CASC-J13

CASC-J13
CASC-J13
CASC-J13

Proceedings of the 13th IJCAR ATP System Competition (CASC-J13)

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-J13 was the thirty-first competition in the CASC series. Twenty three 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) [122] 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-J13 was held on 27th July 2026, as part of the 13th International Joint Conference on Automated Reasoning (IJCAR 2024)[1], in Lisbon, Portugal. It was the thirty-first competition in the CASC series; see [152, 158, 155, 90, 92, 151, 149, 150, 97, 99, 101, 104, 107, 109, 111, 113, 115, 117, 119, 157, 121, 124, 127, 130, 132, 142, 143, 144, 138, 139] and the CASC web site `tptp.org/CASC`, for information about previous CASCs. CASC-J13 was organized by Geoff Sutcliffe, and overseen by a panel consisting of Aart Middeldorp, Cláudia Nalon, and Martina Seidl. The competition was run on computers provided by the StarExec project [84] at the University of Miami. The CASC-J13 web site `tptp.org/CASC/J13` provides access to all the resources used before, during, and after the event.

The design and organization of CASC has evolved over the years to a sophisticated state [152, 153, 148, 154, 88, 89, 91, 93, 94, 95, 96, 98, 100, 103, 106, 108, 110, 112, 114, 116, 118, 120, 123, 126, 128, 129, 131, 133, 134, 135, 137]. Important changes for CASC-J13 were (for readers already familiar with the general design of CASC):

- The TFA and EPR divisions went on hiatus.
- The FNT division returned from hiatus.

[1] CADE was a constituent conference of the 13th IJCAR, hence CASC-"J13", which in turn was part of the Federated Logic Conference 2026

- Inference steps in proofs may not list parents that are not used in the inference.
- Proofs should graft in subproofs from external systems.
- A new performance measure, proof granularity, is recorded and reported.
- The ProoVer competition for proof verifiers was held for the first time in 2026, as a companion competition to CASC-J13.

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. Section 3 describes the competition infrastructure. Section 4 describes how the systems are evaluated. Section 5 describes the practical steps for entering a system. Section 6 lists system properties that entrants should check. Section 7 provides descriptions (written by the entrants) of the systems. Section 8 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-J13 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) [156]. Each division uses problems that have certain logical, language, and syntactic characteristics, so that the ATP 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-J13. The example problems can be viewed online at `tptp.org/cgi-bin/SeeTPTP?Category=Problems`. Section 3.2 explains what problems are eligible for use in each division and category. The competition rankings are at (only) the division level, as explained in Section 4.

ATP systems that cannot run in the competition divisions for any reason, e.g., the system cannot run on the competition computers, 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. Demonstration division systems can run on the competition computers, or on computers supplied by the entrant. The demonstration division results are presented along with the competition divisions' results, but might not be comparable with those results.

Twenty three ATP systems competed in the various divisions of CASC-J13. The division winners of the previous CASC (CASC-30) and the Prover9 1109a system are automatically entered into the FOF demonstration division 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

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 Example: `NUM738^1` **TEQ** – **T**HF with **EQ**uality Example: `SET171^3`
FOF	**F**irst-**O**rder **F**orm theorems (axioms with a provable conjecture).	**FNE** – **F**OF with **N**o **E**quality. Example: `COM003+1` **FEQ** – **F**OF with **EQ**uality. Example: `SEU147+3`
FNT	**F**OF **N**on-**T**heorems (axioms with a counter-satisfiable conjecture, and satisfiable axioms without a conjecture).	**FNN** – **FN**T with **N**o equality. Example: `KRS173+1` **FNQ** – **FN**T with e**Q**uality. Example: `MGT033+2`
UEQ	**U**nit **EQ**uality theorems in clause normal form (unsatisfiable clause sets).	Example: `RNG026-7`

division. System descriptions are in these competition proceedings [135] and on the CASC-J13 web site.

3 Infrastructure

3.1 Computers

The StarExec Miami computers used for the competition have two octa-core Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz CPUs, with hyperthreading (two threads per core), 256GiB memory, the Ubuntu 24.04.3 LTS operating system, with Linux kernel 6.8.0-71-generic. There were 30 computers available for CASC-J13, 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 [11, 27]. Systems can use all the cores on the CPU. In CASC-J13 all the demonstration division systems used the competition computers.

The StarExec Miami 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.

WWW TO HERE, AND ALSO GRANULARITY

3.2 Problems for the TPTP-based Divisions

Problems for the THF, FOF, FNT, and UEQ divisions are taken from the latest release of the Thousands of Problems for Theorem Provers (TPTP) problem library [125]. For CASC-J13 that was release v9.3.0. The TPTP version used for CASC is released only 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 competition.

Table 2: The ATP systems and entrants

ATP System	Divisions				Entrant (Associates)	Entrant's Affiliation
Connect++ 0.7.2		FOF	FNT		Dr Sean B Holden	University of Cambridge
CSI++ 1.0		FOF		UEQ	Guoyan Zeng (Yang Xu, Baojie Rui, Stephan Schulz, Peiyao Liu, Hailin Guo, Jun Liu, Shuwei Chen)	Xihua University
Drodi 4.1.1		FOF	FNT	UEQ	Oscar Contreras	Amateur Programmer
E 3.5.1	THF	FOF	FNT	UEQ	Stephan Schulz	DHBW Stuttgart
FindProof 0.1		FOF		UEQ	Nik Murzin	Wolfram Institute
FMB4J 0.1			FNT		Michael Rawson (Joe Hall)	University of Southampton
iProver 3.9.4		FOF	FNT	UEQ	Konstantin Korovin	University of Manchester
LEO-II 1.7.0	THF				Alexander Steen (Christoph Benzmüller)	University of Greifswald
Leo-III 1.9.0	THF				Alexander Steen (Christoph Benzmüller)	University of Greifswald
LisaST 0.9		*FOF*			Simon Guilloud	EPFL
Mace4 2026-6A			FNT		Jeff Machado (Larry Lesyna)	Independent Researcher
mrs 0.2.0		FOF		UEQ	Olivier Roland	Independent Researcher
Prover9 1109a		*FOF*			CASC (Bob Veroff, Bill McCune)	CASC fixed point
Prover9 2026-6A		FOF		UEQ	Jeff Machado (Larry Lesyna)	Independent Researcher
SATResetCoP 1.0		FOF			Martin Fixman (Fredrik Rømming, Sean Holden)	University of Cambridge
SPASS-SCL 0.1.1		FOF			Simon Schwarz (Yasmine Briefs, Lorenz Leutgeb, Christoph Weidenbach, Fabian Wobito)	Max Planck Institute for Informatics
SUPr 1.0		FOF			Teddy Kim (Adam Pease)	Naval Postgraduate School
Twee 2.7				UEQ	Nick Smallbone (Guy Axelrod)	Chalmers University of Technology
Vampire 4.8			*FNT*		CASC	CASC-29 winner
Vampire 5.0	*THF*	*FOF*		*UEQ*	CASC	CASC-30 winner
Vampire 5.0.1	THF	FOF	FNT	UEQ	Márton Hajdu (Filip Bártek, Ahmed Bhayat, Jonas Bodingbauer, Robin Coutelier, Matthias Hetzenberger, Petra Hozzová, Laura Kovács, Jack McKeown, Merisa Mustajbasic, Axel Polaczek, Jakob Rath, Michael Rawson, Giles Reger, Martin Riener, Martin Suda, Johannes Schoisswohl, Andrei Voronkov, Eva Maria Wagner)	TU Wien
VIP 1.71828459		FOF			Ilies Nokrani (David Delahaye, Clémence Gerardin)	Université Montpellier - LIRMM
Zipperp'n 2.1.9999	THF	FOF			Jasmin Blanchette (Alexander Bentkamp, Simon Cruanes, Petar Vukmirović)	Ludwig-Maximilians-Universität München

- 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) [156]. Problems with ratings in the range 0.21 to 0.99 are eligible. The upper bound of 0.99 excludes problems that probably would not be solved by any of the systems, and thus would not differentiate between the 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 not differentiate between the systems. Problems of lesser and greater ratings are made eligible if there are not enough problems with ratings in that range. There were none in CASC-J13.

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 obfuscated by:

- stripping out all comment lines, including the problem header
- randomly reordering the formulae/clauses (`include` directives are left before formulae, type declarations and definitions are left before the symbols' uses)
- randomly swapping the arguments of associative connectives, and randomly reversing implications
- randomly reversing 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 competition organizer. 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 [90]. 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 VSP list.
- In order to combat excessive tuning towards problems that are in the preceding 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 Time Limits

In the THF, FOF, FNT, and UEQ divisions a wall clock time limit is imposed for each problem. The minimal time limit for each problem is 120s 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 [156, 136]. The time limit is chosen within the range allowed according to the judgement of the competition organizers, and is announced at the competition. No CPU time limits are imposed, so it can be advantageous to use all the cores on the CPU when a wall clock time limit is used.

