# Proceedings of the 5th IJCAR ATP System Competition CASC-J5

Geoff Sutcliffe
University of Miami, USA

**Abstract**

The CADE ATP System Computer (CASC) evaluates the performance of sound, fully automatic, classical 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 specified time limits on solution attempts. The 5th IJCAR ATP System Competition (CASC-J5) was held on 17th July 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-J5 was held on 17th July 2010, as part of the 5th International Joint Conference on Automated Reasoning (IJCAR 2010)[1], in Edinburgh, United Kingdom. It is the fifteenth competition in the CASC series [113, 118, 116, 87, 89, 112, 110, 111, 94, 96, 98, 100, 103, 105].

CASC evaluates the performance of sound, fully automatic, classical 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 [104], and
- specified time limits on solution attempts.

Thirty-eight ATP systems and variants, listed in Table 1, entered into the various competition and demonstration divisions. The winners of the CASC-22 (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).[2]

The design and procedures of this CASC evolved from those of previous CASCs [113, 114, 109, 115, 85, 86, 88, 90, 91, 92, 93, 95, 97, 99, 102]. Important changes for this CASC were:

- The THF division became a full competition division.

[1]CADE was a constituent conference of IJCAR, hence CASC-"J5".

[2]The CASC-22 LTB winner, Vampire-LTB 11.0, was only roughly adapted to the changes to the organization of the CASC-J5 LTB division. It's performance was thus compromised.

Table 1: The ATP systems and entrants

| ATP System | Divisions | Entrants (Associates) | Entrant's Affiliation |
|---|---|---|---|
| Ayane 2 | FOF FNT CNF EPR UEQ | Russell Wallace | Independent researcher |
| Currahee(E,iProver) 0.1 | LTB | Matthias Schmalz (Jann Röder) | ETH Zurich |
| Darwin 1.4.5 | EPR | Peter Baumgartner | NICTA |
| E/EP 1.2pre | FOF FNT CNF EPR UEQ LTB | Stephan Schulz | Technische Universität München |
| E-Darwin 1.3 | FOF FNT CNF EPR | Björn Pelzer | Universität Koblenz-Landau |
| E-KRHyper 1.1.4 | FOF FNT CNF EPR | Björn Pelzer | Universität Koblenz-Landau |
| Equinox 5.0 | FOF FNT CNF EPR UEQ | Koen Claessen | Chalmers University |
| Geo 2010C | FOF FNT CNF EPR UEQ | Hans de Nivelle | Uniwersytetu Wroclawskiego |
| iProver 0.7 | EPR | CASC | CASC-22 EPR winner |
| iProver(-SInE) 0.8 | FOF FNT CNF EPR UEQ LTB | Konstantin Korovin | University of Manchester |
| iProver-Eq(-SInE) 0.6 | FOF FNT CNF EPR UEQ LTB | Christoph Sticksel (Konstantin Korovin) | University of Manchester |
| Isabelle/HOL 2009-2 | THF | Jasmin Christian Blanchette (Larry Paulson, Tobias Nipkow, Makarius Wenzel, Stefan Berghofer) | Technische Universität München |
| leanCoP(-SInE) 2.2 | FOF LTB | Jens Otten (Thomas Raths) | University of Potsdam |
| leanCoP-Ω 0.1 | TFA | Jens Otten (Holger Trölenberg, Thomas Raths) | University of Potsdam |
| LEO-II 1.2 | THF FOF CNF | Christoph Benzmüller (Frank Theiss) | Articulate Software |
| Metis 2.2 | FOF FNT CNF EPR UEQ | Joe Hurd | Galois, Inc. |
| MetiTarski 1.3 | TFA | Larry Paulson | University of Cambridge |
| Muscadet 4.0 | FOF | Dominique Pastre | University Paris Descartes |
| omkbTT 1.0 | UEQ | Sarah Winkler | University of Innsbruck |
| Otter 3.3 | FOF CNF UEQ | CASC (William McCune) | CASC |
| Paradox 3.0 | FNT | CASC | CASC-22 FNT winner |
| Paradox 4.0 | FNT EPR | Koen Claessen | Chalmers University |
| Satallax 1.4 | THF | Chad E. Brown (Julian Backes, Gert Smolka) | Saarland University |
| SNARK 20080805r027 | TFA | Mark Stickel | SRI International |
| SPASS+T 2.2.12 | TFA | Uwe Waldmann (Stephan Zimmer) | Max Planck Institut für Informatik |
| SPASS-XDB 3.01X0.5 | TFA | Geoff Sutcliffe (Martin Suda) | University of Miami |
| TPS 3.080227G1d | THF | Chad E. Brown (Peter B. Andrews) | CASC-22 THF winner |
| Vampire 10.0 | CNF | CASC | CASC-22 CNF winner |
| Vampire 11.0 | FOF | CASC | CASC-22 FOF winner |
| Vampire-LTB 11.0 | LTB | CASC | CASC-22 LTB winner |
| Vampire 0.6 | FOF CNF EPR UEQ LTB | Andrei Voronkov (Kryštof Hoder) | University of Manchester |
| Waldmeister C09a | UEQ | CASC | CASC-22 UEQ winner |
| Waldmeister 710 | UEQ | Thomas Hillenbrand | Max-Planck-Institut für Informatik |
| Zenon 0.6.3 | FOF | Damien Doligez | INRIA |

- The FOF division had three problem categories - the existing FNE and FEQ categories, and a new FEP category for FOF effectively propositional problems (which were removed from the FNE and FEQ problem categories). Problems were placed in the FEP category if their CNF is EPR (using E's FOF to CNF converter).
- The SAT division was suspended.
- The LTB division had only an assurance ranking class.
- A TFA (Typed First-order with Arithmetic) demonstration division was added.
- The problems in each division and LTB problem category were given in increasing order of TPTP difficulty rating (this is aesthetic in the non-LTB divisions, but practically important in the LTB problem categories where it is possible to learn from proofs found earlier in the batch).
- In the LTB division:
  - The division was run on quad-core computers. Systems were encouraged to take advantage of the multiple cores.
  - The batch specification file had a new configuration section.
  - There was a per-problem wall clock time limit, and an overall wall clock time limit that was the per-problem limit multiplied by the number of problems. The per-problem wall clock time limit was announced to be in the range 30s-60s. These time limits were given in the configuration section at the start of the batch specification file. The overall wall clock time limit was also (as previously) available as a command line parameter. There were no CPU time limits.
- The efficiency measure was changed, to use the average of the individual solution rates rather than the overall solution rate.
- Timing was to 100ths of a second.

