IJCAR 2008

4th International Joint Conference on Automated Reasoning

Sydney, Australia, August 10–15, 2008

CASC-J4

The 4th
2 = A NUMBER
1 = A NUMBER
2 = 1
10 - 15 August 2008
http://2008.IJCAR.org
The 4th International Joint Conference on Automated Reasoning, Sydney, Australia, 10-15 August 2008

The 4th IJCAR ATP System Competition

Geoff Sutcliffe, Panel, Entrants

August 13

The 4th IJCAR ATP System Competition (CASC-J4)

Geoff Sutcliffe
University of Miami, USA

Abstract

The CADE ATP System Computer (CASC) evaluates the performance of sound, fully automatic, classical first-order logic, ATP systems. The evaluation is in terms of the number of problems solved, the number of acceptable proofs and models produced, and the average runtime for problems solved, in the context of a bounded number of eligible problems chosen from the TPTP problem library, and a specified time limit for each solution attempt. The 4th IJCAR ATP System Competition (CASC-J4) was held on 13th August 2008. The design of the competition and it's rules, and information regarding the competing systems, are provided in this report.

1 Introduction

The CADE conferences are the major forum for the presentation of new research in all aspects of automated deduction. In order to stimulate ATP research and system development, and to expose ATP systems within and beyond the ATP community, the CADE ATP System Competition (CASC) is held at each CADE conference. CASC-J4 was held on 13th August 2008, as part of the 4th International Joint Conference on Automated Reasoning[1], in Sydney, Australia. It is the thirteenth competition in the CASC series (lucky for some) [SS97a, SS98d, SS99, Sut00b, Sut01b, SSP02, SS03, SS04, Sut05b, Sut06b, Sut07b, Sut08].

CASC evaluates the performance of sound, fully automatic, classical first-order logic, ATP systems. The evaluation is in terms of:

- the number of problems solved,
- the number of acceptable proofs and models produced, and
- the average runtime for problems solved;

in the context of:

- a bounded number of eligible problems, chosen from the TPTP problem library [SS98c], and
- specified time limits on solution attempts.

Twenty-seven ATP systems and variants, listed in Table 1, entered into the various competition and demonstration divisions. The winners of the CASC-21 (the previous CASC) divisions were automatically entered into those divisions, to provide benchmarks against which progress can be judged (the competition archive provides access to the systems' executables and source code).

The design and procedures of this CASC evolved from those of previous CASCs [SS97c, SS97b, SS98a, SS98b, Sut99, Sut00a, Sut01a, Sut02, Sut03, Sut04, Sut05a, Sut06a, Sut07a]. Important changes for this CASC were:

- The LTB - Large Theory Batch - division was added. Each category of this division uses a theory in which there are many functors and predicates, many axioms of which typically only a few are required for the proof of a theorem, and many theorems to be proved using a common core set of axioms. The problems of each category are provided to the ATP systems as a batch, allowing the

[1]CADE was a constituent conference of IJCAR, hence CASC-"J4".

Table 1: The ATP systems and entrants

ATP System	Divisions	Entrants	Affiliation
CHewTPTP 1.0	FNT (demo)	Eric McGregor (Christopher Lynch)	Clarkson University
Darwin 1.3	EPR	CASC	*CASC-21 EPR winner*
E 1.0pre	FOF FNT CNF SAT EPR UEQ LTB	Stephan Schulz	Technische Universität München
EP 1.0pre	FOF* LTB*		*E 1.0pre variant*
E-Darwin 1.0	CNF EPR	Björn Pelzer	University Koblenz-Landau
E-KRHyper 1.1	FOF FNT CNF SAT EPR LTB	Björn Pelzer (Peter Baumgartner, Alexander Fuchs, Cesare Tinelli)	University Koblenz-Landau (NICTA, The University of Iowa)
Equinox 3.0	FOF CNF UEQ	Koen Claessen	Chalmers University of Technology
Infinox 0.1	FNT SAT (demo)	Koen Claessen (Ann Lillieström)	Chalmers University of Technology
iProver 0.5	FOF FNT CNF SAT EPR LTB	Konstantin Korovin	The University of Manchester
MaLARea 0.3	LTB*	Josef Urban	Charles University in Prague
MetaProver 1.0	FNT SAT	Matthew Streeter	Carnegie Mellon University
Metis 2.1	FOF* FNT* CNF SAT EPR UEQ LTB*	Joe Hurd	Galois, Inc.
Muscadet 3.0	FOF* LTB*	Dominique Pastre	Université René Descartes Paris-5
OSHL-S 0.1	FOF FNT CNF EPR	Hao Xu (David Plaisted)	University of North Carolina at Chapel Hill
Otter 3.3	FOF CNF UEQ	CASC (William McCune)	CASC (Argonne National Laboratory)
Paradox 1.3	SAT	CASC	*CASC-21 SAT winner*
Paradox 2.2	FNT*	CASC	*CASC-21 FNT* winner*
Paradox 3.0	FNT* SAT EPR	Koen Claessen	Chalmers University of Technology
randoCoP 1.1	FOF* LTB*	Jens Otten (Thomas Raths)	University of Potsdam
SInE 0.3	LTB*	Krystof Hoder	Charles University in Prague
SInE-VD 0.3	LTB (demo)	Krystof Hoder	Charles University in Prague
Vampire 8.1	CNF	CASC	*CASC-21 CNF winner*
Vampire 9.0	FOF*	CASC	*CASC-21 FOF* winner*
Vampire 10.0	FOF* CNF EPR UEQ LTB*	Andrei Voronkov	University of Manchester
Waldmeister 806	UEQ	CASC	*CASC-21 UEQ winner*
wgandalf 0.98	CNF EPR UEQ LTB*	Tanel Tammet	Tallinn University of Technology
Zenon 0.5.0	FOF*	Damien Doligez	INRIA

A * superscript on a division indicates participation in the division's proof/model class - see Section 2.

ATP systems to load and preprocess the common core set of axioms just once, and to share logical and control results between proof searches. Articulate Software provided $3000 of prize money for the SMO category of the LTB division.

- The FPE category - FOF Pure Equality - was split off from the FEQ category.
- The CNF proof class and SAT model class were discontinued.
- The distinguished strings output to indicate what solution has been found, or that no conclusion has been reached, had to be different for:
  - Proved conjectures of FOF problems
  - Disproved conjectures of FOF problems
  - Unsatisfiable sets of formulae (FOF problems without conjectures) and unsatisfiable set of clauses (CNF problems)
  - Satisfiable sets of formulae (FOF problems without conjectures) and satisfiable set of clauses (CNF problems)
- The distinguished strings output to delimit the start and end of proofs/models had to be different for:
  - Proofs (SZS output forms `Proof`, `Refutation`, `CNFRefutation`)
  - Models (SZS output forms `Model`, `FiniteModel`, `InfiniteModel`, `Saturation`)
- An open source license is explicitly encouraged for entered systems.

The competition organizer is Geoff Sutcliffe. The competition is overseen by a panel of knowledgeable researchers who are not participating in the event; the panel members were Alessandro Armando, Christoph Benzmüller, and John Slaney. The rules, specifications, and deadlines are absolute. Only the panel has the right to make exceptions. The competition was run on computers provided by the Department of Computer Science at the University of Manchester. The CASC-J4 WWW site provides access to resources used before, during, and after the event: `http://www.tptp.org/CASC/J4`

2 Divisions

CASC is run in divisions according to problem and system characteristics. There are *competition* divisions in which systems are explicitly ranked, and a *demonstration* division in which systems demonstrate their abilities without being formally ranked. Some divisions are further divided into problem categories, which make it possible to analyze, at a more fine grained level, which systems work well for what types of problems. The problem categories have no effect on the competition rankings, which are made at only the division level.

2.1 The Competition Divisions

The competition divisions are open to ATP systems that meet the required system properties described in Section 6.1. Systems that rely essentially on running other ATP systems without adding value are deprecated; the competition panel may disallow or move such systems to the demonstration division.

Each competition division uses problems that have certain logical, language, and syntactic characteristics, so that the ATP systems that compete in the division are, in principle, able to attempt all the problems in the division.

The **FOF** division: First-order form non-propositional theorems (axioms with a provable conjecture). The FOF division has two problem categories:

- The **FNE** category: FOF with No Equality
- The **FEQ** category: FOF with some (but not pure) Equality

- The **FPE** category: FOF Pure Equality

The **FNT** division: First-order form non-propositional non-theorems (axioms with an unprovable conjecture, and satisfiable axioms sets). The FNT division has two problem categories:

- The **FNN** category: FNT with no Equality
- The **FNQ** category: FNT with Equality

The **CNF** division: Clause normal form really non-propositional theorems (unsatisfiable clause sets), but not unit equality problems (see the UEQ division below). *Really non-propositional* means with an infinite Herbrand universe. The CNF division has five problem categories:

- The **HNE** category: Horn with No Equality
- The **HEQ** category: Horn with some (but not pure) Equality
- The **NNE** category: Non-Horn with No Equality
- The **NEQ** category: Non-Horn with some (but not pure) Equality
- The **PEQ** category: Pure Equality

The **SAT** division: Clause normal form really non-propositional non-theorems (satisfiable clause sets). The SAT division has two problem categories:

- The **SNE** category: SAT with No Equality
- The **SEQ** category: SAT with Equality

The **EPR** division: Effectively propositional clause normal form theorems and non-theorems (clause sets). *Effectively propositional* means non-propositional with a finite Herbrand Universe. The EPR division has two problem categories:

- The **EPT** category: Effectively Propositional Theorems (unsatisfiable clause sets)
- The **EPS** category: Effectively Propositional non-theorems (Satisfiable clause sets)

The **UEQ** division: Unit equality clause normal form really non-propositional theorems (unsatisfiable clause sets).