4 System Evaluation

For each ATP system, for each problem, the following items of data are recorded:

- whether or not the problem was solved
- whether or not a solution (proof or model) was output
- the CPU and wall clock times taken to solve the problem and the times taken to output the solution (taken from the time stamps prepended to each line of the system's `stdout` by StarExec's `runsolver` utility)
- the maximal number of parents in any inference

The systems are ranked at (only) the division level, according to the number of problems solved with an acceptable solution output. Ties are broken by the average time taken over problems solved. Trophies are awarded to the competition divisions' winners. 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. In the demonstration division the systems are not ranked, and no trophies are awarded.

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

- Proofs should not use a TPTP language that is more expressive than that of the problem.
- Proofs must be in TPTP format[2] [146]. This is checked using TPTP4X [102]. Entrants can do this in SystemB4TSTP[3].
- Proofs must be structurally correct:
  - Annotated formulae must be uniquely named.
  - Proofs must be acyclic.
  - Proofs must show only relevant inference steps.
  - Proofs must have formulae from the problem as leaves, and end at the conjecture (for axiomatic proofs) or `$false` formulae (for proofs by contradiction, e.g., CNF refutations and closed tableau).
  - Proofs that negate the conjecture must correctly annotate the step as `status(cth)` and have a a single parent with the role `conjecture`.
  - Inference steps must list exactly those parents that are used in the inference.
  - Proofs that make `assumption`s must propagate and eventually discharge the assumptions.

  Proof structure is checked using GDV [140]. This can be done in SystemOnTSTP.
- Translations from one form to another, e.g., FOF to CNF, must be adequately documented.
- Inference steps must be reasonably fine-grained. The maximal number of parents in any inference is considered in this regard.
- Proofs should graft in subproofs from external systems.
- An unsatisfiable set of ground instances of clauses is acceptable for establishing the unsatisfiability of a set of clauses.
- Models must be complete, documenting the domain and symbol maps. The domain and symbol maps may be specified by explicit ground lists (of mappings), or by any clear, terminating algorithm. Saturations are acceptable if their models are a subset of the models of the problem formulae. Use of the TPTP format for interpretations [147] is encouraged.

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

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 (excluding the previous year's winners that are earlier versions of competing systems). 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 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 (the reciprocal of the time taken to solve it) over the problems it solves, multiplied by the fraction of problems solved. 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 the problems it solves with a raw core usage greater than 1.0. The competition ran on octa-core computers, thus the maximal core usage was 8.0. The ability to solve problems quickly before multi-core search is used is also of interest, and the number of problems solved with a raw core usage less than 1.0 is also noted.
- The *granularity* measures the extent to which inference steps combine multiple parent formulae into a single inferred formula. The granularity of a proof is the maximal number of parents in an inference, and the overall granularity is the average over the the problems it solves with proof output.

5 System Registration, Delivery, and Execution

The ATP systems are registered for the competition by the registration deadline, using the system registration form, Systems can be entered at only the division level, and can be entered into more than one division. Entering many similar variants of a system is deprecated, and entrants may be required to limit the number of variants that they enter. Systems that rely essentially on running other ATP systems without adding value are deprecated, and such systems might be disallowed or moved to the demonstration division. 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.

The ATP systems are delivered to the competition organizers by the system delivery deadline, as StarExec `.tgz` installation packages. The installation packages may contain only the components necessary for running the system, i.e., not including source code, etc. The competition organizers install and test the packages on StarExec. Source code is delivered separately, under the trusting assumption that the installation package corresponds to the source code. The entrants must email a `.tgz` file of the source code and any files required for building the StarExec installation package to the competition organizers by the system delivery deadline.

After the competition all competition division systems' StarExec and source code packages are made available on the competition web site. This allows anyone to use the systems on StarExec, and to examine the source code. Entrants are encouraged to make a public release of their systems after the competition, so that users can enjoy the latest capabilities. 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. In the demonstration division the entrant specifies whether or not the source code is placed on the web site.

Execution of the ATP systems is controlled by StarExec. StarExec copies the systems and problems to the compute nodes before starting execution, and timing starts when execution starts. 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 before the competition by submitting non-theorems to the systems in the THF, FOF, and UEQ divisions, and theorems to the systems in the FNT division.

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. The execution of demonstration division systems is supervised by their entrants.

After the competition the systems are tested over the whole TPTP. If a system is found to be unsound, 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 disqualified. After the competition the systems' solutions are checked. If any of the solutions are found to be unacceptable, but before the competition report is published, then the ranking is revised. All disqualifications and ranking changes are explained in the competition report.

5.1 System Descriptions

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

- Architecture. This section introduces the system, and describes its calculus and inference rules.
- 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 6).
- Implementation. This section describes the implementation of the 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 predictions about the performance of the 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 7).

5.2 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 is required, and use of the TPTP format for interpretations is encouraged. The competition panel decides whether or not each system's solutions are acceptable (see Section 4).

Proof/model samples are required as follows:

- THF: `SET014^4`
- FOF: `SEU140+2`
- FNT: `NLP042+1` and `SWV017+1`
- UEQ: `BOO001-1`

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

6 System Properties

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.
- 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 [105] for this. For example