The competition organizer was Geoff Sutcliffe. The competition was overseen by a panel of knowledgeable researchers who were not participating in the event; the CASC-J5 panel members were Nikolaj Bjørner, John Slaney, and Christoph Weidenbach. The CASC rules, specifications, and deadlines are absolute. Only the panel has the right to make exceptions. The competition was run on computers provided by the Max-Planck-Insitut für Informatik, Saarbrücken, Germany, and the Department of Computer Science, University of Manchester, United Kingdom. The CASC-J5 web site provides access to resources used before, during, and after the event: `http://www.tptp.org/CASC/J5`

It is assumed that all entrants have read the web pages related to the competition, and have complied with the competition rules. Non-compliance with the rules could lead to disqualification. A "catch-all" rule is used to deal with any unforeseen circumstances: *No cheating is allowed.* The panel is allowed to disqualify entrants due to unfairness, and to adjust the competition rules in case of misuse.

## 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 analyse, 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. 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 **THF** division: Typed Higher-order Form non-propositional theorems (axioms with a provable conjecture), using only the THF0 syntax. The THF division has two problem categories:

- The **TNE** category: THF with No Equality
- The **TEQ** category: THF with EQuality

The **FOF** division: First-Order Form non-propositional theorems (axioms with a provable conjecture). The FOF division has three problem categories:

- The **FNE** category: FOF with No Equality, not (obviously) effectively propositional
- The **FEQ** category: FOF with Equality, not (obviously) effectively propositional
- The **FEP** category: FOF Effectively Propositional

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 **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 in Batches. 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`, and `CSR118`.

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 (e.g., the system requires special hardware, or the entrant is an organizer) 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 demonstration division 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. The entry specifies which competition divisions' problems are to be used.

In CASC-J5, in addition to the competition divisions, there was an additional demonstration division:

- The **TFA** division: Typed First-order with Arithmetic theorems (axioms with a provable conjecture, using the TFF0 syntax and the proposed TPTP arithmetic format). These were limited to integer arithmetic problems.

## 3 Infrastructure

### 3.1 Computers

The non-LTB division computers were Dual-Opteron computers, each having:

- Two AMD Opteron(tm) Processor 250, 2390MHz CPUs
- 7.5GB memory
- Linux mpicl3-04 2.6.30.10.1.amd64-smp operating system

The LTB division computers were quad-core Dell server computers, each having:

- Four Intel(R) Xeon(R) L5410, 2.333GHz CPUs
- 12GB memory
- Linux 2.6.29.4-167.fc11.x86_64 operating system

### 3.2 Problems

#### 3.2.1 Problem Selection

The problems were taken from the TPTP problem library, version v4.0.0. 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 [117]:
  - 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 (especially the LTB division, because the TPTP problem ratings are computed from sequential mode results). 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.
- In the LTB division, the problems are selected so that there is consistent symbol usage between problems in each category, but there may not be consistent axiom naming between problems.

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 [43] (the competition organizers have to ensure that there are sufficient computers available to run the ATP systems on this minimal number of problems). The minimal numbers of problems is used in determining the time limits 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 per-problem time limit, according to the following relationship:

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

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

The numbers of problems used in the categories in the various divisions are (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

The problems are in TPTP format, with `include` directives (included files are found relative to the `TPTP` environment variable). The problems in each division and LTB problem category are given in increasing order of TPTP difficulty rating (this is aesthetic in the non-LTB divisions, but practically important in the LTB problem categories where it is possible to learn from proofs found earlier in the batch).

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

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

In order 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. `CCC` is the division or problem category name, and `NNN` runs from `001` to the number of problems in the respective division or problem 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 preprocessing applied. However, the original fille names can be retained for systems running on computers provided by the entrant.

In the LTB division, the problems for each category are listed in a *batch specification file*, containing:

- A header line `% SZS start BatchConfiguration`
- The problem category is given in a line of the form `division.category LTB.`*category_mnemonic*
- The per-problem wall clock time limit is given in a line of the form `limit.time.problem.wc` *limit_in_seconds*
- The overall wall clock time limit is given in a line of the form `limit.time.overall.wc` *limit_in_seconds*
- A terminator line `% SZS end BatchConfiguration`
- A header line `% SZS start BatchIncludes`
- `include` directives that are used in every problem. Problems in the batch have all these `include` directives, and can also have other `include` directives that are not listed here.
- A terminating line `% SZS end BatchIncludes`
- A header line `% SZS start BatchProblems`
- Pairs of absolute problem file names, and absolute output file names where the output for the problem must be written.
- A terminator line `% SZS end BatchProblems`

## 3.3 Resource Limits

**Non-LTB 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 *NumberOfProblems*. 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. The time limits are imposed individually on each solution attempt.

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.

**LTB division**
There is a per-problem wall clock time limit, and an overall wall clock time limit that is the per-problem limit multiplied by the number of problems. The per-problem wall clock time limit is in the range 30s-60s. These time limits are given in the configuration section at the start of the batch specification file. The overall wall clock time limit is also available as a command line parameter. There are no CPU time limits.

## 4 System Evaluation

For each ATP system, for each problem, four items of data are recorded: whether or not a solution was found, the CPU time taken, the wall clock time taken, and whether or not a solution (proof or model) was output. In the LTB division, time spent before starting the first problem, and times spent between ending a problem and starting the next, are not part of the time taken on problems.

The systems are ranked at the division level from the 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 and FNT divisions additionally have 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 time over problems solved (CPU time for the non-LTB divisions, wall clock time for the LTB division). In the competition divisions, class winners are announced and prizes are awarded.

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.
- 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 are highlighted in the presentation of results.

In addition to the ranking criteria, two other measures are made and presented in the results: The *state-of-the-art contribution* (SOTAC) quantifies the unique abilities of the systems. For each problem solved by a system, its SOTAC for the problem is the inverse of the number of systems that solved the problem, and a system's overall SOTAC is the average SOTAC over the problems it solves. The *efficiency measure* balances the number of problems solved with the CPU time taken. It is the average of the inverses of the times for problems solved (CPU times for the non-LTB divisions, wall clock times for the LTB division, with times less that the timing granularity rounded up to that granularity, to avoid skewing caused by very low times) multiplied by the fraction of problems solved. This can be interpreted intuitively as the average of the solution rates (for problems solved) multiplied by the fraction of problems solved.

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). 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, 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. 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. 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. The division winners from the previous CASC are automatically entered into their divisions, to provide benchmarks against which progress can be judged.

### 5.1 System Description

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

- Architecture. This section introduces the ATP system, and describes the calculus and inference rules used.
- Strategies. This section describes the search strategies used, why they are effective, and how they are selected for given problems. Any strategy tuning that is based on specific

problems' characteristics must be clearly described (and justified in light of the tuning restrictions described in Section 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. The availability of system is described here.
- Expected competition performance. This section makes some predictions about the performance of the ATP system in each of the divisions and categories in which it is competing.
- References.

