CASC-30

CASC-30

CASC-30

CASC-30

Proceedings of CASC-30 – the CADE-30 ATP System Competition

Geoff Sutcliffe

University of Miami, USA

Abstract

The CADE ATP System Competition (CASC) is the annual evaluation of fully automatic, classical logic, Automated Theorem Proving (ATP) systems - the world championship for such systems. CASC-30 was the thirtieth competition in the CASC series. Twenty ATP systems competed in the various divisions. These proceedings present the competition design and information about the competing systems.

1 Introduction

The CADE ATP System Competition (CASC) [117] is the annual evaluation of fully automatic, classical logic, ATP systems – the world championship for such systems. One purpose of CASC is to provide a public evaluation of the relative capabilities of ATP systems. Additionally, CASC aims to stimulate ATP research, motivate development and implementation of robust ATP systems that can be easily and usefully deployed in applications, provide an inspiring environment for personal interaction between ATP researchers, and expose ATP systems within and beyond the ATP community. CASC evaluates the performance of the ATP systems in terms of the number of problems solved, the number of acceptable solutions (proofs and models) output, and the average time taken for problems solved, in the context of a bounded number of eligible problems and specified time limits.

CASC is held at each CADE (the International Conference on Automated Deduction) and IJCAR (the International Joint Conference on Automated Reasoning) conference – the major forums for the presentation of new research in all aspects of automated deduction. CASC-30 was held on 30th July 2025, as part of the 30th International Conference on Automated Deduction (CADE-30). in Nancy, France. It was the thirtieth competition in the CASC series; see [141, 147, 144, 86, 88, 140, 138, 139, 93, 95, 97, 99, 102, 104, 106, 108, 110, 112, 114, 146, 116, 119, 122, 125, 127, 132, 133, 134, 131] and the CASC web site `tptp.org/CASC`, for information about previous CASCs. CASC-30 was organized by Geoff Sutcliffe, and overseen by a panel consisting of Aart Middeldorp, Cláudia Nalon, and Tjark Weber. The competition was run on computers provided by the StarExec project [81] at the University of Miami. The CASC-30 web site provides access to all the resources used before, during, and after the event: `tptp.org/CASC/30`.

The design and organization of CASC has evolved over the years to a sophisticated state [141, 142, 137, 143, 84, 85, 87, 89, 90, 91, 92, 94, 96, 98, 101, 103, 105, 107, 109, 111, 113, 115, 118, 121, 123, 124, 126, 128, 129, 130]. Important changes for CASC-30 were (for readers already familiar with the general design of CASC):

- Proofs had to be output in TPTP, and had to pass a structural verification check.
- The EPR returned from hiatus.

The CASC rules, specifications, and deadlines are absolute. Only the panel has the right to make exceptions. It is assumed that all entrants have read the documentation related to the competition, and have complied with the competition rules. Non-compliance with the rules can

lead to disqualification. A "catch-all" rule is used to deal with any unforeseen circumstances: *No cheating is allowed*. The panel is allowed to disqualify entrants due to unfairness, and to adjust the competition rules in case of misuse.

These proceedings are organized as follows: Section 2 describes the competition divisions and the ATP systems that entered the various divisions. Sections 3 and 4 describe the competition infrastructure and the requirements for the ATP systems. Section 5 describes how the systems are evaluated. Sections 6 and 7 describe the practical steps for registering a system and checking that is has the required properties. Section 8 provides descriptions (written by the entrants) of the systems entered into CASC-30. Section 9 concludes.

A Tense Note: Attentive readers will notice changes between the present and past tenses in this paper. Many parts of CASC are established and stable – they are described in the present tense (the rules are the rules). Aspects that were particular to CASC-30 are described in the past tense so that they make sense when reading this after the event.

2 Divisions and Systems

CASC is divided into divisions according to problem and system characteristics, in a coarse version of the TPTP problem library's Specialist Problem Classes (SPCs) [145]. Each division uses problems that have certain logical, language, and syntactic characteristics, so that the systems that compete in a division are, in principle, able to attempt all the problems in the division. Some divisions are further divided into problem categories that make it possible to analyze, at a more fine-grained level, which systems work well for what types of problems. Table 1 catalogs the divisions and problem categories of CASC-30. The example problems can be viewed online at `tptp.org/cgi-bin/SeeTPTP?Category=Problems`. Sections 3.2, 3.3, and 3.4 explain what problems are eligible for use in each division and category.

Systems that cannot be entered into the competition divisions (e.g., the system requires special hardware, or the entrant is an organizer) can be entered into the demonstration division. The demonstration division uses the same problems as the competition divisions, and the entry specifies which competition divisions' problems are to be used.

Twenty ATP systems competed in the various divisions of CASC-30. The division winners from the previous CASC (CASC-J12) and the Prover9 1109a system are automatically entered into the corresponding demonstration divisions to provide benchmarks against which progress can be judged. The systems, the divisions in which they were entered, and their entrants, are listed in Table 2. A division acronym in *italics* indicates the system was in the demonstration division. System descriptions are in the competition proceedings [130] and on the CASC-30 web site.

3 Infrastructure

3.1 Computers

The StarExec computers used for the competition have:

- Two octa-core Intel(R) Xeon(R) E5-2667, 3.20GHz CPUs, without hyperthreading
- 256GiB memory
- The CentOS Linux release 7.4.1708 (Core) operating system, with Linux kernel 3.10.0-693.el7.x86_64.

Table 1: Divisions and Problem categories

Division	Problems	Problem categories
THF	**T**yped (monomorphic) **H**igher-order **F**orm theorems (axioms with a provable conjecture).	**TNE** – **T**HF with **N**o **E**quality, e.g., `NUM738^1`. **TEQ** – **T**HF with **EQ**uality, e.g., `SET171^3`.
TFA	**T**yped (monomorphic) **F**irst-order form theorems with **A**rithmetic (axioms with a provable conjecture).	**TFI** – **TF**A with only **I**nteger arithmetic, e.g., `DAT016_1`. **TFE** – **TF**A with only r**E**al arithmetic, e.g., `MSC022_2`.
TFN	**T**yped **F**irst-order form **N**on-theorems (axioms with a countersatisfiable conjecture, and satisfiable axiom sets without a conjecture) without arithmetic.	E.g., `COM002_20`.
FOF	**F**irst-**O**rder **F**orm theorems (axioms with a provable conjecture).	**FNE** – **F**OF with **N**o **E**quality, e.g., `COM003+1`. **FEQ** – **F**OF with **EQ**uality, e.g., `SEU147+3`.
EPR	**E**ffectively **PR**opositional clause normal form theorems and non-theorems (clause sets).	**EPU** – **EP**R **U**nsatisfiable clause sets e.g., `GEO079-1`. **EPS** – **EP**R **S**atisfiable clause sets e.g., `PUZ001-3`.
UEQ	**U**nit **EQ**uality theorems in clause normal form (unsatisfiable clause sets).	E.g., `RNG026-7`.
SLH	Typed (monomorphic) higher-order theorems (axioms with a provable conjecture), generated by Isabelle's **SL**edge**H**ammer system [57].	The problems were generated from sessions in Isabelle's Archive of Formal Proofs [12].
ICU	First-order theorems (axioms with a provable conjecture) provided by the entrants.	The problems supplied by an entrant were expected to be easy enough for that entrant's ATP system, but difficult for the other entrants' systems, i.e., each entrant is saying to the others: "I Challenge yoU!".

The StarExec computers used for CASC are the same as are publicly available to the TPTP community, which allows system developers to test and tune their systems in exactly the same environment as is used for the competition. There were 30 computers available for CASC-30, i.e., 60 CPUs.

One ATP system runs on one CPU at a time. StarExec uses Linux's `sched_setaffinity` to restrict each system run to a single CPU, and `setrlimit` to limit memory use to 128 GiB. This separation avoids contention that might affect performance measurements. Systems can use all the cores on the CPU, which can be advantageous in divisions where a wall clock time limit is used. StarExec copies the systems and problems to the compute nodes before starting execution, so that there are no network delays.

Table 2: The ATP systems and entrants

ATP System	Divisions	Entrant (Associates)	Entrant's Affiliation
Connect++ 0.6.1	FOF ICU	Sean Holden	University of Cambridge
CSE_E 1.7	FOF EPR UEQ ICU	Peiyao Liu (Baojie Rui, Yang Xu, Stephan Schulz, Jun Lui, Shuwei Chen)	Xihua University
CSI_Enigma 1.0.5	*FOF* EPR *UEQ* *ICU*	(Guoyan Zeng (Yang Xu, Hailin Guo, Peiyao Liu, Guanfeng Wu, Jan Jakubuv, Stephan Schulz, Jun Liu, Shuwei Chen, Feng Cao, Jian Zhong, Xingxing He, Peng Xu)	Southwest Jiaotong University
cvc5 1.3.0	*THF TFA* TFN *FOF* *SLH*	Andrew Reynolds (Cesare Tinelli, Clark Barrett, Lydia Kondylidou)	University of Iowa
Drodi 4.1.0	FOF EPR UEQ ICU	Oscar Contreras	Amateur Programmer
E 3.3.0	THF FOF EPR UEQ SLH ICU	Stephan Schulz	DHBW Stuttgart
hopCoP 0.1	FOF	Michael Rawson (Clemens Eisenhofer, Laura Kovács)	University of Southampton
iProver 3.9	*TFN*	CASC	CASC-J12 winner
iProver 3.9.4	TFA TFN FOF EPR UEQ ICU	Konstantin Korovin	University of Manchester
LastButNotLeast 0	FOF	Julie Cailler	University of Lorraine, CNRS, Inria, LORIA
Leo-III 1.7.19	THF *SLH*	Alexander Steen (Christoph Benzmüller)	University of Greifswald
LisaTT 0.9.1	FOF	Simon Guilloud (Sankalp Gambhir)	EPFL
Prover9 1109a	*FOF*	CASC (Bob Veroff, Bill McCune)	CASC fixed point
SPASS-SCL 0.1	*FOF* EPR	Christoph Weidenbach (Simon Schwarz, Yasmine Briefs, Lorenz Leutgeb, Fabian Wobito, Charlotte Weidenbach	Max Planck Institute for Informatics
Toma 0.7	UEQ	Teppei Saito (Nao Hirokawa)	Japan Advanced Institute Science and Technology
Twee 2.6.0	UEQ	Nick Smallbone	Chalmers University of Technology
Vampire 4.9	*THF TFA* *FOF* *UEQ SLH ICU*	CASC	CASC-J12 winner
Vampire 5.0	THF TFA TFN FOF EPR UEQ SLH ICU	Michael Rawson (Filip Bártek, Ahmed Bhayat, Robin Coutelier, Márton Hajdú, Matthias Hetzenberger, Petra Hozzová, Laura Kovács, Jakob Rath, Giles Reger, Martin Riener, Martin Suda, Johannes Schoisswohl, Andrei Voronkov)	University of Southampton
Zipperp'n 2.1.9999	THF FOF	Jasmin Blanchette (Alexander Bentkamp, Simon Cruanes, Petar Vukmirović)	Ludwig-Maximilians-Universität München

3.2 Problems for the TPTP-based Divisions

The problems for the THF, TFA, TFN, FOF, EPR, and UEQ divisions were taken from the Thousands of Problems for Theorem Provers (TPTP) problem library [120], v9.1.0. The TPTP version used for CASC is released after the competition has started, so that new problems in the release have not been seen by the entrants. The problems have to meet certain criteria to be eligible for use:

- The TPTP tags problems that are designed specifically to be suited or ill-suited to some ATP system, calculus, or control strategy as *biased* They are excluded from the compe-

tition.

- The problems must be syntactically non-propositional.
- The TPTP uses system performance data in the Thousands of Solutions from Theorem Provers (TSTP) solution library to compute problem difficulty ratings in the range 0.00 (easy) to 1.00 (unsolved) [145] – Problems with ratings in the range 0.21 to 0.99 are eligible – the upper bound of 0.99 excludes problems that cannot be solved by any system and thus don't differentiate between systems; the lower bound of 0.21 was chosen (many years ago, and it has worked successfully) to exclude problems that would be solved by most of the systems and thus don't differentiate between systems.

  Problems of lesser and greater ratings are made eligible if there are not enough problems with ratings in that range. In the CASC-30 TFN division 68 problems with rating 0.00 and 96 problems with rating 1.00 were made eligible, because there were only 47 eligible problems with rating 0.21 to 0.99 (there were no problems with rating 0.01 to 0.20). Similarly, in the CASC-30 EPS problem category 191 problems with rating 0.00, 2 problems with rating 0.14, and 12 problems with rating 1.00 were made eligible, because there were only 46 eligible problems with rating 0.21 to 0.99. The organizer considered making these additional problems eligible to be acceptably useful: solving easy problems would be encouraging for weaker systems, and solving hard problems would be encouraging for stronger systems.

  Systems can be submitted before the competition so that their performance data is used in computing the problem ratings – problems that are newly solved get a rating less than 1.00 and thus become eligible (until the rating drops below 0.21). The rating calculation also uses performance data from ATP systems that are not entered into the competition, which can produce ratings that make some problems eligible for selection but easy or unsolvable for the systems in the competition. Using problems that are solved by all or none of the competition systems does not affect the competition rankings, has the benefit of placing the systems' performances in the context of the state-of-the-art in ATP, but does reduce the differentiation between the systems in the competition.