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

  or

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

  A system has solved a problem iff it outputs its status string within the time limit. The string specifying the status must be output before the start of a solution.
- 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
  ```

  A system has produced a solution iff it outputs its end-of-solution string within the time limit.
- 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 [146] is required, and use of the TPTP format for interpretations [147] is encouraged.
- Proofs are checked for syntax and structure (see Section 4).

Resource Usage

- Systems that run on the competition computers 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 process hierarchy, then one second later to all processes (even if they have disconnected from the process 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 files anywhere. If a system is terminated by a `SIGXCPU` or `SIGALRM` it may not leave 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.

7 The ATP Systems

These system descriptions were written by the entrants.

7.1 Connect++ 0.7.2

Dr Sean B 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 [60]. 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 [60], 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.

This is the first version of Connect++ that, in addition to using connection calculus, employs ideas from [64], for periodically grounding clauses and using a SAT solver to complete a proof by finding a finite unsatisfiable grounded set.

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. Since version 0.7.0 it has also implemented the standard TPTP format for recording connection proofs proposed in [145]. During the build process cmake reads a specified version of the CaDiCaL [13] SAT solver from its GitHub repository, builds a competition version as a library, and links directly to this to perform SAT-solving.

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. As new copies of clauses are often needed, clauses are themselves also retained during backtracking and reused where possible to minimize the need to make new ones.

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 [26], 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 0.7.0.

7.2 CSI++ 1.0

Guoyan Zeng
Xihua University, China

Architecture

CSI++ is an automated theorem prover for first-order logic, integrating contradiction separation inference with superposition-based reasoning. The system combines the CSI inference framework and the E (or GKC) prover in a cooperative architecture. CSI is a multi-layer inverse and parallel prover based on the Contradiction Separation Based Dynamic Multi-Clause Synergized Automated Deduction (S-CS) framework [165], while E and GKC mainly provide efficient reasoning capabilities for superposition, equality, and resolution. The cooperation mechanism works as follows. CSI and E (or GKC) are first applied sequentially to the given problem. If either component succeeds, the proof search terminates successfully. Otherwise, CSI generates additional inferred clauses, especially clauses with at most two literals and unit clauses, which are then supplied to the superposition reasoning component together with the original clause set for further proof search. This integration is intended to combine the strengths of both reasoning styles. CSI is effective at generating useful unit clauses and simplifying the search space, while the superposition component provides strong equality handling ability. Their cooperation aims to improve performance on difficult first-order theorem proving problems.

Strategies

The CSI inference component adopts strategies similar to those used in the standalone CSE 1.6 system, including clause/literal selection, strategy scheduling, and CSC strategies. In the integrated system, dedicated equality handling strategies of the original CSE component are disabled in favor of the superposition-based equality reasoning module. The main additional strategies in the integrated system include:

- Complementary ratio strategy: a measure and calculation method for estimating complementary relations between clauses, guiding clause selection and deduction path planning effectively.
- Portfolio strategy: different clause selection schemes are applied during different stages of proof search.

- Multi-Goal Extraction Strategy. We use the following strategy to extract the multiple goal literals inside the S-SC (called multi-goal extraction strategy). This literal selection strategy is used to get the inverse deduction goal clause.

Implementation

CSI++ is implemented mainly in C++, while Java is used for batch problem execution. The coordination and job dispatch between the contradiction separation inference module and the superposition reasoning module are implemented in C++.

Expected Competition Performance

We expect CSIPlusPlus to solve some difficult problems that cannot be solved by traditional superposition provers alone and to achieve competitive overall performance.

Acknowledgement: Development of CSIPlusPlus has been partially supported by the National Natural Science Foundation of China (NSFC) (Grant No. 62106206, 62206227), and the Key Project of Sichuan Science and Technology Innovation and Entrepreneurship Seeding Program (Grant No. 2024JDRC0084).

7.3 Drodi 4.1.1

Oscar Contreras
Amateur Programmer, Spain

Architecture

Drodi 4.1.1 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 [59].
- AVATAR architecture with a SAT solver [159].
- Limited Resource Strategy [71].
- Discrimination trees.
- KBO, non recursive and lexicographic reduction orderings. KBO has been rewriten using the polynomial-time algorithm in [49].
- Literal selection including lookahead as in [35].
- SInE distance for clauses and symbols as in [85].
- Layered clause selection as in [29].
- Stochastic strategy inspired in [86].
- Goal transformation for Unit Equality problems as in [82].
- SAT solver based subsumption and subsumption resolution inspired in [19].
- Learning data is now embedded in the executable. An external file with learning data is no longer required.
- 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 [71] saturation algorithms.
- A basic implementation of AVATAR architecture [159].

- Several literal and term reduction orderings.
- Several literal selection options [35].
- Several layered clause selection heuristics with adjustable selection ratios [29].
- Classical clause relevancy pruning.
- Drodi V4 has new strategy portfolio inspired (but not exactly equal than) by [18]. 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 [86].
- 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

Due to the SAT based subsumption and subsumption resolution Drodi 4.1.1 solves around 5FOF problems than last year's version. Due to new goal transformation Drodi solves around 30Also all syntax problems in the proof have been hopefully corrected. It is expected better results than in last year but probably not enough to improve the ranking position.

7.4 E 3.5.1

Stephan Schulz
DHBW Stuttgart, Germany

Architecture

E [75, 79, 81] 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 pre-processing 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 [160, 162]. 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 [12] 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-J13, 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 1615 different strategies have been thoroughly evaluated on various TPTP versions, 420 of which made it into the prover either into the automatic mode or into 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 [80]. Subsumption and contextual literal cutting (also known as subsumption resolution) is supported using feature vector indexing [78]. Superposition and backward rewriting use fingerprint indexing [77], 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 3.5 has a new strategy hacked into it manually at the last moment. If this works out, we expect performance to be at least as good as last year's version, though it might do better than last year in UEQ. The system is expected to perform well in most proof classes, but will at best complement top systems in the disproof classes.

7.5 FindProof 0.1

Nik Murzin
Wolfram Institute, USA

Architecture

FindProof [57] is an equational theorem prover for unit-equality problems, based on unfailing (ordered) Knuth-Bendix completion [3]. It continues the Waldmeister line of provers [34], which won the CASC unit-equality division at every competition in which it was ranked between 1997 and 2014; the completion loop, critical-pair selection, queue management, and normalisation reproduce the Waldmeister design, and on classical algebraic problems the default configuration reproduces Waldmeister's critical-pair selection sequence exactly.

The axioms are saturated towards a convergent rewrite system while the goals are kept normalised, and a proof is reported when the two sides of every goal are joined; a conjecture given as several negative units is proved as a multi-goal conjunction against one saturation. Equations that cannot be oriented by the reduction ordering are used unfailingly, so that both faces are superposed, which preserves refutational completeness. The Knuth-Bendix ordering and the lexicographic path ordering are supported, with symbol precedences generated automatically. Redundant critical pairs are discarded by ground-joinability testing, the connectedness criterion, forward and backward subsumption and demodulation, and right-hand-side interreduction. Proofs are reconstructed from the recorded completion trace, the closing rewrite chain re-derived against the final rule set, and rendered as a TPTP CNFRefutation.

The system is not restricted to unit equality. Full first-order input is handled by equationalization, in which propositions are Skolemized and encoded as equations over a Boolean-algebra axiomatisation, proved by the same completion core, and decoded back into a predicate-logic proof. The present entry competes in the UEQ division.

Strategies

The search is ordered by a weighted critical-pair selection heuristic over a priority queue, with a Waldmeister-style selection ratio that interleaves a first-in-first-out pick at a fixed cadence, so that fairness, and hence completeness, is retained. The default configuration is the Waldmeister strategy stack. The full strategy surface, some fifty-five switches covering the reduction ordering, critical-pair weighting, weight limits, redundancy criteria, goal treatment, and queue management, is exposed as command-line options, and named configurations modelled on the published defaults of Waldmeister, Twee, and Vampire are provided. In competition a fixed schedule of such configurations is run, each in its own child process under a slice of the wall-clock limit, stopping at the first proof. The schedule and its slice allocation are derived from coverage measurements on the problem sets of previous competitions; strategy selection is not tuned to individual problems.

Implementation

FindProof is a single self-contained C program with no external dependencies. Terms are represented as flatterms over a term bank; rewriting and subsumption retrieval use perfect discrimination trees; the critical-pair queue is a binary heap over a compact twelve-byte critical-pair encoding; and the completion trace is kept off-heap in a packed store from which proof objects are reconstructed on demand. Problems in TPTP CNF syntax [136] are read directly, with include directives resolved against the problem directory and the TPTP environment variable, and positive units taken as axioms and negative units as goals. Result lines follow the SZS ontology, with Satisfiable claimed only under a strategy with no completeness-breaking pruning and a ground goal. An optional Wolfram Language layer drives the same engine and adds proof objects verified by internal replay and the predicate-logic equationalization; it is a design feature, not a dependency, and the competition binary is the standalone C build. The system is available from the author.

Expected Competition Performance

FindProof is entered in the UEQ division. The Waldmeister default solves the standard equational classes, including Boolean and ternary Boolean algebra, group and ring theory, lattice theory, and combinatory logic, and the schedule adds coverage from ground-joinability and connectedness configurations on the associative-commutative classes where pure completion saturates slowly. The system is expected to trail the leading portfolio provers, whose unit-equality schedules have benefited from years of tuning, and a mid-field placement is anticipated.

7.6 FMB4J 0.1

Michael Rawson
University of Southampton, United Kingdom

Architecture

FMB4J is a finite model builder in the tradition of MACE. It performs a reduction from the question "does this first-order set of clauses have a model of size n?" to SAT. The value of k begins at 1 and counts upwards, looking for progressively-larger finite models.

Strategies

Only one strategy is attempted.

Implementation

The finite model builder is implemented in Java using SAT4J as a solver. FMB4J currently uses the Vampire system as a clausifier, with various flags that (i) axiomatise equality and (ii) may (not) improve FMB4J's performance on the generated CNF. It is available from:

`https://github.com/jh4n23/FMB4J`

Expected Competition Performance

This system was implemented as an undergraduate project (!). It is not the state of the art in finite model building. However, the use of Vampire as a frontend boosts performance considerably. It can solve around a third of the satisfiable fragment of TPTP in 1 second.

7.7 iProver 3.9.4

Konstantin Korovin
The University of Manchester, United Kingdom

Architecture

iProver [44, 22] is a theorem prover for quantified first-order logic with theories. iProver interleaves instantiation calculus Inst-Gen [45, 44, 28] with ordered resolution and superposition calculi [22]. iProver approximates first-order clauses using propositional abstractions that are solved using MiniSAT [26] 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 [32, 33]. First-order clauses are exchanged between calculi during the proof search.

Recent features in iProver include:

- Ground joinability and connectedness in the superposition calculus [24].
- Support for quantified reasoning with arithmetic and arrays.
- Gound joinability and connectedness [24].
- AC joinability and AC normalisation [23].
- Superposition calculus with simplifications including: demodulation, light normalisation, subsumption, subsumption resolution and global subsumption. iProver's simplification set up [22] 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 [26] and Z3 [21]. iProver accepts TFF, FOF, and CNF formats. Vampire [47, 67] and E prover [79] are used for proof-producing clausification of FOF/TFF problems. Vampire is also used for SInE axiom selection [36]. ¡!– and for theory axioms in the TFA division. –¿ iProver is available at:

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

An online version is available in iProver Live at:

`https://www.iprover-live.com/`

Compared to the previous year there have been: fixes in the proof output, a number of efficiency related fixes and updated schedules.

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. We also expect proofs to pass the GDV checker.

7.8 LEO-II 1.7.0

Alexander Steen
University of Greifswald, Germany

Architecture

LEO-II [8], the successor of LEO [7], is a higher-order ATP system based on extensional higher-order resolution. More precisely, LEO-II employs a refinement of extensional higher-order RUE resolution [6]. LEO-II is designed to cooperate with specialist systems for fragments of higher-order logic. By default, LEO-II cooperates with the first-order ATP system E [74]. LEO-II is often too weak to find

a refutation amongst the steadily growing set of clauses on its own. However, some of the clauses in LEO-II's search space attain a special status: they are first-order clauses modulo the application of an appropriate transformation function. Therefore, LEO-II launches a cooperating first-order ATP system every n iterations of its (standard) resolution proof search loop (e.g., 10). If the first-order ATP system finds a refutation, it communicates its success to LEO-II in the standard SZS format. Communication between LEO-II and the cooperating first-order ATP system uses the TPTP language and standards.

Strategies

LEO-II employs an adapted "Otter loop". Moreover, LEO-II uses some basic strategy scheduling to try different search strategies or flag settings. These search strategies also include some different relevance filters.

Implementation

LEO-II is implemented in OCaml 4, and its problem representation language is the TPTP THF language [9]. In fact, the development of LEO-II has largely paralleled the development of the TPTP THF language and related infrastructure [141]. LEO-II's parser supports the TPTP THF0 language and also the TPTP languages FOF and CNF.

Unfortunately the LEO-II system still uses only a very simple sequential collaboration model with first-order ATPs instead of using the more advanced, concurrent and resource-adaptive OANTS architecture [10] as exploited by its predecessor LEO.

The LEO-II system is distributed under a BSD style license, and it is available from

`http://www.leoprover.org`

Expected Competition Performance

LEO-II is not actively being developed anymore, hence there are no expected improvements to last year's CASC results.

7.9 Leo-III 1.8.0

Alexander Steen
University of Greifswald, Germany

Architecture

Leo-III [83], the successor of LEO-II [8], 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 [75, 79] 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. There is no time slicing, no strategy scheduling, or similar.

Implementation

¡style type="text/css"¿ .leopard font-variant: small-caps; ¡/style¿ Leo-III utilizes and instantiates the associated ¡span class="leopard"¿ LeoPARD¡/span¿ system platform [164] for higher-order (HO) deduction systems implemented in Scala (currently using Scala 2.13 and running on a JVM with Java ¿= 11). The prover makes use of ¡span class="leopard"¿LeoPARD¡/span¿'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 [14, 43]. Since version 1.6.X (X ¿= 0) Leo-III also accepts non-classical problem input represented in non-classical TPTP, see ...