The **LTB** division: First-order form non-propositional theorems (axioms with a provable conjecture) from large theories, presented to the ATP systems in batches. A large theory has many functors and predicates, has many axioms of which typically only a few are required for the proof of a theorem, and many theorems to be proved using a common core set of axioms (but not all of the common core axioms are in every problem). The batch presentation allows the ATP systems to load and preprocess the common core set of axioms just once, and to share logical and control results between proof searches. The LTB division has three problem categories:

- The **CYC** category: Problems taken from the Cyc contribution to the `CSR` domain of the TPTP. These are problems `CSR025` to `CSR074`.
- The **MZR** category: Problems taken from the Mizar Problems for Theorem Proving (MPTP) contribution to the TPTP. These are problems `ALG214` to `ALG234`, `CAT021` to `CAT037`, `GRP618` to `GRP653`, `LAT282` to `LAT380`, `SEU406` to `SEU451`, and `TOP023` to `TOP048`.
- The **SMO** category: Problems taken from the Suggested Upper Merged Ontology (SUMO) contribution to the `CSR` domain of the TPTP. These are problems `CSR075` to `CSR109`.

Section 3.2 explains what problems are eligible for use in each division and category. Section 4 explains how the systems are ranked in each division.

2.2 The Demonstration Division

ATP systems that cannot run in the competition divisions for any reason can be entered into the demonstration division. Demonstration division systems can run on the competition computers, or the computers can be supplied by the entrant. Computers supplied by the entrant may be brought to CASC, or may be accessed via the internet.

The entry specifies which competition divisions' problems are to be used. The results are presented along with the competition divisions' results, but may not be comparable with those results. The systems are not ranked and no prizes are awarded.

3 Infrastructure

3.1 Computers

The non-LTB division computers are Dell computers, each having:

- Intel Core 2 Duo E4600, 2.4GHz CPU
- 1GB RAM
- Linux 2.6 operating system

The LTB division computers are Hewlett-Packard computers, each having:

- Intel Pentium 4, 2.80GHz CPU
- 1GB RAM
- Linux 2.6 operating system

3.2 Problems

3.2.1 Problem Selection

The problems were from the TPTP problem library, version v3.5.0. The TPTP version used for the competition is not released until after the system delivery deadline, so that new problems have not seen by the entrants.

The problems have to meet certain criteria to be eligible for selection:

- The TPTP uses system performance data to compute problem difficulty ratings, and from the ratings classifies problems as one of [SS01]:
  - Easy: Solvable by all state-of-the-art ATP systems
  - Difficult: Solvable by some state-of-the-art ATP systems
  - Unsolved: Solvable by no ATP systems
  - Open: Theoremhood unknown

  Difficult problems with a rating in the range 0.21 to 0.99 are eligible. Problems of lesser and greater difficulty ratings might also be eligible in some divisions. Performance data from systems submitted by the system submission deadline is used for computing the problem ratings for the TPTP version used for the competition.
- The TPTP distinguishes versions of problems as one of standard, incomplete, augmented, especial, or biased. All except biased problems are eligible.

The problems used are randomly selected from the eligible problems at the start of the competition, 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.

- The selection mechanism is biased to select problems that are new in the TPTP version used, until 50% of the problems in each category have been selected, after which random selection (from old and new problems) continues. The actual percentage of new problems used depends on how many new problems are eligible and the limitation on very similar problems.

3.2.2 Number of Problems

The minimal numbers of problems that have to be used in each division and category, to ensure sufficient confidence in the competition results, are determined from the numbers of eligible problems in each division and category [GS96] (the competition organizers have to ensure that there is sufficient CPU time available to run the ATP systems on this minimal number of problems). The minimal numbers of problems is used in determining the CPU time limit imposed on each solution attempt - see Section 3.3.

A lower bound on the total number of problems to be used is determined from the number of computers available, the time allocated to the competition, the number of ATP systems to be run on the competition computers over all the divisions, and the CPU time limit, according to the following relationship:

$$NumberOfProblems = \frac{NumberOfComputers * TimeAllocated}{NumberOfATPSystems * CPUTimeLimit}$$

It is a lower bound on the total number of problems because it assumes that every system uses all of the CPU time limit for each problem. Since some solution attempts succeed before the CPU time limit is reached, more problems can be used.

The numbers of problems used in the categories in the various divisions is (roughly) proportional to the numbers of eligible problems than can be used in the categories, after taking into account the limitation on very similar problems.

The numbers of problems used in each division and category are determined according to the judgement of the competition organizers.

3.2.3 Problem Preparation, non-LTB divisions

In order to ensure that no system receives an advantage or disadvantage due to the specific presentation of the problems in the TPTP, the tptp2X utility (distributed with the TPTP) is used to:

- rename all predicate and function symbols to meaningless symbols
- randomly reorder the clauses and literals in CNF problems
- randomly reorder the formulae in FOF problems
- randomly reverse the equalities in UEQ problems
- remove equality axioms that are not needed by the ATP systems
- add equality axioms that are needed by the ATP systems
- output the problems in the formats required by the ATP systems. (The clause type information, one of `axiom`, `hypothesis`, or `conjecture`, may be included in the final output of each formula.)

Further, to prevent systems from recognizing problems from their file names, symbolic links are made to the selected problems, using names of the form `CCCNNN-1.p` for the symbolic links, with `NNN` running from `001` to the number of problems in the respective division or category. The problems are specified to the ATP systems using the symbolic link names.

In the demonstration division the same problems are used as for the competition divisions, with the same tptp2X transformations applied. However, the original file names are retained.

3.2.4 Problem Preparation, LTB division

The problems are in their original TPTP form (which may provide information that helps find solutions), with `include` directives. There is consistent symbol usage between problems in each category, but there may not be consistent axiom naming between problems (although there obviously is for axioms from the same `included` file)

3.3 Resource Limits

In the competition divisions, CPU and wall clock time limits are imposed. A minimal CPU time limit of 240 seconds per problem is used. The maximal CPU time limit per problem is determined using the relationship used for determining the number of problems, with the minimal number of problems as the $NumberOfRoblems$. The CPU time limit is chosen as a reasonable value within the range allowed, and is announced at the competition. The wall clock time limit is imposed in addition to the CPU time limit, to limit very high memory usage that causes swapping. The wall clock time limit is double the CPU time limit. In the non-LTB competition divisions the time limits are imposed individually on each solution attempt. In the LTB division the total time limits (the individual time limits multiplied by the number of problems) are imposed on each category.

In the demonstration division, each entrant can choose to use either a CPU or a wall clock time limit, whose value is the CPU time limit of the competition divisions.

4 System Evaluation

For each ATP system, for each problem, three items of data are recorded: whether or not a solution was found, the CPU time taken, and whether or not a solution (proof or model) was output on `stdout`. The systems are ranked from this performance data. All the divisions have an assurance ranking class, ranked according to the number of problems solved (a "yes" output, giving an assurance of the existence of a proof/model). The FOF, FNT, and LTB divisions additionally have and a proof/model ranking class, ranked according to the number of problems solved with an acceptable proof/model output. Ties are broken according to the average CPU times over problems solved. All systems are automatically ranked in the assurance classes, and are ranked in the proof/model classes if they output acceptable proofs/models. A system that wins a proof/model ranking class might also win the corresponding assurance ranking class. In the competition divisions class winners are announced and prizes are awarded.

- Articulate Software has provided $3000 of prize money for the SMO category of the LTB division. (Employees of Articulate Sofware, its subcontractors, and funded partners, are not eligible for this prize money.) In each ranking class the winner will receive $750, the second place $500, and the third place $250.

The competition panel decides whether or not the systems' proofs and models are acceptable for the proof/model ranking classes. The criteria include:

- Derivations must be complete, starting at formulae from the problem, and ending at the conjecture (for axiomatic proofs) or a $false$ formula (for proofs by contradiction, including CNF refutations).
- For proofs of FOF problems by CNF refutation, the conversion from FOF to CNF must be adequately documented.
- Derivations must show only relevant inference steps.
- Inference steps must document the parent formulae, the inference rule used, and the inferred formula.

- Inference steps must be reasonably fine-grained.
- In the LTB division the proofs must be in TPTP format.
- 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, function maps, and predicate maps. The domain, function maps, and predicate maps may be specified by explicit ground lists (of mappings), or by any clear, terminating algorithm.

In the assurance ranking classes the ATP systems are not required to output solutions (proofs or models). However, systems that do output solutions to `stdout` are highlighted in the presentation of results.

If a system is found to be unsound during or after the competition, but before the competition report is published, and it cannot be shown that the unsoundness did not manifest itself in the competition, then the system is retrospectively disqualified. At some time after the competition, all high ranking systems in each division are tested over the entire TPTP. This provides a final check for soundness (see Section 6.1 regarding soundness checking before the competition). At some time after the competition, the proofs and models from the winners of the proof/model ranking classes are checked by the panel. If any of the proofs or models are unacceptable, i.e., they are significantly worse than the samples provided, then that system is retrospectively disqualified. All disqualifications are explained in the competition report.

5 System Entry

To be entered into CASC, systems have to be registered using the CASC system registration form. No registrations are accepted after the registration deadline. For each system entered, an entrant has to be nominated to handle all issues (including execution difficulties) arising before and during the competition. The nominated entrant must formally register for CASC. However, it is not necessary for entrants to physically attend the competition.

Systems can be entered at only the division level, and can be entered into more than one division (a system that is not entered into a competition division is assumed to perform worse than the entered systems, for that type of problem - wimping out is not an option). Entering many similar versions of the same system is deprecated, and entrants may be required to limit the number of system versions that they enter. The division winners from the previous CASC are automatically entered into their divisions, to provide benchmarks against which progress can be judged.

It is assumed that each entrant has read the WWW pages related to the competition, and has complied with the competition rules. Non-compliance with the rules could lead to disqualification. A "catch-all" rule is used to deal with any unforseen 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.

5.1 System Description

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

- Architecture. This section introduces the ATP system, and describes the calculus and inference rules used.
- Strategies. This section describes the search strategies used, why they are effective, and how they are selected for given problems. Any strategy tuning that is based on specific problems' characteristics must be clearly described (and justified in light of the tuning restrictions described in Section 6.1).