In order to ensure that no system receives an advantage or disadvantage due to the specific presentation of the problems in the TPTP, the problems are preprocessed to

- strip out all comment lines (in particular, the problem header)
- randomly reorder the formulae (`include` directives are left before the formulae, and type declarations are left before the symbols' uses)
- randomly swap the arguments of associative connectives and randomly reverse implications
- randomly reverse equalities

The numbers of problems used in each division and problem category are constrained by the numbers of eligible problems, the numbers of systems entered in the divisions, the number of CPUs available, the time limits, and the time available for running the competition live in one conference day, i.e., in about 6 hours. The numbers of problems used are set within these constraints according to the judgement of the organizers. The problems used are randomly selected from the eligible problems based on a seed supplied by the competition panel:

- The selection is constrained so that no division or category contains an excessive number of very similar problems, according to the "very similar problems" (VSP) lists distributed with the TPTP problem library [86]: For each problem category in each division, if the category is going to use N problems and there are L VSP lists that have an intersection of

at least $N/(L+1)$ with the eligible problems for the category, then maximally $N/(L+1)$ problems are taken from each of those VSP lists.
- In order to combat excessive tuning towards problems that were already in the preceding released TPTP version, the selection is biased to select problems that are new in the TPTP version used until 50% of the problems in each problem category have been selected or there are no more new problems to select, after which random selection from old and new problems continues. The number of new problems used depends on how many new problems are eligible and the limitation on very similar problems.
- Problems with rating 0.21 to 0.99 are selected before problems with other ratings.

The problems are given to the ATP systems as files in TPTP format, with `include` directives, in increasing order of TPTP difficulty rating.

3.3 Problems for the SLH Division

For the SLH division of CASC-29, Isabelle's Sledgehammer system was used to generate 8400 problems that could be used, of which 1000 appropriately difficult problems were selected based on performance data [129, 134]. For the SLH division of CASC-J12, the same problem set was used, and 1000 problems not used in CASC-29 were selected. For the SLH division of CASC-30, the same problem set was used, and 1000 problems not used in CASC-29 or CASC-J12 were selected. This reuse of the problem set was announced in advance of CASC-30, so that developers could tune their systems using the CASC-29 and CASC-J12 problems. The problems are given in a roughly estimated increasing order of difficulty.

3.4 Problems for the ICU Division

For the ICU division each entrant had to submit 20 FOF theorems (axioms with a provable conjecture). The problems had to be provided in decreasing order of desired use in the division, i.e., probably from hardest to easiest for other systems. The problems had to all be different, as assessed by the competition organizer. At least five problems were taken from each entrant's submission, in the order specified, and the same total number of problems were taken from the CASC-J12 ICU problem set (avoiding duplicates). Reuse of the CASC-J12 problems allows for an assessment of progress. The problems were given in reverse order of desired use, so that the "easier" problems were used before "harder" ones. It was expected that each entrant would submit problems that are easy enough for that entrant's system, but difficult for the other entrants' systems, i.e., each entrant is saying to the others: "I Challenge yoU!".

3.5 Time Limits

In the THF, TFA, TFN, FOF, EPR, UEQ, and ICU based divisions a wall clock time limit is imposed for each problem. The minimal time limit for each problem is 120 s, except for the ICU division where the minimal time limit is 300 s. The maximal time limit for each problem is constrained by the same factors that constrain the numbers of problems that are used, taking into account the phenomenon that ATP systems solve most problems quickly and very few slowly. The time limit is chosen within the range allowed according to the judgement of the organizers, and is announced at the competition. No CPU time limits are imposed (so that it can be advantageous to use all the cores on the CPU).

In the SLH division a CPU time limit is imposed for each problem. The limit is between 15 s and 90 s, which is the range of CPU time that can be usefully allocated for a proof attempt

in the Sledgehammer context.[1] The time limit is chosen within the range allowed according to the judgement of the organizers, and is announced at the competition.

In the ICU division a wall clock time limit is imposed for each problem. The limit is between 300 s and 600 s, which is a range that gives the systems sufficient time (2400-4800 s CPU time on the octa-core CPUs) to attempt the difficult problems submitted. The time limit is chosen within the range allowed according to the judgement of the organizer, and is announced at the competition.

4 System Entry, Delivery, and Execution

Systems can be entered at only the division level, and can be entered into more than one division. A system that is not entered into a division is assumed to perform worse than the entered systems, for that type of problem – wimping out is not an option. Entering many similar versions of the same system is deprecated, and entrants can be required to limit the number of system versions that they enter. Systems that rely essentially on running other ATP systems without adding value are deprecated; such systems might be disallowed or moved to the demonstration division.

The ATP systems entered into CASC are delivered to the competition organizer as StarExec installation packages, which the organizer installs and tests on StarExec. Source code is delivered separately, under the trusting assumption that the installation package corresponds to the source code. After the competition all competition division systems' StarExec and source code packages are made available on the CASC web site. This allows anyone to use the systems on StarExec, and to examine the source code. An open source license is encouraged, to allow the systems to be freely used, modified, and shared. Many of the StarExec packages include statically linked binaries that provide further portability and longevity of the systems.

The ATP systems must be fully automatic. They are executed as black boxes, on one problem at a time. Any command line parameters have to be the same for all problems in each division. The ATP systems must be sound, and are tested for soundness by submitting non-theorems to the systems in the THF, TFA, FOF, EPR, UEQ, SLH, and ICU divisions, and theorems to the systems in the TFN and EPR divisions. Claiming to have found a proof of a non-theorem or a disproof of a theorem indicates unsoundness. If a system fails the soundness testing it must be repaired by the unsoundness repair deadline or be withdrawn.

5 System Evaluation

CASC ranks the ATP systems at only the division level. For each ATP system, for each problem, four items of data are recorded: whether or not the problem was solved, the CPU and wall clock times taken (as measured by StarExec's `runsolver` utility [71], and prepended to each line of the system's `stdout`), and whether or not a solution (proof or model) was output. The systems are ranked according to the number of problems solved with an acceptable solution output, except in the EPR division that is ranked according to the number of problems solved but not necessarily accompanied by a solution (systems that do output solutions are highlighted in the presentation of results). Ties are broken according to the average time taken over problems solved. Trophies are awarded to the competition divisions' winners. The demonstration division results are presented along with the competition divisions' results, but might not be comparable with those results. The demonstration division systems are not ranked.

[1] According to a personal communication from Jasmin Blanchette, and he should know.

The competition panel decides whether or not the systems' solutions are "acceptable".

- Models must be complete, documenting the domain, function maps, and predicate maps. The domain, function maps, and predicate maps may be specified by explicit ground lists (of mappings), or by any clear, terminating algorithm.

The criteria for "acceptable" proofs are stricter that in CASC-J12:

- Inference steps must be reasonably fine-grained.
- For proofs that use translations from one form to another, e.g., translation of FOF problems to CNF, the translations must be adequately documented.
- Proofs must be in TPTP format[2] [135]. TPTP formatting is checked using TPTP4X [?]. This can be done in SystemB4TSTP[3].
- Proofs must be structurally correct:
  - Annotated formulae in a proof must be uniquely named.
  - Proofs must be acyclic.
  - Proofs must have formulae from the problem as leaves, and end at the conjecture (for axiomatic proofs) or `$false` formula (for proofs by contradiction, e.g., CNF refutations).
  - Proofs that negate the conjecture must correctly annotate the step as `status(cth)` and have a a single parent with the role `conjecture`.
  - Proofs that make `assumption`s must propagate and eventually discharge the assumptions.
  - Proofs must show only relevant inference steps.

  Proof structure is checked using GDV[?]. This can be done in SystemOnTSTP[4].

In addition to the ranking data, other measures are made and presented in the results:

- The *state-of-the-art contribution* (SotAC) quantifies the unique abilities of each system. For each problem solved by a system, its SotAC for the problem is the fraction of systems that do not solve the problem. A system's overall SotAC is the average SotAC over the problems it solves but that are not solved by all the systems.
- The *efficiency* balances the number of problems solved with the time taken. It is the average solution rate over the problems solved multiplied by the fraction of problems solved (the solution rate for one problem is the reciprocal of the time taken to solve it). Efficiency is computed for both CPU time and wall clock time, to measure how efficiently the systems use one core and multiple cores respectively.
- The *core usage* measures the extent to which the systems take advantage of multiple cores. The raw core usage is the ratio of CPU time to wall clock time used, and the average core usage is over problems solved with a raw core usage greater than 1.0. The number of problems solved with a lesser raw core usage is also noted, because the ability to solve problems quickly before multi-core search is necessary is also of interest. The competition ran on octa-core computers, thus the maximal core usage was 8.0. While high core usage can be seen as a strength of an ATP system, the ability to solve problems quickly before multi-core search is started is also a strength - the number of such problems is simply the difference between the number solved and the number solved with core usage greater than 1.0.

[2] `tptp.org/UserDocs/QuickGuide/Derivations.html`
[3] `tptp.org/cgi-bin/SystemB4TPTP`
[4] `tptp.org/cgi-bin/SystemOnTSTP`

The demonstration division results are presented along with the competition divisions' results, but might not be comparable with those results. The demonstration division systems are not ranked.

At some time after the competition all high ranking systems in each division are tested over the entire TPTP. This provides a final check for soundness. If a system is found to be unsound during or after the competition, but before the competition report is published, and it cannot be shown that the unsoundness did not manifest itself in the competition, then the system is retrospectively disqualified. At some time during or after the competition, the solutions from the winners of divisions ranked by the numbers of solutions output are checked. Proofs are checked for structure by GDV, and models are checked by the panel. If any of the solutions are unacceptable then the victory is rescinded. All disqualifications and ranking changes are explained in the competition report.

6 System Registration

ATP systems must be registered for the competition using the CASC system registration form, by the registration deadline. For each system an entrant must be nominated to handle all issues (e.g., installation and execution difficulties) arising before, during, and after the competition. The nominated entrant must formally register for CASC. It is not necessary for entrants to physically attend the competition.

6.1 System Descriptions

A system description has to be provided for each ATP system, using the HTML schema supplied on the CASC web site. The schema has the following sections:

- Architecture. This section introduces the ATP system, and describes the calculus and inference rules used.
- Strategies. This section describes the search strategies used, why they are effective, and how they are selected for given problems. Any strategy tuning that is based on specific problems' characteristics must be clearly described (and justified in light of the tuning restrictions described in Section 7).
- Implementation. This section describes the implementation of the ATP system, including the programming language used, important internal data structures, and any special code libraries used. The availability of the system is also given here.
- Expected competition performance. This section makes some predictions about the performance of the ATP system for each of the divisions and categories in which it is competing.
- References.

The system description has to be emailed to the competition organizers by the system description deadline. The system descriptions form part of the competition proceedings (Section 8).

6.2 Sample Solutions

For systems in the divisions that require solution output, representative sample solutions must be emailed to the competition organizers by the sample solutions deadline. Use of the TPTP format for proofs and finite interpretations is encouraged. The competition panel decides whether or not each system's solutions are acceptable (see Section 5).

Proof/model samples are required as follows:

- THF and SLH: `SET014^4`
- TFA: `DAT013=1`
- TFN: `DAT335_2` and `SWW469_10`
- FOF and ICU : `SEU140+2`
- UEQ: `BOO001-1`

An explanation must be provided for any non-obvious features.

7 System Requirements

Entrants must ensure that their systems execute in the competition environment, and have the following properties. Entrants are advised to finalize their installation packages and check these properties well in advance of the system delivery deadline. This gives the competition organizers time to help resolve any difficulties encountered.

Execution, Soundness, and Completeness

- Systems must be fully automatic, i.e., all command line switches have to be the same for all problems in each division.
- Systems' performances must be reproducible by running the system again.
- Systems must be sound.
- Systems do not have to be complete in any sense, including calculus, search control, implementation, or resource requirements.
- All techniques used must be general purpose, and expected to extend usefully to new unseen problems. The precomputation and storage of information about individual problems that might appear in the competition, or their solutions, is not allowed. Strategies and strategy selection based on individual problems or their solutions are not allowed. If machine learning procedures are used to tune a system, the learning must ensure that sufficient generalization is obtained so that no there is no specialization to individual problems. The system description must explain any such tuning or training that has been done. The competition panel may disqualify any system that is deemed to be problem specific rather than general purpose.

Output

- All solution output must be to `stdout`.
- For each problem, the system must output a distinguished string indicating what solution has been found or that no conclusion has been reached. Systems must use the SZS ontology and standards [100] for this. For example

  ```
  SZS status Theorem for SYN075+1
  ```

  or

  ```
  SZS status GaveUp for SYN075+1
  ```

- When outputting a solution, the start and end of the solution must be delimited by distinguished strings. Systems must use the SZS ontology and standards for this. For example

```
SZS output start CNFRefutation for SYN075-1.p
  ...
SZS output end CNFRefutation for SYN075-1.p
```

The string specifying the problem status must be output before the start of a solution.