```
<A HREF=‘‘https://tptp.org/NonClassicalLogic/’’>https://tptp.org/NonClassicalLogic/</A>
```

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 1.8.0 is, for all intents and purposes of CASC, equivalent to the version from previous years, except that some minor proof output bugs were fixed, and the support for reasoning in various quantified non-classical logics (not relevant to CASC) was 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).

7.10 Mace4 2026-6A

Jeff Machado
Independent Researcher, USA

Architecture

Mace4 [53], version 2026-6A [51], is a finite model finder for first-order logic with equality, based on William McCune's LADR (Library for Automated Deduction Research) codebase. Given a set of clauses or formulas, Mace4 searches for a finite interpretation that satisfies them. For a fixed domain size it ground-instantiates the problem and applies a decision procedure based on ground rewriting

modulo the input equalities, together with Davis-Putnam-style backtracking and negative propagation. Equalities are preserved through the search rather than flattened to propositional variables, so reasoning about function table cells stays first-order until a contradiction or a model is reached.

The `set(arithmetic)` command optionally enables an integer arithmetic interpretation of designated symbols, so that common operations such as sum and product are interpreted as such. This is provided as a convenience and is off by default.

Strategies

Mace4 starts at a small domain size (default 2) and iterates upward, building a ground model for each domain in turn, until a satisfying interpretation is found, the configured end_size is reached, or the wall-clock budget expires. Smallest-first iteration is effective on the finite-model problem profile, where most interpretations of interest are small and the search terminates quickly once the right domain size is reached. Within each domain, Mace4 prunes the search with least-number-heuristic (LNH) isomorphism reduction [166] to break domain symmetry, ground propagation, negative-assignment propagation (the neg_assign and neg_assign_near flags), and negative elimination (neg_elim_near). Propagation proceeds by ground rewriting and derivation of negated equalities, detecting inconsistency early. A cell-overflow safety break stops the domain iterator when the function tables for the next domain size would exceed available memory, preventing runaway expansion on problems with no small finite model. These pruning techniques are enabled by default.

For parallel search, the `-cores N` option (and `-casc`, which implies `-cores 8`) directs Mace4 to race up to N domain sizes at once rather than running them sequentially. Because the smallest model is the desired answer, Mace4 retains the smallest model found so far, keeps smaller sizes running while wall-clock time remains, and reports that model when it is proven minimal or when the time limit is reached. Racing sizes makes productive use of the available cores under a wall-clock limit, and the keep-smallest rule preserves the minimality of the reported interpretation.

Mace4 applies a single fixed configuration to every problem in a division, with the same command line for all problems. It performs no per-problem tuning, uses no machine learning, and stores no precomputed information about individual problems or their solutions. All techniques are general purpose.

Implementation

Mace4 is coded in C (C89), using the LADR libraries shared with Prover9. The 2026 version adds approximately 2,000 lines to McCune's 2009 Mace4 sources, concentrated in TPTP input handling, SZS status output, the TFI model emitter, the parallel domain-size scheduler, and competition-mode command-line plumbing, while remaining backward compatible with existing LADR input files. The ground search uses an estack-based representation of partial assignments to cell variables. The parallel scheduler is a single Mace4 process that forks one child per domain size; the children share the parsed and clausified problem in memory, and the parent collects results, retains the smallest model, and relays it for the winning size. The single-size search core is unchanged from McCune's original design; only this orchestration is new. Checkpoint and resume support lets the user halt and resume a search and recover from unplanned outages.

For TPTP input, Mace4 emits SZS status lines (`Satisfiable` for a satisfiable axiom set, `CounterSatisfiable` for a conjecture refuted by a finite model) and TFI-format interpretations in TPTP syntax, following the specification of Sutcliffe et al. [147]. Each interpretation gives the domain together with the function and predicate denotations, wrapped in SZS output delimiters for verification by tools such as AGMV. The `-ladr_out` option additionally produces the original LADR-format model for tools that consume the legacy representation.

Mace4 compiles and runs at the command line with simple, documented commands, and installation instructions are given on the website. It has been tested on a variety of platforms and runs on essentially all commonly used operating systems, including Linux, Windows, and macOS 10.4 or later. Mace4 will

also run directly in a modern web browser, with no download or compilation required; the code is fetched once and then executes on the local device, so an input file prepared in any text editor can be run and the results saved or verified without using the command line.

`https://prover9.org`

Expected Competition Performance

In the FNT division, Mace4 is expected to be effective on problems with small finite models: its home territory of algebra (group theory, ring theory, lattice theory) and combinatorics, where smallest-first iteration reaches the right domain size quickly and the parallel scheduler covers several sizes at once. Against the modern SAT-based and finite-model-building systems, whose propositional encodings scale to larger domains, Mace4 is expected to trail overall; internal evaluation at competition-style limits supports a modest overall solve count concentrated on the small-model profile. For problems that require large domains or have no finite model, Mace4 will not produce a result within the time limit. It reports SZS Timeout when the time budget is exhausted, SZS GaveUp when the configured domain range is exhausted or the next domain size would exceed available memory, and SZS MemoryOut when memory is exhausted during search.

7.11 mrs 0.2.0

Olivier Roland
Independent Researcher, France

Architecture

mrs 0.2.0 is an automated theorem prover for first-order logic with equality, implementing the superposition calculus [4]. within an Otter-style given-clause loop [75]. The core inference rules are ordered binary resolution, factoring, equality resolution, equality factoring, and superposition (into both terms and literals), oriented by a Knuth-Bendix Ordering (KBO) or Lexicographic Path Ordering (LPO) with dynamic, rarity-based symbol precedence. Clause splitting on non-Horn clauses is delegated to the AVATAR architecture [159], backed by the CaDiCaL CDCL SAT solver. EPR-structured problems are handled lazily through AVATAR splitting rather than eager ground pre-expansion, which previously caused memory exhaustion on large Effectively Propositional problems.

Term retrieval for unification and matching uses perfect discrimination trees that track variable bindings through traversal [54], eliminating false-positive candidates at the index level. Subsumption and subsumption resolution are accelerated by Feature Vector Indexing [76]. Redundancy elimination includes forward/backward demodulation, forward/backward subsumption, subsumption resolution, condensation, tautology deletion, and global subsumption with orphan elimination (removal of the entire derived subtree of a clause that is later found to be subsumed).

mrs runs a portfolio of independent search strategies concurrently on separate threads, one per available CPU core, each maintaining its own clause set; the first strategy to find a refutation signals the others to stop via a shared atomic flag. Strategies additionally share a pool of derived unit equalities across threads to accelerate sibling searches without duplicating work; each shared entry carries its full justifying ancestor chain back to the original problem's axioms, so a clause adopted from a sibling thread is spliced into the receiving thread's own proof record with a complete, locally self-consistent derivation rather than an opaque fact.

Strategies

For CASC, mrs selects a division-tuned, data-driven portfolio of 8 strategies (matching the 8-core StarExec hardware) via `--auto-schedule`, which classifies the input problem as FNE, FEQ, or UEQ

using rule-based syntactic checks (presence of equality literals, presence of function symbols of arity greater than 0, unit-equality-only clause sets) and dispatches to the matching `casc_*` schedule.

Each per-division priority order was derived empirically: every one of mrs's 15 base strategies (varying clause weight function, literal selection, term ordering, Set-of-Support depth, and AVATAR on/off) was run solo against a large corpus of representative FOF/UEQ problems at the official time limit, and a greedy set-cover algorithm selected the minimal-redundancy 8-strategy subset that maximizes problems solved when run in parallel. This tuning is strictly at the division level (FNE/FEQ/UEQ), never at the level of individual problems or their solutions, and generalizes to unseen problems of the same syntactic class. The baseline strategies use age-weight ratios (e.g., every 5th or 6th given-clause pick by FIFO age, the rest by weight) to balance breadth and depth of search, several distinct clause weight functions (including symbol-count, function-depth penalties, Horn-clause penalties, and conjecture-symbol boosting), and Set-of-Support restrictions that skip inferences between two clauses both far from the negated conjecture.

Implementation

mrs is implemented entirely in Rust (edition 2024), organized as a Cargo workspace of single-purpose crates: a zero-copy TPTP/TSTP parser built on the winnow parser-combinator library, a clausification pipeline (NNF/Skolemization/definitional CNF), a Robinson unification engine, the inference and ordering crate, the discrimination-tree/feature-vector indexing crate, the given-clause search loop and strategy scheduler, and a proof-extraction/TSTP-output crate. AVATAR splitting uses the cadical Rust bindings to the CaDiCaL SAT solver. mrs oes not depend on or invoke any external ATP system; all reasoning is performed in-process. mrs produces a TSTP-format refutation proof on success.