- Implementation. This section describes the implementation of the ATP system, including the programming language used, important internal data structures, and any special code libraries used.
- Expected competition performance. This section makes some predictions about the performance of the ATP system in each of the divisions and categories in which the system is competing.
- References.

The system description has to be emailed to the competition organizers by the system description deadline. The system descriptions, along with information regarding the competition design and procedures, form the proceedings for the competition.

5.2 Sample Solutions

For systems in the proof/model classes representative sample solutions must be emailed to the competition organizers before the sample solutions deadline. Proof samples for the FOF and LTB proof classes must include a proof for `SYN075+1`. Model samples for the the FNT model class must include models for `MGT019+2` and `SWV010+1`. The sample solutions must illustrate the use of all inference rules. A key must be provided if any non-obvious abbreviations for inference rules or other information are used.

6 System Requirements

6.1 System Properties

Systems are required to have the following properties:

- Systems have to run on a single locally provided standard UNIX computer (the *competition computers* - see Section 3.1). ATP systems that cannot run on the competition computers can be entered into the demonstration division.
- Systems must be fully automatic, i.e., any command line switches have to be the same for all problems.
- Systems must be sound. At some time before the competition all the systems in the competition divisions are tested for soundness. Non-theorems are submitted to the systems in the FOF, CNF, EPR, UEQ, and LTB divisions, and theorems are submitted to the systems in the FNT, SAT and EPR divisions. Finding 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 soundness testing eliminates the possibility of a system simply delaying for some amount of time and then claiming to have found a solution. At some time after the competition, all high ranking systems in the competition divisions are tested over the entire TPTP. This provides a final check for soundness. For systems running on entrant supplied computers in the demonstration division, the entrant must perform the soundness testing and report the results to the competition organizers.
- Systems do not have to be complete in any sense, including calculus, search control, implementation, or resource requirements.
- Systems must be executable by a single command line, using an absolute path name for the executable, which might not be in the current directory. In the non-LTB divisions the command line arguments are the absolute path name of a symbolic link as the problem file name, the time limit (if required by the entrant), and entrant specified system switches (the same for all problems). In the LTB division the command line arguments are the absolute path name of a file containing pairs of absolute problem file names and absolute output file names (where the output for the problem

must be written), the time limit (if required by the entrant), and entrant specified system switches. No shell features, such as input or output redirection, may be used in the command line. No assumptions may be made about the format of the problem file name.

- The systems that run on the competition computers have to be interruptable by a `SIGXCPU` signal, so that the CPU time limit can be imposed on each solution attempt, and interruptable by a `SIGALRM` signal, so that the wall clock time limit can be imposed on each solution attempt. For systems that create multiple processes, the signal is sent first to the process at the top of the hierarchy, then one second later to all processes in the hierarchy. The default action on receiving these signals is to exit (thus complying with the time limit, as required), but systems may catch the signals and exit of their own accord. If a system runs past a time limit this is noticed in the timing data, and the system is considered to have not solved that problem.
- In the non-LTB divisions all solution output must be to `stdout`. In the LTB division all solution output must be to the named output file for each problem (output to `stdout` and `stderr` will be redirected to `/dev/null`).
- When terminating of their own accord, the systems have to output a distinguished string (specified by the entrant) on `stdout` indicating either that a solution has been found, or that no conclusion has been reached. The distinguished strings the problem status should use the SZS ontology, in a line starting `SZS status`. For example

  ```
  SZS status Unsatisfiable for SYN075+1
  ```

  or

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

  Regardless of whether the SZS status values are used, the distinguished strings must be different for:
  - Proved theorems of FOF problems (SZS status `Theorem`)
  - Disproved conjectures of FNT problems (SZS status `CounterSatisfiable`)
  - Unsatisfiable sets of formulae (FOF problems without conjectures) and unsatisfiable set of clauses (CNF problems) (SZS status `Unsatisfiable`)
  - Satisfiable sets of formulae (FNT problems without conjectures) and satisfiable set of clauses (SAT problems) (SZS status `Satisfiable`)

  The first such string is recognized, and accepted as the system's claimed result.
- When outputing proofs/models for the proof/model ranking classes, the start and end of the proof/-model must be identified by distinguished strings (specified by the entrant). The distinguished strings should specify the precise output form named in the SZS ontology, using lines starting `SZS output start` and `SZS output end`. For example

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

  Regardless of whether the SZS output forms are used, the distinguished strings must be different for:
  - Proofs (SZS output forms `Proof`, `Refutation`, `CNFRefutation`)
  - Models (SZS output forms `Model`, `FiniteModel`, `InfiniteModel`, `Saturation`)

  The string specifying the problem status must be output before the start of a proof/model.

- If an ATP system terminates of its own accord, it may not leave any temporary or other output files. If an ATP system is terminated by a `SIGXCPU` or `SIGALRM`, it may not leave any temporary or other output files anywhere other than in `/tmp`. Multiple copies of the ATP systems have to be executable concurrently on different machines but in the same (NFS cross mounted) directory. It is therefore necessary to avoid producing temporary files that do not have unique names, with respect to the machines and other processes. An adequate solution is a file name including the host machine name and the process id.
- 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. The limit is at least 10MB per system.
- The precomputation and storage of any information specifically about TPTP problems is not allowed. Strategies and strategy selection based on the characteristics of a few specific TPTP problems are not allowed, i.e., strategies and strategy selection must be general purpose and expected to extend usefully to new unseen problems. If automatic strategy learning procedures are used, the learning must ensure that sufficient generalization is obtained, and that no learning at the individual problem level is performed.
- The system's performance must be reproducible by running the system again.

Entrants must ensure that their systems execute in a competition-like environment, according to the system checks described in Section 6.4. Entrants are advised to perform these checks well in advance of the system delivery deadline. This gives the competition organizers time to help resolve any difficulties encountered. Entrants will not have access to the competition computers.

6.2 System Delivery

For systems running on the competition computers, entrants must email an installation package to the competition organizers by the system delivery deadline. The installation package must be a `.tar.gz` file containing the system source code, any other files required for installation, and a `ReadMe` file. The `ReadMe` file must contain:

- Instructions for installation
- Instructions for executing the system
- Format of problem files, in the form of tptp2X format and transformation parameters.
- Command line, using `%s` and `%d` to indicate where the problem file name and CPU time limit must appear.
- The distinguished strings output.

The installation procedure may require changing path variables, invoking `make` or something similar, etc, but nothing unreasonably complicated. All system binaries must be created in the installation process; they cannot be delivered as part of the installation package. The system is installed onto the competition computers by the competition organizers, following the instructions in the `ReadMe` file. Installation failures before the system delivery deadline are passed back to the entrant. (i.e., delivery of the installation package before the system delivery deadline provides an opportunity to fix things if the installation fails!). After the system delivery deadline no further changes or late systems are accepted.

For systems running on entrant supplied computers in the demonstration division, entrants must deliver a source code package to the competition organizers by the start of the competition. The source code package must be a `.tgz` file containing the system source code.

After the competition all competition division systems' source code is made publically available on the CASC WWW site. In the demonstration division, the entrant specifies whether or not the source code is placed on the CASC WWW site. An open source license is encouraged.

6.3 System Execution

Execution of the ATP systems on the competition computers is controlled by a `perl` script, provided by the competition organizers. The jobs are queued onto the computers so that each computer is running one job at a time. In the non-LTB divisions, all attempts at the Nth problems in all the divisions and categories are started before any attempts at the (N+1)th problems. In the LTB division all attempts in each category in the division are started before any attempts at the next category.

During the competition a `perl` script parses the systems' outputs. If any of an ATP system's distinguished strings are found then the CPU time used to that point is noted. A system has solved a problem iff it outputs its termination string within the CPU time limit, and a system has produced a proof/model iff it outputs its end-of-proof/model string within the CPU time limit. The result and timing data is used to generate an HTML file, and a WWW browser is used to display the results.

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

6.4 System Checks

- Check: The ATP system can run on a computer that has the same configuration as the competition computers. The competition computers' configuration, obtained from `uname`, is:

  ```
  > uname -mp -sr
  Linux 2.6.25.9-76.fc9.i686 i686 i686
  ```

  If the ATP system requires any special software, libraries, etc, which not part of a standard installation, the competition organizers must be told in the system registration.

- Check: The ATP system can be run by an absolute path name for the executable.

  ```
  prompt> pwd
  /home/tptp
  prompt> which MyATPSystem
  /home/tptp/bin/MyATPSystem
  prompt> /home/tptp/bin/MyATPSystem /home/tptp/TPTP/Problems/SYN/SYN075-1.p
  SZS status Unsatisfiable for SYN075-1
  ```

- Check: The ATP system accepts an absolute path name of a symbolic link as the problem file name.

  ```
  prompt> cd /home/tptp/tmp
  prompt> ln -s /home/tptp/TPTP/Problems/SYN/SYN075-1.p CCC001-1.p
  prompt> cd /home/tptp
  prompt> /home/tptp/bin/MyATPSystem /home/tptp/tmp/CCC001-1.p
  SZS status Unsatisfiable for CCC001-1
  ```

- Check: The ATP system makes no assumptions about the format of the problem file name.

  ```
  prompt> ln -s /home/tptp/TPTP/Problems/GRP/GRP001-1.p \_foo-Blah
  prompt> /home/tptp/bin/MyATPSystem \_foo-Blah
  SZS status Unsatisfiable for _foo-Blah
  ```

- Check: The ATP system can run under the `TreeLimitedRun` program (sources are available from the CASC WWW site).

```
prompt> which TreeLimitedRun
/home/tptp/bin/TreeLimitedRun
prompt> /home/tptp/bin/TreeLimitedRun -q0 200 400 /home/tptp/bin/MyATPSystem
        /home/tptp/tmp/CCC001-1.p
TreeLimitedRun: ------------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 200s
TreeLimitedRun: WC  time limit is 400s
TreeLimitedRun: PID is 4867
TreeLimitedRun: ------------------------------------------------
SZS status Unsatisfiable for CCC001-1
FINAL WATCH: 147.8 CPU 150.0 WC
```