The system description has to be emailed to the competition organizers by the system description deadline. The system descriptions, 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 by the sample solutions deadline. Use of the TPTP format for proofs and finite interpretations is encouraged. Proof samples for the FOF proof class must include a proof for `SYN075+1`. Model samples for the FNT model class must include models for `MGT019+2` and `SWV010+1`. The sample solutions must illustrate the use of all inference rules. An explanation must be provided for any non-obvious features.

## 6 System Requirements

### 6.1 System Properties

Systems are required to have the following properties. Entrants must ensure that their systems execute in a competition-like environment, and have the following properties. Entrants are advised to check these well in advance of the system delivery deadline. This gives the competition organizers time to help resolve any difficulties encountered. Entrants do not have access to the competition computers.

#### 6.1.1 Soundness and Completeness

- 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 THF, FOF, CNF, EPR, UEQ, and LTB divisions, and theorems are submitted to the systems in the FNT 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.

- All techniques used must be general purpose, and expected to extend usefully to new unseen problems. The precomputation and storage of information about individual TPTP problems and axiom sets is not allowed. Strategies and strategy selection based on individual TPTP problems is not allowed. If machine learning procedures are used, the learning must ensure that sufficient generalization is obtained so that no there is no specialization to individual problems or their solutions.
- The system's performance must be reproducible by running the system again.

### 6.1.2 Execution

- 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 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. In the LTB division the command line arguments are the absolute path name of the batch specification file, the aggregated batch 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.
- Systems must be fully automatic, i.e., any command line switches have to be the same for all problems in each division.
- In the LTB division the systems must attempt the problems in the order given in the batch specification file. Systems may not start any attempt on a problem, including reading the problem file, before ending the attempt on the preceding problem.

### 6.1.3 Output

- 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.
- For each problem, the systems have to output a distinguished string (specified by the entrant), indicating what solution has been found or that no conclusion has been reached. The distinguished strings should use the SZS ontology and standards [101]. 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) (SZS status `Satisfiable`)

  The first distinguished string output is accepted as the system's result.

- When outputting proofs/models, the start and end of the proof/model must be delimited by distinguished strings (specified by the entrant). The distinguished strings should use the SZS ontology and standards. 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. Use of the TPTP format for proofs and finite interpretations is encouraged [107].
- In the LTB division the systems must print SZS notification lines to `stdout` when starting and ending all work on a problem (including any cleanup work, such as deleting temporary files). It is recommended that the result for the problem be output as the last thing before the ending notification line (note, the result must also be output to the solution file anyway). For example

  ```
  % SZS status Started for /home/graph/tptp/TPTP/Problems/CSR/CSR075+2.p
    ... (system churns away, result and solution output to file)
  % SZS status Theorem for /home/graph/tptp/TPTP/Problems/CSR/CSR075+2.p
  % SZS status Ended for /home/graph/tptp/TPTP/Problems/CSR/CSR075+2.p
  ```

  Once the `Ended` notification is received the output file is time stamped and may not be changed. Systems may spend any amount of time before starting the first problem (e.g., preloading and analysing the batch axioms), and any amount of time between ending a problem and starting the next (e.g., learning from the proof just found). This time is not part of the time used for any problem, but is part of the overall time for the batch.

### 6.1.4 Resource Usage

- The systems that run on the competition computers have to be interruptible by a `SIGXCPU` signal, so that the CPU time limit can be imposed, and interruptible by a `SIGALRM` signal, so that the wall clock time limit can be imposed. For systems that create multiple processes, the signal is sent first to the process at the top of the hierarchy, then one second later to all processes in the hierarchy. The default action on receiving these signals is to exit (thus complying with the time limit, as required), but systems may catch the signals and exit of their own accord. If a system runs past a time limit this is noticed in the timing data, and the system is considered to have not solved that problem.
- If an ATP system terminates of its own accord, it may not leave any temporary or intermediate output files. If an ATP system is terminated by a `SIGXCPU` or `SIGALRM`, it may not leave any temporary or intermediate files anywhere other than in `/tmp`. Multiple copies of the ATP systems have to be executable concurrently on different computers 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 computers and other processes. An adequate solution is a file name including the host computer 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.

## 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 `.tgz` 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, using `%s` and `%d` to indicate where the problem file name and time limit must appear in the command line.
- The distinguished strings indicating what solution has been found, and delimiting proofs/models.

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. If the ATP system requires any special software, libraries, etc, which is not part of a standard installation, the competition organizers must be told in the system registration. 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 web site. In the demonstration division, the entrant specifies whether or not the source code is placed on the CASC web 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 time used to that point is noted. A system has solved a problem iff it outputs its termination string within the time limit, and a system has produced a proof/model iff it outputs its end-of-proof/model string within the time limit. The result and timing data is used to generate an HTML file, and a web browser is used to display the results.

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

## 7 The ATP Systems

These system descriptions were written by the entrants.

### 7.1 Ayane 2

Russell Wallace
Independent researcher, Ireland

**Architecture**
Ayane is an automated theorem prover for first-order logic. It uses refutation by saturation, converting the input to clause normal form then generating additional clauses until a contradiction is found or until no further inference can be performed. Inference is based on the superposition calculus.

**Strategies**
The current version uses a straightforward implementation of the given clause algorithm, with clauses selected for processing in order of size (smallest first) and basic filtering of generated clauses for duplication and tautologies.

**Implementation**
Ayane is written in C#, and has been tested on Linux (Mono 2.4.4, should work with later or somewhat earlier versions) and Windows (.Net); it is expected to work on any platform that supports a reasonably up to date version of the CLR.

The system is available from `http://code.google.com/p/ayane/`.

**Expected Competition Performance**
Ayane is not expected to win the competition yet, as it is still in early development and much work on efficient inference remains to be done.

### 7.2 Currahee(E,iProver) 0.1

Matthias Schmalz, Jann Röder
ETH Zurich, Switzerland

**Architecture**
Currahee(E,iProver) 0.1 applies a number of axiom selection strategies to a problem in parallel, and passes the resulting smaller problems to E and iProver. Currahee(E,iProver) relies on the assumption that several axiom selection strategies and theorem provers are more successful than one axiom selection strategy with one theorem prover. The challenge was to find out the strengths of the individual selection strategies, i.e., when to apply which strategy.

**Strategies**
Currahee(E,iProver) applies the following axiom selection strategies (with slight adjustments):

- the algorithm underlying SInE,
- contextual indirect relevance [81],

- Meng and Paulson's relevance filter for Sledgehammer [60],
- combinations of these strategies.

