to construct past portfolios allows comparing portfolio performance with and without new options. We return to this idea later, but it appears very promising as an evaluation method.

On the other hand, we note that it is often the case that the developers of some systems (the main system that comes to mind here is CVC4) tend to focus on a small number of new, orthogonal options. As a result, evaluation is simplified somewhat, and experiments are usually relatively straightforward, comparing the solver with new options against other ATPs, e.g. [26]. This leaves developers free to select or generate appropriate benchmarks. In a different world altogether, SAT solvers tend to have very few (if any) system options, and the benefit of a decidable setting. This leads to some interesting evaluation methods [21] that the ATP community could learn from.

2.4 Summary Statistics

Once an experiment has finished, most publications contain some summary information to indicate the success[5] of the techniques described. Sometimes, a technique simply produces a much greater total number of solved problems, which is hard to argue with. If not, a researcher can hope that the modified system can prove something that nothing else can (“uniques”), which is made somewhat easier if few other runs are made. The argument made here — often using the word “complementary” — is that the feature can be gainfully included into a hypothetical future portfolio, although this is rarely actually carried out to support the claim in the publication. Even more summaries may be computed, particularly in competition:

Virtual Best Solver, a comparison against the hypothetical solver that chose the best approach between different systems, such as in [42, 16]. In SMT-COMP this is used to give the *Largest Contribution* ranking, which can be thought of as measuring the *regret* of not having a particular solver available.

SOTAC, State Of The Art Contribution, is mostly used in the TPTP/CASC world [34]. SOTAC for a problem/system pair is $1 - f$, where f is the fraction of systems that solved that problem, and SOTAC for a system as a whole is the mean SOTAC for problems it solved, excluding problems solved by all systems[6].

***u*-score** “accumulates for each problem solved by a strategy the reciprocal of the number of strategies which solve that problem” [13]. Here u stands for *u*niquely solved and this is meant to be a measure of how unique the solution is e.g. it is 1 if a single strategy solves the problem.

Biggest Lead, rewards the solver that *won by the most* (introduced to SMT-COMP in 2019). Motivated by the observation that many “wins” are quite close, this comparison aims to identify the solver that creates the largest gap between itself and the closest runner-up.

[5] failure is in principle possible but rarely seen, mediocrity rephrased as qualified success

[6] recently updated slightly after discussion with the community noted that considering problems solved by all systems is not useful, see http://www.tptp.org/CASC/J10/Proceedings.pdf

3 Undesired Problems and Problematic Desires

As mentioned in the abstract, the issue of evaluation is not a new issue and continues to be the subject of much discussion, but it shouldn't be. We would rather not have to think too hard about ATP evaluation and would rather be spending this time developing new and exciting theorem provers. Therefore, to help ourselves and similar researchers, we'd like to sort the matter out once and for all. Below, in no particular order, we enumerate some problems encountered when evaluating ATPs and indulge in some wishful thinking about the characteristics enjoyed by the future evaluation section of our dreams. We suggest directions towards this future evaluation approach in the next section but don't claim to have the answer yet.

Simplicity. The typical approach has the benefit of total simplicity. It is easy to re-implement, quite tolerant of systematic error, and easy to review. Any replacement must be (almost) as simple so as to retain these characteristics and gain acceptance in the community. We must remember that "worse is better" [10]: flawed-yet-simple systems have better survival characteristics than complex, fully-correct systems. Simple methods are also likely to be fun and easy to use, characteristics that endear them to researchers and foster adoption.

Rapid approximate evaluation. In a similar vein, researchers would like to rapidly evaluate ATPs during development [18] but are discouraged from doing so by the time it requires. While full competition-style evaluation is likely to remain computationally-expensive, a fast approximation may suffice for development.

Energy consumption. ATP evaluation consumes considerable electricity, which presently comes at environmental cost. Researchers have an obligation to at least consider whether their work could continue with a reduced energy budget.

Incentives. Not everyone agrees on what the field of ATP development as a whole should aim towards [23]. Whatever you think, presumably one's chosen evaluation methods should incentivise progress towards one's research goals.

Dissemination of ideas. It is difficult at present to publish an idea without spending a significant amount of effort and CPU time evaluating it. The existence of highly-developed systems such as Vampire also make it difficult to justify early fundamental work, such as entirely-new approaches to reasoning.

The scrapheap. Some ideas are never investigated or published due to disappointing results. Some good directions that "do not turn out to be practically useful" may simply require a different experimental setting (benchmarks, time limit, etc.) to shine[7]. We remark here that even the totally useless can be beautiful or interesting!