- Check: The ATP system's CPU time can be limited using the `TreeLimitedRun` program.

```
prompt> /home/tptp/bin/TreeLimitedRun -q0 10 20 /home/tptp/bin/MyATPSystem
        /home/tptp/tmp/CCC001-1.p
TreeLimitedRun: ------------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 10s
TreeLimitedRun: WC  time limit is 20s
TreeLimitedRun: PID is 5827
TreeLimitedRun: ------------------------------------------------
CPU time limit exceeded
FINAL WATCH: 10.7 CPU 13.1 WC
```

- Check: The ATP system's wall clock time can be limited using the `TreeLimitedRun` program.

```
prompt> /home/tptp/bin/TreeLimitedRun -q0 20 10 /home/tptp/bin/MyATPSystem
        /home/tptp/tmp/CCC001-1.p
TreeLimitedRun: ------------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 20s
TreeLimitedRun: WC  time limit is 10s
TreeLimitedRun: PID is 5827
TreeLimitedRun: ------------------------------------------------
Alarm clock
FINAL WATCH: 9.7 CPU 10.1 WC
```

- Check: The system outputs a distinguished string when terminating of its own accord.

```
prompt> /home/tptp/bin/TreeLimitedRun -q0 200 400 /home/tptp/bin/MyATPSystem
        /home/tptp/tmp/CCC001-1.p
TreeLimitedRun: ------------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 200s
TreeLimitedRun: WC  time limit is 400s
TreeLimitedRun: PID is 5827
TreeLimitedRun: ------------------------------------------------
SZS status Unsatisfiable for CCC001-1
FINAL WATCH: 147.8 CPU 150.0 WC
```

Similar checks should be made for the cases where the system gives up.

- Check: The system outputs distinguished strings at the start and end of its solution.

```
prompt> /home/tptp/bin/TreeLimitedRun -q0 200 400 /home/tptp/bin/MyATPSystem
        -output_proof /home/tptp/tmp/CCC001-1.p
TreeLimitedRun: -----------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 200s
TreeLimitedRun: WC  time limit is 400s
TreeLimitedRun: PID is 5827
TreeLimitedRun: -----------------------------------------------
SZS status Unsatisfiable for CCC001-1
SZS output start CNFRefutation for CCC001-1
    ... acceptable proof/model here ...
SZS output end CNFRefutation for CCC001-1
FINAL WATCH: 147.8 CPU 150.0 WC
```

- Check: No temporary or other files are left if the system terminates of its own accord, and no temporary or other files are left anywhere other than in `/tmp` if the system is terminated by a `SIGXCPU` or `SIGALRM`. Check in the current directory, the ATP system's directory, the directory where the problem's symbolic link is located, and the directory where the actual problem file is located.

```
prompt> pwd
/home/tptp
prompt> /home/tptp/bin/TreeLimitedRun -q0 200 400 /home/tptp/bin/MyATPSystem
        /home/tptp/tmp/CCC001-1.p
TreeLimitedRun: -----------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 200s
TreeLimitedRun: WC  time limit is 400s
TreeLimitedRun: PID is 13526
TreeLimitedRun: -----------------------------------------------
SZS status Unsatisfiable for CCC001-1
FINAL WATCH: 147.8 CPU 150.0 WC
prompt> ls /home/tptp
     ... no temporary or other files left here ...
prompt> ls /home/tptp/bin
     ... no temporary or other files left here ...
prompt> ls /home/tptp/tmp
     ... no temporary or other files left here ...
prompt> ls /home/tptp/TPTP/Problems/GRP
     ... no temporary or other files left here ...
prompt> ls /tmp
     ... no temporary or other files left here by decent systems ...
```

- Check: Multiple concurrent executions do not clash.

```
prompt> (/bin/time /home/tptp/bin/TreeLimitedRun -q0 200 400
        /home/tptp/bin/MyATPSystem /home/tptp/tmp/CCC001-1.p) &
        (/bin/time /home/tptp/bin/TreeLimitedRun -q0 200 400
        /home/tptp/bin/MyATPSystem /home/tptp/tmp/CCC001-1.p)
TreeLimitedRun: -----------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 200s
```

```
TreeLimitedRun: WC  time limit is 400s
TreeLimitedRun: PID is 5827
TreeLimitedRun: -----------------------------------------------
TreeLimitedRun: -----------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 200s
TreeLimitedRun: WC  time limit is 400s
TreeLimitedRun: PID is 5829
TreeLimitedRun: -----------------------------------------------
SZS status Unsatisfiable for CCC001-1
FINAL WATCH: 147.8 CPU 150.0 WC

SZS status Unsatisfiable for CCC001-1
FINAL WATCH: 147.8 CPU 150.0 WC
```

7 The ATP Systems

These system descriptions were written by the entrants.

7.1 CHewTPTP 1.0

Eric McGregor
Clrakson University, USA

Architecture

ChewTPTP version 1.0 implements the method described in [BKL$^+$08, DHJ$^+$07]. ChewTPTP transforms a set of first order clauses into a propositional encoding (modulo recursive type theory) of the existence of a rigid first order connection tableau and the satisfiability of unification constraints, which is then fed to Yices. For the unification constraints, terms are represented as recursive datatypes, and unification constraints are equations on terms. Additional instances of the first order clauses may be added for the non-rigid case.

Strategies

The strategy is as follows:

1. Encode the existence of a rigid connection tableau with unification constraints.
2. Run SMT solver on encoding.
3. If the solver returns satisfiable, we verify that the model represents a rigid tableau. If it does we return unsatisfiable, otherwise we add additional clauses and goto 2.
4. If the solver returns unsatisfiable we add additional instances of the initial clauses to the problem and goto 1.

Implementation

ChewTPTP is written in C++ and uses Yices SMT solver. Yices requires GMP, the GNU Multiprecision Library.

Expected Competition Performance

The system is a demonstration of the method described above and is not expected to win its division.

7.2 Darwin 1.3

Peter Baumgartner[1], Alexander Fuchs[2], Cesare Tinelli[2]
[1]NICTA Australia,
[2]University of Iowa, USA

Architecture

Darwin [BFT04, BFT06a] is an automated theorem prover for first order clausal logic. It is the first implementation of the Model Evolution Calculus [BT03]. The Model Evolution Calculus lifts the propositional DPLL procedure to first-order logic. One of the main motivations for this approach was the possibility of migrating to the first-order level some of those very effective search techniques developed by the SAT community for the DPLL procedure.

The current version of Darwin implements first-order versions of unit propagation inference rules analogously to a restricted form of unit resolution and subsumption by unit clauses. To retain completeness, it includes a first-order version of the (binary) propositional splitting inference rule.

Proof search in Darwin starts with a default interpretation for a given clause set, which is evolved towards a model or until a refutation is found.

Implementation

The central data structure is the *context*. A context represents an interpretation as a set of first-order literals. The context is grown by using instances of literals from the input clauses. The implementation of Darwin is intended to support basic operations on contexts in an efficient way. This involves the handling of large sets of candidate literals for modifying the current context. The candidate literals are computed via simultaneous unification between given clauses and context literals. This process is sped up by storing partial unifiers for each given clause and merging them for different combinations of context literals, instead of redoing the whole unifier computations. For efficient filtering of unneeded candidates against context literals, discrimination tree or substitution tree indexing is employed. The splitting rule generates choice points in the derivation which are backtracked using a form of backjumping similar to the one used in DPLL-based SAT solvers.

Improvements to the previous version include additional preprocessing steps, less memory requirements, and lemma learning [BFT06b].

Darwin is implemented in OCaml and has been tested under various Linux distributions (compiled but untested on FreeBSD, MacOS X, Windows). It is available from `http://goedel.cs.uiowa.edu/Darwin/`.

Strategies

Darwin traverses the search space by iterative deepening over the term depth of candidate literals. Darwin employs a uniform search strategy for all problem classes.

Expected Competition Performance

Darwin 1.3 is the CASC-21 EPR division winner.

7.3 E and EP 1.0pre

Stephan Schulz
Technische Universität München, Germany

Architecture

E 1.0pre [Sch02, Sch04b] is a purely equational theorem prover. The core proof procedure operates on formulas in clause normal form, using a calculus that combines superposition (with selection of negative literals) and rewriting. No special rules for non-equational literals have been implemented, i.e., resolution is simulated via paramodulation and equality resolution. The basic calculus is extended with rules for AC redundancy elimination, some contextual simplification, and pseudo-splitting with definition caching. The latest versions of E also supports simultaneous paramodulation, either for all inferences or for selected inferences.

E is based on the DISCOUNT-loop variant of the *given-clause* algorithm, i.e., a strict separation of active and passive facts. Proof search in E is primarily controlled by a literal selection strategy, a clause evaluation heuristic, and a simplification ordering. The prover supports a large number of preprogrammed literal selection strategies, many of which are only experimental. Clause evaluation heuristics can be constructed on the fly by combining various parameterized primitive evaluation functions, or can be selected from a set of predefined heuristics. Supported term orderings are several parameterized instances of Knuth-Bendix-Ordering (KBO) and Lexicographic Path Ordering (LPO).

The prover uses a preprocessing step to convert formulas in full first order format to clause normal form. This step may introduce (first-order) definitions to avoid an exponential growth of the formula. Preprocessing also unfolds equational definitions and performs some simplifications on the clause level.

The automatic mode determines literal selection strategy, term ordering, and search heuristic based on simple problem characteristics of the preprocessed clausal problem.

EP 1.0pre is just a combination of E 1.0pre in verbose mode and a proof analysis tool extracting the used inference steps.