- Solutions may not have irrelevant output (e.g., from other threads running in parallel) interleaved in the solution.
- Use of the TPTP format for proofs [135] is required, and use of the TPTP format for interpretations [136] is encouraged.
- Proofs will be check for syntax and structure (see Section 5).

Resource Usage

- Systems must be interruptible by a `SIGXCPU` signal so that CPU time limits can be imposed, and interruptible by a `SIGALRM` signal so that wall clock time limits can be imposed. For systems that create multiple processes the signal is sent first to the process at the top of the hierarchy, then one second later to all processes in the hierarchy. The default action on receiving these signals is to exit (thus complying with the time limit, as required), but systems may catch the signals and exit of their own accord. If a system runs past a time limit this is noticed in the timing data, and the system is considered to have not solved the problem.
- If a system terminates of its own accord it may not leave any temporary or intermediate output files. If a system is terminated by a `SIGXCPU` or `SIGALRM` it may not leave any temporary or intermediate files anywhere other than in `/tmp`.
- For practical reasons excessive output from an ATP system is not allowed. A limit, dependent on the disk space available, is imposed on the amount of output that can be produced.

8 The ATP Systems

These system descriptions were written by the entrants.

8.1 ConnectPP 0.6.1

Sean Holden
University of Cambridge, United Kingdom

Architecture

Connect++ Version 0.6.1 is the current publicly-available release of the Connect++ prover for first-order logic, introduced in [39]. It is a connection prover using the same calculus and inference rules as leanCoP [56]. That is, it uses the connection caclulus with regularity and (optionally) with lemmas.

Strategies

The (default) proof search is essentially the same as that used by leanCoP Version 2.1. That is, it employs a search trying left branches of extensions first, with restricted backtracking as described in [56], and using the leftmost literal of the relevant clause for all inference rules. It does not attempt to exactly reproduce leanCoP's search order. When not using a schedule, command-line options allow many of these choices to be altered; for example allowing backtracking restriction for extensions while still fully exploring left branches, or other modification of the backtracking restrictions. If run with its default schedule it uses one similar to that of leanCoP Version 2.1, including the various applications of definitional clause conversion. Alternatively it can read and apply arbitrary schedules if desired. At present Connect++ does not attempt to tune its proof search based on the characteristics of individual problems.

Implementation

Connect++ is implemented in C++ - minimally the 2017 standard - and built using cmake. Libraries from the Boost collection are used for parsing, hashing, some random number generation, and processing of command-line options. The system has a built-in proof checker for verifying its own output, but also includes a standalone checker implemented in SWI Prolog.

As substitutions need to apply to an entire proof tree the system only represents each variable once and shares the representation, simultaneously maintaining a stack of substitutions making removal of substitutions under backtracking trivial. It also creates subterms only once and shares them; these are indexed allowing constant-time lookup, and nothing is ever removed from the index, meaning that if a term is constructed again after its initial construction no new memory allocation takes place and the term itself is obtained in constant time. At the same time, fresh copies of variables are recycled under backtracking - these two design choices appear to interact very effectively, as the recycling of the variables seems to make it quite likely that subterms already in the index can be reused.

By default a standard recursive unification algorithm is used, but a polynomial-time version is optional.

If a schedule is used, it is assumed that different approaches to definitional clause conversion may be needed - typically all clauses, conjecture clauses only, or no clauses. As these choices

can lead to different matrices, and the conversion itself can be expensive, the system stores and switches between the different matrices rather than converting multiple times.

As the system was developed with two guiding aims - to provide a clear implementation easily modified by others, somewhat in the spirit of MiniSAT [27], and to support experiments in machine learning for guiding the proof search - the implementation avoids the use of direct recusion in favour of a pair of stacks and an iterative implementation based on these, as described in [39]. This allows complete and arbitrary control of backtracking restriction and other modifications to the proof search using typically quite simple modifications to the code.

The source and documentation are available at

`http://www.cl.cam.ac.uk/~sbh11/connect++.html`

Expected Competition Performance

The system remains at an early stage of development and is currently undergoing systematic profiling and improvement. It is not expected at this stage to be competitive with the state-of-the-art, but is expected to be a distinct improvement on last year's version 0.6.0.

8.2 CSE_E 1.7

Peiyao Liu
Xihua University, China

Architecture

CSE_E 1.7 is an automated theorem prover for first-order logic by combining CSE 1.8 and E 2.6, where CSE 1.8 is based on the Contradiction Separation Based Dynamic Multi-Clause Synergized Automated Deduction (S-CS) [155], and E is mainly based on superposition. The combination mechanism is like this: E and CSE are applied to the given problem sequentially. If either prover solves the problem, then the proof process completes. If neither CSE nor E can solve the problem, some inferred clauses with no more than two literals, especially unit clauses, by CSE will be fed to E as lemmas, along with the original clauses, for further proof search. This kind of combination is expected to take advantage of both CSE and E, and produce a better performance. Concretely, CSE is able to generate a good number of unit clauses, based on the fact that unit clauses are helpful for proof search and equality handling. On the other hand, E has a good ability on equality handling.

Strategies

The primary enhancement in version CSE_E 1.7, relative to its predecessors, lies in the adoption of a parallelization strategy. The core concept of this strategy is to leverage CSE's inherent capability to efficiently derive a large number of unit clauses. First, CSE is invoked on a single processor core to generate a set of unit clauses, which serves as the initial lemma set. Subsequently, multiple partitioning schemes are applied to divide this lemma set into N distinct subsets. N processor cores are then activated, with each core executing the theorem prover E to independently check the unsatisfiability of the combination of the original CNF clause set and its assigned subset of lemmas.

Implementation

CSE_E 1.7 is implemented mainly in C++. The job dispatch between CSE and E is implemented in C++.

Expected Competition Performance

We expect CSE_E 1.7 to solve some hard problems that E cannot solve and have a satisfying performance.

Acknowledgements

Development of CSE_E 1.7 has been supported by the Key Project of Sichuan Science and Technology Innovation and Entrepreneurship Seeding Program (Grant No. 2024JDRC0084). Stephan Schulz for his kind permission on using his E prover that makes CSE_E possible.

8.3 CSI_Enigma 1.0.5

Guoyan Zeng
Southwest Jiaotong University, China

Architecture

CSI_Enigma 1.0.5 is an automated theorem prover for first-order logic, combining CSI 1.1 and Enigma, where CSI 1.1 is a multi-layer inverse and parallel prover based on the Contradiction Separation Based Dynamic Multi-Clause Synergized Automated Deduction (S-CS) [155], and Enigma is an efficient implementation of learning-based guidance for given clause selection in saturation-based automated theorem provers. This kind of combination is expected to take advantage of both CSI and Enigma, and produce a better performance. Concretely, CSI is able to construct a good number of unit clauses, based on the fact that unit clauses are helpful for proof search and equality handling. On the other hand, Enigma has a good ability.

Strategies

The CSI part of CSI_Enigma 1.0.5 takes almost the same strategies as in that in CSE 1.6 standalone, e.g., clause/literal selection, strategy selection, and CSC strategy. The only difference is that equality handling strategies of CSE part of the combined system are blocked. The main new strategies for the combined systems are:

- Complementary ratio strategy. A measure and calculation method of complementary relation between two clauses, which can guide the selection of clauses to participate in deduction and plan the deduction path effectively.
- Portfolio strategy clause selection schemes for different run stages.

Implementation

CSI_Enigma 1.0.5 is implemented mainly in C++.

Expected Competition Performance

We expect CSI_Enigma 1.0.5 to solve some hard problems that CSI_E cannot solve and have a satisfying performance. CSI_Enigma 1.0.5 can solve Satisfiable and Unsatisfiable problems.

Acknowledgement
Development of CSI_Enigma 1.0.5 has been partially supported by the National Natural Science Foundation of China (NSFC) (Grant No. 62106206, 62206227).

8.4 cvc5 1.3.0

Andrew Reynolds
University of Iowa, USA

Architecture

cvc5 [4] is the successor of CVC4 [7]. It is an SMT solver based on the CDCL(T) architecture [54] that includes built-in support for many theories, including linear arithmetic, arrays, bit vectors, datatypes, finite sets and strings. It incorporates approaches for handling universally quantified formulas. For problems involving free function and predicate symbols, cvc5 primarily uses heuristic approaches based on conflict-based instantiation and E-matching for theorems, and finite model finding approaches for non-theorems.

Like other SMT solvers, cvc5 treats quantified formulas using a two-tiered approach. First, quantified formulas are replaced by fresh Boolean predicates and the ground theory solver(s) are used in conjunction with the underlying SAT solver to determine satisfiability. If the problem is unsatisfiable at the ground level, then the solver answers "unsatisfiable". Otherwise, the quantifier instantiation module is invoked, and will either add instances of quantified formulas to the problem, answer "satisfiable", or return unknown. Finite model finding in cvc5 targets problems containing background theories whose quantification is limited to finite and uninterpreted sorts. In finite model finding mode, cvc5 uses a ground theory of finite cardinality constraints that minimizes the number of ground equivalence classes, as described in [69]. When the problem is satisfiable at the ground level, a candidate model is constructed that contains complete interpretations for all predicate and function symbols. It then adds instances of quantified formulas that are in conflict with the candidate model, as described in [69]. If no instances are added, it reports "satisfiable".

cvc5 has native support for problems in higher-order logic, as described in [6]. It uses a pragmatic approach for HOL, where lambdas are eliminated eagerly via lambda lifting. The approach extends the theory solver for quantifier-free uninterpreted functions (UF) and E-matching. For the former, the theory solver for UF in cvc5 now handles equalities between functions using an extensionality inference. Partial applications of functions are handle using a (lazy) applicative encoding where some function applications are equated to the applicative encoding. For the latter, several of the data structures for E-matching have been modified to incorporate matching in the presence of equalities between functions, function variables, and partial function applications.

cvc5 integrates MBQI-Enum [44], a new instantiation strategy that combines model-based quantifier instantiation [29] with syntax-guided synthesis [67]. This approach improves HOL support by generating instantiations that include lambda-terms, identity functions, and terms

with uninterpreted symbols. Additionally, cvc5 incorporates an extended version of MBQI-Enum that supports Hilbert's choice operator. This extension improves cvc5's success on higher-order logic benchmarks by generating instantiations that involve choice terms.

Strategies

For handling theorems, cvc5 primarily uses conflict-based quantifier instantiation [68, 5], enumerative instantiation [66] and E-matching. cvc5 uses a handful of orthogonal trigger selection strategies for E-matching, and several orthogonal ordering heuristics for enumerative instantiation. For handling non-theorems, cvc5 primarily uses finite model finding techniques. Since cvc5 with finite model finding is also capable of establishing unsatisfiability, it is used as a strategy for theorems as well.

Implementation

cvc5 is implemented in C++. The code is available from

`https://github.com/cvc5/cvc5`

Expected Competition Performance

This year, we have added several new strategies for the THF division, including an extension of MBQI-Enum [44] for reasoning about Hilbert's choice operator, which significantly improves cvc5's performance on benchmarks involving higher-order quantifiers and choice terms. We continue to rely on a conversion from TPTP to smt2 as a preprocess step. Overall, our performance should mostly be the same as last year on the other divisions apart from THF.

8.5 Drodi 4.1.0

Oscar Contreras
Amateur Programmer, Spain

Architecture

Drodi 4.1.0 is a very basic and lightweight automated theorem prover. It implements the following main features:

- Ordered resolution and equality paramodulation inferences as well as demodulation and some other standard simplifications.
- A basic implementation of clausal normal form conversion as in [55].
- AVATAR architecture with a SAT solver [148].
- Limited Resource Strategy [70].
- Discrimination trees.
- KBO, non recursive and lexicographic reduction orderings. KBO has been rewriten using the polynomial-time algorithm in [18].
- Literal selection including lookahead as in [35].
- SInE distance for clauses and symbols as in [82].
- Layered clause selection as in [30].

- Stochastic strategy inspired in [83].
- Drodi produces a (hopefully) verifiable proof in TPTP format.

Strategies

Drodi has a fair number of selectable strategies including but not limited to the following:

- Otter, Discount and Limited Resource Strategy [70] saturation algorithms.
- A basic implementation of AVATAR architecture [148].
- Several literal and term reduction orderings.
- Several literal selection options [35].
- Several layered clause selection heuristics with adjustable selection ratios [30].
- Classical clause relevancy pruning.
- Drodi V4 has new strategy portfolio inspired (but not exactly equal than) by [19]. The strategies have now unequal time slices and the set of strategies used depends on the problem features detected during the preprocessing.
- Some strategies are run a second time with a previously applied randomization to the problem [83].
- Drodi can generate learning data from successful proofs and use the data to guide clause selection strategy. It is based in the enhanced ENIGMA method. However, unlike ENIGMA, the learning data is completely general and can be used with any kind of problems. This generality allows the use of the same learning data in both FOF and UEQ CASC competition divisions. The learning data is generated over a set of TPTP problems before the CASC competition using built-in Drodi functions that include a L2 Support Vector Machine. Drodi integrated learning functions are a generalization of ENIGMA [41, 42]. Literals polarity, equality, skolem and variable occurrences are stored in clause feature vectors. Unlike ENIGMA, instead of storing the specific functions and predicates themselves only the SinE distance and arity of functions and non equality predicates are stored in clause feature vectors with different features assigned to predicates and functions.