mrs is open source (MIT OR Apache-2.0) and available from:

`https://github.com/newca12/mrs`

Expected Competition Performance

mrs is entered in the FOF and UEQ divisions only. In local benchmarking against a representative CASC-30-era problem set (not run under official competition conditions), mrs solved 45/100 FNE-category, 98/400 FEQ-category, and roughly 39-40/300 UEQ problems, compared to reference local runs of Vampire 5.0.1 (82 FNE, 361 FEQ, 243 UEQ) and E 3.3.3 (67 FNE, 236 FEQ, 186 UEQ) on the same local harness. mrs is expected to be competitive with, but behind, the leading systems in both divisions.

7.12 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 [56]. 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 [55]. (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 [76] 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.

7.13 Prover9 2026-6A

Jeff Machado
Independent Researcher, USA

Architecture

Prover9 [53], version 2026-6A [51], is a resolution/paramodulation prover for first-order logic with equality, based on William McCune's LADR (Library for Automated Deduction Research) codebase. The system follows the "given clause" algorithm, in which not-yet-given clauses are available for rewriting and for other inference operations (the "Otter loop"). The prover provides positive ordered (and nonordered) resolution and paramodulation, negative ordered (and nonordered) resolution, factoring, positive and negative hyperresolution, UR-resolution, and demodulation (term rewriting). Term ordering options include LPO, RPO, or KBO. Selection of candidate clauses uses an age-weight ratio. The 2026 version adds native TPTP input parsing with automatic SZS status output, SInE [36] for large-theory problems, a machine-learned strategy selector, and a multi-core parallel portfolio scheduler. The underlying inference algorithms are unchanged from McCune's original design.

For TPTP input, proof output is in TPTP/TSTP format with SZS status lines and `CNFRefutation` delimiters. SIGXCPU and SIGALRM are handled for clean termination under both CPU and wall-clock time limits. In competition mode (`-casc T`), the system automatically configures TPTP output, multi-core scheduling, the ML strategy selector, and wall-clock timeout T for the entire portfolio run.

Strategies

Prover9 2026-6A employs a portfolio of 96 strategies executed via a sliding-window parallel scheduler. A compact decision tree (147 nodes, 74 leaves) selects and orders strategies for each input problem from a fixed vector of general syntactic features: formula and clause counts, symbol counts and symbols-per-formula, term-depth and literal-count estimates, an axiom-to-symbol ratio, presence of equality, and unit/Horn classification. The selector reads only these structural features and is given no problem name, file identifier, hash, or solution. The tree was built by running the 96 strategies over the TPTP library and learning which regions of this feature space favor which strategies. Its 74 leaves partition problems into broad structural classes, so the learned feature-to-strategy mapping is coarse by construction and generalizes across families of problems rather than fitting any one of them; it is expected to apply equally to new, unseen problems. The same trained selector is applied unchanged to all problems in all divisions. Strategy 0 (auto_default) runs McCune's original automatic mode, which adjusts inference rules and parameters according to syntactic properties of the input (presence of equality, non-Horn clauses, etc.). The remaining 95 specialist strategies vary parameters such as term ordering (KBO/LPO/RPO), age-weight ratio, inference rule selection (hyperresolution, UR-resolution, paramodulation), demodulation settings, literal selection, and SInE tolerance levels.

The parallel scheduler assigns each strategy a time slice and runs multiple strategies concurrently using a sliding-window approach. An initial breadth phase exposes the problem to diverse strategies, followed by a priority-based depth phase that allocates remaining time to the most promising strategies based on progress metrics such as given and kept clause counts and memory use.

SInE is enabled automatically for TPTP problems with more than 128 axioms, reducing large-theory problems to manageable subsets before search. Several specialist strategies vary the SInE tolerance to explore different axiom subsets.

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

Implementation

Prover9 is coded in C (C89), using the LADR libraries. Some of the code descended from EQP [55]. The 2026 version adds approximately 10,000 lines to the Prover9-specific search code in McCune's original codebase and approximately 13,500 lines to the shared LADR libraries (TPTP parser and writer, SInE, demodulation enhancements, and proof-output machinery), while maintaining strict backward compatibility with existing LADR input files. Memory addressing and management are significantly improved from the 2009 version; over a trillion generated clauses have been processed effectively.

Indexing uses 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 is used for forward and backward nonunit subsumption. To accelerate lookups at high-fanout index nodes, both FPA/Path and discrimination tree nodes can switch from linear child scanning to hash-table lookup; the switchover threshold is a runtime parameter (`fpa_hash_threshold`, default 4 children, and `discrim_hash_threshold`, off by default) so strategies can tune it per problem family. The hash-table code paths are unconditionally compiled in.

The multi-core scheduler uses anonymous shared mmap regions and signals (SIGSTOP/SIGCONT/SIGALRM) for cooperative child process management, with open_memstream for capturing child proof output. Each child process runs an independent Prover9 instance with its own strategy configuration. Checkpoint and resume support lets the user halt and resume a search and recover from

unplanned outages.

Prover9 may be compiled and run easily at the command line with documented, simple commands familiar to most users. Installation instructions are displayed conveniently on the website. Prover9 has been tested on a variety of platforms, and runs on essentially all commonly used computer operating systems including Linux, Windows, and macOS versions 10.4 or later. Prover9 will also run directly from any modern browser connected to the internet, without the need for the user to download or compile any code. Code execution takes place on the local device. Any local text editor may be used to prepare the input file, and the command line may be entirely avoided. Prover9 results are directly reported to the screen, and may be saved, verified, or processed according to the user's needs.

`https://prover9.org`

Expected Competition Performance

In the FOF division, Prover9 2026-6A is expected to clearly outperform the fixed Prover9 1109a reference, on the strength of the multi-core portfolio, ML strategy selection, and SInE axiom selection; internal evaluation on rating-banded TPTP samples at competition-style limits shows an improvement of roughly a third over the 2009 baseline. The portfolio provides robustness across diverse problem types, and SInE brings large-theory problems into reach. In the UEQ division, Prover9's paramodulation and demodulation engine, a descendant of the EQP system that solved the Robbins Problem [55], remains a capable, well-tested equational reasoner, with the portfolio adding coverage through varied term orderings and demodulation strategies. A respectable showing is expected, though the leading dedicated equational systems are expected to remain ahead. Overall, we expect Prover9 2026-6A to finish comfortably ahead of the fixed Prover9 1109a reference, by roughly a third on internal evaluation, while remaining well behind the modern superposition portfolio systems. Its particular strengths are equational and algebraic problems.

7.14 SATResetCoP 1.0

Martin Fixman
University of Cambridge, United Kingdom

Architecture

SATResetCoP is a connection prover for untyped first-order formulas without equality. It is coupled to an incremental SAT solver: ground instances of clauses encountered during tableau construction are accumulated as propositional clauses, and a theorem is reported when that clause set is propositionally unsatisfiable. The current SAT model is used to rank eligible tableau steps. When a tableau attempt has generated new ground clauses, it is reset to explore a fresh part of the search space while retaining the accumulated SAT state. This design builds on SATCoP [64] and the Connections framework [72].

Strategies

At a tableau dead end, a reset at the current depth is selected when new ground clauses have been generated. Otherwise the depth bound is increased and a new tableau is started. Thus, local backtracking is traded for diverse clause generation while the SAT state is retained. The depth bound is increased by at most one when an iteration has stopped producing new ground clauses, following the iterative-deepening discipline used by leanCoP [61]. This keeps early searches shallow while allowing deeper tableaux when they are required. The strategy is intended to generate a sufficiently informative set of ground clauses for a SAT refutation, rather than to close one particular tableau.

Implementation

SATResetCoP is implemented in Python using a fork of the Connections library [72], which provides a framework for creating and testing connection provers. The fork adds a classical SAT-assisted calculus with leanCoP-style cuts and a strategy interface for selecting different prover controllers. These changes are currently being backported to newer versions of the library. The incremental SAT state is maintained by CaDiCaL through the pydical Python binding. The packaged system also depends on python-sat for optional experimental strategies. The final version can be downloaded from GitHub:

`https://github.com/mfixman/satresetcop`

Expected Competition Performance

As a first version, SATResetCoP is not expected to be competitive with mature saturation provers such as Vampire in the general FOF category. Strong performance is expected against other connection provers on problems for which shallow tableau exploration quickly generates an unsatisfiable set of ground propositional clauses. The experimental results suggest particular strength on the TPTP SYN, SWC, and SWV categories, with improvements also seen on GEO and NUM. No category-specific tuning is used.

7.15 SPASS-SCL 0.1.1

Simon Schwarz
Max Planck Institute for Informatics, Germany