Strategies

E has been optimized for performance over the TPTP. The automatic mode of E 1.0pre is inherited from previous version and is based on about 90 test runs over TPTP 3.1.1. It consists of the selection of one of about 40 different strategies for each problem. All test runs have been performed on SUN Ultra 60/300 machines with a time limit of 300 seconds (or roughly equivalent configurations). All individual strategies are refutationally complete. The worst one solves about 49of TPTP 3.0.1, the best one about 60

E distinguishes problem classes based on a number of features, all of which have between 2 and 4 possible values. The most important ones are:

- Is the most general non-negative clause unit, Horn, or Non-Horn?
- Is the most general negative clause unit or non-unit?
- Are all negative clauses unit clauses?
- Are all literals equality literals, are some literals equality literals, or is the problem non-equational?
- Are there a few, some, or many clauses in the problem?
- Is the maximum arity of any function symbol 0, 1, 2, or greater?
- Is the sum of function symbol arities in the signature small, medium, or large?

Wherever there is a three-way split on a numerical feature value, the limits are selected automatically with the aim of splitting the set of problems into approximately equal sized parts based on this one feature.

For classes above a threshold size, we assign the absolute best heuristic to the class. For smaller, non-empty classes, we assign the globally best heuristic that solves the same number of problems on this class as the best heuristic on this class does. Empty classes are assigned the globally best heuristic. Typically, most selected heuristics are assigned to more than one class.

Implementation

E is implemented in ANSI C, using the GNU C compiler. At the core is a implementation of aggressively shared first-order terms in a *term bank* data structure. Based on this, E supports the global sharing of rewrite steps. Rewriting is implemented in the form of *rewrite links* from rewritten to new terms. In effect, E is caching rewrite operations as long as sufficient memory is available. Other important features are the use of *perfect discrimination trees* with age and size constraints for rewriting and unit-subsumption, *feature vector indexing* [Sch04a] for forward- and backward subsumption and contextual literal cutting, and a new polynomial implementation of LPO [Loe04].

The program has been successfully installed under SunOS 4.3.x, Solaris 2.x, HP-UX B 10.20, MacOS-X, and various versions of Linux. Sources of the latest released version are available freely from `http://www.eprover.org`.

EP 1.0pre is a simple Bourne shell script calling E and the postprocessor in a pipeline.

Expected Competition Performance

In the last years, E performed well in most proof categories. We believe that E will again be among the stronger provers in the FOF and CNF categories. We hope that E will at least be a useful complement to dedicated systems in the other categories.

EP 1.0p will be hampered by the fact that it has to analyse the inference step listing, an operation that typically is about as expensive as the proof search itself. Nevertheless, it should be competitive in the FOF proof class.

7.4 E-KRHyper 1.1

Björn Pelzer
University Koblenz-Landau, Germany

Architecture

E-KRHyper 1.1 [PW07] is a theorem proving and model generation system for first-order logic with equality. It is an implementation of the E-hyper tableau calculus [BFP07], which integrates a superposition-based handling of equality [BG98] into the hyper tableau calculus [BFN96]. The system is an extension of the KRHyper theorem prover [Wer03], which implements the original hyper tableau calculus.

An E-hyper tableau is a tree whose nodes are labeled with clauses and which is built up by the application of the inference rules of the E-hyper tableau calculus. The calculus rules are designed such that most of the reasoning is performed using positive unit clauses. A branch can be extended with new clauses that have been derived from the clauses of that branch.

A positive disjunction can be used to split a branch, creating a new branch for each disjunct. No variables may be shared between branches, and if a case-split creates branches with shared variables, then these are immediately substituted by ground terms. The grounding substitution is arbitrary as long as the terms in its range are irreducible: the branch being split may not contain a positive equational unit which can simplify a substituting term, i.e., rewrite it with one that is smaller according to a reduction ordering. When multiple irreducible substitutions are possible, each of them must be applied in consecutive splittings in order to preserve completeness.

Redundancy rules allow the detection and removal of clauses that are redundant with respect to a branch.

The hyper extension inference from the original hyper tableau calculus is equivalent to a series of E-hyper tableau calculus inference applications. Therefore the implementation of the hyper extension in KRHyper by a variant of semi-naive evaluation [Ull89] is retained in E-KRHyper, where it serves as a shortcut inference for the resolution of non-equational literals.

Strategies

E-KRHyper uses a uniform search strategy for all problems. The E-hyper tableau is generated depth-first, with E-KRHyper always working on a single branch. Refutational completeness and a fair search control are ensured by an iterative deepening strategy with a limit on the maximum term weight of generated clauses.

Implementation

E-KRHyper is implemented in the functional/imperative language OCaml, and Runs on Unix and MS-Windows platforms. The system accepts input in the TPTP format and in the TPTP-supported Protein format. The calculus implemented by E-KRHyper works on clauses, so first order formula input is converted into CNF by an algorithm which follows the transformation steps as used in the clausification of Otter [McC94b]. E-KRHyper operates on an E-hyper tableau which is represented by linked node records. Several layered discrimination-tree based indexes (both perfect and non-perfect) provide access to the clauses in the tableau and support backtracking.

E-KRHyper is available under the GNU Public License from `http://www.uni-koblenz.de/~bpelzer/ekrhyper`.

Expected Competition Performance

Most work on E-KRHyper since version 1.0 has been concerning its interfaces and embedding in natural language processing applications, so only a minor improvement in comparison to last year's performance can be expected.

7.5 Equinox 3.0

Koen Claessen
Chalmers University of Technology, Sweden

Architecture

Equinox is an experimental theorem prover for pure first-order logic with equality. It finds ground proofs of the input theory, by solving successive ground instantiations of the theory using an incremental SAT-solver. Equality is dealt with using a Nelson-Oppen framework.

Implementation

The main part of Equinox is implemented in Haskell using the GHC compiler. Equinox also has a built-in incremental SAT solver (MiniSat) which is written in C++. The two parts are linked together on the object level using Haskell's Foreign Function Interface.

Strategies

There is only one strategy in Equinox:

1. Give all ground clauses in the problem to a SAT solver.
2. Run the SAT-solver.
3. If a contradiction is found, we have a proof and we terminate.
4. If a model is found, we use the model to indicate which new ground instantiations should be added to the SAT-solver.
5. Goto 2.

Expected Competition Performance

Equinox should perform reasonably well. There should be problems that it can solve that few other provers can handle.

7.6 iProver 0.5

Konstantin Korovin
The University of Manchester, England

Architecture

iProver is an automated theorem prover which is based on an instantiation calculus Inst-Gen [GK03, Kor08a], complete for first-order logic. One of the distinctive features of iProver is a modular combination of first-order reasoning with ground reasoning. In particular, iProver currently integrates MiniSat for reasoning with ground abstractions of first-order clauses. In addition to instantiation, iProver implements ordered resolution calculus and a combination of instantiation and ordered resolution, see [Kor08b] for the implementation details. The saturation process is implemented as a modification of a given clause algorithm. We use non-perfect discrimination trees for the unification indexes, priority queues for passive clauses and a compressed vector index for subsumption and subsumption resolution (both forward and backward). The following redundancy eliminations are implemented: blocking non-proper instantiations; dismatching constraints [GK04, Kor08b]; global subsumption [Kor08b]; resolution-based simplifications and propositional-based simplifications. Equality is dealt with (internally) by adding the necessary axioms of equality.

Strategies

iProver v0.5 has around 40 options to control the proof search including options for literal selections, passive clause selections, frequency of calling the SAT solver, simplifications and options for combination of instantiation with resolution. At the CASC competition iProver will execute a fixed schedule of selected options.

Implementation

iProver is implemented in OCaml and for the ground reasoning uses MiniSat. iProver accepts cnf and fof formats. In the case of fof format, either Vampire or E prover is used for clausification.

iProver is available at `http://www.cs.man.ac.uk/~korovink/iprover/`.

Expected Competition Performance

The instantiation method behind iProver is a decision procedure for the EPR class, and we expect a good performance in this class. We also expect a reasonable performance in FOF and CNF classes.

7.7 MaLARea 0.3

Josef Urban
Charles University in Prague, Czech Republic

Architecture

MaLARea 0.3 [Urb07, USPV08] is a metasystem for ATP in large theories where symbol and formula names are used consistently. It uses several deductive systems (now E,SPASS,Paradox,Mace), as well as complementary AI techniques like machine learning (the SNoW system) based on symbol-based similarity, model-based similarity, term-based similarity, and obviously previous successful proofs.

Strategies

The basic strategy is to run ATPs on problems, then use the machine learner to learn axiom relevance for conjectures from solutions, and use the most relevant axioms for next ATP attempts. This is iterated, using different timelimits and axiom limits. Various features are used for learning, and the learning is complemented by other criteria like model-based reasoning, symbol and term-based similarity, etc.

Implementation

The metasystem is implemented in ca. 2500 lines of Perl. It uses many external programs - the above mentioned ATPs and machine learner, TPTP utilities, LADR utilities for work with models, and some standard Unix tools. MaLARea is available at `http://kti.ms.mff.cuni.cz/cgi-bin/viewcvs.cgi/MPTP2/MaLARea/`.

The metasystem's Perl code is released under GPL2.

Expected Competition Performance

Thanks to machine learning, MaLARea is strongest on batches of many related problems with many redundant axioms where some of the problems are easy to solve and can be used for learning the axiom relevance. MaLARea is not very good when all problems are too difficult (nothing to learn from), or the problems (are few and) have nothing in common. Some of its techniques (selection by symbol and term-based similarity, model-based reasoning) could however make it even there slightly stronger than standard ATPs. MaLARea has a very good performance on the MPTP Challenge, which is a predecessor of the LTB division.

7.8 MetaProver 1.0

Matthew Streeter
Carnegie Mellon University, USA

Architecture

MetaProver is a hybridization of solvers entered in last year's CASC competition. When given a problem instance, MetaProver runs one or more solvers subject to time limits, according to a fixed schedule. The following is an example of such a schedule.