For a given problem the strategies are chosen depending on the problem's size.

**Implementation**
Problems are parsed using Andrei Tchaltsev's TPTP parser (in Java). The axiom selection strategies have been reimplemented in Java. The underlying theorem provers are E 1.1 and iProver 0.8pre.

After the competition, Currahee(E,iProver) can be obtained from `http://www.infsec.ethz.ch/people/mschmalz`.

**Expected Competition Performance**
Based on preliminary measurements, we expect Currahee(E,iProver) to solve at least half of the problems from last year's LTB division (applying this year's resource restrictions).

### 7.3 Darwin 1.4.5

Peter Baumgartner
NICTA, Australia

**Architecture**
Darwin [11, 12] is an automated theorem prover for first order clausal logic. It is an implementation of the Model Evolution calculus [15]. 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, analogous 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.

**Strategies**
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. 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.

**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 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/`.

Changes to the previous version consist of minor bug fixes.

**Expected Competition Performance**
We expect its performance to be very strong in the EPR division.

## 7.4 E/EP 1.2pre

Stephan Schulz
Technische Universität München, Germany

**Architecture**
E 1.2pre [82, 83] is a purely equational theorem prover. The core proof procedure operates on formulae 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. 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 formulae in full first order format to clause normal form. This step may introduce (first-order) definitions to avoid an exponential growth of formulae. Preprocessing also unfolds equational definitions and performs some simplifications on the clause level.

EP 1.2pre is just a combination of E 1.2pre in verbose mode and a proof analysis tool extracting the used inference steps.

**Strategies**
The automatic mode determines literal selection strategy, term ordering, and search heuristic based on simple problem characteristics of the preprocessed clausal problem. E has been optimized for performance over the TPTP. The automatic mode of E 1.2pre is partially inherited from previous version and is based on about 60 test runs over TPTP 4.0.1. It consists of the selection of one of about 40 different strategies for each problem. All test runs have been performed on Linux/Intel machines with a time limit roughly equivalent to 300 seconds on 300MHz Sun SPARC machines, i.e., around 30 seconds on 2Ghz class machines. All individual strategies are refutationally complete.

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?

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.

For the LTB part of the competition, E will use a relevancy-based pruning approach and attempt to solve the problems with successively more complete specifications until it succeeds or runs out of time.

**Implementation**
E is implemented in ANSI C, using the GNU C compiler. At the core is an 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. E uses *perfect discrimination trees* with age and size constraints for rewriting and unit-subsumption, *feature vector indexing* [83] for forward- and backward subsumption and contextual literal cutting, and a new technique called *fingerprint indexing* for backward rewriting and (hopefully) paramodulation. Knuth-Bendix Ordering and Lexicographic Path Ordering are implemented using the linear and polynomial algorithms described by Bernd Löchner [54, 55].

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 1.2pre 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 1.2pre 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 systems.

## 7.5 E-Darwin 1.3

Björn Pelzer
Universität Koblenz-Landau, Germany

**Architecture**
E-Darwin is an automated theorem prover for first order clausal logic with equality. It is a modified version of the Darwin prover [11], intended as a testbed for variants of the Model Evolution calculus [15]. Among other things, it implements the Model Evolution with Equality

calculus [16]. Also, since the last version, a new calculus has been implemented, using a different approach to incorporating equality reasoning into Model Evolution. This new calculus will be used for CASC this year. Three principal data structures are used: the context (a set of rewrite literals), the set of constrained clauses, and the set of derived candidates. The prover always selects one candidate, which may be a new clause or a new context literal, and exhaustively computes inferences with this candidate and the context and clause set, moving the results to the candidate set. Afterwards the candidate is inserted into one of the context or the clause set, respectively, and the next candidate is selected. The inferences are superposition-based. Demodulation and various means of redundancy detection are used as well.

**Strategies**
The uniform search strategy is identical to the one employed in the original Darwin, slightly adapted to account for derived clauses.

**Implementation**
E-Darwin is implemented in the functional/imperative language OCaml. The system has been tested on Unix and is available under the GNU Public License from `http://www.uni-koblenz.de/~bpelzer/edarwin`.

Darwin's method of storing partial unifiers has been adapted to equations and subterm positions for the superposition inferences in E-Darwin. A combination of perfect and non-perfect discrimination tree indexes is used to store the context and the clauses.

**Expected Competition Performance**
The new calculus used this year differs significantly from the previous versions , which still had a lot in common with the original Model Evolution. The implementation was completed very recently. As such we do not yet know what performance to expect.

## 7.6 E-KRHyper 1.1.4

Björn Pelzer
Universität Koblenz-Landau, Germany

**Architecture**
E-KRHyper [76] is a theorem proving and model generation system for first-order logic with equality. It is an implementation of the E-hyper tableau calculus [14], which integrates a superposition-based handling of equality [9] into the hyper tableau calculus [13]. The system is an extension of the KRHyper theorem prover [123], 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 [119] 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. The system runs on Unix and MS-Windows platforms and is available under the GNU Public License, from `http://www.uni-koblenz.de/~bpelzer/ekrhyper`.

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 [58]. 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.

**Expected Competition Performance**

Most of the work done on E-KRHyper since the last version is only related to its operation as a reasoning server within the question-answering system LogAnswer [40], and these modifications have no effect on CASC-performance. Therefore we expect it to be comparable to Otter, like last year.

## 7.7 Equinox 5.0

Koen Claessen
Chalmers University of Technology, Sweden

No system description supplied.

## 7.8 Geo 2010C

Hans de Nivelle
Uniwersytetu Wroclawskiego, Poland

**Architecture**

Geo is based on geometric resolution [38, 36, 37]. A geometric formula is a formula without function symbols and constants, which has form:

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

The `Ai` are atoms, which are not equality atoms or inequality atoms. The arguments of the atoms `Ai` and the inequalities `vj != wj` must be among the variables `x1, ..., xn`. `Z(x1,...,xn)` must have one of the following three forms:

1. The constant `FALSE`.
2. A disjunction `B1`
   `/ ...`
   `/ Br`, in which the atoms `Bk` are non-equality atoms, and the arguments of each atom `Bk` are among the variables `x1,..., xn`.
3. An existential quantification `EXISTS y B`, in which `y` is a variable distinct from `x1,...,xn`, and `B` is a non-equality atom that has all its variables among `x1,...,xn,y`.

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

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

During search, Geo searches for a model of the geometric formulae 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.

**Strategies**

Currently, Geo has only one strategy: It searches for a model by depth-first search, and learns a new lemma at each backtracking point. Lemmas never expire, but they are sometimes simplified into smaller lemmas.