Alternative progress. While it is probably too much to hope for to expect ATP researchers to submit, review and publish *bona fide* negative results, it would be interesting if evaluation methods could detect when one technique is subsumed by others, which also encourages a reduction in the size of ATP systems. Current methods do not typically achieve this.

Scalar evaluations. Other fields have the luxury of widely-used numerical measures of success, such as the GLUE benchmark [41] from natural language processing. In principle this is also possible for ATP systems, but no single measure is used comparatively in the same way.

[7] This is not a new problem, merely an old one reappearing in a new guise! One reported motivation for the creation of TPTP was observing good ideas abandoned due to poor results on very small problem sets.

Reproducible runs. Results differ from machine to machine, and even from run to run, even if they "should not". This is undesirable.

Magic numbers. Systems often implement parameterised algorithms. Manual tuning of these parameters to obtain fixed "magic numbers" that work well on certain problem sets is fragile and not especially interesting from a research standpoint, so evaluation methods should encourage exposing these as user options.

Aesthetics and Occam's Razor. Given two systems that are equally powerful but differing in complexity, clarity or aesthetics, most researchers have an innate sense that the simpler, clearer, more beautiful system is preferable. This is by no means a well-defined criterion, but nonetheless it is something to consider.

Reduction of experimental parameters. There is an abundance of experimental parameters that must be chosen, such as the benchmark set chosen and the time limit. This makes it impossible to compare systems directly[8] from literature and less likely that researchers coincide on their choices.

Unclear conclusions. A typical experiment only offers a very specific conclusion such as "Vampire's current portfolio mode can be augmented with feature F to prove X more problems while only losing Y problems on the supported fragment of a certain version of TPTP, in the context of a certain amount of time and memory on a certain machine". We would surely like to conclude something a little more concrete.

Community stimulus. We are reliably informed that competitions are designed to provide both evaluation and stimulation, in order that hard work and good systems are rewarded, but also to provide inspiration and discussion for others. We consider this one of the best aspects of the current approach, and one that must be retained regardless of evaluation.

4 Some Suggestions

Here we suggest some ideas that may help with the above. They may well be controversial, ill-considered, already done elsewhere, or simply not work, but we hope to hear about this in discussion at ARCADE!

Reproduction by emulation. Emulation software such as DOSBox[9] provide software implementations of instructions and peripheral responses for an emulated machine. Given this, it may be possible to emulate ATP runs such that sources of non-determinism such as CPU cache or disk access are eliminated, regardless of the host machine. We are not yet sure of the practical or performance implications of such a contraption.

Case study problems. Given the limitations of the typical approach (particularly with respect to portfolio modes), one way forward might be a community shift to demonstrating practicality by highlighting specific "case study" examples that can be solved (faster) using a new technique [23], leaving overall performance for competitions. This is easily-understood, traditional, and arguably more convincing than a large table of quantities such as the u-score. It also motivates the creation of difficult problems.

[8] For example, competitions often changing their time limits and benchmark selection approach between years can complicate comparison.

[9] https://www.dosbox.com/

Statistical estimation. By treating the evaluation problem not as an enormous space, but as a source of random configuration/problem samples, some insight may be gained. One could imagine conclusions such as "with 95% confidence, enabling feature F solves problems X% faster" (c.f. [11]). This also allows for incremental approximate evaluation.

Single-mode systems. A very simple way around the problem of evaluating portfolio modes is to forbid them! This is not a new idea: recent iterations of the SAT and SMT competitions explicitly rule out some kinds of portfolio system[10]. This encourages development of strong single-mode systems that may be later combined in separate portfolio systems.

The "Glucose hack" track. The SAT competition includes a division where entries must be minor modifications to the Glucose SAT solver [1]. The lack of similar competitions for e.g. first-order systems suggests we could copy the idea to some extent, although the diversity of first-order systems and calculi suggests we would need at least an "E hack", "iProver hack" and "leanCoP hack"! Perhaps the new PyRes teaching system [29] will offer a level playing field for saturation-based systems[11].

Evaluation by fast portfolio creation. As mentioned above, a possible solution to evaluate portfolios is to create a fresh portfolio with access to a new option, then compare the fresh portfolio to a baseline portfolio created without access. If done consistently, this provides convincing evidence that the new feature improves overall performance in the presence of portfolios. If general-purpose algorithm configuration tools and techniques (see later) prove sufficiently powerful and flexible, a standard technique could be prescribed to allow comparison. This could also allow competitions on unseen problems with automatically-tuned portfolios, removing this aspect from competition.