1. Run E 0.999 for 0.01 seconds.
2. Run Geo 2007f for 0.02 seconds.
3. Run Metis 2.0 for 0.01 seconds.
4. Run E 0.999 for 0.03 seconds.
5. Run Geo 2007f for 0.06 seconds.
6. Run Paradox 2.2 for 0.3 seconds.
7. Run E 0.999 for 0.07 seconds.
8. Run Paradox 2.2 for 1.48 seconds.
9. Run Metis 2.0 for 0.43 seconds.
10. Run Paradox 2.2 for 3.34 seconds.
11. Run iProver 0.2 for 32.75 seconds.
12. Run E 0.999 for 164.54 seconds.

Note that the average time such a schedule requires to solve a problem instance can be much lower than that of any of the algorithms used in the schedule.

To compute an effective schedule, the solvers entered in last year's CASC competition were run on all the TPTP-v3.4.1 benchmark instances in the SAT and FNT divisions, subject to a five minute time limit per instance (the SAT and FNT divisions were chosen because preliminary experiments indicated that a scheduling approach would work well in those divisions). Using the timing data obtained from these runs, a schedule was then computed that, if run on each benchmark instance, would solve the instances in the minimum average time. Two schedules were computed, one for the FNT instances and one for the SAT instances. The schedules were computed using the greedy approximation algorithm described in [SGS07].

Because the time required to solve a particular benchmark instance varies across machines, a schedule must be calibrated for use on a particular machine. This calibration is performed as part of MetaProver's installation script. The example schedule shown above is the schedule used by MetaProver for instances in the FNT division, calibrated for execution on an Intel Xeon 3.6 GHz machine with 4 GB of memory.

Strategies

The strategies used by MetaProver include those used by its component algorithms: DarwinFM 1.4.1, E 0.999, Geo 2007f, iProver 0.2, Metis 2.0, Paradox 1.3, and Paradox 2.2. At a higher level, MetaProver adopts the strategy of running its component algorithms according to a schedule derived from the algorithms' performance on the relevant TPTP benchmark instances, as described in the previous section. This schedule is computed using performance data for a large number of benchmark instances, and thus it is reasonable to expect the schedule's performance to generalize well to new, previously unseen instances.

Implementation

MetaProver is implemented as a set of bash scripts, and runs on Linux. It is available online at `http://www.cs.cmu.edu/~matts/MetaProver`.

Expected Competition Performance

MetaProver outperforms last year's SAT division winner (Paradox 1.3) and last year's FNT division winner (Paradox 2.2) on the relevant TPTP benchmarks.

7.9 Metis 2.1

Joe Hurd
Galois, Inc., USA

Architecture

Metis 2.1 [Hur03] is a proof tactic used in the HOL4 interactive theorem prover. It works by converting a higher order logic goal to a set of clauses in first order logic, with the property that a refutation of the clause set can be translated to a higher order logic proof of the original goal.

Experiments with various first order calculi [Hur03] have shown a *given clause algorithm* and ordered resolution to best suit this application, and that is what Metis 2.1 implements. Since equality often appears in interactive theorem prover goals, Metis 2.1 also implements the ordered paramodulation calculus.

Strategies

Metis 2.1 uses a fixed strategy for every input problem. Negative literals are always chosen in favour of positive literals, and terms are ordered using the Knuth-Bendix ordering with uniform symbol weight and precedence favouring reduced arity.

Implementation

Metis 2.1 is written in Standard ML, for ease of integration with HOL4. It uses indexes for resolution, paramodulation, (forward) subsumption and demodulation. It keeps the *Active* clause set reduced with respect to all the unit equalities so far derived.

In addition to standard size and distance measures, Metis 2.1 uses finite models to weight clauses in the *Passive* set. When integrated with higher order logic, a finite model is manually constructed to interpret standard functions and relations in such a way as to make many axioms true and negated goals false. Non-standard functions and relations are interpreted randomly, but with a bias towards making negated goals false. Since it is part of the CASC competition rules that standard functions and relations are obfuscated, Metis 2.1 will back-off to interpreting all functions and relations randomly (except equality), using a finite model with 4 elements.

Metis 2.1 reads problems in TPTP format and outputs detailed proofs in TSTP format, where each proof step is one of 6 simple inference rules. Metis 2.1 implements a complete calculus, so when the set of clauses is saturated it can soundly declare the input problem to be unprovable (and outputs the saturation set).

Metis 2.1 is free software, released under the GPL. It can be downloaded from `http://www.gilith.com/software/metis`.

Expected Competition Performance

The major change between Metis 2.0, which was entered into CASC-21, and Metis 2.1 is the TSTP proof format. There were only minor changes to the core proof engine, so Metis 2.1 is expected to perform at approximately the same level and end up in the lower half of the table.

7.10 Muscadet 3.0

Dominique Pastre
Université René Descartes Paris-5, France

Architecture

The MUSCADET theorem prover is a knowledge-based system. It is based on Natural Deduction, following the terminology of [Ble71] and [Pas78], and uses methods which resembles those used by humans. It is composed of an inference engine, which interprets and executes rules, and of one or several bases of facts, which are the internal representation of "theorems to be proved". Rules are either universal and put into the system, or built by the system itself by metarules from data (definitions and lemmas). Rules may add new hypotheses, modify the conclusion, create objects, split theorems into two or more subtheorems or build new rules which are local for a (sub-)theorem.

Strategies

There are specific strategies for existential, universal, conjonctive or disjunctive hypotheses and conclusions. Functional symbols may be used, but an automatic creation of intermediate objects allows deep subformulae to be flattened and treated as if the concepts were defined by predicate symbols. The successive steps of a proof may be forward deduction (deduce new hypotheses from old ones), backward deduction (replace the conclusion by a new one) or refutation (only if the conclusion is a negation).

The system is also able to work with second order statements. It may also receive knowledge and know-how for a specific domain from a human user; see [Pas89] and [Pas93]. These two possibilities are not used while working with the TPTP Library.

Implementation

Muscadet 3.0 [Pas01] is implemented in SWI-Prolog. Rules are written as declarative Prolog clauses. Metarules are written as sets of Prolog clauses, more or less declarative. The inference engine includes the Prolog interpreter and some procedural Prolog clauses.

Proofs are given in natural style (for each step, that is for each action or rule application, the system gives the new fact, the precedent facts its comes from and an explanation).

Muscadet 3.0 is available from `http://www.math-info.univ-paris5.fr/~pastre/muscadet/muscadet.html`.

Expected Competition Performance

The best performances of Muscadet will be for problems manipulating many concepts in which all statements (conjectures, definitions, axioms) are expressed in a manner similar to the practice of humans, especially of mathematicians [Pas02, Pas07]. It will have poor performances for problems using few concepts but large and deep formulas leading to many splittings.

Muscadet 3.0 will probably have the same performances as Muscadet 2.7a (2007 CASC-21 version + bugfixed), but will give an out proof for most solved problems.

7.11 OSHL-S 0.1

Hao Xu, David Plaisted
University of North Carolina at Chapel Hill, USA

Architecture

OSHL-S is a theorem prover based on the architecture and strategy introduced in [PZ00] with a few improvements. A preliminary form of type inference is employed to reduce the number of instances that are generated before a contradicting instance is found.

Strategies

OSHL-S employs a uniform strategy for all problems, which is similar to OSHL: It starts with a all negative or all positive model. In each iteration, it finds a contradicting instance according to a relaxed ordering and type information, and then modifies the model to make the instance satisfiable, if possible.

Implementation

OSHL-S is implemented in Java using Java SE 6 SDK and the Netbeans IDE. The profiler and the GUI provided in Java SE 6 and the Netbeans IDE are employed to profile the system, which provide performance data that are used for optimization. The implementation of OSHL-S combines a few strategies for improving the performance of the system, including caching of terms, clauses, and solutions to integer programming problems, lazy-evaluation, and backtracking.

Expected Competition Performance

The performance should be good for near propositional problems; otherwise, the competition performance is unknown.

7.12 Otter 3.3

William McCune
Argonne National Laboratory, USA

Architecture

Otter 3.3 [McC03] is an ATP system for statements in first-order (unsorted) logic with equality. Otter is based on resolution and paramodulation applied to clauses. An Otter search uses the "given clause algorithm", and typically involves a large database of clauses; subsumption and demodulation play an important role.

Strategies

Otter's original automatic mode, which reflects no tuning to the TPTP problems, will be used.

Implementation

Otter is written in C. Otter uses shared data structures for clauses and terms, and it uses indexing for resolution, paramodulation, forward and backward subsumption, forward and backward demodulation, and unit conflict. Otter is available from `http://www-unix.mcs.anl.gov/AR/otter/`.

Expected Competition Performance

Otter has been entered into CASC as a stable benchmark against which progress can be judged (there have been only minor changes to Otter since 1996 [MW97], nothing that really affects its performace in CASC). This is not an ordinary entry, and we do not hope for Otter to do well in the competition.

Acknowledgments: Ross Overbeek, Larry Wos, Bob Veroff, and Rusty Lusk contributed to the development of Otter.

7.13 Paradox 1.3

Koen Claessen, Niklas Sörensson
Chalmers University of Technology and Gothenburg University, Sweden

Architecture

Paradox [CS03] is a finite-domain model generator. It is based on a MACE-style [McC94a] flattening and instantiating of the first-order clauses into propositional clauses, and then the use of a SAT solver to solve the resulting problem.

Paradox incorporates the following features: Polynomial-time *clause splitting heuristics*, the use of *incremental SAT*, *static symmetry reduction* techniques, and the use of *sort inference*.

The main differences with Paradox 1.0 are: a better SAT-solver, better memory behaviour, and a faster clause instantiation algorithm.

Strategies

There is only one strategy in Paradox:

1. Analyze the problem, finding an upper bound N on the domain size of models, where N is possibly infinite. A finite such upper bound can be found, for example, for EPR problems.
2. Flatten the problem, and split clauses and simplify as much as possible.
3. Instantiate the problem for domain sizes 1 up to N, applying the SAT solver incrementally for each size. Report "SATISFIABLE" when a model is found.
4. When no model of sizes smaller or equal to N is found, report "CONTRADICTION".

In this way, Paradox can be used both as a model finder and as an EPR solver.