Implementation

Drodi is implemented in C. It includes discrimination trees and hashing indexing. All the code is original, without special code libraries or code taken from other sources.

Expected Competition Performance

Drodi 4.1.0 solves around 5Anyway we expect that performance will be similar to last year's version.

8.6 E 3.3.0

Stephan Schulz
DHBW Stuttgart, Germany

Architecture

E [73, 77, 78] is a purely equational theorem prover for many-sorted first-order logic with equality, and for monomorphic higher-order logic. It consists of an (optional) clausifier for preprocessing full first-order formulae into clausal form, and a saturation algorithm implementing an instance of the superposition calculus with negative literal selection and a number of redundancy elimination techniques, optionally with higher-order extensions [149, 151]. E is based on the DISCOUNT-loop variant of the given-clause algorithm, i.e., a strict separation of active and passive facts. No special rules for non-equational literals have been implemented. Resolution is effectively simulated by paramodulation and equality resolution. As of E 2.1, PicoSAT [10] can be used to periodically check the (on-the-fly grounded) proof state for propositional unsatisfiability.

Strategies

Proof search in E is primarily controlled by a literal selection strategy, a clause selection heuristic, and a simplification ordering. The prover supports a large number of pre-programmed literal selection strategies. Clause selection heuristics can be constructed on the fly by combining various parameterized primitive evaluation functions, or can be selected from a set of predefined heuristics. Clause evaluation heuristics are based on symbol-counting, but also take other clause properties into account. In particular, the search can prefer clauses from the set of support, or containing many symbols also present in the goal. Supported term orderings are several parameterized instances of Knuth-Bendix-Ordering (KBO) and Lexicographic Path Ordering (LPO), which can be lifted in different ways to literal orderings.

For CASC-30, E implements a two-stage multi-core strategy-scheduling automatic mode. The total CPU time available is broken into several (unequal) time slices. For each time slice, the problem is classified into one of several classes, based on a number of simple features (number of clauses, maximal symbol arity, presence of equality, presence of non-unit and non-Horn clauses, possibly presence of certain axiom patterns, ...). For each class, a schedule of strategies is greedily constructed from experimental data as follows: The first strategy assigned to a schedule is the the one that solves the most problems from this class in the first time slice. Each subsequent strategy is selected based on the number of solutions on problems not already solved by a preceding strategy. The strategies are then scheduled onto the available cores and run in parallel.

About 140 different strategies have been thoroughly evaluated on all untyped first-order problems from TPTP 7.3.0. We have also explored some parts of the heuristic parameter space with a short time limit of 5 seconds. This allowed us to test about 650 strategies on all TPTP problems, and an extra 7000 strategies on UEQ problems from TPTP 7.2.0. About 100 of these strategies are used in the automatic mode, and about 450 are used in at least one schedule.

Implementation

E is build around perfectly shared terms, i.e., each distinct term is only represented once in a term bank. The whole set of terms thus consists of a number of interconnected directed acyclic graphs. Term memory is managed by a simple mark-and-sweep garbage collector. Unconditional (forward) rewriting using unit clauses is implemented using perfect discrimination trees with size and age constraints. Whenever a possible simplification is detected, it is added as a rewrite link in the term bank. As a result, not only terms, but also rewrite steps are shared. Subsumption and contextual literal cutting (also known as subsumption resolution) is supported using feature vector indexing [76]. Superposition and backward rewriting use fingerprint indexing [75], a new technique combining ideas from feature vector indexing and path indexing. Finally, LPO and KBO are implemented using the elegant and efficient algorithms developed by Bernd Löchner in [49, 50]. The prover and additional information are available at

`https://www.eprover.org`

Expected Competition Performance

E 3.3.0 is basically E 3.2.0 with bug fixes. We have not yet been able to evaluate and integrate new search strategies making full use of all new features. As a result, we expect performance to be similar to last year's version. The system is expected to perform well in most proof classes, but will at best complement top systems in the disproof classes.

8.7 hopCoP 0.1

Michael Rawson
University of Souhhampton, United Kingdom

Architecture

hopCoP is a system in the connection family of theorem provers that includes SETHEO, leanCoP, and this year's Connect++. These systems all attempt to produce a proof object from beginning to end, backtracking one step on failure. However, hopCoP differs by analysing the position that caused the failure and *learning* (in the sense of CDCL rather than gradient descent) a reason for failure. This reason is recorded (and used to avoid similar failures in future), then the system *backjumps*, undoing multiple decisions at once until the reasons for failure are corrected [58] [REK25].

Strategies

hopCoP does not employ strategy scheduling. Search begins at the conjecture (or positive clauses, failing that), and continues with some pseudo-random aspects until either a proof is found or the system determines that there is not a proof within a certain tableau depth. In the latter case, the depth is increased and search tried again. Multiple cores are used to search multiple iterative deepening levels at once, a trick learned from Vampire's finite model builder.

Implementation

The implementation is imperative in nature and the skeleton was lifted from SATCoP [59], but it has been adapted considerably to support a new core search routine of around 500 lines. The source is available at:

`http://github.com/MichaelRawson/hopcop`

and should be somewhat hackable.

Expected Competition Performance

Having gone to some effort to improve the behaviour of connection-driven proof search, we hope to prove more theorems than the closest-related system leanCoP would have done. However, we expect fierce competition from Connect++ and to be roundly beaten by state-of-the art systems.

8.8 iProver 3.9

Konstantin Korovin
University of Manchester, United Kingdom

Architecture

iProver [45, 23] is a theorem prover for quantified first-order logic with theories. iProver interleaves instantiation calculus Inst-Gen [46, 45, 28] with ordered resolution and superposition calculi [23]. iProver approximates first-order clauses using propositional abstractions that are solved using MiniSAT [27] or Z3 [21] and refined using model-guided instantiations. iProver also implements a general abstraction-refinement framework for under-and over-approximations of first-order clauses [33, 34]. First-order clauses are exchanged between calculi during the proof search.

Recent features in iProver include:

- Ground joinability and connectedness in the superposition calculus [25].
- Support for quantified reasoning with arithmetic and arrays.
- AC joinability and AC normalisation [24].
- Superposition calculus with simplifications including: demodulation, light normalisation, subsumption, subsumption resolution and global subsumption. iProver's simplification set up [23] is tunable via command line options and generalises common architectures such as Discount or Otter.
- HOS-ML framework for learning heuristics using combination of hyper-parameter optimisation and dynamic clustering together with schedule optimisation using constraint solving [38, 37]

Strategies

iProver has around 100 options to control the proof search including options for literal selection, passive clause selection, frequency of calling the SAT/SMT solvers, simplifications, and options for combination of instantiation with resolution and superposition. For the competition

HOS-ML [38] was used to build a multi-core schedule from heuristics learnt over a sample of FOF problems. Some theories and fragments are recognised such as EPR, UEQ, Horn, groups, rings and lattices for which options are adapted accordingly.

Implementation

iProver is implemented in OCaml. For the ground reasoning uses MiniSat [27] and Z3 [21]. iProver accepts FOF, TFF and CNF formats. Vampire [48, 62] and E prover [77] are used for proof-producing clausification of FOF/TFF problems. Vampire is also used for SInE axiom selection [36] in the LTB division and for theory axioms in the TFA division. iProver is available at:

`https://gitlab.com/korovin/iprover`

Expected Competition Performance

iProver 3.9 is the CASC-J12 TFN winner.

8.9 iProver 3.9.3

Konstantin Korovin
The University of Manchester, United Kingdom

Architecture

iProver [45, 23] is a theorem prover for quantified first-order logic with theories. iProver interleaves instantiation calculus Inst-Gen [46, 45, 28] with ordered resolution and superposition calculi [23]. iProver approximates first-order clauses using propositional abstractions that are solved using MiniSAT [27] or Z3 [21] and refined using model-guided instantiations. iProver also implements a general abstraction-refinement framework for under-and over-approximations of first-order clauses [33, 34]. First-order clauses are exchanged between calculi during the proof search.

Recent features in iProver include:

- Ground joinability and connectedness in the superposition calculus [25].
- Support for quantified reasoning with arithmetic and arrays.
- AC joinability and AC normalisation [24].
- Superposition calculus with simplifications including: demodulation, light normalisation, subsumption, subsumption resolution and global subsumption. iProver's simplification set up [23] is tunable via command line options and generalises common architectures such as Discount or Otter.
- HOS-ML framework for learning heuristics using combination of hyper-parameter optimisation and dynamic clustering together with schedule optimisation using constraint solving [38, 37]

Strategies

iProver has around 100 options to control the proof search including options for literal selection, passive clause selection, frequency of calling the SAT/SMT solvers, simplifications, and

options for combination of instantiation with resolution and superposition. For the competition HOS-ML [38] was used to build a multi-core schedule from heuristics learnt over a sample of FOF problems. Some theories and fragments are recognised such as EPR, UEQ, Horn, groups, rings and lattices for which options are adapted accordingly.

Implementation

iProver is implemented in OCaml. For the ground reasoning uses MiniSat [27] and Z3 [21]. iProver accepts FOF, TFF and CNF formats. Vampire [48, 62] and E prover [77] are used for proof-producing clausification of FOF/TFF problems. Vampire is also used for SInE axiom selection [36] of theory axioms in the TFA division. iProver is available at:

`https://gitlab.com/korovin/iprover`

Expected Competition Performance

iProver is regularly in the top 3 in FOF, EPR, UEQ, TFN, TFA. We expect an improved performance compared to the previous year due to efficiency improvements and heuristic optimization.

8.10 LastButNotLeast 0

Julie Cailler
University of Lorraine, CNRS, Inria, LORIA, Nancy, France

Architecture

LastButNotLeast: The fastest way to give up - now with 0 bugs! Probably the fastest prover in the competition:

- Runs flawlessly.
- Achieves absolutely nothing.
- 100
- Zero proof, infinite potential.
- Proudly earns you a cool CASC T-shirt.

Unfortunately, designing this prover may force you to share a meal with very weird people - but hey, that's the price of true innovation.

Strategies

The strategy is simple: always return ¡code¿GaveUp¡/code¿. It's fast, reliable, and guarantees we never solve anything - by design.

Implementation

```
#!/usr/bin/env python

print(''% LastButNotLeast: The fastest way to give up!'')
print(''% For best results, do not expect results.'')
print(''% SZS status GaveUp'')
print(''% It's not a bug - it's a philosophical stance.'')
print(''% Thanks for trying LastButNotLeast :)'')
```

Expected Competition Performance

The prover is expected to come last, and any other result would be genuinely surprising - possibly even concerning. In fact, LastButNotLeast proudly exists to ensure that no other prover - no matter how underwhelming - has to carry the burden of being last.

8.11 Leo-III 1.7.19

Alexander Steen
University of Greifswald, Germany

Architecture

Leo-III [80], the successor of LEO-II [9], is a higher-order ATP system based on extensional higher-order paramodulation with inference restrictions using a higher-order term ordering. The calculus contains dedicated extensionality rules and is augmented with equational simplification routines that have their intellectual roots in first-order superposition-based theorem proving. The saturation algorithm is a variant of the given clause loop procedure inspired by the first-order ATP system E.

Leo-III cooperates with external first-order ATPs that are called asynchronously during proof search; a focus is on cooperation with systems that support typed first-order (TFF) input. For this year's CASC E [73, 77] is used as external system. However, cooperation is in general not limited to first-order systems. Further TPTP/TSTP-compliant external systems (such as higher-order ATPs or counter model generators) may be included using simple command-line arguments. If the saturation procedure loop (or one of the external provers) finds a proof, the system stops, generates the proof certificate and returns the result.

Strategies

Leo-III comes with several configuration parameters that influence its proof search by applying different heuristics and/or restricting inferences. These parameters can be chosen manually by the user on start-up. Leo-III implements a very naive time slicing approach in which at most three manually fixed parameter configurations are used, one after each other. In practice, this hardly ever happens and Leo-III will just run with its default parameter setting.

Implementation

Leo-III utilizes and instantiates the associated ¡span class="leopard"¿ LeoPARD¡/span¿ system platform [154] for higher-order (HO) deduction systems implemented in Scala (currently using Scala 2.13 and running on a JVM with Java >= 8). The prover makes use of LeoPARD's data structures and implements its own reasoning logic on top. A hand-crafted parser is provided that supports all TPTP syntax dialects. It converts its produced concrete syntax tree to an internal TPTP AST data structure which is then transformed into polymorphically typed lambda terms. As of version 1.1, Leo-III supports all common TPTP dialects (CNF, FOF, TFF, THF) as well as their polymorphic variants [13, 43]. Since version 1.6.X (X >= 0) Leo-III also accepts non-classical problem input represented in non-classical TPTP[5].

The term data structure of Leo-III uses a polymorphically typed spine term representation augmented with explicit substitutions and De Bruijn-indices. Furthermore, terms are perfectly shared during proof search, permitting constant-time equality checks between alpha-equivalent terms.