**Implementation**

Geo is written in C++. We try to keep the code readable and reasonably efficient at the same time. Currently, Geo has no sophisticated data structures for efficiency. Our priority lies with the theoretical understanding of geometric resolution, adding a rich type system with partial functions to Geo, implementing full natural deduction proof output, and adding other features that might be useful in interactive verification.

**Expected Competition Performance**

There is no significant difference with earlier versions of Geo. We expect to end somewhere in the middle.

### 7.9 iProver 0.7

Konstantin Korovin
University of Manchester, United Kingdom

**Architecture**
iProver is an automated theorem prover based on an instantiation calculus Inst-Gen [41, 49] which is 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 [39] 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 [48] 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 [42, 48]; global subsumption [48]; resolution-based simplifications and propositional-based simplifications. We implemented a compressed feature vector index for efficient forward/backward subsumption and subsumption resolution. Equality is dealt with (internally) by adding the necessary axioms of equality.

**Strategies**
iProver has around 40 options to control the proof search including options for literal selection, passive clause selection, frequency of calling the SAT solver, simplifications and options for combination of instantiation with resolution. At CASC iProver will execute a small number of fixed schedules of selected options depending on general syntactic properties such as Horn/non-Horn, equational/non-equational, and maximal term depth.

**Implementation**
iProver is implemented in OCaml and for the ground reasoning uses MiniSat. iProver accepts FOF and CNF formats, in the case of FOF format, E prover is used for clausification. iProver is available from `http://www.cs.man.ac.uk/~korovink/iprover/`.

**Expected Competition Performance**
iProver 0.7 is the CASC-22 EPR division winner.

### 7.10 iProver(-SInE) 0.8

Konstantin Korovin
University of Manchester, United Kingdom

**Architecture**
iProver is an automated instantiation-based theorem prover for first-order logic. We refer to description of iProver 0.7 for general information. Major additions in the current version are:

- Model output using first-order definitions in term algebra.
- Incrementality wrt. model changes in the SAT solving part.
- New index for dismatching constraints.

For the LTB division, axiom selection and clausification is done by Vampire 0.6. SInE scripts are used for problem scheduling and iProver for reasoning.

**Implementation**
iProver accepts FOF and CNF formats, where either Vampire [80] or E prover [83] is used for clausification of FOF problems. iProver is available from `http://www.cs.man.ac.uk/~korovink/iprover/`.

**Expected Competition Performance**
iProver 0.8 is expected to have strong performance in FOF, CNF, FNT, SAT and LTB divisions, and to considerably outperform iProver 0.7 in the EPR division.

## 7.11 iProver-Eq(-SInE) 0.6

Konstantin Korovin, Christoph Sticksel
University of Manchester, United Kingdom

**Architecture**
iProver-Eq [50] extends the iProver system [48] with built-in equational reasoning, along the lines of [42]. As in the iProver system, first-order reasoning is combined with ground satisfiability checking, where the latter is delegated to an off-the-shelf ground solver.

iProver-Eq consists of three core components: i) ground reasoning by an SMT solver, ii) first-order equational reasoning on literals in a candidate model by a labelled unit superposition calculus [50] and iii) instantiation of clauses with substitutions obtained by ii).

Given a set of first-order clauses, iProver-Eq first abstracts it to a set of ground clauses which are passed to the ground solver. If the ground abstraction is unsatisfiable, then the set of first-order clauses is also unsatisfiable. Otherwise, literals are selected from the first-order clauses based on the model of the ground solver. The labelled unit superposition calculus checks whether the selected literals are conflicting. If they are conflicting, then clauses are instantiated such that the ground solver has to refine its model in order to resolve the conflict. Otherwise, satisfiability of the initial first-order clause set is shown.

Clause selection and literal selection in the unit superposition calculus are implemented in separate given clause algorithms. Relevant substitutions are accumulated in labels during unit superposition inferences and then used to instantiate clauses. For redundancy elimination iProver-Eq uses demodulation and dismatching constraints. In order to efficiently propagate redundancy elimination from instantiation into unit superposition, we implemented different representations of labels based on sets, AND/OR-trees and OBDDs. Non-equational resolution inferences provide further simplifications.

For the LTB division, axiom selection and clausification is done by Vampire 0.6. SInE scripts are used for problem scheduling and iProver-Eq for reasoning.

**Strategies**
Proof search options in iProver-Eq control clause and literal selection in the respective given clause algorithms. Equally important is the global distribution of time between the inference engines and the ground solver. At CASC, iProver-Eq will execute a fixed schedule of selected options.

If no equational literals occur in the input, iProver-Eq falls back to the inference rules of iProver, otherwise the latter are disabled and only unit superposition is used. If all clauses are unit equations, no instantiations need to be generated and the calculus is run without the otherwise necessary bookkeeping.

**Implementation**
iProver-Eq is implemented in OCaml and uses CVC3 [15] for the ground reasoning in the equational case and MiniSat [39] in the non-equational case. iProver-Eq accepts FOF and CNF formats, where either Vampire [80] or the E prover [83] is used for clausification of FOF problems. iProver-Eq is available from `http://www.cs.man.ac.uk/~korovink/iprover`.

**Expected Competition Performance**
iProver-Eq has significantly improved over the version in the previous CASC. We expect reasonably good performance in all divisions, including the EPR divisions where instantiation-based methods are particularly strong.

### 7.12 Isabelle/HOL 2009-1

Jasmin C. Blanchette[1], Lawrence C. Paulson[2], Tobias Nipkow[1], Makarius Wenzel[1], Stefan Berghofer[1]
[1]Technische Universität München, Germany
[2]University of Cambridge, United Kingdom

**Architecture**
Isabelle/HOL 2009-1 [63] is the higher-order logic incarnation of the generic proof assistant Isabelle2009-1. Isabelle/HOL provides several automatic proof tactics, notably an equational reasoner [62], a classical reasoner [75], a tableau prover [74], and a first-order resolution-based prover [46].

Although Isabelle is designed for interactive proof development, it is a little known fact that it is possible to run Isabelle from the command line, passing in a theory file with a formula to solve. Isabelle theory files can include Standard ML code to be executed when the file is processed. The `TPTP2X` Isabelle format module outputs a THF problem in Isabelle/HOL syntax, augmented with ML code that (1) runs the ten tactics in sequence, each with a CPU time limit, until one succeeds or all fail, and (2) reports the result and proof (if found) using the SZS standards. A Perl script is used to insert the CPU time limit (equally divided over the ten tactics) into `TPTP2X`'s Isabelle format output, and then run the command line `isabelle-process` on the resulting theory file.

