CASC-21

Proceedings of the

CADE-21
ATP System Competition

Bremen, German, 17th July 2007

CASC-21

The CADE-21 ATP System Competition (CASC-21)

Geoff Sutcliffe
Department of Computer Science
University of Miami
geoff@cs.miami.edu

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 CADE-21 ATP System Competition (CASC-21) was held on 17th July 2007. 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-21 was held at the 21st International Conference on Automated Deduction, on 17th July 2007. CASC-21 is the twelfth such ATP system competition [SS97a, SS98d, SS99, Sut00b, Sut01b, SSP02, SS03, SS04, Sut05b, Sut06b, Sut07].

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
- a specified time limit for each solution attempt.

Twenty ATP systems and variants, listed in Table 1, entered into the various competition and demonstration divisions. The winners of the CASC-J3 (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]. Important changes for this CASC are:

Table 1: The ATP systems and entrants

ATP System	Divisions	Entrants	Affiliation
Darwin 1.3	EPR	CASC	*CASC-J3 EPR winner*
Darwin 1.4	EPR	Peter Baumgartner,	NICTA,
		Alexander Fuchs, Cesare Tinelli	University of Iowa
FM-Darwin 1.4	FNT* SAT*		*Darwin 1.4 variant*
E 0.999	FOF CNF FNT SAT EPR UEQ	Stephan Schulz	Technische Universität München
EP 0.999	FOF* CNF*		*E 0.999 variant*
E-KRHyper 1.0	CNF* EPR	Björn Pelzer	University Koblenz-Landau
Equinox 1.2	FOF CNF EPR UEQ	Koen Claessen	Chalmers University of Technology
Fampire 1.3	FOF (demo)	Josef Urban	Charles University
Geo 2007j	FOF* FNT* CNF* SAT* EPR UEQ	Hans de Nivelle	University of Wroclaw
iProver 0.2	FOF CNF FNT SAT EPR	Konstantin Korovin	The University of Manchester
leanCoP 2.0	FOF	Jens Otten, Thomas Raths	University of Potsdam
ileanCoP 1.2	FOF (demo)		*leanCoP 2.0 variant*
Metis 2.0	FOF CNF* FNT* SAT* EPR UEQ	Joe Hurd	Oxford University
Muscadet 2.7	FOF	Dominque Pastre	Université René Descartes Paris-5
Otter 3.3	FOF CNF* UEQ	CASC, William McCune	Argonne National Laboratory
Paradox 1.3	SAT*	CASC	*CASC-J3 SAT* winner*
Paradox 2.2	FNT* SAT* EPR	Koen Claessen, Niklas Sörensson	Chalmers University of Technology
Vampire 8.1	FOF* CNF*	CASC	*CASC-J3 FOF*, CNF* winner*
Vampire 9.0	FOF* CNF* EPR UEQ	Andrei Voronkov	University of Manchester
Waldmeister 806	UEQ	CASC	*CASC-J3 UEQ winner*

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

- The FNT division - First order form Non-Theorems - was added. This division uses counter-satisfiable and satisfiable FOF problems.
- The problems used the full set of TPTP roles defined in the TPTP language: `axiom`, `lemma`, `conjecture`, etc.
- The distinguished strings used to report the problem status should (developers were encouraged to do this, not absolutely required) 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
  ```

- The distinguished strings used to delimit proofs/models should (developers were encouraged to do this, not absolutely required) 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
  ```

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 Christoph Benzmüller, Alan Bundy, and John Slane. 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-21 WWW site provides access to resources used before, during, and after the event:

`http://www.tptp.org/CASC/21`

2 Divisions

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

- The **FNN** category: FNT with no Equality
- The **FNQ** category: FNT with 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 clauses)
- The EPS category: Effectively Propositional non-theorems (Satisfiable clauses)

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

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 on the competition computers, or cannot be entered into the competition divisions for any other 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.