Implementation

The main part of Paradox is implemented in Haskell using the GHC compiler. Paradox also has a built-in incremental SAT solver which is written in C++. The two parts are linked together on the object level using Haskell's Foreign Function Interface.

Expected Competition Performance

Paradox 1.3 is the CASC-21 SAT division winner.

7.14 Paradox 2.2 and 3.0

Koen Claessen, Niklas Sörensson
Chalmers University of Technology, Sweden

Architecture

Paradox 2.2 is a rewrite of Paradox 1.3. Paradox 2.2 does not have all the features yet that Paradox 1.3 has. Some experimental features, such as type-based model finding, have been added. Paradox 3.0 has the same description as Paradox 2.2. See the description of Paradox 1.3 for general information.

Expected Competition Performance

Paradox 2.2 is the CASC-21 FNT division winner.

7.15 randoCoP 1.1

Jens Otten, Thomas Raths
University of Potsdam, Germany

Architecture

randoCoP [RO08] is an automated theorem prover for classical first-order logic. It is an extension of the leanCoP [Ott08b, OB03] prover, which is a very compact implementation of the connection calculus. It integrates a technique that randomly alters the proof search order by reordering the axioms of the given problem and the literals within its clausal form.

Strategies

The shell script of randoCoP repeatedly reorders the axioms and the literals within the clausal form, before invoking the two most successful variants of the leanCoP 2.0 core prover. These variants use restricted backtracking [Ott08a] and benefit significantly from the reordering techniques used by randoCoP.

Implementation

randoCoP is implemented in Prolog. The essential extensions consist of a modified shell script, which is used to invoke the leanCoP 2.0 core prover, and a few predicates that realize the reordering of axioms and literals. A preprocessing component translates first-order formulas into a (definitional) clausal form. Equality can be handled by adding the equality axioms. Version 1.1 of randoCoP returns a compact connection proof, which is then translated into a readable proof. The source code of randoCoP is available at `http://www.leancop.de`.

Expected Competition Performance

The performance of randoCoP is in particular good on problems involving large theories, i.e., problems that contain a large number of axioms.

7.16 SInE 0.3 and SInE-VD 0.3

Krystof Hoder
Charles University in Prague, Czech Republic

Architecture

SiNE 0.3 is an axiom selection system for first order theories. It uses a syntactic approach based on symbols presence in axioms and conjecture. (When we say symbols, we mean functional, predicate and constant symbols taken together.) A relation D (as in "Defines") is created between symbols and axioms which represents the fact that for a symbol there are some axioms that "give it its meaning". When the relation is constructed, the actual axiom selection starts. At the beginning only the conjecture is selected, in each iteration the selection is extended by all axioms that are D-related to any symbol used in already selected axioms. The iteration goes until no more axioms are selected. Then the selected axioms are handed to an underlying inference engine.

Strategies

The construction of the D relation is inspired by the idea that general symbols are more likely to define the meaning of more specific axioms than vice-versa. So given a generality measure on symbols, SiNE puts each axiom into the relation with the least general of its symbols. (When there are more of them, all are put in the relation.) The generality measure used is the number of axioms in which the symbol occurs. (General symbols as `s_Entity` are likely to be used more often than specific symbols like `s_Monday`.) One slight optimisation for SUMO problems is also used: When we run into a symbol ending with "_M" we remove the suffix for the selection process. The "_M" suffix is used when a predicate symbol should be used as functional. This strategy selects about 2% of axioms on problems CSR(075-109).

Implementation

The axiom selection is implemented in Python. The problem file is read and include directives are extracted. Then problems which include the same sets of axioms are grouped together and the groups are processed separately. In each group the axioms which would be included by problems are loaded and preprocessed, constructing the D relation. (This takes most of time of the whole axiom selection.) Then for each problem a set of axioms is selected based on all symbols that occur in the problem file and an underlying prover is called. There are two underlying provers supported: EP and Vampire 9. EP will be used in the competition division and Vampire 9 in the demonstration division (as the usage of Vampire in the competition division was not allowed by its developers).

The batch mode isn't fully finished yet. As vast majority of proved CSR(025-109) problems is proved in first 10 seconds of underlying engine's run (and all but three in first 30secs), I'll probably add some more iterations which would be less restrictive during the axiom selection.

Expected Competition Performance

The following table shows results when time-limit for conventional provers was set to 300s.

CSR theorems (max=23)	E 0.999	Vampire 9	SInE with E0.999	SInE with Vampire 9
+1	3	17	12	17
+2		13	12	17
+3		3	13	18

This suggests that SInE could perform well especially on larger problems.

This could also improve with further optimisation of the batch mode - four of the six problems where SInE with V9 failed were solved by V9 itself. Therefore another attempt with less restrictive axiom selection could help.

7.17 Vampire 8.1

Andrei Voronkov
University of Manchester, England

No system description supplied. However, see the description of Vampire 8.0¡/A¿ for general information. Minor changes have been made, including a bugfix in the FOF to CNF conversion.

Expected Competition Performance

Vampire 8.1 is the CASC-21 CNF division winner.

7.18 Vampire 9.0

Andrei Voronkov
University of Manchester, England

No system description supplied.

Expected Competition Performance

Vampire 9.0 is the CASC-21 FOF division winner.

7.19 Waldmeister 806

Thomas Hillenbrand[1], Bernd Löchner[2]
[1]Max-Planck-Institut für Informatik Saarbrücken, Germany
[2]Technische Universität Kaiserslautern, Germany

No system description supplied.

Expected Competition Performance

Waldmeister 806 is the CASC-21 UEQ division winner.

7.20 Zenon 0.5.0

Damien Doligez
INRIA, France

Architecture

Zenon 0.5.0 [BDD07] is based on the tableau method with free variables. It uses a nondestructive way of handling free variables, which enables a purely local search procedure: each branch is closed before the next one is explored.

Zenon outputs totally formal proofs that can be checked by Coq.

Strategies

Implementation

Zenon is written in Objective Caml. It can be downloaded from `http://focal.inria.fr/zenon`.

Expected Competition Performance

Zenon is still in the prototype stage and we don't really expect brilliant results at this point.

8 Conclusion

The 4th IJCAR ATP System Competition is the thirteenth large scale competition for classical first-order logic ATP systems. The organizers believe that CASC fulfills its main motivations: stimulation of research, motivation for improving implementations, evaluation of relative capabilities of ATP systems, and providing an exciting event. Through the continuity of the event and consistency in the 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 provided an overview of the implementation state of running, fully automatic, first order ATP systems.

References

[BDD07] R. Bonichon, D. Delahaye, and D. Doligez. Zenon : An Extensible Automated Theorem Prover Producing Checkable Proofs. In N. Dershowitz and A. Voronkov, editors, *Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 4790 in Lecture Notes in Artificial Intelligence, pages 151–165, 2007.

[BFN96] P. Baumgartner, U. Furbach, and I. Niemelä. Hyper Tableaux. In J. Alferes, L. Pereira, and E. Orlowska, editors, *Proceedings of JELIA'96: European Workshop on Logic in Artificial Intelligence*, number 1126 in Lecture Notes in Artificial Intelligence, pages 1–17. Springer-Verlag, 1996.

[BFP07] P. Baumgartner, U. Furbach, and B. Pelzer. Hyper Tableaux with Equality. In F. Pfenning, editor, *Proceedings of the 21st International Conference on Automated Deduction*, number 4603 in Lecture Notes in Artificial Intelligence, pages 492–507. Springer-Verlag, 2007.

[BFT04] P. Baumgartner, A. Fuchs, and C. Tinelli. Darwin - A Theorem Prover for the Model Evolution Calculus. In G. Sutcliffe, S. Schulz, and T. Tammet, editors, *Proceedings of the Workshop on Empirically Successful First Order Reasoning, 2nd International Joint Conference on Automated Reasoning*, 2004.

[BFT06a] P. Baumgartner, A. Fuchs, and C. Tinelli. Implementing the Model Evolution Calculus. *International Journal on Artificial Intelligence Tools*, 15(1):21–52, 2006.

[BFT06b] P. Baumgartner, A. Fuchs, and C. Tinelli. Lemma Learning in the Model Evolution Calculus. In M. Hermann and A. Voronkov, editors, *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 4246 in Lecture Notes in Artificial Intelligence, pages 572–585, 2006.

[BG98] L. Bachmair and H. Ganzinger. Equational Reasoning in Saturation-Based Theorem Proving. In W. Bibel and P.H. Schmitt, editors, *Automated Deduction, A Basis for Applications*, volume I Foundations - Calculi and Methods of *Applied Logic Series*, pages 352–397. Kluwer Academic Publishers, 1998.

[BKL$^+$08] J. Bongio, C. Katrak, H. Lin, C. Lynch, and R. McGregor. Encoding First Order Proofs in SMT. In S. Krstic and A. Oliveras, editors, *Proceedings of the 5th International Workshop on Satisfiability Modulo Theories*, volume 198 of *Electronic Notes in Theoretical Computer Science*, pages 71–84, 2008.

[Ble71] W.W. Bledsoe. Splitting and Reduction Heuristics in Automatic Theorem Proving. *Artificial Intelligence*, 2:55–77, 1971.

[BT03] P. Baumgartner and C. Tinelli. The Model Evolution Calculus. In F. Baader, editor, *Proceedings of the 19th International Conference on Automated Deduction*, number 2741 in Lecture Notes in Artificial Intelligence, pages 350–364. Springer-Verlag, 2003.

[CS03] K. Claessen and N. Sorensson. New Techniques that Improve MACE-style Finite Model Finding. In P. Baumgartner and C. Fermueller, editors, *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*, 2003.

[DHJ$^+$07] T. Deshane, W. Hu, P. Jablonski, H. Lin, C. Lynch, and R. McGregor. Encoding First Order Proofs in SAT. In F. Pfenning, editor, *Proceedings of the 21st International Conference on Automated Deduction*, number 4603 in Lecture Notes in Artificial Intelligence, pages 476–491. Springer-Verlag, 2007.

[GK03] 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, 2003.