**Strategies**
The Isabelle/HOL wrapper submitted to the competition tries the following tactics sequentially:

**simp** Performs equational reasoning using rewrite rules.

**blast** Searches for a proof using a fast untyped tableau prover and then attempts to reconstruct the proof using Isabelle tactics.

**auto** Combines simplification and classical reasoning under one roof.

**metis** Combines ordered resolution and ordered paramodulation. The proof is then reconstructed using Isabelle tactics.

**fast** Searches for a proof using sequent-style reasoning, performing a depth-first search. Unlike `blast` and `metis`, they construct proofs directly in Isabelle. That makes them slower, but enables them to work in the presence of the more unusual features of HOL, such as type classes and function unknowns.

**fastsimp** Combines `fast` and `simp`.

**best** Similar to `fast`, except that it performs a best-first search.

**force** Similar to `auto`, but more exhaustive.

**meson** Implements Loveland's MESON procedure [56]. Constructs proofs directly in Isabelle.

**smt** Invokes the Z3 SMT solver [35] developed at Microsoft Research, and reconstructs the proofs in Isabelle [27].

**Implementation**

Isabelle is a generic theorem prover written in Standard ML. Its meta-logic, Isabelle/Pure, provides an intuitionistic fragment of higher-order logic. The HOL object logic extends pure with a more elaborate version of higher-order logic, complete with the familiar connectives and quantifiers. Other object logics are available, notably FOL (first-order logic) and ZF (Zermelo-Fraenkel set theory).

The implementation of Isabelle relies on a small LCF-style kernel, meaning that inferences are implemented as operations on an abstract `theorem` data type. Assuming the kernel is correct, all values of type `theorem` are correct by construction.

Most of the code for Isabelle was written by the Isabelle teams at the University of Cambridge and the Technische Universität München. A notable exception is the `metis` proof method, which was taken from the HOL4 theorem prover (also implemented in ML).

Isabelle/HOL is available for all major platforms under a BSD-style license, from `http://www.cl.cam.ac.uk/research/hvg/Isabelle`.

**Expected Competition Performance**

Results from last year would suggest that Isabelle will finish third in the THF category, after TPS and LEO-II. However, since last year, we have added the `smt` proof method, which we expect will increase our theorem count by about 50.

## 7.13 leanCoP(-SInE) 2.2

Jens Otten
University of Potsdam, Germany

**Architecture**

leanCoP [66, 64] is an automated theorem prover for classical first-order logic with equality. It is a very compact implementation of the connection (tableau) calculus [23, 53].

For the LTB division, leanCoP-SInE runs leanCoP as the underlying inference engine of SInE 0.4, i.e., axiom selection is done by SInE, and proof attempts are done by leanCoP. SInE is developed by Kryštof Hoder; leanCoP-SInE is co-developed by Thomas Raths.

**Strategies**
The reduction rule of the connection calculus is applied before the extension rule. Open branches are selected in a depth-first way. Iterative deepening on the proof depth is used to achieve completeness. Additional inference rules and strategies are regularity, lemmata, and restricted backtracking [65]. leanCoP uses an optimized structure-preserving transformation into clausal form [65] and a fixed strategy scheduling.

**Implementation**
leanCoP is implemented in Prolog (ECLiPSe, SICStus, and SWI Prolog are currently supported). The source code of the core prover is only a few lines long and fits on half a page. Prolog's built-in indexing mechanism is used to quickly find connections.

leanCoP can read formulae using the leanCoP syntax as well as the (raw) TPTP syntax format. Equality axioms are automatically added if required. The core leanCoP prover returns a very compact connection proof, which is translated into a readable proof. Several output formats are available. As the main enhancement, leanCoP 2.2 now supports the output of proofs in the (proposed) TPTP syntax for representing derivations in connection (tableau) calculi [67].

The source code of leanCoP 2.2, which is available under the GNU general public license, together with more information can be found on the leanCoP web site `http://www.leancop.de`.

**Expected Competition Performance**
As the core prover has not changed, we expect the performance of leanCoP 2.2 and leanCoP-SInE 2.2 to be similar to the performance of leanCoP 2.1 and leanCoP-SInE 2.1, respectively.

## 7.14 leanCoP-Ω 0.1

Jens Otten, Holger Trölenberg, Thomas Raths
University of Potsdam, Germany

**Architecture**
leanCoP-Ω is an automated theorem prover for classical first-order logic with equality and interpreted functions and predicates for integer arithmetic. It combines version 2.1 of the compact connection prover leanCoP [64, 65] with version 1.2 of the Omega test system [79].

**Strategies**
The Omega test determines whether there is an integer solution to an arbitrary set of linear equalities and inequalities. It is based on an extension of Fourier-Motzkin variable elimination (a linear programming method) to integer programming, and has worst-case exponential time complexity [79]. Omega test is invoked from the leanCoP prover if equalities or inequalities are contained in the active path of a connection derivation.

**Implementation**
The leanCoP prover is implemented in Prolog; see the leanCoP 2.2 description for more details. The Omega test system is implemented in C++. If leanCoP-Ω finds a proof it returns a very compact connection proof.

### 7.15 LEO-II 1.2

Christoph Benzmüller[1], Frank Theiss[2]
[1]Articulate Software, USA, [2]Saarland University, Germany

**Architecture**
LEO-II [20], the successor of LEO [19], is a higher-order ATP system based on extensional higher-order resolution. More precisely, LEO-II employs a refinement of extensional higher-order RUE resolution [17]. LEO-II is designed to cooperate with specialist systems for fragments of higher-order logic. By default, LEO-II cooperates with the first-order ATP system E [82]. LEO-II is often too weak to find a refutation amongst the steadily growing set of clauses on its own. However, some of the clauses in LEO-II's search space attain a special status: they are first-order clauses modulo the application of an appropriate transformation function. The default transformation is Hurd's fully typed translation [46]. Therefore, LEO-II launches a cooperating first-order ATP system every n iterations of its (standard) resolution proof search loop (e.g., n = 10). If the first-order ATP system finds a refutation, it communicates its success to LEO-II in the standard SZS format. Communication between LEO-II and the cooperating first-order ATP system uses the TPTP language and standards.

**Strategies**
LEO-II employs an adapted "Otter loop". In contrast to its competitor systems (such as Satallax, TPS, and IsabelleP) LEO-II so far only employs a monolithic search strategy, that is, it does not use strategy scheduling to try different search strategies or flag settings. However, LEO-II version 1.2 for the first time includes some very naive relevance filtering and selectively applies some simple scheduling for different relevance filters.