5 Good Thinking and Useful Tools

We consider some tools and techniques we think may be useful for future evaluation techniques.

The problem of running system/problem pairs in a reproducible fashion does not seem too difficult when compared to the other problems of evaluation. We note here excellent software such as StarExec [31] and the `runsolver` tool [28], but with emulation/monitoring tools such as `perf` [9], QEMU [5] and Valgrind [20] it may be possible to go further.

An effort is already underway to apply the SMAC [15] algorithm configuration tool to Vampire, perhaps improving on the existing script we use for this purpose. We will consider whether this technique can also be used for evaluating portfolio ATPs: there is existing work on ablation analysis [6] for similar domains that may prove interesting. We are also interested in work applying SMAC and other techniques to iProver [14]. Strategy-invention tools such as BliStr [39] and extremely-general techniques such as Bayesian optimisation (already widely-used for parameter optimisation with extremely expensive samples in machine learning [30]) may also prove useful in this endeavour.

The SAT and SMT communities continues to provide inspiration. Work by Nikolić [21] seems very promising for a statistical/sampling approach to evaluation. The Sparkle system [40] already aims to solve at least some of our problems in a SAT context.

[10] The difficultly comes in defining a portfolio system. Vampire may run 100s of complementary strategies using a range of different calculi, which may be more diverse then combining two different solvers.

[11] There is a tutorial at CADE in 2021 that encourages participants to hack PyRes and submit the result to CASC.

References

[1] Gilles Audemard and Laurent Simon. On the glucose SAT solver. *International Journal on Artificial Intelligence Tools*, 27(01):1840001, 2018.

[2] Clark Barrett, Leonardo De Moura, and Aaron Stump. SMT-COMP: Satisfiability modulo theories competition. In *International Conference on Computer Aided Verification*, pages 20–23. Springer, 2005.

[3] Clark Barrett, Aaron Stump, Cesare Tinelli, et al. The SMT-LIB standard: Version 2.0. In *Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, England)*, volume 13, page 14, 2010.

[4] Gilles Barthe, Renate Eilers, Pamina Georgiou, Bernhard Gleiss, Laura Kovács, and Matteo Maffei. Verifying relational properties using trace logic. In *2019 Formal Methods in Computer Aided Design (FMCAD)*, pages 170–178. IEEE, 2019.

[5] Fabrice Bellard. QEMU, a fast and portable dynamic translator. In *USENIX annual technical conference, FREENIX Track*, volume 41, page 46. California, USA, 2005.

[6] André Biedenkapp, Marius Lindauer, Katharina Eggensperger, Frank Hutter, Chris Fawcett, and Holger Hoos. Efficient parameter importance analysis via ablation with surrogates. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[7] Koen Claessen and Ann Lillieström. Automated inference of finite unsatisfiability. *Journal of Automated Reasoning*, 47(2):111–132, 2011.

[8] David R Cok, Aaron Stump, and Tjark Weber. The 2013 evaluation of SMT-COMP and SMT-LIB. *Journal of Automated Reasoning*, 55(1):61–90, 2015.

[9] Maria Dimakopoulou, Stéphane Eranian, Nectarios Koziris, and Nicholas Bambos. Reliable and efficient performance monitoring in Linux. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 396–408. IEEE, 2016.

[10] Richard Gabriel. The rise of "worse is better". *Lisp: Good News, Bad News, How to Win Big*, 2(5), 1991.

[11] M. Greiner and M. Schramm. A Probablistic Stopping Criterion for the Evaluation of Benchmarks. Technical Report I9638, Institut für Informatik, Technische Universität München, München, Germany, 1996.

[12] Andrew Healy, Rosemary Monahan, and James F Power. Evaluating the use of a general-purpose benchmark suite for domain-specific SMT-solving. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 1558–1561, 2016.

[13] Kryštof Hoder, Giles Reger, Martin Suda, and Andrei Voronkov. Selecting the selection. In *International Joint Conference on Automated Reasoning*, pages 313–329. Springer, 2016.

[14] Edvard K Holden and Konstantin Korovin. SMAC and XGBoost your theorem prover. In *Proc. 4th Conference on Artificial Intelligence and Theorem Proving (AITP 2019)*, pages 93–95, 2019.

[15] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

[16] Mikoláš Janota, Haniel Barbosa, Pascal Fontaine, and Andrew Reynolds. Fair and adventurous enumeration of quantifier instantiations. *arXiv preprint arXiv:2105.13700*, 2021.