3 Infrastructure

3.1 Computers

The competition computers are Dell computers, each having:

- AMD Athlon XP 3000+, 2091MHz CPU
- 503MB memory
- Linux 2.6.20-1.2307.fc5 operating system

3.2 Problems

3.2.1 Problem Selection

The problems are from the TPTP problem library, version v3.2.0. The TPTP version used for the competition is not released until after the system installation 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. 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

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.

3.3 Resource Limits

In the competition divisions, CPU and wall clock time limits are imposed on each solution attempt. A minimal CPU time limit of 240 seconds is used. The maximal CPU time limit 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 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

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, CNF, FNT, and SAT divisions additionally have and a proof/model ranking class, ranked according to the number of problems solved with an acceptable proof/model output on `stdout`. 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.

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. Division and class winners are announced and prizes are awarded.

The competition panel decides whether or not the systems' proofs and models are acceptable. 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.
- 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 classes, and the EPR and UEQ divisions, 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 Properties regarding soundness checking before the competition). At some time after the competition, the proofs from the winners of the CNF and FOF division proof classes, and the models from the winners of the FNT and SAT division model 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 system description must fit onto two pages, using 12pt times font. The schema has the following sections:

- Architecture. This section introduces the ATP system, and describes the calculus and inference rules used.
- 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.
- 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).
- 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 and model classes, representative sample solutions must be emailed to the competition organizers before the sample solutions deadline. Proof samples for the CNF division must include a proof for `SYN075-1`. Proof samples for the FOF division must include a proof for `SYN075+1`. Model samples for the FNT division must include models for `MGT019+2` and `SWV010+1`. Model samples for the SAT division must include a model for `MGT031-1`. 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:

- The ATP 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 have to be fully automatic, i.e., any command line switches have to be the same for all problems.
- Systems have to 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, and UEQ 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 an ATP system fails the soundness testing it must be repaired by the unsoundness repair deadline or be withdrawn. The soundness testing eliminates the possibility of an ATP 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.
- The ATP systems have to be executable by a single command line, using an absolute path name for the executable, which might not be in the current directory. 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). 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 ATP 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. Any orphan processes are killed after that, using `SIGKILL`. 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.
- When terminating of their own accord, the ATP 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
  ```

  Only 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). These pairs of strings must be different for proofs and models. 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
```

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 the ATP systems is not allowed. A limit, dependent on the disk space available, is imposed on the amount of `stdout` and `stderr` output that can be produced. The limit is at least 10KB per problem (averaged over all problems so that it is possible to produce *some* long proofs).
- 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.
- For every problem solved, the system's solution process 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 installation 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 installation 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 reinstalled onto the competition computers by the competition

organizers, following the instructions in the `ReadMe` file. Installation failures before the installation deadline are passed back to the entrant. After the installation deadline access to the competition computers is denied, and no further changes or late systems are accepted (i.e., deliver your installation package before the installation deadline so if the installation fails you have a chance to fix it!). If you are in doubt about your installation package or procedure, please email the competition organizers.

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 `.tar.gz` 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.

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. All attempts at the Nth problems in all the divisions and categories are started before any attempts at the (N+1)th problems.

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 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 2.6.20-1.2307.fc5 i686 athlon
  ```

  If the ATP system requires any special software, libraries, etc, which not part of a standard installation, please let the competition organizers know in the system registration.

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

  ```
  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. For example

```
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. For example

```
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). For example

```
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. For example

```
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. For example

```
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. The SZS format, described in Section 6.1, has been used here. If appropriate, similar checks should be made for the cases where no solution exists and where no conclusion is reached.

```
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
```

- Check: The system outputs distinguished strings at the start and end of its solution. The SZS format, described in Section 6.1, has been used here.