Leo-III's saturation procedure may at any point invoke external reasoning tools. To that end, Leo-III includes an encoding module which translates (polymorphic) higher-order clauses to polymorphic and monomorphic typed first-order clauses, whichever is supported by the external system. While LEO-II relied on cooperation with untyped first-order provers, Leo-III exploits the native type support in first-order provers (TFF logic) for removing clutter during translation and, in turn, higher effectivity of external cooperation.

Leo-III is available on GitHub:

`https://github.com/leoprover/Leo-III`

Expected Competition Performance

Version 1.7.19 is, for all intents and purposes of CASC, equivalent to the version from the previous year except that some minor bugs were fixed, and the support for reasoning in various quantified non-classical logics (not relevant to CASC) has been improved. We do not expect Leo-III to be strongly competitive against more recent higher-order provers, as Leo-III does not implement several standard features of effective systems (including time slicing and proper axiom selection).

8.12 LisaTT 0.9.1

Simon Guilloud
EPFL, Switzerland

Architecture

Lisa is a proof assistant based on first order logic and set theory. Lisa offers a unified proof script, tactic and implementation by implementing an expressive DSL directly within the Host language Scala. Lisa also foundationally relies on Orthologic, a generalization of Boolean Algebra without the distributivity law that admits quadratic time normalization and validity checking. This makes proofs shorter and simpler, and crucially improve on automation, in particular in Lisa's Tableau tactic.

[5] `tptp.org/NonClassicalLogic`

Strategies

Lisa's Tableau tactic, is, unsurprisingly, a tableau-based solver. It's main strategy and differentiating point is the use of Orthologic, which simplifies the initial input. We know of classes that Orthologic solve quickly but that are difficult for ATP's and Sat solver, though this tends to be more significant. Beyond this, the implementation is standard, partially inspired from the Zenon prover for branch elimination.

Implementation

The implementation (in Scala) is functional and produces low level proofs that are accepted by Lisa's kernel. It is available from:

`https://github.com/epfl-lara/lisa/tree/main`

Expected Competition Performance

We do not expect great performances from the system, as it is the result of two weeks of work of one PhD student and the heuristic are rather simple. Nonetheless, it could in theory prove problems that no other ATP is expected to solve, thanks to orthologic normalization (we do not know if such problems exist in the TPTP library).

8.13 Prover9 1109a

Bob Veroff on behalf of William McCune
University of New Mexico, USA

Architecture

Prover9, Version 2009-11A, is a resolution/paramodulation prover for first-order logic with equality. Its overall architecture is very similar to that of Otter-3.3 [53]. It uses the "given clause algorithm", in which not-yet-given clauses are available for rewriting and for other inference operations (sometimes called the "Otter loop").

Prover9 has available positive ordered (and nonordered) resolution and paramodulation, negative ordered (and nonordered) resolution, factoring, positive and negative hyperresolution, UR-resolution, and demodulation (term rewriting). Terms can be ordered with LPO, RPO, or KBO. Selection of the "given clause" is by an age-weight ratio.

Proofs can be given at two levels of detail: (1) standard, in which each line of the proof is a stored clause with detailed justification, and (2) expanded, with a separate line for each operation. When FOF problems are input, proof of transformation to clauses is not given.

Completeness is not guaranteed, so termination does not indicate satisfiability.

Strategies

Prover9 has available many strategies; the following statements apply to CASC.

Given a problem, Prover9 adjusts its inference rules and strategy according to syntactic properties of the input clauses such as the presence of equality and non-Horn clauses. Prover9 also does some preprocessing, for example, to eliminate predicates.

For CASC Prover9 uses KBO to order terms for demodulation and for the inference rules, with a simple rule for determining symbol precedence.

For the FOF problems, a preprocessing step attempts to reduce the problem to independent subproblems by a miniscope transformation; if the problem reduction succeeds, each subproblem is clausified and given to the ordinary search procedure; if the problem reduction fails, the original problem is clausified and given to the search procedure.

Implementation

Prover9 is coded in C, and it uses the LADR libraries. Some of the code descended from EQP [52]. (LADR has some AC functions, but Prover9 does not use them). Term data structures are not shared (as they are in Otter). Term indexing is used extensively, with discrimination tree indexing for finding rewrite rules and subsuming units, FPA/Path indexing for finding subsumed units, rewritable terms, and resolvable literals. Feature vector indexing [74] is used for forward and backward nonunit subsumption. Prover9 is available from

`http://www.cs.unm.edu/~mccune/prover9/`

Expected Competition Performance

Prover9 is the CASC fixed point, against which progress can be judged. Each year it is expected do worse than the previous year, relative to the other systems.

8.14 SPASS-SCL 0.1

Christoph Weidenbach
Max Planck Institute for Informatics, Germany

Architecture

SPASS-SCL-FOL 0.1 is a prototype implementing SCL(FOL) [16] for first-order logic without equality and an SMT-style support for equality in the case of BSR. Equality beyond BSR is not supported. Currently, proof documentation is not supported. Models are always given explicitly [15]. SPASS-SCL-FOL 0.1 is mainly built to study the properties of SCL-based calculi.

Strategies

SPASS-SCL-FOL 0.1 does not have any complex strategy parameters. It is a pure SCL(FOL) solver without the use of a portfolio. It is in no way adapted to the problems of the TPTP. It runs five different SCL(FOL) strategies in parallel.

Implementation

SPASS-SCL-FOL is implemented in C on top of the SPASS-Workbench and will become our next system after our SAT solver SPASS-SAT and our SMT solver SPASS-SATT. Due to its prototypical status, it is currently not published on our website.

Expected Competition Performance

SPASS-SCL-FOL 0.1 won't win any TPTP category, but should show reasonable performance in the sense that even our prototype version should not be subsumed by any other ATP.

8.15 Toma 0.7

Teppei Saito
Japan Advanced Institute of Science and Technology, Japan

Architecture

Toma is an automatic equational theorem prover based on a DISCOUNT loop [22] implementing ordered completion [3]. In addition to LPO and KBO, the tool also implements non-standard term orders such as weighted path orders [156] and monotonic semantic path orders [14].

Strategies

For this competition, the tool tries two runs for a given problem: one with a fixed KBO and another with an LPO. Based on an idea from [153], the precedence of the LPO is chosen in a way that minimizes the number of critical pairs between axioms, using the SMT solver Z3.

Implementation

For efficiency, Toma implements fingerprint indexing [75], Waidmeister's AC heuristics [1], and a connectedness testing [2, 79]. The source code is available at

`https://www.jaist.ac.jp/project/maxcomp/.`

Expected Competition Performance

Toma would solve almost half as many problems as state-of-the-art solvers.

8.16 Twee 2.6.0

Nick Smallbone
Chalmers University of Technology, Sweden

Architecture

Twee 2.6.0 [79] is a theorem prover for unit equality problems based on unfailing completion [3]. It implements a DISCOUNT loop, where the active set contains rewrite rules (and unorientable equations) and the passive set contains critical pairs. The basic calculus is not goal-directed, but Twee implements a transformation which improves goal direction for many problems.

Twee features ground joinability testing [51] and a connectedness test [2], which together eliminate many redundant inferences in the presence of unorientable equations. The ground joinability test performs case splits on the order of variables, in the style of [51], and discharges individual cases by rewriting modulo a variable ordering. This year's version has fixes for some completeness bugs which allows some slightly more aggressive redundancy criteria to be used.

Strategies

Twee's strategy is simple and it does not tune its heuristics or strategy based on the input problem. The term ordering is always KBO; by default, functions are ordered by number of occurrences and have weight 1. The proof loop repeats the following steps:

- Select and normalise the lowest-scored critical pair, and if it is not redundant, add it as a rewrite rule to the active set.
- Normalise the active rules with respect to each other.
- Normalise the goal with respect to the active rules.

Each critical pair is scored using a weighted sum of the weight of both of its terms. Terms are treated as DAGs when computing weights, i.e., duplicate subterms are counted only once per term.

For CASC, to take advantage of multiple cores, several versions of Twee run in parallel using different parameters (e.g., with the goal-directed transformation on or off).

Implementation

Twee is written in Haskell. Terms are represented as array-based flatterms for efficient unification and matching. Rewriting uses a perfect discrimination tree.

The passive set is represented compactly (12 bytes per critical pair) by storing only the information needed to reconstruct the critical pair, not the critical pair itself. Because of this, Twee can run for an hour or more without exhausting memory.

Twee uses an LCF-style kernel: all rules in the active set come with a certified proof object which traces back to the input axioms. When a conjecture is proved, the proof object is transformed into a human-readable proof. Proof construction does not harm efficiency because the proof kernel is invoked only when a new rule is accepted. In particular, reasoning about the passive set does not invoke the kernel.

Twee can be downloaded as open source from:

`https://nick8325.github.io/twee/`

Expected Competition Performance

Competing with the top provers.

8.17 Vampire 4.9

Michael Rawson
TU Wien, Austria

There have been a number of improvements since Vampire 4.8, although it is still the same beast. For the first time this year, Vampire's schedules were constructed mostly using the

Snake strategy selection tool, although a return of the traditional Spider is still possible in future. Improvements from the past year include:

- A runtime-specialised version of unidirectional term ordering checks
- Improvements to unification with abstraction
- Surprising improvements to Vampire's basic routines such as renaming and unification
- A simple interactive mode
- Revitalisation of code trees
- Experimental features not yet fully understood, mostly aimed at unit-equational reasoning.
- Portability: Vampire is much more standards-compliant and portable than previously, with much-reduced dependence on platform-specific APIs and hardware architectures, aided by C++17

Vampire's higher-order support remains very similar to last year, although a re-implementation intended for mainline Vampire is already underway.

Architecture

Vampire [48] is an automatic theorem prover for first-order logic with extensions to theory-reasoning and higher-order logic. Vampire implements the calculi of ordered binary resolution, and superposition for handling equality. It also implements a MACE-style finite model builder for finding finite counter-examples [63]. Splitting in resolution-based proof search is controlled by the AVATAR architecture which uses a SAT or SMT solver to make splitting decisions [148, 60]. A number of standard redundancy criteria and simplification techniques are used for pruning the search space: subsumption, tautology deletion, subsumption resolution and rewriting by ordered unit equalities. The reduction ordering is the Knuth-Bendix Ordering. Substitution tree and code tree indexes are used to implement all major operations on sets of terms, literals and clauses. Internally, Vampire works only with clausal normal form. Problems in the full first-order logic syntax are clausified during preprocessing [64]. Vampire implements many useful preprocessing transformations including the SInE axiom selection algorithm. When a theorem is proved, the system produces a verifiable proof, which validates both the clausification phase and the refutation of the CNF.

Strategies

Vampire 4.9 provides a very large number of options for strategy selection. The most important ones are:

- Choices of saturation algorithm:
  - Limited Resource Strategy [70]
  - DISCOUNT loop
  - Otter loop
  - MACE-style finite model building with sort inference
- Splitting via AVATAR [148]
- A variety of optional simplifications.
- Parameterized reduction orderings.
- A number of built-in literal selection functions and different modes of comparing literals [35].

- Age-weight ratio that specifies how strongly lighter clauses are preferred for inference selection. This has been extended with a layered clause selection approach [30].
- Set-of-support strategy with extensions for theory reasoning.
- For theory-reasoning:
  - Ground equational reasoning via congruence closure.
  - Addition of theory axioms and evaluation of interpreted functions [61].
  - Use of Z3 with AVATAR to restrict search to ground-theory-consistent splitting branches [60].
  - Specialised theory instantiation and unification [65].
  - Extensionality resolution with detection of extensionality axioms

Implementation

Vampire 4.9 is implemented in C++. It makes use of fixed versions of Minisat and Z3. See the GitHub repository[6] and associated wiki for more information.

Expected Competition Performance

Vampire 4.9 is the CASC-J12 THF, TFA, FOF, UEQ, SLH, and ICU winner.

8.18 Vampire 5.0

Michael Rawson
University of Southampton, United Kongdom

Vampire 5.0 remains similar in spirit to all previous versions, but a bumper crop of changes have been merged this competition cycle. Various non-competition improvements to Vampire including a *program synthesis* mode [40] and partial support for the polymorphic SMT-LIB 2.7 standard landed, but for the competition we mention:

- ALASCA [47] for reasoning with linear arithmetic, with further VIRAS extensions [72] for quantifier elimination.
- Partial redundancy calculi [32]
- Stabilised and greatly enhanced runtime-specialised unidirectional term ordering checks [31]
- A variant of the ground joinability redundancy elimination rule, used in forward simplification.
- Subsumption (resolution) via code trees.
- Integration of the CaDiCaL SAT solver [11] alongside Minisat.
- More detailed output, including proofs that are (more) TSTP-compliant, reporting non-trivial preprocessing in saturations, and producing completely faithful finite models of the input.
- Portability: Vampire is much more standards-compliant and portable than previously, with much-reduced dependence on platform-specific APIs and hardware architectures, aided by C++17.

Vampire's higher-order support remains very similar to last year, although a re-implementation intended for mainline Vampire is being merged in stages.