[17] Michael Kinyon, Robert Veroff, and Petr Vojtěchovskỳ. Loops with abelian inner mapping groups: An application of automated deduction. In *Automated Reasoning and Mathematics*, pages 151–164. Springer, 2013.

[18] Qinghua Liu, Zishi Wu, Zihao Wang, and Geoff Sutcliffe. Evaluation of axiom selection techniques. In *PAAR+ SC^2@ IJCAR*, pages 63–75, 2020.

[19] Jan Łukasiewicz. The shortest axiom of the implicational calculus of propositions. In *Proceedings of the Royal Irish Academy. Section A: Mathematical and Physical Sciences*, volume 52, pages 25–33. JSTOR, 1948.

[20] Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. *ACM SIGPLAN notices*, 42(6):89–100, 2007.

[21] Mladen Nikolić. Statistical methodology for comparison of SAT solvers. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 209–222. Springer, 2010.

[22] Sebastian Rautila. Evaluating the 2015 SMT competition. Master's thesis, Uppsala University, 2017.

[23] Giles Reger. Boldly going where no prover has gone before. *arXiv preprint arXiv:1912.12958*, 2019.

[24] Giles Reger and Martin Riener. What is the point of an SMT-LIB problem? In *16th International Workshop on Satisfiability Modulo Theories*, 2018.

[25] Giles Reger, Martin Suda, and Andrei Voronkov. The challenges of evaluating a new feature in Vampire. In Laura Kovács and Andrei Voronkov, editors, *Proceedings of the 1st and 2nd Vampire Workshops*, volume 38 of *EPiC Series in Computing*, pages 70–74. EasyChair, 2016.

[26] Andrew Reynolds, Tim King, and Viktor Kuncak. Solving quantified linear arithmetic by counterexample-guided instantiation. *Formal Methods Syst. Des.*, 51(3):500–532, 2017.

[27] Alexandre Riazanov and Andrei Voronkov. Limited resource strategy in resolution theorem proving. *Journal of Symbolic Computation*, 36(1-2):101–115, 2003.

[28] Olivier Roussel. Controlling a solver execution with the runsolver tool. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(4):139–144, 2011.

[29] Stephan Schulz and Adam Pease. Teaching automated theorem proving by example: PyRes 1.2. In *International Joint Conference on Automated Reasoning*, pages 158–166. Springer, 2020.

[30] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

[31] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. Starexec: A cross-community infrastructure for logic solving. In *International joint conference on automated reasoning*, pages 367–373. Springer, 2014.

[32] Geoff Sutcliffe. The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, 43(4):337, 2009.

[33] Geoff Sutcliffe. The CADE ATP System Competition — CASC. *AI Magazine*, 37(2):99–101, 2016.

[34] Geoff Sutcliffe. The 10th IJCAR automated theorem proving system competition — CASC-J10. *AI Communications*, pages 1–15, 2021.

[35] Geoff Sutcliffe and Francis Jeffry Pelletier. System description: JGXYZ: An ATP system for gap and glut logics. *Lecture notes in computer science*, 2019.

[36] Geoff Sutcliffe and Christian Suttner. Evaluating general purpose automated theorem proving systems. *Artificial intelligence*, 131(1-2):39–54, 2001.

[37] Dmitry Tishkovsky, Renate A Schmidt, and Mohammad Khodadadi. The tableau prover generator MetTeL2. In *European Workshop on Logics in Artificial Intelligence*, pages 492–495. Springer, 2012.

[38] Josef Urban. MPTP 0.2: Design, implementation, and initial experiments. *Journal of Automated Reasoning*, 37(1-2):21–43, 2006.

[39] Josef Urban. BliStr: The blind strategymaker. *arXiv preprint arXiv:1301.2683*, 2013.

[40] Koen van der Blom, Chuan Luo, and Holger H Hoos. Sparkle: Towards automated algorithm configuration for everyone. *Configuration and Selection of Algorithms (COSEAL)*, 2019.

[41] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[42] Tjark Weber, Sylvain Conchon, David Déharbe, Matthias Heizmann, Aina Niemetz, and Giles Reger. The SMT competition 2015–2018. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):221–259, 2019.

[43] Christoph Wernhard and Wolfgang Bibel. Learning from Łukasiewicz and Meredith: Investigations into proof structures. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction: CADE-28*, LNCS (LNAI). Springer, 2021. (To appear, preprint (extended version): https://arxiv.org/abs/2104.13645).