```
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. For example

```
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 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-J3 EPR division winner.

7.2 Darwin 1.4

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 an 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. Darwin is implemented in OCaml and has been tested under various Linux distributions. 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

We expect its performance to be very strong in the EPR division.

7.3 FM-Darwin 1.4

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

Architecture

FM-Darwin [BFdNT06] is a finite model finder for first order clausal logic with equality, in the spirit of Paradox [CS03].

For each domain size the problem is transformed into an equisatisfiable function-free clause set, which is decided by the Darwin prover [BFT06a].

Implementation

FM-Darwin is implemented as an extension to Darwin, which can now be used in its default mode or as a finite model finder. Like Paradox, the system uses clause splitting, term definitions, static symmetry reduction, sort inference, and lemmas. In contrast, clause splitting and term definitions are only applied in a restricted way, that is for variable disjunct clause and ground terms, as ground instantiation is not performed and thus the exponential increase in the size of the clause set does not happen.

FM-Darwin is available from

`http://goedel.cs.uiowa.edu/Darwin/`

Strategies

FM-Darwin follows a popular strategy of finite domain finders. The first-order clausal problem is transformed into a decidable logic, here function-free clause logic. Then, the system tries to find a finite domain model for sizes 1 .. n. A given domain is encoded by adding the axioms of totality and functionality for the symbols which have been converted from function to predicate symbols, such that any finite model of the original problem can be translated into a model of the transformed problem, and vice versa.

Like Paradox, FM-Darwin is a complete method for function-free clause sets, and can also detect unsatisfiability if the totality axioms are not used in a refutation.

Expected Competition Performance

We expect its performance to be good in the SAT and FNT divisions.

7.4 E and EP 0.999

Stephan Schulz
Technische Universität München, Germany

Architecture

E 0.999 [Sch02a, 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. 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 0.999 is just a combination of E 0.999 in verbose mode and a proof analysis tool extracting the used inference steps.

Implementation

E is implemented in ANSI C, using the GNU C compiler. The most outstanding feature is the global sharing of rewrite steps. Current versions of E add 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 0.999 is a simple Bourne shell script calling E and the postprocessor in a pipeline.

Strategies

E's automatic mode is optimized for performance on TPTP. The optimization of E 0.999 is based on about 90 test runs over the library (and previous experience) and 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 49We expect similar optimization for E 0.999. 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.

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 CNF category. Performance on FOF problems should be competitive. We hope that E will at least be a useful complement to dedicated systems in the other categories.

EP 0.999 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 among the FOF* and CNF* systems.

7.5 E-KRHyper 1.0

Björn Pelzer, Christoph Wernhard
Universität Koblenz-Landau, Germany

Architecture

E-KRHyper 1.0 [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.

Implementation

E-KRHyper is implemented in the functional/imperative language OCaml with additional preprocessing scripts in Prolog. The system runs on Unix and MS-Windows platforms and is available under the GNU Public License from the E-KRHyper website at

`http://www.uni-koblenz.de/ bpelzer/ekrhyper`

The system accepts clause input in the TPTP-supported Protein-format. E-KRHyper operates on an E-hyper tableau that is represented by linked node records. A layered discrimination-tree based index provides access to the clauses in the tableau and supports backtracking.

Strategies

This first version of 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.

Expected Competition Performance

E-KRHyper has been completed recently, with limited testing and optimization done, so the expectations are humble.

7.6 Equinox 1.2

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.

7.7 Fampire 1.3

Josef Urban
Charles University of Prague, Czech Republic

Architecture

Fampire 1.3. is the Vampire 8.1 prover using the FLOTTER (SPASS) clausifier.

Implementation

Vampire and (a slightly modified) FLOTTER are linked together using a simple Perl script, Geoff Sutcliffe's tptp4X utility, and slightly modified dfg2tptp utility.

Strategies

The strategy is to use the FLOTTER clausifier, and run Vampire on the CNF problem.

Expected Competition Performance

The system outperforms Vampire 8.1 on Mizar problems, however it is weaker than Vampire 8.1 on the TPTP library (for which is Vampire pre-optimized). It could solve some problems that Vampire 8.1 will not solve.

7.8 Geo 2007j

Hans de Nivelle
University of Wroclaw, Poland

Architecture

Geo [dNM06, dN07] is based on geometric resolution. A geometric formula is a formula of form:

```
FORALL x1, ..., xn
[ A1 /\ ... /\ Ap /\ v1 != w1 /\ ... /\ vq != wq -&gt; Z(x1,...,xn) ].
```

The `Ai` are atoms not containing (dis)equality. The arguments of the atoms `Ai` and the inequalities `vj != wj` are among the variables `x1, ..., xn`. Function symbols and constant symbols are not allowed.

`Z(x1,...,xn)` must have one of the following three forms:

1. The constant `FALSE`.
2. A disjunction `B1 ... Br`. The atoms `Bk` must be non-equality atoms. The arguments of the atoms `Bk` are among the variables `x1, ..., xn`. There are no function symbols, and no constants in the atoms `Bk`.
3. An existential quantification `EXISTS y B`. `y` is a variable distinct from `x1,...,xn`. `B` is an atom which has all arguments among `x1,...,xn,y`. There are no function symbols and constants in `B`.

As input, Geo accepts geometric formulas and first-order formulas. First-order formulas are transformed into geometric formulas. During this translation, function symbols and constants are removed, and replaced by relations.

Geo accepts formulas either in its own syntax or in TPTP-syntax. Because CNF has no special status for Geo, TPTP-formulas in CNF form are treated as ordinary first-order formulas.

During search, Geo searches for a model of the geometric formulas by a combination of backtracking and learning. Whenever it cannot extend an interpretation into a model, Geo learns a new rule of Type 1, which ensures that Geo will not explore similar models in the future.

In case Geo finds a finite model, it simply prints the model. In case no model exists, it will eventually learn the rule -¿ FALSE, so that it is able to output a proof.

The final aim of geometric resolution, and of Geo, is to obtain a prover that is both good at finding proofs, and at finding finite models.

Differences with last year (2006)

- Geo2007j has lemma simplification. Learnt lemmas are used to simplify each other. In addition, inductive consequences of the initial rules are used as simplification rules.
- The heuristics for rule selection have been improved.

Implementation

Geo is written in C++. We try to keep the code readable and reasonably efficient at the same time. Geo has no sophisticated datastructures. At this moment the priority lies with the theoretical understanding of the calculus.

Strategies

Currently, Geo has only one strategy: It searches for a model by exhaustive search, and learns a new rule at each backtracking point. Lemmas do not expire, but they can be simplified.

Expected Competition Performance

Geo has become up to a hundred times faster on hard problems, due to the added simplification rules and the improved selection rules. So we expect that it will solve some more problems than last year.

7.9 leanCoP 2.0

Jens Otten
University of Potsdam, Germany

Architecture

leanCoP is an automated theorem prover for classical first-order logic. It is a very compact implementation of the connection calculus as described in the original leanCoP paper [OB03], which is similar to the connection tableau calculus [LS01].

Implementation

leanCoP is implemented in Prolog. The source code of the core prover is only a few lines long and fits on half a page. In a preprocessing step first-order formulas are translated into clausal form; equality can be handled by adding the equality axioms. leanCoP 2.0 keeps the input clauses in Prolog's database thus integrating Prolog's indexing mechanism into the lean theorem proving framework. leanCoP is available at

`http://www.leancop.de`

Strategies

The reduction rule of the connection calculus is applied before the extension rule is applied. Open branches are selected in a depth-first way. Iterative deepening on the proof depth is used to achieve completeness. leanCoP 2.0 integrates inference rules for regularity, factorization and an additional technique to limit backtracking.

Expected Competition Performance

We expect the performance of leanCoP 2.0 to be state-of-the-art according to the TPTP standards, i.e. it is not subsumed by any other theorem prover on the given problem set.

Acknowledgments: Thomas Raths contributed to the development of leanCoP *2.0 by performing comprehensive benchmark tests on several problem libraries.*

7.10 ileanCoP 1.2

Jens Otten
University of Potsdam, Germany

Architecture

ileanCoP is an automated theorem prover for intuitionistic first-order logic. It is a very compact implementation of the clausal connection calculus for intuitionistic logic [Ott05], prefixes to capture the specific restrictions of intuitionistic logic.

Implementation

ileanCoP is implemented in Prolog and based on the classical prover leanCoP. Only a few minor additions are necessary to turn leanCoP into a theorem prover for intuitionistic logic. The source code of the core prover, which includes the prefix unification algorithm, is only a few lines long and fits on one page. In a preprocessing step formulas are translated into clausal form and a prefix is added to each literal. Equality can be handled by adding the equality axioms. ileanCoP is available at