[6] github.com/vprover/vampire

Architecture

Vampire [17] is an automatic theorem prover for first-order logic with extensions to theory-reasoning and higher-order logic. Vampire implements several extensions of a core superposition calculus. It also implements a MACE-style finite model builder for finding finite counter-examples [63]. Splitting in saturation-based proof search is controlled by the AVATAR architecture which uses a SAT or SMT solver to make splitting decisions [148, 60]. A number of standard redundancy criteria and simplification techniques are used for pruning the search space: subsumption, tautology deletion, subsumption resolution and rewriting by ordered unit equalities. Substitution tree and code tree indices are used to implement all major operations on sets of terms, literals and clauses. Internally, Vampire works only with clausal normal form: problems not already in CNF are clausified during preprocessing [64]. Vampire implements many preprocessing transformations, including the SInE axiom selection algorithm for large theories and blocked clause elimination.

Strategies

Vampire 5.0 provides a very large number of options for strategy selection. The most important ones are:

- Choices of saturation algorithm:
  - Limited Resource Strategy [70]
  - DISCOUNT loop
  - Otter loop
  - MACE-style finite model building with sort inference
- Splitting via AVATAR [148]
- A variety of optional simplifications.
- Parameterized reduction orderings KBO and LPO.
- A number of built-in literal selection functions and different modes of comparing literals [35].
- Age-weight ratio that specifies how strongly lighter clauses are preferred for inference selection. This has been extended with a layered clause selection approach [30].
- The set-of-support strategy with extensions for theory reasoning.
- For theory reasoning:
  - Specialised calculi such as ALASCA.
  - Addition of theory axioms and evaluation of interpreted functions [61].
  - Use of Z3 with AVATAR to restrict search to ground-theory-consistent splitting branches [60].
  - Specialised theory instantiation and unification [65].
  - Extensionality resolution with detection of extensionality axioms

Implementation

Vampire 5.0 is implemented in C++. It makes use of fixed versions of Minisat, CaDiCaL, GMP, VIRAS, and Z3. See the GitHub repository[7] and associated wiki for more information.

[7] github.com/vprover/vampire

Expected Competition Performance

Vampire 5.0 should be an improvement on the previous version. A reasonably strong performance across all divisions is therefore expected. In the higher-order divisions, performance should be the same as last year.

8.19 Zipperposition 2.1.9999

Jasmin Blanchette
Ludwig-Maximilians-Universität München, Germany

Architecture

Zipperposition is a superposition-based theorem prover for typed first-order logic with equality and for higher-order logic. It is a pragmatic implementation of a complete calculus for full higher-order logic [8]. It features a number of extensions that include polymorphic types, user-defined rewriting on terms and formulas ("deduction modulo theories"), a lightweight variant of AVATAR for case splitting [26], and Boolean reasoning [152]. The core architecture of the prover is based on saturation with an extensible set of rules for inferences and simplifications. Zipperposition uses a full higher-order unification algorithm that enables efficient integration of procedures for decidable fragments of higher-order unification [150]. The initial calculus and main loop were imitations of an earlier version of E [73]. With the implementation of higher-order superposition, the main loop had to be adapted to deal with possibly infinite sets of unifiers [149].

Strategies

The system uses various strategies in a portfolio. The strategies are run in parallel, making use of all CPU cores available. We designed the portfolio of strategies by manual inspection of TPTP problems. Zipperposition's heuristics are inspired by efficient heuristics used in E. Various calculus extensions are used by the strategies [149]. The portfolio mode distinguishes between first-order and higher-order problems. If the problem is first-order, all higher-order prover features are turned off. In particular, the prover uses standard first-order superposition calculus and disables collaboration with the backend prover (described below). Other than that, the portfolio is static and does not depend on the syntactic properties of the problem.

Implementation

The prover is implemented in OCaml. Term indexing is done using fingerprints for unification, perfect discrimination trees for rewriting, and feature vectors for subsumption. Some inference rules such as contextual literal cutting make heavy use of subsumption. For higher-order problems, some strategies use the E prover as an end-game backend prover.

Zipperposition's code can be found at

`https://github.com/sneeuwballen/zipperposition`

and is entirely free software (BSD-licensed).

Zipperposition can also output graphic proofs using graphviz. Some tools to perform type inference and clausification for typed formulas are also provided, as well as a separate library for dealing with terms and formulas [20].

Expected Competition Performance

The prover is expected to perform well on THF, about as well as last year's version. We expect to beat E.

9 Conclusion

The CADE-30 ATP System Competition was the thirtieth large scale competition for classical logic ATP systems. The organizers believe that CASC fulfills its main motivations: evaluation of relative capabilities of ATP systems, stimulation of research, motivation for improving implementations, and providing an exciting event. Through the continuity of the event and consistency in the reporting of the results, performance comparisons with previous and future years are easily possible. The competition provides exposure for system builders both within and outside of the community, and provides an overview of the implementation state of running, fully automatic, classical logic ATP systems.

References

[1] J. Avenhaus, T. Hillenbrand, and B. Löchner. On Using Ground Joinable Equations in Equational Theorem Proving. *Journal of Symbolic Computation*, 36(1-2):217–233, 2003.

[2] L. Bachmair and N. Dershowitz. Critical Pair Criteria for Completion. *Journal of Symbolic Computation*, 6(1):1–18, 1988.

[3] L. Bachmair, N. Dershowitz, and D.A. Plaisted. Completion Without Failure. In H. Ait-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, pages 1–30. Academic Press, 1989.

[4] H. Barbosa, C. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, and Y. Zohar. cvc5: A Versatile and Industrial-Strength SMT Solver. In D. Fisman and G. Rosu, editors, *Proceedings of the 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 13243 in Lecture Notes in Computer Science, pages 415–442. Springer, 2022.

[5] H. Barbosa, P. Fontaine, and A. Reynolds. Congruence Closure with Free Variables. In A. Legay and T. Margaria, editors, *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 10205 in Lecture Notes in Computer Science, pages 2134–230. Springer-Verlag, 2017.

[6] H. Barbosa, A. Reynolds, D. El Ouraoui, C. Tinelli, and C. Barrett. Extending SMT Solvers to Higher-Order Logic. In P. Fontaine, editor, *Proceedings of the 27th International Conference on Automated Deduction*, number 11716 in Lecture Notes in Computer Science, pages 35–54. Springer-Verlag, 2019.

[7] C. Barrett, C. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. In G. Gopalakrishnan and S. Qadeer, editors, *Proceedings of the 23rd International Conference on Computer Aided Verification*, number 6806 in Lecture Notes in Computer Science, pages 171–177. Springer-Verlag, 2011.

[8] A. Bentkamp, J. Blanchette, S. Tourret, and P. Vukmirović. Superposition for Full Higher-order Logic. In A. Platzer and G. Sutcliffe, editors, *Proceedings of the 28th International Conference on Automated Deduction*, number 12699 in Lecture Notes in Computer Science, pages 396–412. Springer-Verlag, 2021.

[9] C. Benzmüller, L. Paulson, F. Theiss, and A. Fietzke. LEO-II - A Cooperative Automatic Theorem Prover for Higher-Order Logic. In P. Baumgartner, A. Armando, and G. Dowek,

editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 162–170. Springer-Verlag, 2008.

[10] A. Biere. PicoSAT Essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008.

[11] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt. CaDiCaL 2.0. In A. Gurfinkel and V. Ganesh, editors, *Proceedings of the 36th International Conference on Computer Aided Verification*, number 14681 in Lecture Notes in Computer Science, pages 133–152. Springer-Verlag, 2024.

[12] J. Blanchette, M. Haslbeck, D. Matichuk, and T. Nipkow. Mining the Archive of Formal Proofs. In M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, and V. Sorge, editors, *Proceedings of the 8th Conference on Intelligent Computer Mathematics*, number 9150 in Lecture Notes in Computer Science, pages 3–17. Springer-Verlag, 2015.

[13] J. Blanchette and A. Paskevich. TFF1: The TPTP Typed First-order Form with Rank-1 Polymorphism. In M.P. Bonacina, editor, *Proceedings of the 24th International Conference on Automated Deduction*, number 7898 in Lecture Notes in Artificial Intelligence, pages 414–420. Springer-Verlag, 2013.

[14] C. Borralleras, M. Ferreira, and A. Rubio. Complete Monotonic Semantic Path Orderings. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, number 1831 in Lecture Notes in Artificial Intelligence, pages 346–364. Springer-Verlag, 2000.

[15] M. Bromberger, F. Krasnopol, S. Moehle, and Weidenbach C. First-Order Automatic Literal Model Generation. In C. Benzmüller, M. Heule, and R. Schmidt, editors, *Proceedings of the 12th International Joint Conference on Automated Reasoning*, number 14739 in Lecture Notes in Artificial Intelligence, pages 133–153, 2024.

[16] M. Bromberger, S. Schwarz, and C. Weidenbach. Exploring Partial Models with SCL. In R. Piskac and A. Voronkov, editors, *Proceedings of the 24th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 94 in EPiC Series in Computing, pages 48–72. EasyChair Publications, 2023.

[17] F. Bártek, A. Bhayat, R. Coutelier, M. Hajdu, M. Hetzenberger, P. Hozzová, L. Kovács, J. Rath, M. Rawson, G. Reger, M. Suda, J. Schoisswohl, and A. Voronkov. The Vampire Diary. arXiv:2506.03030, 2025.

[18] F. Bártek and M. Suda. How Much Should This Symbol Weigh? A GNN-Advised Clause Selection. In R. Piskac and A. Voronkov, editors, *Proceedings of the 24th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 94 in EPiC Series in Computing, pages 96–111. EasyChair Publications, 2023.

[19] F. Bártek and M. Suda. Robust Strategy Schedule Optimization for an Automatic Theorem Prover. In C. Benzmüller, M. Heule, and R. Schmidt, editors, *Proceedings of the 8th Conference on Artificial Intelligence and Theorem Proving*, 2023.

[20] S. Cruanes. *Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond*. PhD thesis, Ecole Polytechnique, Paris, France, 2015.

[21] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In C. Ramakrishnan and J. Rehof, editors, *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 4963 in Lecture Notes in Artificial Intelligence, pages 337–340. Springer-Verlag, 2008.

[22] J. Denzinger, M. Kronenburg, and S. Schulz. DISCOUNT: A Distributed and Learning Equational Prover. *Journal of Automated Reasoning*, 18(2):189–198, 1997.

[23] A. Duarte and K. Korovin. Implementing Superposition in iProver. In N. Peltier and V. Sofronie-Stokkermans, editors, *Proceedings of the 10th International Joint Conference on Automated Reasoning*, number 12167 in Lecture Notes in Artificial Intelligence, pages 388–397, 2020.

[24] A. Duarte and K. Korovin. AC Simplifications and Closure Redundancies in the Superposition Calculus. In A. Das and S. Negri, editors, *Proceedings of the 30th International Conference on*

Automated Reasoning with Analytic Tableaux and Related Methods, number 12842 in Lecture Notes in Artificial Intelligence, pages 200–217. Springer-Verlag, 2021.

[25] A. Duarte and K. Korovin. Ground Joinability and Connectedness in the Superposition Calculus. In J. Blanchette, L. Kovács, and D. Pattinson, editors, *Proceedings of the 11th International Joint Conference on Automated Reasoning*, number 13385 in Lecture Notes in Artificial Intelligence, pages 169–187, 2022.

[26] G. Ebner, J. Blanchette, and S. Tourret. A Unifying Splitting Framework. In A. Platzer and G. Sutcliffe, editors, *Proceedings of the 28th International Conference on Automated Deduction*, number 12699 in Lecture Notes in Computer Science, pages 344–360. Springer-Verlag, 2021.

[27] N. Eén and N. Sörensson. An Extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing*, number 2919 in Lecture Notes in Computer Science, pages 502–518. Springer-Verlag, 2004.

[28] H. Ganzinger and K. Korovin. New Directions in Instantiation-based Theorem Proving. In P. Kolaitis, editor, *Proceedings of the 18th IEEE Symposium on Logic in Computer Science*, pages 55–64. IEEE Press, 2003.

[29] Y. Ge and L. de Moura. Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories. In A. Bouajjani and O. Maler, editors, *Proceedings of the 21st International Conference on Computer Aided Verification*, number 5643 in Lecture Notes in Computer Science, pages 306–320. Springer-Verlag, 2009.

[30] B. Gleiss and M. Suda. Layered Clause Selection for Theory Reasoning. In N. Peltier and V. Sofronie-Stokkermans, editors, *Proceedings of the 10th International Joint Conference on Automated Reasoning*, number 12166 in Lecture Notes in Computer Science, pages 402–409, 2020.

[31] M. Hajdu, R. Coutelier, L. Kovács, and A. Voronkov. Term Ordering Diagrams. In C. Barrett and U. Waldmann, editors, *Proceedings of the 30th International Conference on Automated Deduction*, Lecture Notes in Computer Science, page To appear. Springer-Verlag, 2025.

[32] M. Hajdu, L. Kovács, and A. Voronkov. Partial Redundancy in Saturation. In C. Barrett and U. Waldmann, editors, *Proceedings of the 30th International Conference on Automated Deduction*, Lecture Notes in Computer Science, page To appear. Springer-Verlag, 2025.