Architecture

SPASS-SCL-FOL 0.1.1 is a prototype implementing SCL(FOL) [16] for first-order logic without equality. The focus of this year's development was not on improving performance, but rather on supporting proofs and models, implementing a first prototype integration of propositional reasoning, and clarifying the structure of the main loop. Unlike the classical main loops of saturation-based provers, this loop is centered on selecting ground literals to extend the trail and handling stuck states, which represent partial saturations.

Strategies

SPASS-SCL-FOL 0.1.1 comes without any complex strategy parameters. The main strategy parameter of SCL(FOL) is the selection of ground literals that should extend the trail. Currently, only very simple strategies are implemented. The main strategy selects ground literals by building linear models [15] and considers literals that conflict during model building. As a prototype, SPASS-SCL-FOL does not use a portfolio.

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 0.1.1 is still an early prototype for studying properties of SCL and has not been tuned for competition performance, so performance will be poor.

7.16 SUPr 1.0

Teddy Kim
Naval Postgraduate School, USA

Architecture

SUPr 1.0 is a saturation-based theorem prover built as a native component of the Sigma knowledge engineering environment (Rust implementation) (SigmaKEE-rs) [63]. SUPr is optimized to reason directly over the Suggested Upper Merged Ontology (SUMO) [58, 62], with additional support to solve standalone TPTP problems. The core search procedure is a given-clause saturation loop implementing ordered resolution and superposition with selective literal selection, forward and backward demodulation, and a Knuth-Bendix reduction ordering. Clauses and their constituent terms are represented in a content-addressed, hash-consed arena. This content based addressing scheme is structured to allow batched, hardware accelerated unification via binary arithmetic.

Strategies

SUPr is built specifically for reasoning over SUMO; consequently, SUPr's proving strategy depends on heavy caching of supporting evidence at the time of ingest for new axioms and use-case-specific semantic decoding.

- Given that SUMO is comprised of hundreds of thousands of axioms, axiom selection is key to proving any problem leveraging the ontology. Novel pre-caching of SUMO Inference Engine (SInE) occurence and trigger indices [36] allows for fast per-problem axiom selection. Axioms are additionally ranked by structural relevance to the conjecture [48] and those which rank highly but dropped by SInE are rescued and introduced to the problem.
- Beneath the given-clause loop, a generalized Datalog($\neg$). model-construction system extracts and caches Horn-clause shaped rules from the axiom set to drive a background ground fact materialization loop to potentially short circuit easy queries against the ontology.
- Individually developed and sound subsystems are used to short circuit the given-clause loop for common SUMO queries like event calculus and transitive closures.
- Search-loop tuning is organized as a small number of strategies running as concurrent, threads which race for the solution. Threads share a single clause stack that remains deconflicted due to the atomicity of the system's content-based hashing scheme.
- SUPr uses a content-based hashing system by which terms are identified by a 64-bit unsigned integer hash based on its content and structure. In addition to serving as a unique identifier for the lifetime of the ontology, it is structured such that unification and paramodulation within the saturation loop can be conducted as bitwise arithmetic and parallelized via "Single Instruction, Multiple Data" (SIMD) on a single processor core.

Implementation

SUPr 1.0 is implemented in Rust. It is shipped as a part of the SigmaKEE-rs v2.0.0+ command line interface application. The system accepts both SUO-KIF and TPTP (CNF, FOF, TFF) syntax as input dialects, parsing either into a common internal sentence which is cached into a Lightning Memory Database (LMDB), which itself is a shared memory B+ Tree which allows for large axiom spaces to share a common memory addressing scheme for systems lacking the resource to hold a large ontology simultaneously in memory. Proofs are emitted in the afforementioned dialects, graphviz compatible notation, or procedurally generated plain language prose. SigmaKEE-rs can be downloaded from its GitHub page:

`https://github.com/ontologyportal/sigma-rs`

Expected Competition Performance

SUPr 1.0 is competing in the first order category. It is expected to perform well on problems drawn from or resembling the SUMO ontology, where its axiom-selection feedback loop and discharge-subsystem prologue are specifically tuned, and on standalone TPTP problems small enough, or easy enough, for at least one portfolio lane to solve quickly. Harder, uniformly slow problems in equational domains are expected to be a weaker category: superposition-based simplification is present but comparatively lightly indexed next to mature systems in this competition.

7.17 Twee 2.7

Nick Smallbone
Chalmers University of Technology, Sweden

Architecture

Twee 2.7 [82] 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 [52] 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 [52], and discharges individual cases by rewriting modulo a variable ordering.

This year's version adds preliminary support for discovering interesting term patterns during proof search [1].

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.

7.18 Vampire 4.8

Michael Rawson
TU Wien, Austria

There have been a number of changes and improvements since Vampire 4.7, although it is still the same beast. Most significant from a competition point of view are long-awaited refreshed strategy schedules. As a result, several features present in previous competitions will now come into full force, including new rules for the evaluation and simplification of theory literals. A large number of completely new features and improvements also landed this year: highlights include a significant refactoring of the substitution tree implementation, the arrival of encompassment demodulation to Vampire, and support for parametric datatypes.

Vampire's higher-order support has also been re-implemented from the ground up. The new implementation is still at an early stage and its theoretical underpinnings are being developed. There is currently no documentation of either.

Architecture

Vampire [47] 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 the Inst-gen calculus and a MACE-style finite model builder [68]. Splitting in resolution-based proof search is controlled by the AVATAR architecture which uses a SAT or SMT solver to make splitting decisions [159, 65]. 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 [69]. 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 4.8 provides a very large number of options for strategy selection. The most important ones are:

- Choices of saturation algorithm:
  - Limited Resource Strategy [71]
  - DISCOUNT loop
  - Otter loop
  - Instantiation using the Inst-Gen calculus
  - MACE-style finite model building with sort inference
- Splitting via AVATAR [159]
- 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 [29].
- 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 [66].
  - Use of Z3 with AVATAR to restrict search to ground-theory-consistent splitting branches [65].
  - Specialised theory instantiation and unification [70].
  - Extensionality resolution with detection of extensionality axioms

The schedule for the new HOL implementation was developed using Snake, a strategy schedule construction tool described in more detail last year. The Snake schedule will this year fully embrace Vampire randomisation support [86] and, in particular, every strategy will independently shuffle the input problem, to nullify (in expectation) the effect of problem scrambling done by the organisers.

Implementation

Vampire 4.8 is implemented in C++. It makes use of fixed versions of Minisat and Z3. See the website for more information and access to the GitHub repository:

`https://vprover.github.io/`

Expected Competition Performance

Vampire 4.8 is the CASC-29 FNT winner.

7.19 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 [46] for reasoning with linear arithmetic, with further VIRAS extensions [73] for quantifier elimination.
- Partial redundancy calculi [31]
- Stabilised and greatly enhanced runtime-specialised unidirectional term ordering checks [30]
- A variant of the ground joinability redundancy elimination rule, used in forward simplification.
- Subsumption (resolution) via code trees.
- Integration of the CaDiCaL SAT solver [13] 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.

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 [68]. Splitting in saturation-based proof search is controlled by the AVATAR architecture which uses a SAT or SMT solver to make splitting decisions [159, 65]. 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 [69]. 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 [71]
  - DISCOUNT loop
  - Otter loop
  - MACE-style finite model building with sort inference
- Splitting via AVATAR [159]
- 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 [29].
- 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 [66].
  - Use of Z3 with AVATAR to restrict search to ground-theory-consistent splitting branches [65].
  - Specialised theory instantiation and unification [70].
  - 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 website for more information and access to the GitHub repository:

`https://vprover.github.io/`

Expected Competition Performance

Vampire 5.0 is the CASC-30 THF, FOF, and UEQ winner.

7.20 Vampire 5.0.1

Márton Hajdu
TU Wien, Austria

Vampire 5.0.1 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 landed, but for the competition we mention:

- Clause-selection guidance based on Graph and Recursive Neural Networks [87].
- Higher-order support has been mostly merged with mainline Vampire, therefore our HOL support enjoys all competition-winning improvements over the last years, while remaining efficient on non-higher-order problems.
- Detailed propositional proof output via the CaDiCaL SAT solver [13].
- More detailed output, including proofs that are (more) TSTP-compliant.
- 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++20.

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