[GK04] H. Ganzinger and K. Korovin. Integrating Equational Reasoning into Instantiation-Based Theorem Proving. In J. Marcinkowski and A. Tarlecki, editors, *Proceedings of the 18th International Workshop on Computer Science Logic, 13th Annual Conference of the EACSL*, number 3210 in Lecture Notes in Computer Science, pages 71–84, 2004.

[GS96] M. Greiner and M. Schramm. A Probablistic Stopping Criterion for the Evaluation of Benchmarks. Technical Report I9638, Institut für Informatik, Technische Universität München, München, Germany, 1996.

[Hur03] J. Hurd. First-Order Proof Tactics in Higher-Order Logic Theorem Provers. In M. Archer, B. Di Vito, and C. Munoz, editors, *Proceedings of the 1st International Workshop on Design and Application of Strategies/Tactics in Higher Order Logics*, number NASA/CP-2003-212448 in NASA Technical Reports, pages 56–68, 2003.

[Kor08a] K. Korovin. An Invitation to Instantiation-Based Reasoning: From Theory to Practice. In A. Podelski, A. Voronkov, and R. Wilhelm, editors, *Volume in Memoriam of Harald Ganzinger*. Springer-Verlag, 2008.

[Kor08b] K. Korovin. iProver - An Instantiation-Based Theorem Prover for First-order Logic (System Description). In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, 2008.

[Loe04] B. Loechner. What to Know When Implementing LPO. In G. Sutcliffe, S. Schulz, and T. Tammet, editors, *Proceedings of the Workshop on Empirically Successful First Order Reasoning, 2nd International Joint Conference on Automated Reasoning*, 2004.

[McC94a] W.W. McCune. A Davis-Putnam Program and its Application to Finite First-Order Model Search: Quasigroup Existence Problems. Technical Report ANL/MCS-TM-194, Argonne National Laboratory, Argonne, USA, 1994.

[McC94b] W.W. McCune. Otter 3.0 Reference Manual and Guide. Technical Report ANL-94/6, Argonne National Laboratory, Argonne, USA, 1994.

[McC03] W.W. McCune. Otter 3.3 Reference Manual. Technical Report ANL/MSC-TM-263, Argonne National Laboratory, Argonne, USA, 2003.

[MW97] W.W. McCune and L. Wos. Otter: The CADE-13 Competition Incarnations. *Journal of Automated Reasoning*, 18(2):211–220, 1997.

[OB03] J. Otten and W. Bibel. leanCoP: Lean Connection-Based Theorem Proving. *Journal of Symbolic*

Computation, 36(1-2):139–161, 2003.

[Ott08a] J. Otten. Restricting Backtracking in Connection Calculi. Technical Report Technical Report, Institut für Informatik, University of Potsdam, Potsdam, Germany, 2008.

[Ott08b] J. Otten. leanCoP 2.0 and ileancop 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic. In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, 2008.

[Pas78] D. Pastre. Automatic Theorem Proving in Set Theory. *Artificial Intelligence*, 10:1–27, 1978.

[Pas89] D. Pastre. Muscadet : An Automatic Theorem Proving System using Knowledge and Metaknowledge in Mathematics. *Artificial Intelligence*, 38:257–318, 1989.

[Pas93] D. Pastre. Automated Theorem Proving in Mathematics. *Annals of Mathematics and Artificial Intelligence*, 8:425–447, 1993.

[Pas01] D. Pastre. Muscadet version 2.3 : User's Manual. http://www.math-info.univ-paris5.fr/ pastre/muscadet/manual-en.ps, 2001.

[Pas02] D. Pastre. Strong and Weak Points of the Muscadet Theorem Prover. *AI Communications*, 15(2-3):147–160, 2002.

[Pas07] D. Pastre. Complementarity of a Natural Deduction Knowledge-based Prover and Resolution-based Provers in Automated Theorem Proving. http://www.math-info.univ-paris5.fr/ pastre/compl-NDKB-RB.pdf, 2007.

[PW07] B. Pelzer and C. Wernhard. System Description: E-KRHyper. In F. Pfenning, editor, *Proceedings of the 21st International Conference on Automated Deduction*, number 4603 in Lecture Notes in Artificial Intelligence, pages 508–513. Springer-Verlag, 2007.

[PZ00] D.A. Plaisted and Y. Zhu. Ordered Semantic Hyper-linking. *Journal of Automated Reasoning*, 25(3):167–217, 2000.

[RO08] T. Raths and J. Otten. randoCoP: Randomizing the Proof Search Order in the Connection Calculus. In R. Schmidt, B. Konev, and S. Schulz, editors, *Proceedings of the Workshop on Practical Aspects of Automated Reasoning, 4th International Joint Conference on Automated Reasoning*, page Accepted, 2008.

[Sch02] 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. AAAI Press, 2002.

[Sch04a] S. Schulz. Simple and Efficient Clause Subsumption with Feature Vector Indexing. In G. Sutcliffe, S. Schulz, and T. Tammet, editors, *Proceedings of the Workshop on Empirically Successful First Order Reasoning, 2nd International Joint Conference on Automated Reasoning*, 2004.

[Sch04b] 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, 2004.

[SGS07] M. Streeter, D. Golovin, and S.F. Smith. Combining Multiple Heuristics Online. In R.C. Holte and A. Howe, editors, *Proceedings of the 22nd Conference on Artificial Intelligence*, pages 1197–1203. AAAI Press, 2007.

[SS97a] G. Sutcliffe and C.B. Suttner. Special Issue: The CADE-13 ATP System Competition. *Journal of Automated Reasoning*, 18(2), 1997.

[SS97b] G. Sutcliffe and C.B. Suttner. The Procedures of the CADE-13 ATP System Competition. *Journal of Automated Reasoning*, 18(2):163–169, 1997.

[SS97c] C.B. Suttner and G. Sutcliffe. The Design of the CADE-13 ATP System Competition. *Journal of Automated Reasoning*, 18(2):139–162, 1997.

[SS98a] 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.

[SS98b] G. Sutcliffe and C.B. Suttner. Proceedings of the CADE-15 ATP System Competition. Lindau, Germany, 1998.

[SS98c] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[SS98d] C.B. Suttner and G. Sutcliffe. The CADE-14 ATP System Competition. *Journal of Automated Reasoning*, 21(1):99–134, 1998.

[SS99] G. Sutcliffe and C.B. Suttner. The CADE-15 ATP System Competition. *Journal of Automated Reasoning*, 23(1):1–23, 1999.

[SS01] G. Sutcliffe and C.B. Suttner. Evaluating General Purpose Automated Theorem Proving Systems. *Artificial Intelligence*, 131(1-2):39–54, 2001.

[SS03] G. Sutcliffe and C. Suttner. The CADE-18 ATP System Competition. *Journal of Automated Reasoning*, 31(1):23–32, 2003.

[SS04] G. Sutcliffe and C. Suttner. The CADE-19 ATP System Competition. *AI Communications*, 17(3):103–182, 2004.

[SSP02] G. Sutcliffe, C. Suttner, and F.J. Pelletier. The IJCAR ATP System Competition. *Journal of Automated Reasoning*, 28(3):307–320, 2002.

[Sut99] G. Sutcliffe. Proceedings of the CADE-16 ATP System Competition. Trento, Italy, 1999.

[Sut00a] G. Sutcliffe. Proceedings of the CADE-17 ATP System Competition. Pittsburgh, USA, 2000.

[Sut00b] G. Sutcliffe. The CADE-16 ATP System Competition. *Journal of Automated Reasoning*, 24(3):371–396, 2000.

[Sut01a] G. Sutcliffe. Proceedings of the IJCAR ATP System Competition. Siena, Italy, 2001.

[Sut01b] G. Sutcliffe. The CADE-17 ATP System Competition. *Journal of Automated Reasoning*, 27(3):227–250, 2001.

[Sut02] G. Sutcliffe. Proceedings of the CADE-18 ATP System Competition. Copenhagen, Denmark, 2002.

[Sut03] G. Sutcliffe. Proceedings of the CADE-19 ATP System Competition. Miami, USA, 2003.

[Sut04] G. Sutcliffe. Proceedings of the 2nd IJCAR ATP System Competition. Cork, Ireland, 2004.

[Sut05a] G. Sutcliffe. Proceedings of the CADE-20 ATP System Competition. Tallinn, Estonia, 2005.

[Sut05b] G. Sutcliffe. The IJCAR-2004 Automated Theorem Proving Competition. *AI Communications*, 18(1):33–40, 2005.

[Sut06a] G. Sutcliffe. Proceedings of the 3rd IJCAR ATP System Competition. Seattle, USA, 2006.

[Sut06b] G. Sutcliffe. The CADE-20 Automated Theorem Proving Competition. *AI Communications*, 19(2):173–181, 2006.

[Sut07a] G. Sutcliffe. Proceedings of the CADE-21 ATP System Competition. Bremen, Germany, 2007.

[Sut07b] G. Sutcliffe. The 3rd IJCAR Automated Theorem Proving Competition. *AI Communications*, 20(2):117–126, 2007.

[Sut08] G. Sutcliffe. The CADE-21 Automated Theorem Proving System Competition. *AI Communications*, 21(1):71–82, 2008.

[Ull89] J. Ullman. *Principles of Database and Knowledge-Base Bystems.* Computer Science Press, Inc., 1989.

[Urb07] J. Urban. MaLARea: a Metasystem for Automated Reasoning in Large Theories. In J. Urban, G. Sutcliffe, and S. Schulz, editors, *Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories*, number 257 in CEUR Workshop Proceedings, pages 45–58, 2007.

[USPV08] J. Urban, G. Sutcliffe, P. Pudlak, and J. Vyskocil. MaLARea SG1: Machine Learner for Automated Reasoning with Semantic Guidance. In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, 2008.

[Wer03] C. Wernhard. System Description: KRHyper. Technical Report Fachberichte Informatik 14–2003, Universität Koblenz-Landau, Koblenz, Germany, 2003.