[33] J. Hernandez and K. Korovin. An Abstraction-Refinement Framework for Reasoning with Large Theories. In D. Galmiche, S. Schulz, and R. Sebastiani, editors, *Proceedings of the 9th International Joint Conference on Automated Reasoning*, number 10900 in Lecture Notes in Computer Science, pages 663–679, 2018.

[34] J. Hernandez and K. Korovin. Towards an Under-Approximation Abstraction-Refinement for Reasoning with Large Theories. In A. Bolotov and F. Kammueller, editors, *Proceedings of the 26th Automated Reasoning Workshop*, page To appear, 2019.

[35] K. Hoder, G. Reger, M. Suda, and A. Voronkov. Selecting the Selection. In N. Olivetti and A. Tiwari, editors, *Proceedings of the 8th International Joint Conference on Automated Reasoning*, number 9706 in Lecture Notes in Artificial Intelligence, pages 313–329, 2016.

[36] K. Hoder and A. Voronkov. Sine Qua Non for Large Theory Reasoning. In V. Sofronie-Stokkermans and N. Bjœrner, editors, *Proceedings of the 23rd International Conference on Automated Deduction*, number 6803 in Lecture Notes in Artificial Intelligence, pages 299–314. Springer-Verlag, 2011.

[37] E. Holden and K. Korovin. SMAC and XGBoost your Theorem Prover. In T. Hales, C. Kaliszyk, R. Kumar, S. Schulz, and J. Urban, editors, *Proceedings of the 4th Conference on Artificial Intelligence and Theorem Proving*, pages 93–95, 2019.

[38] E. Holden and K. Korovin. Heterogeneous Heuristic Optimisation and Scheduling for First-Order Theorem Proving. In F. Kamareddine and C. Sacerdoti, editors, *Proceedings of the 14th International Conference on Intelligent Computer Mathematics*, number 12833 in Lecture Notes in Computer Science, pages 107–123. Springer-Verlag, 2021.

[39] S. Holden. Connect++: A New Automated Theorem Prover Based on the Connection Calculus. In J. Otten and W. Bibel, editors, *Proceedings of the 1st International Workshop on Automated Reasoning with Connection Calculi*, number 3613 in CEUR Workshop Proceedings, pages 95–106, 2023.

[40] P. Hozzová, D. Amrollahi, M. Hajdu, L. Kovács, A. Voronkov, and Wagner E. Synthesis of Recursive Programs in Saturation. In C. Benzmüller, M. Heule, and R. Schmidt, editors, *Proceedings of the 12th International Joint Conference on Automated Reasoning*, number 14739 in Lecture Notes in Artificial Intelligence, pages 154–171, 2024.

[41] 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, *Proceedings of the 10th International Conference on Intelligent Computer Mathematics*, number 10383 in Lecture Notes in Artificial Intelligence, pages 292–302. Springer-Verlag, 2017.

[42] J. Jakubuv and J. Urban. Enhancing ENIGMA Given Clause Guidance. In F. Rabe, W. Farmer, G. Passmore, and A. Youssef, editors, *Proceedings of the 11th International Conference on Intelligent Computer Mathematics*, number 11006 in Lecture Notes in Artificial Intelligence, pages 118–124. Springer-Verlag, 2018.

[43] C. Kaliszyk, G. Sutcliffe, and F. Rabe. TH1: The TPTP Typed Higher-Order Form with Rank-1 Polymorphism. In P. Fontaine, S. Schulz, and J. Urban, editors, *Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning*, number 1635 in CEUR Workshop Proceedings, pages 41–55, 2016.

[44] L. Kondylidou, A. Reynolds, and J. Blanchette. Augmenting Model-Based Instantiation with Fast Enumeration. In A. Gurfinkel and M. Heule, editors, *Proceedings of the 31st International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 15696 in Lecture Notes in Computer Science, pages 85–103. Springer-Verlag, 2025.

[45] K. Korovin. iProver - An Instantiation-based Theorem Prover for First-order Logic (System Description). In P. Baumgartner, A. Armando, and G. Dowek, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 292–298, 2008.

[46] K. Korovin. Inst-Gen - A Modular Approach to Instantiation-based Automated Reasoning. In A. Voronkov and C. Weidenbach, editors, *Programming Logics, Essays in Memory of Harald Ganzinger*, number 7797 in Lecture Notes in Computer Science, pages 239–270. Springer-Verlag, 2013.

[47] K. Korovin, L. Kovács, G. Reger, and J. Schoisswohl. ALASCA: Reasoning in Quantified Linear Arithmetic. In S. Sankaranarayanan and N. Sharygina, editors, *Proceedings of the 29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 13993 in Lecture Notes in Computer Science, pages 647–665. Springer-Verlag, 2023.

[48] L. Kovács and A. Voronkov. First-Order Theorem Proving and Vampire. In N. Sharygina and H. Veith, editors, *Proceedings of the 25th International Conference on Computer Aided Verification*, number 8044 in Lecture Notes in Artificial Intelligence, pages 1–35. Springer-Verlag, 2013.

[49] B. Loechner. Things to Know When Implementing KBO. *Journal of Automated Reasoning*, 36(4):289–310, 2006.

[50] B. Loechner. Things to Know When Implementing LBO. *Journal of Artificial Intelligence Tools*, 15(1):53–80, 2006.

[51] U. Martin and T. Nipkow. Ordered Rewriting and Confluence. In M.E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, number 449 in Lecture Notes in Artificial Intelligence, pages 366–380. Springer-Verlag, 1990.

[52] W.W. McCune. Solution of the Robbins Problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.

[53] W.W. McCune. Otter 3.3 Reference Manual. Technical Report ANL/MSC-TM-263, Argonne

National Laboratory, Argonne, USA, 2003.

[54] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.

[55] A. Nonnengart and C. Weidenbach. Computing Small Clause Normal Forms. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 335–367. Elsevier Science, 2001.

[56] J. Otten. Restricting Backtracking in Connection Calculi. *AI Communications*, 23(2-3):159–182, 2010.

[57] L. Paulson and J. Blanchette. Three Years of Experience with Sledgehammer, a Practical Link between Automatic and Interactive Theorem Provers. In G. Sutcliffe, E. Ternovska, and S. Schulz, editors, *Proceedings of the 8th International Workshop on the Implementation of Logics*, number 2 in EPiC Series in Computing, pages 1–11. EasyChair Publications, 2010.

[58] M. Rawson, C. Eisenhofer, and L. Kovács. Constraint Learning for Non-confluent Proof Search. In G. Pozzato and T. Uustalu, editors, *Proceedings of the 34th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, Lecture Notes in Artificial Intelligence, page To appear. Springer-Verlag, 2025.

[59] M. Rawson and G. Reger. Eliminating Models during Model Elimination. In A. Das and S. Negri, editors, *Proceedings of the 30th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, Lecture Notes in Artificial Intelligence, page To appear. Springer-Verlag, 2021.

[60] G. Reger, N. Bjørner, M. Suda, and A. Voronkov. AVATAR Modulo Theories. In C. Benzmüller, G. Sutcliffe, and R. Rojas, editors, *Proceedings of the 2nd Global Conference on Artificial Intelligence*, number 41 in EPiC Series in Computing, pages 39–52. EasyChair Publications, 2016.

[61] G. Reger, J. Schoisswohl, and A. Voronkov. Making Theory Reasoning Simpler. In J. Groote and K. Larsen, editors, *Proceedings of the 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 12652 in Lecture Notes in Computer Science, pages 164–180. Springer-Verlag, 2021.

[62] G. Reger, M. Suda, and A. Voronkov. Playing with AVATAR. In A. Felty and A. Middeldorp, editors, *Proceedings of the 25th International Conference on Automated Deduction*, number 9195 in Lecture Notes in Computer Science, pages 399–415. Springer-Verlag, 2015.

[63] G. Reger, M. Suda, and A. Voronkov. Finding Finite Models in Multi-Sorted First Order Logic. In N. Creignou and D. Le Berre, editors, *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing*, number 9710 in Lecture Notes in Computer Science, pages 323–341. Springer-Verlag, 2016.

[64] G. Reger, M. Suda, and A. Voronkov. New Techniques in Clausal Form Generation. In C. Benzmüller, G. Sutcliffe, and R. Rojas, editors, *Proceedings of the 2nd Global Conference on Artificial Intelligence*, number 41 in EPiC Series in Computing, pages 11–23. EasyChair Publications, 2016.

[65] G. Reger, M. Suda, and A. Voronkov. Unification with Abstraction and Theory Instantiation in Saturation-based Reasoning. In D. Beyer and M. Huisman, editors, *Proceedings of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 10805 in Lecture Notes in Computer Science, pages 3–22. Springer-Verlag, 2018.

[66] A. Reynolds, H. Barbosa, and P. Fontaine. Revisiting Enumerative Instantiation. In D. Beyer and M. Huisman, editors, *Proceedings of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 10805 in Lecture Notes in Computer Science, pages 112–131. Springer-Verlag, 2018.

[67] A. Reynolds, H. Barbosa, A. Nötzli, C. Barrett, and C. Tinelli. cvc4sy: Smart and Fast Term Enumeration for Syntax-Guided Synthesis. In I. Dillig and S. Tasiran, editors, *Proceedings of the 31st International Conference on Computer Aided Verification*, number 11562 in Lecture Notes

in Computer Science, pages 74–83. Springer-Verlag, 2019.

[68] A. Reynolds, C. Tinelli, and L. de Moura. Finding Conflicting Instances of Quantified Formulas in SMT. In K. Claessen and V. Kuncak, editors, *Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design*, pages 195–202, 2014.

[69] A. Reynolds, C. Tinelli, A. Goel, S. Krstic, M. Deters, and C. Barrett. Quantifier Instantiation Techniques for Finite Model Finding in SMT. In M.P. Bonacina, editor, *Proceedings of the 24th International Conference on Automated Deduction*, number 7898 in Lecture Notes in Artificial Intelligence, pages 377–391. Springer-Verlag, 2013.

[70] A. Riazanov and A. Voronkov. Limited Resource Strategy in Resolution Theorem Proving. *Journal of Symbolic Computation*, 36(1-2):101–115, 2003.

[71] O. Roussel. Controlling a Solver Execution with the `runsolver` Tool. *Journal of Satisfiability, Boolean Modeling and Computation*, 7(4):139–144, 2011.

[72] J. Schoisswohl, L. Kovács, and K. Korovin. VIRAS: Conflict-Driven Quantifier Elimination for Integer-Real Arithmetic. In N. Bjørner, M. Heule, and A. Voronkov, editors, *Proceedings of the 25th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 100 in EPiC Series in Computing, pages 147–164. EasyChair Publications, 2024.

[73] S. Schulz. E: A Brainiac Theorem Prover. *AI Communications*, 15(2-3):111–126, 2002.

[74] S. Schulz. System Abstract: E 0.81. In M. Rusinowitch and D. Basin, editors, *Proceedings of the 2nd International Joint Conference on Automated Reasoning*, number 3097 in Lecture Notes in Artificial Intelligence, pages 223–228. Springer-Verlag, 2004.

[75] S. Schulz. Fingerprint Indexing for Paramodulation and Rewriting. In B. Gramlich, D. Miller, and U. Sattler, editors, *Proceedings of the 6th International Joint Conference on Automated Reasoning*, number 7364 in Lecture Notes in Artificial Intelligence, pages 477–483. Springer-Verlag, 2012.

[76] S. Schulz. Simple and Efficient Clause Subsumption with Feature Vector Indexing. In M.P. Bonacina and M. Stickel, editors, *Automated Reasoning and Mathematics: Essays in Memory of William W. McCune*, number 7788 in Lecture Notes in Artificial Intelligence, pages 45–67. Springer-Verlag, 2013.

[77] S. Schulz. System Description: E 1.8. In K. McMillan, A. Middeldorp, and A. Voronkov, editors, *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 8312 in Lecture Notes in Computer Science, pages 477–483. Springer-Verlag, 2013.

[78] S. Schulz, S. Cruanes, and P. Vukmirović. Faster, Higher, Stronger: E 2.3. In P. Fontaine, editor, *Proceedings of the 27th International Conference on Automated Deduction*, number 11716 in Lecture Notes in Computer Science, pages 495–507. Springer-Verlag, 2019.

[79] N. Smallbone. Twee: An Equational Theorem Prover (System Description). In A. Platzer and G. Sutcliffe, editors, *Proceedings of the 28th International Conference on Automated Deduction*, number 12699 in Lecture Notes in Computer Science, pages 602–613. Springer-Verlag, 2021.

[80] A. Steen and C. Benzmüller. Extensional Higher-Order Paramodulation in Leo-III. *Journal of Automated Reasoning*, 65(6):775–807, 2021.

[81] A. Stump, G. Sutcliffe, and C. Tinelli. StarExec: a Cross-Community Infrastructure for Logic Solving. In S. Demri, D. Kapur, and C. Weidenbach, editors, *Proceedings of the 7th International Joint Conference on Automated Reasoning*, number 8562 in Lecture Notes in Artificial Intelligence, pages 367–373, 2014.