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 [68]. Splitting in saturation-based proof search is controlled by the AVATAR architecture which uses a SAT or SMT solver to make splitting decisions [159, 65]. 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 [69]. Vampire implements many preprocessing transformations, including the SInE axiom selection algorithm for large theories and blocked clause elimination.

Strategies

Vampire 5.0.1 provides a very large number of options for strategy selection. The most important ones are:

- Choices of saturation algorithm:
  - Limited Resource Strategy [71]
  - DISCOUNT loop
  - Otter loop
  - MACE-style finite model building with sort inference
- Splitting via AVATAR [159]
- 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 [29]. Optionally, instead of age-weight alternation, clause selection can be fully delegated to a guiding neural network [87].
- 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 [66].
- Use of Z3 with AVATAR to restrict search to ground-theory-consistent splitting branches [65].
- Specialised theory instantiation and unification [70].
- Extensionality resolution with detection of extensionality axioms

For CASC, we prepared several clause-selection guiding networks trained on the TPTP library. As described in [87], we were independently checking test performance on randomly held-out fraction of the library during the training. The test performance, although lower than the train performance, is greater than that of the baseline, which means that generalization to problems similar to those used in training is achieved.

Implementation

Vampire 5.0.1 is implemented in C++. It makes use of fixed versions of Minisat, CaDiCaL, GMP, VIRAS, and Z3. See the website for more information and access to the GitHub repository:

`https://vprover.github.io/`

The neural clause-selection guidance relies on the LibTorch library (v2.10), which is statically linked into the competition submission so that it can run on StarExec without requiring the library to be installed there.

Expected Competition Performance

Vampire 5.0.1 should be an improvement on the previous version. A reasonably strong performance across all divisions is therefore expected.

7.21 VIP 1.718

Ilies Nokrani
Université Montpellier - LIRMM, France

Architecture

VIP 2026 is an automatic theorem prover for first-order logic in TPTP FOF syntax. It was developed under a short time constraint as a standalone prover, with extensive LLM-assisted implementation. The submitted system does not call another ATP system as a backend.

The core is a given-clause saturation loop. Problems are parsed from TPTP files, include directives are expanded, formulae are clausified, and clauses are processed by resolution-style and superposition-style rules. The main inferences are binary resolution, factoring, equality resolution, equality factoring, and equality-oriented paramodulation/superposition steps. Standard simplifications include tautology deletion, demodulation, forward simplification, subsumption, subsumption resolution, condensation, and contextual literal cutting. VIP contains a simple legacy engine and a more recent engine with stronger equality handling, indexing, and clause selection; the competition portfolio combines both. The system also includes SInE-style axiom selection, conservative AVATAR-style splitting, layered clause selection, and limited deduction-modulo-inspired preprocessing for selected definitional equivalences.

Strategies

VIP uses a sequential portfolio. The main CASC-oriented portfolio is `casc-150`. Each stage receives a fixed fraction of the available time and runs with fixed options for the FOF division. The submitted StarExec script uses the same command line for every FOF problem.

Strategy scheduling is based only on general syntactic characteristics of the input, such as equality density, clause shape, unit-clause ratio, polarity patterns, and symbol occurrence information. It is not based on problem names, file paths, comments, TPTP headers, or stored information about individual problems or their solutions. The portfolio includes FEQ-oriented equality stages, FNE recovery stages using the legacy engine, SInE axiom-selection stages with different widths, legacy-guided modern stages, layered age/weight passive selection, and conservative splitting stages. The submitted StarExec script does not hardcode the CASC time limit; it accepts the announced wall-clock budget as a wrapper argument or environment value. The default generated-clause limit is 75000.

Implementation

VIP is implemented in OCaml and is built with Dune. The executable installed in the StarExec package is `vip`; `ip` is kept as a compatibility alias in the source tree. The delivered binary is statically linked.

TPTP include files are resolved according to the standard TPTP convention: first relative to the problem file, and otherwise relative to the TPTP root supplied by the TPTP environment variable. Internal data structures include feature-vector style indices for subsumption candidates, discrimination-style indices for rewriting candidates, and KBO-style term ordering for equality reasoning.

In competition mode, VIP writes all result information to standard output, emits an SZS status line, and, when a refutation is found, prints a TPTP/TSTP-style proof delimited by SZS output markers. The run script expects the problem file as its first argument and relies on the StarExec/TPTP environment for include-file resolution and resource enforcement. VIP is available from:

`https://github.com/delahayd/vip`

Expected Competition Performance

VIP is expected to solve a useful subset of FOF theorem problems, with better performance on unsatisfiable problems where axiom selection, equality simplification, and staged saturation interact well. It is not expected to match mature ATP systems such as Vampire, E, or Zipperposition.

¿P¿

7.22 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 [5]. 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 [25], and Boolean reasoning [163]. 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 [161]. The initial calculus and main loop were imitations of an earlier version of E [75]. With the implementation of higher-order superposition, the main loop had to be adapted to deal with possibly infinite sets of unifiers [160].

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 [160]. 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.

8 Conclusion

CASC-J13 was the thirty-first large scale competition for classical logic ATP systems. The competition 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] G. Axelrod, M. Johansson, and N. Smallbone. Twitch: Learning Abstractions for Equational Theorem Proving. In A. Biere, C. Lutz, and S. Negri, editors, *Proceedings of the 13th International Joint Conference on Automated Reasoning*, number 16688 in Lecture Notes in Artificial Intelligence, page To appear, Berlin, Germany, 2026. Springer.

[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] L. Bachmair and H. Ganzinger. Rewrite-based Equational Theorem Proving with Selection and Simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[5] 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, Berlin, Germany, 2021. Springer.

[6] C. Benzmüller. Extensional Higher-order Paramodulation and RUE-Resolution. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 399–413, Berlin, Germany, 1999. Springer.

[7] C. Benzmüller and M. Kohlhase. LEO - A Higher-Order Theorem Prover. In C. Kirchner and H. Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction*, number 1421 in Lecture Notes in Artificial Intelligence, pages 139–143, Berlin, Germany, 1998. Springer.

[8] 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, Berlin, Germany, 2008. Springer.

[9] C. Benzmüller, F. Rabe, and G. Sutcliffe. THF0 - The Core TPTP Language for Classical 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 491–506, Berlin, Germany, 2008. Springer.

[10] C. Benzmüller, V. Sorge, M. Jamnik, and M. Kerber. Combined Reasoning by Automated Cooperation. *Journal of Applied Logic*, 6(3):318–342, 2008.

[11] D. Beyer, S. Löwe, and P. Wendler. Reliable Benchmarking: Requirements and Solutions. *International Journal on Software Tools for Technology Transfer*, 21:1–29, 2019.

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

[13] 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 Com-*

puter Aided Verification, number 14681 in Lecture Notes in Computer Science, pages 133–152, Berlin, Germany, 2024. Springer.

[14] 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, Berlin, Germany, 2013. Springer.

[15] M. Bromberger, F. Krasnopol, S. Moehle, and C. Weidenbach. 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. SCL(FOL) Revisited. 10.48550/arXiv.2302.05954, 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. In R. Piskac and Z. Rakamaric, editors, *Proceedings of the 37th International Conference on Computer Aided Verification*, number 15933 in Lecture Notes in Computer Science, pages 57–71, 2025.

[18] 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.

[19] R. Coutelier, J. Rath, M. Rawson, and L. Kovács. SAT Solving for Variants of First-order Subsumption. *Formal Methods in System Design*, 67:27–70, 2024.

[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, Berlin, Germany, 2008. Springer.

[22] 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.

[23] 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, Berlin, Germany, 2021. Springer.

[24] 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.

[25] 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, Berlin, Germany, 2021. Springer.

[26] 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, Berlin, Germany, 2004. Springer.

[27] J. Fichte, T. Geibinger, M. Hecher, and M. Schlögel. Parallel Empirical Evaluations: Resilience despite Concurrency. In M. Wooldridge, J. Dy, and S. Natarajan, editors, *Proceedings of the 38th AAAI Conference on Artificial Intelligence*, volume 38 of *Proceedings of the AAAI Conference*

on Artificial Intelligence, pages 8004–8012, Palo Alto, USA, 2024. AAAI Press.

[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] 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.

[30] 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*, number 15943 in Lecture Notes in Artificial Intelligence, pages 552–569, Berlin, Germany, 2025. Springer.

[31] 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*, number 15943 in Lecture Notes in Artificial Intelligence, pages 532–551, Berlin, Germany, 2025. Springer.

[32] 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.

[33] 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.