`http://www.leancop.de/ileancop`

Strategies

ileanCoP uses a classical proof search within the connection calculus to prove classical validity first, since every formula that is valid in intuitionistic logic is valid in classical logic as well. After a classical proof has been found the prefixes of the connected literals are unified using a prefix unification algorithm [OK96]. If unification of all prefixes succeeds the formula is valid in intuitionistic logic. ileanCoP 1.2 integrates some additional techniques from leanCoP 2.0, i.e. inference rules for regularity, factorization and an additional technique to limit backtracking.

Expected Competition Performance

This is the first time an automated theorem prover for intuitionistic logic participates. We expect the performance of ileanCoP 1.2 to be similar to former state-of-the-art theorem provers for classical logic. This assumes that most of the given problems are intuitionistically valid as well. Comprehensive tests on the TPTP library and the ILTP library [ROK07] have shown that this is the case for many problems. Whenever the classical component finds a proof for a given problem and the problem is intuitionistically valid, ileanCoP should find a proof.

Acknowledgments: Thomas Raths contributed to the development of ileanCoP *by performing comprehensive benchmark tests on several problem libraries.*

7.11 iProver 0.2

Konstantin Korovin
The University of Manchester, England

Architecture

iProver is based on the instantiation calculus [GK03], which is complete for first-order logic.

iProver successively instantiates a given set of clauses, and uses a ground SAT solver for checking whether the current ground abstraction of clauses is satisfiable. If the ground solver finds unsatisfiability then the prover terminates, proving unsatisfiability of the initial set of clauses; otherwise the model for ground clauses is used to guide further instantiations. The saturation process is implemented as a modification of a given clause algorithm. We use non-perfect discrimination trees for the unification index. The following redundancy eliminations are implemented: blocking non-proper instantiations; dismatching constraints [GK04]; resolution-based simplifications; propositional-based simplifications, and removing duplicates. Equality is dealt with (internally) by adding the necessary axioms of equality.

Implementation

iProver is implemented in OCaml and for the ground reasoning uses MiniSat [ES05]. iProver accepts cnf and fof formats. In the case of fof format, E prover [Sch02b] is used for clausification.

Strategies

iProver has parameters to control literal selection, passive clause selection, frequency of calling the SAT solver, interleaving instantiation with resolution.

Expected Competition Performance

Since the instantiation method behind iProver is a decision procedure for the EPR class, we expect a reasonable performance in this class.

7.12 Metis 2.0

Joe Hurd
Oxford University, UK

Architecture

Metis 2.0 [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 2.0 implements. Since equality often appears in higher order logic goals, Metis 2.0 also implements the ordered paramodulation calculus.

Implementation

Metis 2.0 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 2.0 uses finite models to weight clauses in the *Waiting* 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. Since it is part of the CASC competition rules that standard functions and relations are obfuscated, Metis 2.0 will back-off to interpreting all functions and relations randomly (except equality).

Metis 2.0 is free software, released under the GPL. It can be downloaded from
`http://www.gilith.com/software/metis`

Strategies

Metis 2.0 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.

Expected Competition Performance

Running previous versions of Metis on previous CNF problem sets has scored in the bottom half of the table, so that is the expected performance. Metis 2.0 always keeps proofs, so the performance will be no different in the proof classes.

Testing on previous UEQ problem sets has shown Metis to be weak on equality problems, so I expect Metis 2.0 to perform relatively worse on the equality divisions, particularly UEQ.

7.13 Muscadet 2.7

Dominique Pastre
Université René Descartes (Paris 5), France

Architecture

The Muscadet theorem prover [Pas01] 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.

Implementation

Muscadet 2 [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

can be obtained in natural style. Muscadet 2 is available from
`http://www.math-info.univ-paris5.fr/ pastre/muscadet/muscadet.html`

Strategies