**Implementation**
LEO-II is implemented in Objective Caml version 3.10, and its problem representation language is the new TPTP THF language [21]. In fact, the development of LEO-II has largely paralleled the development of the TPTP THF language and related infrastructure [106].

The improved performance of LEO-II in comparison to its predecessor LEO (implemented in LISP) is due to several novel features including the exploitation of term sharing and term indexing techniques [18], support for primitive equality reasoning (extensional higher-order RUE resolution), and improved heuristics at the calculus level. One recent development is LEO-II's new parser: in addition to the TPTP THF language, this parser now also supports the TPTP FOF and CNF languages. Hence, LEO-II can now also be used for FOF and CNF problems. Unfortunately the LEO-II system still uses only a very simple sequential collaboration model with first-order ATPs instead of using the more advanced, concurrent and resource-adaptive OANTS architecture [22] as exploited by its predecessor LEO.

The LEO-II system is distributed under a BSD style license, and it is available from `http://leoprover.org`.

**Expected Competition Performance**
LEO-II (resp. its main developer C. Benzmüller) is currently on parental leave. As a result LEO-II has not much improved over the last year (except for its parser), while competitors such as IsabelleP and the new Satallax prover seem to have significantly gained on strength.

This year LEO-II will also participate in the FOF and CNF categories in order to evaluate its performance for these fragments. For this, note that LEO-II still employs its own input processing and normalization techniques, and that calls to prover E are applied only modulo Hurd's fully typed translation. The evaluation will thus reveal the performance loss in comparison to E and it will likely point to relevant future work. Another reason for entering THF, FOF, and CNF is to demonstrate that integrated higher-order-first-order theorem provers are generally feasible.

### 7.16 Metis 2.2

Joe Hurd
Galois, Inc., USA

**Architecture**
Metis 2.2 [46] 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 [46] have shown a given clause algorithm and ordered resolution best suit this application, and that is what Metis 2.2 implements. Since equality often appears in interactive theorem prover goals, Metis 2.2 also implements the ordered paramodulation calculus.

**Strategies**
Metis 2.2 uses a fixed strategy for every input problem. Negative literals are always chosen over positive literals, and terms are ordered using the Knuth-Bendix ordering with uniform symbol weight and precedence favouring reduced arity.

**Implementation**
Metis 2.2 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 age and size measures, Metis 2.2 uses finite models to weight clauses in the *Passive* set. When integrated with higher order logic, an interpretation of known functions and relations is manually constructed to make many of their standard properties valid in the finite model. For example, if the domain of the model is the set *0,...,7*, and the higher order logic arithmetic functions are interpreted in the model modulo *8*. Unknown functions and relations are interpreted randomly, but with a bias towards making supporting theorems valid in the model. The finite model strategy carries over to TPTP problems, by manually interpreting a collection of functions and relations that appear in TPTP axiom files in such a way as to make the axioms valid in the model.

Metis 2.2 reads problems in TPTP format and outputs detailed proofs in TSTP format, where each refutation step is one of 6 simple inference rules. Metis 2.2 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 2.2 is free software, released under the GPL. It is available from `http://www.gilith.com/software/metis`.

**Expected Competition Performance**
There have been only minor changes to Metis 2.2 since CASC-22, so it is expected to perform at

approximately the same level in CASC-J5 and end up in the lower half of the table. However, it is possible that Metis 2.2 might get a boost from its finite model clause weighting, since this year the CASC competition rules have changed so that function and relation names are no longer obfuscated.

## 7.17 MetiTarski 1.3

Larry Paulson
University of Cambridge, United Kingdom

**Architecture**
MetiTarski [1, 30] is an automatic theorem prover based on a combination of resolution and QEPCAD-B [30], a decision procedure for the theory of real closed fields. It is designed to prove theorems involving real-valued special functions such as log, exp, sin, cos and sqrt. In particular, it is designed to prove universally quantified inequalities involving such functions. This problem is undecidable: MetiTarski is incomplete. MetiTarski is a modified version of Joe Hurd's theorem prover, Metis [46].

**Strategies**
MetiTarski employs resolution, augmented with axiom files that specify upper and lower bounds of the special functions mentioned in the problem. MetiTarski also has code to simplify polynomials and put them into canonical form. The resolution calculus is extended with a literal deletion rule: if the decision procedure finds a literal to be inconsistent with its context (which consists of known facts and the negation of the other literals in the clause), then it is deleted.

**Implementation**
MetiTarski, like Metis, is implemented in Standard ML. QEPCAD is implemented in C and C++ (and unfortunately is very difficult to build on 64 bit machines). The latest version of MetiTarski can be downloaded from `http://www.cl.cam.ac.uk/~lp15/papers/Arith/`.

## 7.18 Muscadet 4.0

Dominique Pastre
University Paris Descartes, France

**Architecture**
The Muscadet theorem prover is a knowledge-based system. It is based on Natural Deduction, following the terminology of [26] and [68], and uses methods that resemble 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 derived 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 that are local for a (sub-)theorem.

**Strategies**
There are specific strategies for existential, universal, conjunctive and disjunctive hypotheses and conclusions, and equalities. 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), refutation (only if the conclusion is a negation), search for objects satisfying the conclusion, or dynamic building of new rules.

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 [69] and [70]. These two possibilities are not used while working with the TPTP Library.

**Implementation**

Muscadet [71] is implemented in SWI-Prolog. Rules are written as more or less declarative Prolog clauses. Metarules are written as sets of Prolog clauses. The inference engine includes the Prolog interpreter and some procedural Prolog clauses. A theorem may be split into several subtheorems, structured as a tree with "and" and "or" nodes. All the proof search steps are memorized as facts including all the elements which will be necessary to later extract the useful steps (the name of the executed action or applied rule, the new facts added or rule dynamically built, the antecedents, and a brief explanation).

Muscadet 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 [72, 73]. It will have poor performances for problems using few concepts but large and deep formulae leading to many splittings. Its best results will be in set theory, especially for functions and relations. It's originality is that proofs are given in natural style.

## 7.19 omkbTT 1.0

Sarah Winkler
University of Innsbruck, Austria

**Architecture**

Based on ordered completion [8], omkbTT 1.0 constitutes a theorem prover for unit equality problems. It stands out from classical completion tools in two respects: (1) automatic termination tools replace a fixed reduction order, and (2) omkbTT employs a multi-completion approach [52] to explore multiple branches of the search tree at once. A detailed description of the underlying inference system can be found in [124].