[34] T. Hillenbrand and B. Löchner. The Next Waldmeister Loop. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, number 2392 in Lecture Notes in Artificial Intelligence, pages 486–500, Berlin, Germany, 2002. Springer.

[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, Berlin, Germany, 2011. Springer.

[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, Berlin, Germany, 2021. Springer.

[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, Berlin, Germany, 2017. Springer.

[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, Berlin, Germany, 2018. Springer.

[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] 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.

[45] 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, Berlin, Germany, 2013.

[46] 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, Berlin, Germany, 2023. Springer.

[47] 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, Berlin, Germany, 2013. Springer.

[48] Q. Liu and Y. Xu. Axiom Selection Over Large Theory Based on New First-order Formula Metrics. *Applied Intelligence*, 52(2):1793–1807, 2022.

[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 LPO. *Journal of Artificial Intelligence Tools*, 15(1):53–80, 2006.

[51] J.P. Machado and L. Lesyna. Enhanced LADR-2026. https://prover9.org.

[52] 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, Berlin, Germany, 1990. Springer.

[53] W.W. McCune. Prover9. http://www.cs.unm.edu/ mccune/prover9/.

[54] W.W. McCune. Experiments with Discrimination-Tree Indexing and Path Indexing for Term Retrieval. *Journal of Automated Reasoning*, 9(2):147–167, 1992.

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

[56] W.W. McCune. Otter 3.3 Reference Manual. Technical Report ANL/MSC-TM-263, Argonne National Laboratory, Argonne, USA, 2003.

[57] N. Murzin. FindProof: Equational Theorem Proving in the Wolfram Language. Wolfram Institute Technical Note, 2026.

[58] I. Niles and A. Pease. Towards a Standard Upper Ontology. In C. Welty and B. Smith, editors, *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems*, pages 2–9, 2001.

[59] 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,

Amsterdam, The Netherlands, 2001.

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

[61] J. Otten. 20 Years of leanCoP - An Overview of the Provers. 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 4–22, 2023.

[62] A. Pease. *Ontology: A Practical Guide*. Articulate Software Press, 2011.

[63] A. Pease and S. Schulz. Knowledge Engineering for Large Ontologies with Sigma KEE 3.0. 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 519–525, Berlin, Germany, 2014. Springer.

[64] 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*, number 12842 in Lecture Notes in Artificial Intelligence, pages 250–265, Berlin, Germany, 2021. Springer.

[65] 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, Manchester, United Kingdom, 2016. EasyChair.

[66] 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, Berlin, Germany, 2021. Springer.

[67] 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, Berlin, Germany, 2015. Springer.

[68] 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, Berlin, Germany, 2016. Springer.

[69] 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, Manchester, United Kingdom, 2016. EasyChair.

[70] 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, Berlin, Germany, 2018. Springer.

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

[72] F. Rømming, J. Otten, and S. Holden. Connections: Markov Decision Processes for Classical, Intuitionistic, and Modal Connection Calculi. 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 107–118, 2023.

[73] 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, Manchester, United Kingdom, 2024. EasyChair.

[74] S. Schulz. A Comparison of Different Techniques for Grounding Near-Propositional CNF Formulae. In S. Haller and G. Simmons, editors, *Proceedings of the 15th International FLAIRS Conference*, pages 72–76, Palo Alto, USA, 2002. AAAI Press.

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

[76] 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, Berlin, Germany, 2004. Springer.

[77] 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, Berlin, Germany, 2012. Springer.

[78] 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, Berlin, Germany, 2013.

[79] 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, Berlin, Germany, 2013. Springer.

[80] S. Schulz. Shared Terms and Cached Rewriting. In K. Korovin, S. Schulz, and M. Rawson, editors, *Proceedings of the 14th and 15th International Workshops on the Implementation of Logics*, number 21 in Kalpa Publications in Computing, pages 18–33, Manchester, United Kingdom, 2025. EasyChair.

[81] 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, Berlin, Germany, 2019. Springer.

[82] 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, Berlin, Germany, 2021. Springer.

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

[84] 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.

[85] 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, Manchester, United Kingdom, 2019. EasyChair.

[86] 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.

[87] M. Suda. Efficient Neural Clause-Selection Reinforcement. In C. Barrett and U. Waldmann, editors, *Proceedings of the 30th International Conference on Automated Deduction*, number 15943 in Lecture Notes in Artificial Intelligence, pages 403–422, Berlin, Germany, 2025. Springer.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[102] G. Sutcliffe. TPTP, TSTP, CASC, etc. In V. Diekert, M. Volkov, and A. Voronkov, editors, *Proceedings of the 2nd International Symposium on Computer Science in Russia*, number 4649 in Lecture Notes in Computer Science, pages 6–22, Berlin, Germany, 2007. Springer.

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

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

[105] 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.

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

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

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

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

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

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

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

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

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

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

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

[117] G. Sutcliffe. The CADE-24 Automated Theorem Proving System Competition - CASC-24. *AI Communications*, 27(4):405–416, 2014.

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

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

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

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

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

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

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

[125] 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.

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

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

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

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

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

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

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

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

[134] G. Sutcliffe. Proceedings of the CADE-29 ATP System Competition. Online, 2023. http://tptp.org/CASC/29/Proceedings.pdf.

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

[136] G. Sutcliffe. Stepping Stones in the TPTP World. 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 30–50, 2024.

[137] G. Sutcliffe. Proceedings of the CADE-30 ATP System Competition. Online, 2025. http://tptp.org/CASC/30/Proceedings.pdf.

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

[139] G. Sutcliffe. The CADE-30 Automated Theorem Proving System Competition - CASC-30. *AI Communications*, page To appear, 2026.

[140] G. Sutcliffe and D. Belfiore. Semantic Derivation Verification. In I. Russell and Z. Markov, editors, *Proceedings of the 18th International FLAIRS Conference*, pages 641–646, Palo Alto, USA, 2005. AAAI Press.

[141] G. Sutcliffe and C. Benzmüller. Automated Reasoning in Higher-Order Logic using the TPTP THF Infrastructure. *Journal of Formalized Reasoning*, 3(1):1–27, 2010.

[142] G. Sutcliffe and M. Desharnais. The CADE-28 Automated Theorem Proving System Competition

- CASC-28. *AI Communications*, 34(4):259–276, 2022.

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

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

[145] G. Sutcliffe, S. Holden, and M. Baksys. The TPTP Format for Clausal Connection Tableaux. In C. Nalon, M. Hajdu, and M. Suda, editors, *Proceedings of the 10th Workshop on Practical Aspects of Automated Reasoning*, CEUR Workshop Proceedings, page To appear, 2026.

[146] 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, Berlin, Germany, 2006. Springer.

[147] G. Sutcliffe, A. Steen, P. Fontaine, and L. Kondylidou. The TPTP Format for Interpretations. In C. Nalon, M. Hajdu, and M. Suda, editors, *Proceedings of the 10th Workshop on Practical Aspects of Automated Reasoning*, CEUR Workshop Proceedings, page To appear, 2026.

[148] 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.

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

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

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

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

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

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

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

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

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

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

[159] 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.

[160] 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, Berlin, Germany, 2021. Springer.

[161] 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.

[162] 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, Berlin, Germany, 2023. Springer.

[163] 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.

[164] 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, Berlin, Germany, 2015. Springer.

[165] 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.

[166] J. Zhang. Constructing Finite Algebras with FALCON. *Journal of Automated Reasoning*, 17(1):1–22, 1996.