[82] M. Suda. Aiming for the Goal with SInE. In C. Benzmüller, C. Lisetti, and M. Theobald, editors, *Proceedings of the 5th and 6th Vampire Workshops*, number 71 in EPiC Series in Computing, page 38–44. EasyChair Publications, 2019.

[83] M. Suda. Vampire Getting Noisy: Will Random Bits Help Conquer Chaos? (System Description). In J. Blanchette, L. Kovács, and D. Pattinson, editors, *Proceedings of the 11th International Joint Conference on Automated Reasoning*, number 13385 in Lecture Notes in Artificial Intelligence,

pages 659–667, 2022.

[84] G. Sutcliffe. Proceedings of the CADE-16 ATP System Competition. Trento, Italy, 1999.

[85] G. Sutcliffe. Proceedings of the CADE-17 ATP System Competition. Pittsburgh, USA, 2000.

[86] G. Sutcliffe. The CADE-16 ATP System Competition. *Journal of Automated Reasoning*, 24(3):371–396, 2000.

[87] G. Sutcliffe. Proceedings of the IJCAR ATP System Competition. Siena, Italy, 2001.

[88] G. Sutcliffe. The CADE-17 ATP System Competition. *Journal of Automated Reasoning*, 27(3):227–250, 2001.

[89] G. Sutcliffe. Proceedings of the CADE-18 ATP System Competition. Copenhagen, Denmark, 2002.

[90] G. Sutcliffe. Proceedings of the CADE-19 ATP System Competition. Miami, USA, 2003.

[91] G. Sutcliffe. Proceedings of the 2nd IJCAR ATP System Competition. Cork, Ireland, 2004.

[92] G. Sutcliffe. Proceedings of the CADE-20 ATP System Competition. Tallinn, Estonia, 2005.

[93] G. Sutcliffe. The IJCAR-2004 Automated Theorem Proving Competition. *AI Communications*, 18(1):33–40, 2005.

[94] G. Sutcliffe. Proceedings of the 3rd IJCAR ATP System Competition. Seattle, USA, 2006.

[95] G. Sutcliffe. The CADE-20 Automated Theorem Proving Competition. *AI Communications*, 19(2):173–181, 2006.

[96] G. Sutcliffe. Proceedings of the CADE-21 ATP System Competition. Bremen, Germany, 2007.

[97] G. Sutcliffe. The 3rd IJCAR Automated Theorem Proving Competition. *AI Communications*, 20(2):117–126, 2007.

[98] G. Sutcliffe. Proceedings of the 4th IJCAR ATP System Competition. Sydney, Australia, 2008.

[99] G. Sutcliffe. The CADE-21 Automated Theorem Proving System Competition. *AI Communications*, 21(1):71–82, 2008.

[100] G. Sutcliffe. The SZS Ontologies for Automated Reasoning Software. In G. Sutcliffe, P. Rudnicki, R. Schmidt, B. Konev, and S. Schulz, editors, *Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and the 7th International Workshop on the Implementation of Logics*, number 418 in CEUR Workshop Proceedings, pages 38–49, 2008.

[101] G. Sutcliffe. Proceedings of the CADE-22 ATP System Competition. Montreal, Canada, 2009.

[102] G. Sutcliffe. The 4th IJCAR Automated Theorem Proving System Competition - CASC-J4. *AI Communications*, 22(1):59–72, 2009.

[103] G. Sutcliffe. Proceedings of the 5th IJCAR ATP System Competition. Edinburgh, United Kingdom, 2010.

[104] G. Sutcliffe. The CADE-22 Automated Theorem Proving System Competition - CASC-22. *AI Communications*, 23(1):47–60, 2010.

[105] G. Sutcliffe. Proceedings of the CADE-23 ATP System Competition. Wroclaw, Poland, 2011.

[106] G. Sutcliffe. The 5th IJCAR Automated Theorem Proving System Competition - CASC-J5. *AI Communications*, 24(1):75–89, 2011.

[107] G. Sutcliffe. Proceedings of the 6th IJCAR ATP System Competition. Manchester, England, 2012.

[108] G. Sutcliffe. The CADE-23 Automated Theorem Proving System Competition - CASC-23. *AI Communications*, 25(1):49–63, 2012.

[109] G. Sutcliffe. Proceedings of the 24th CADE ATP System Competition. Lake Placid, USA, 2013.

[110] G. Sutcliffe. The 6th IJCAR Automated Theorem Proving System Competition - CASC-J6. *AI Communications*, 26(2):211–223, 2013.

[111] G. Sutcliffe. Proceedings of the 7th IJCAR ATP System Competition. Vienna, Austria, 2014.

[112] G. Sutcliffe. The CADE-24 Automated Theorem Proving System Competition - CASC-24. *AI*

Communications, 27(4):405–416, 2014.

[113] G. Sutcliffe. Proceedings of the CADE-25 ATP System Competition. Berlin, Germany, 2015. http://tptp.org/CASC/25/Proceedings.pdf.

[114] G. Sutcliffe. The 7th IJCAR Automated Theorem Proving System Competition - CASC-J7. *AI Communications*, 28(4):683–692, 2015.

[115] G. Sutcliffe. Proceedings of the 8th IJCAR ATP System Competition. Coimbra, Portugal, 2016. http://tptp.org/CASC/J8/Proceedings.pdf.

[116] G. Sutcliffe. The 8th IJCAR Automated Theorem Proving System Competition - CASC-J8. *AI Communications*, 29(5):607–619, 2016.

[117] G. Sutcliffe. The CADE ATP System Competition - CASC. *AI Magazine*, 37(2):99–101, 2016.

[118] G. Sutcliffe. Proceedings of the 26th CADE ATP System Competition. Gothenburg, Sweden, 2017. http://tptp.org/CASC/26/Proceedings.pdf.

[119] G. Sutcliffe. The CADE-26 Automated Theorem Proving System Competition - CASC-26. *AI Communications*, 30(6):419–432, 2017.

[120] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

[121] G. Sutcliffe. Proceedings of the 9th IJCAR ATP System Competition. Oxford, United Kingdom, 2018. http://tptp.org/CASC/J9/Proceedings.pdf.

[122] G. Sutcliffe. The 9th IJCAR Automated Theorem Proving System Competition - CASC-J9. *AI Communications*, 31(6):495–507, 2018.

[123] G. Sutcliffe. Proceedings of the CADE-27 ATP System Competition. Natal, Brazil, 2019. http://tptp.org/CASC/27/Proceedings.pdf.

[124] G. Sutcliffe. Proceedings of the 10th IJCAR ATP System Competition. Online, 2020. http://tptp.org/CASC/J10/Proceedings.pdf.

[125] G. Sutcliffe. The CADE-27 Automated Theorem Proving System Competition - CASC-27. *AI Communications*, 32(5-6):373–389, 2020.

[126] G. Sutcliffe. Proceedings of the CADE-28 ATP System Competition. Online, 2021. http://tptp.org/CASC/28/Proceedings.pdf.

[127] G. Sutcliffe. The 10th IJCAR Automated Theorem Proving System Competition - CASC-J10. *AI Communications*, 34(2):163–177, 2021.

[128] G. Sutcliffe. Proceedings of the 11th IJCAR ATP System Competition. Online, 2022. http://tptp.org/CASC/J11/Proceedings.pdf.

[129] G. Sutcliffe. Proceedings of the CADE-29 ATP System Competition. Online, 2023.

[130] G. Sutcliffe. Proceedings of the 12th IJCAR ATP System Competition. Online, 2024. http://tptp.org/CASC/J12/Proceedings.pdf.

[131] G. Sutcliffe. The 12th IJCAR Automated Theorem Proving System Competition - CASC-J12. *AI Communications*, 38(1):3–20, 2025.

[132] G. Sutcliffe and M. Desharnais. The CADE-28 Automated Theorem Proving System Competition - CASC-28. *AI Communications*, 34(4):259–276, 2022.

[133] G. Sutcliffe and M. Desharnais. The 11th IJCAR Automated Theorem Proving System Competition - CASC-J11. *AI Communications*, 36(2):73–91,, 2023.

[134] G. Sutcliffe and M. Desharnais. The CADE-29 Automated Theorem Proving System Competition - CASC-29. *AI Communications*, 37(4):485–503, 2024.

[135] G. Sutcliffe, S. Schulz, K. Claessen, and A. Van Gelder. Using the TPTP Language for Writing Derivations and Finite Interpretations. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, pages 67–81. Springer, 2006.

[136] G. Sutcliffe, A. Steen, and P. Fontaine. The New TPTP Format for Interpretations.

arXiv:2406.06108, 2024.

[137] G. Sutcliffe and C. Suttner. The CADE-14 ATP System Competition. Technical Report 98/01, Department of Computer Science, James Cook University, Townsville, Australia, 1998.

[138] G. Sutcliffe and C. Suttner. The CADE-18 ATP System Competition. *Journal of Automated Reasoning*, 31(1):23–32, 2003.

[139] G. Sutcliffe and C. Suttner. The CADE-19 ATP System Competition. *AI Communications*, 17(3):103–182, 2004.

[140] G. Sutcliffe, C. Suttner, and F.J. Pelletier. The IJCAR ATP System Competition. *Journal of Automated Reasoning*, 28(3):307–320, 2002.

[141] G. Sutcliffe and C.B. Suttner. Special Issue: The CADE-13 ATP System Competition. *Journal of Automated Reasoning*, 18(2), 1997.

[142] G. Sutcliffe and C.B. Suttner. The Procedures of the CADE-13 ATP System Competition. *Journal of Automated Reasoning*, 18(2):163–169, 1997.

[143] G. Sutcliffe and C.B. Suttner. Proceedings of the CADE-15 ATP System Competition. Lindau, Germany, 1998.

[144] G. Sutcliffe and C.B. Suttner. The CADE-15 ATP System Competition. *Journal of Automated Reasoning*, 23(1):1–23, 1999.

[145] G. Sutcliffe and C.B. Suttner. Evaluating General Purpose Automated Theorem Proving Systems. *Artificial Intelligence*, 131(1-2):39–54, 2001.

[146] G. Sutcliffe and J. Urban. The CADE-25 Automated Theorem Proving System Competition - CASC-25. *AI Communications*, 29(3):423–433, 2016.

[147] C.B. Suttner and G. Sutcliffe. The CADE-14 ATP System Competition. *Journal of Automated Reasoning*, 21(1):99–134, 1998.

[148] A. Voronkov. AVATAR: The New Architecture for First-Order Theorem Provers. In A. Biere and R. Bloem, editors, *Proceedings of the 26th International Conference on Computer Aided Verification*, number 8559 in Lecture Notes in Computer Science, pages 696–710, 2014.

[149] P. Vukmirović, A. Bentkamp, J. Blanchette, S. Cruanes, V. Nummelin, and S. Tourret. Making Higher-order Superposition Work. In A. Platzer and G. Sutcliffe, editors, *Proceedings of the 28th International Conference on Automated Deduction*, number 12699 in Lecture Notes in Computer Science, pages 415–432. Springer-Verlag, 2021.

[150] P. Vukmirović, A. Bentkamp, and V. Nummelin. Efficient Full Higher-order Unification. In Z.M. Ariola, editor, *Proceedings of the 5th International Conference on Formal Structures for Computation and Deduction*, number 167 in Leibniz International Proceedings in Informatics, pages 5:1–5:20. Dagstuhl Publishing, 2020.

[151] P. Vukmirović, J. Blanchette, and S. Schulz. Extending a High-Performance Prover to Higher-Order Logic. In S. Sankaranarayanan and N. Sharygina, editors, *Proceedings of the 29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 13994 in Lecture Notes in Computer Science, page 111–129. Springer-Verlag, 2023.

[152] P. Vukmirović and V. Nummelin. Boolean Reasoning in a Higher-Order Superposition Prover. In P. Fontaine, P. Rümmer, and S. Tourret, editors, *Proceedings of the 7th Workshop on Practical Aspects of Automated Reasoning*, number 2752 in CEUR Workshop Proceedings, pages 148–166, 2020.

[153] S. Winkler and G. Moser. MaedMax: A Maximal Ordered Completion Tool. In D. Galmiche, S. Schulz, and R. Sebastiani, editors, *Proceedings of the 9th International Joint Conference on Automated Reasoning*, number 10900 in Lecture Notes in Computer Science, pages 388–404, 2018.

[154] M. Wisniewski, A. Steen, and C. Benzmüller. LeoPARD - A Generic Platform for the Implementation of Higher-Order Reasoners. In M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, and V. Sorge, editors, *Proceedings of the International Conference on Intelligent Computer Mathematics*, number 9150 in Lecture Notes in Computer Science, pages 325–330. Springer-Verlag,

2015.

[155] Y. Xu, J. Liu, S. Chen, X. Zhong, and X. He. Contradiction Separation Based Dynamic Multi-clause Synergized Automated Deduction. *Information Sciences*, 462:93–113, 2018.

[156] A. Yamada, K. Kusakari, and T. Sakabe. A Unified Ordering for Termination Proving. *Science of Computer Programming*, 111(1):110–134, 2015.