There are specific strategies for existential, universal, conjunctive 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, Pas93]. These two possibilities are not used while working with the TPTP Library.

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.

7.14 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.

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/`

Strategies

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

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.

7.15 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 [McC94] 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.

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.

Strategies

There is only one strategy in Paradox:

- 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.
- Flatten the problem, and split clauses and simplify as much as possible.
- 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.
- 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.

Expected Competition Performance

Paradox 1.3 is the CASC-J3 SAT division winner.

7.16 Paradox 2.2

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.

See the description of Paradox 1.3 for general information.

Expected Competition Performance

Paradox 2.2 will probably perform comparably to Paradox 1.3.

7.17 Vampire 8.1

Andrei Voronkov
University of Manchester, England

No system description supplied.

Expected Competition Performance

Vampire 8.1 is the CASC-J3 FOF and CNF division winner.

7.18 Vampire 9.0

Andrei Voronkov
University of Manchester, England

No system description supplied.

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-J3 UEQ division winner.

8 Conclusion

The CADE-21 ATP System Competition is the twelfth 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

[BFdNT06] P. Baumgartner, A. Fuchs, H. de Nivelle, and C. Tinelli. Computing Finite Models by Reduction to Function-Free Clause Logic. In W. Ahrendt, P. Baumgartner, and H. de Nivelle, editors, *Proceedings of the 3rd Workshop on Disproving - Non-Theorems, Non-Validity, Non-Provability, 3rd International Joint Conference on Automated Reasoning*, 2006.

[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. 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*, Lecture Notes in Artificial Intelligence, 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.

[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.

[dN07] H. de Nivelle. Redundancy for Geometric Resolution. In W. Ahrendt, P. Baumgartner, and H. de Nivelle, editors, *Proceedings of CADE-21 Workshop on DISPROVING - Non-Theorems, Non-Validity, Non-Provability*, 2007.

[dNM06] H. de Nivelle and J. Meng. Geometric Resolution: A Proof Procedure Based on Finite Model Search. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, pages 303–317. Springer-Verlag, 2006.

[ES05] N. Eén and N. Sörensson. MiniSat - A SAT Solver with Conflict-Clause Minimization. In F. Bacchus and T. Walsh, editors, *Posters of the 8th International Conference on Theory and Applications of Satisfiability Testing*, 2005.

[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.

[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.

[LS01] R. Letz and G. Stenz. System Description: DCTP - A Disconnection Calculus Theorem Prover. In R. Gore, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning*, number 2083 in Lecture Notes in Artificial Intelligence, pages 381–385. Springer-Verlag, 2001.

[McC94] 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.

[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.

[OK96] J. Otten and C. Kreitz. T-String-Unification: Unifying Prefixes in Non-Classical Proof Methods. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Proceedings of the 5th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, number 1071 in Lecture Notes in Computer Science, pages 244–20. Springer-Verlag, 1996.

[Ott05] J. Otten. Clausal Connection-Based Theorem Proving in Intuitionistic First-Order Logic. In B. Beckert, editor, *Proceedings of the 14th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, number 3702 in Lecture Notes in Artificial Intelligence, pages 245–261. Springer-Verlag, 2005.

[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. Springer-Verlag, 2007.

[ROK07] T. Raths, J. Otten, and C. Kreitz. The ILTP Problem Library for Intuitionistic Logic - Release v1.1. *Journal of Automated Reasoning*, 38(1-2):261–271, 2007.

[Sch02a] 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.

[Sch02b] S. Schulz. E: A Brainiac Theorem Prover. *AI Communications*, 15(2-3):111–126, 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.

[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.

[Sut07] G. Sutcliffe. The 3rd IJCAR Automated Theorem Proving Competition. *AI Communications*, page To appear, 2007.

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

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