**Strategies**

omkbTT keeps track of multiple ordered completion *processes* using different reduction orders developed during the deduction. The approach of multi-completion allows sharing of many inference steps among these processes. A deduction succeeds as soon as one of the processes does.

omkbTT is parameterized by a termination strategy for the termination checks performed internally, and a selection strategy to choose the equation to be processed next. To ensure that a saturated set of equations and rules is ground-complete, the default termination strategy comprises a number of termination techniques inducing total termination such as Knuth-Bendix orders, lexicographic path orders, polynomial interpretations, and multiset path orders. The default selection strategy first chooses a process for which the size of its current equation and constraint set is minimal. Then a node for this process is selected by considering the term size and timestamp, the latter to ensure fairness of the deduction.

Since in the setting with termination tools the reduction order developed along the way is not known in advance, the set of ordered critical pairs is not computable during the deduction. Instead, critical consequences are over-approximated using the embedding relation [124]. Furthermore, omkbTT uses multi-completion variants of critical pair criteria [7] to restrict the number of equational consequences, and isomorphic processes are filtered out to restrict the search space. Details about these optimizations can be found in [125].

**Implementation**

Due to the multi-completion approach, equations and rewrite rules as well as goals are represented using *nodes*. This data structure connects a pair of terms with the sets of processes which consider it as an equation or rule, respectively. The inference rules of omkbTT working on sets on nodes are combined in a DISCOUNT-like control loop. In order to accelerate the retrieval of adequate terms for rewriting and the computation of overlaps, omkbTT relies on code trees [120] as its default term indexing technique.

omkbTT is written in OCaml. It interfaces the termination tool TTT2 [51] internally to perform termination checks. Version 1.0 is available from `http://cl-informatik.uibk.ac.at/software/omkbtt`.

**Expected Competition Performance**

As a prototype of the described approach, the system is to demonstrate the use of termination tools in a multi-completion setting. omkbTT is expected to solve most easy problems and a moderate number of the more difficult instances (but is not assumed to win its division).

## 7.20 Otter 3.3

William McCune
Argonne National Laboratory, USA

**Architecture**

Otter 3.3 [58] 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.cs.unm.edu/~mccune/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 [59], nothing that really affects its performance 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.*

## 7.21 Paradox 3.0

Koen Claessen, Niklas Sörensson
Chalmers University of Technology, Sweden

**Architecture**
Paradox [34] is a finite-domain model generator. It is based on a MACE-style [57] 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.*

**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 3.0 is the CASC-22 FNT division winner.

## 7.22 Paradox 4.0

Koen Claessen
Chalmers University of Technology, Sweden

No system description supplied.

### 7.23 Satallax 1.4

Chad E. Brown
Saarland University, Germany

**Architecture**
Satallax is an automated theorem prover for higher-order logic. The particular form of higher-order logic supported by Satallax is Church's simple type theory with extensionality and choice operators. The SAT solver MiniSat [39] is responsible for much of the search for a proof.

The theoretical basis of search is a complete ground tableau calculus for higher-order logic [33] with a choice operator [10]. A problem is given using an S-expression variant of the THF format. A branch is formed from the axioms of the problem and negation of the conjecture (if any is given). From this point on, Satallax tries to determine unsatisfiability or satisfiability of this branch.

Satallax progressively generates higher-order formulae and corresponding propositional clauses. These formulae and propositional clauses correspond to instances of the tableau rules. Satallax uses the SAT solver MiniSat as an engine to test the current set of propositional clauses for unsatisfiability. If the clauses are unsatisfiable, then the original branch is unsatisfiable. If there are no quantifiers at function types, the generation of higher-order formulae and corresponding clauses may terminate [32, 32]. In such a case, if MiniSat reports the final set of clauses as satisfiable, then the original set of higher-order formulae is satisfiable (by a standard model in which all types are interpreted as finite sets).

**Strategies**
There are a number of flags that control the order in which instances of tableau rules are considered. Other flags activate some optional extensions to the basic proof procedure. Three such extensions have proven particularly successful.

1. A preprocessing stage may split the original problem into several problems to be solved independently. This is helpful, for example, when the claim to be proven is an equivalence.
2. The set of all (ground) subterms of the problem can be preemptively included in the set of all allowed instantiations for quantifiers. This is helpful when strictly following the tableau calculus rules would require taking a roundabout path to licensing a subterm as a legal instantiation.
3. Some universally quantified formulae can be treated as higher-order clauses with strict occurrences of higher-order variables. Whenever new (ground) formulae are considered, pattern matching can be used to create ground instances of higher-order clauses. These ground clauses are included in the clause set sent to MiniSat.

A collection of flag settings is called a mode. Approximately 250 modes have been tried so far. Regardless of the mode, the search procedure is sound and complete for higher-order logic with choice. This implies that if search terminates with a particular mode, then we can conclude that the original set of formulae is unsatisfiable or satisfiable.

A strategy is an ordered collection of modes with information about how much time the mode should be allotted. Satallax tries each of the modes for a certain amount of time sequentially. Satallax 1.4 uses a default strategy consisting of 22 modes determined through experimentation using the THF problems in the TPTP library.

**Implementation**
Satallax is implemented in Steel Bank Common Lisp. Satallax calls MiniSat which is implemented in C++. Satallax is available from `http://satallax.com`.

**Expected Competition Performance**
Since Satallax supports theorem proving with a choice operator, there should be a few problems only Satallax can solve. In addition, the use of MiniSat to do the propositional part of the search seems to be very effective. Preliminary tests show Satallax to be competitive. On the other hand, the primary instantiation technique for higher-order quantifiers is essentially a generate-and-test method. This means most problems that require nontrivial higher-order instantiations will be out of reach for the time being.

## 7.24 SNARK—20080805r027

Mark Stickel
SRI International, USA

**Architecture**
SNARK 20080805r027 is a resolution and paramodulation theorem prover for first-order predicate calculus. Procedural attachment is used to evaluate arithmetic expressions. The objective to date is to add some arithmetic calculation in support of SNARK's applications, not to duplicate or incorporate symbolic algebra systems to extend SNARK's mathematical range.

SNARK has a rudimentary type system that includes subtypes but not polymorphism. Thus, the mapping onto TPTP arithmetic types, which lack subtypes, and TPTP arithmetic operations, which are polymorphic, is approximate. It uses Common Lisp's unlimited precision rational arithmetic. Integers are a subtype. Inexact floating-point arithmetic is not used; real number inputs are converted to rationals. Real numbers are a supertype so that irrational numbers can be represented symbolically.

Procedural attachment is used to evaluate arithmetic expressions that are fully instantiated with numeric arguments. For example, `($$SUM 2 2)` and `($$LESS 2 3)` can be rewritten to 4 and TRUE. The equality relation is also procedurally attached when an argument is numeric so that, for example, `(= ($$SUM ?X 2) 4)` can be solved.

**Implementation**
SNARK is written in Common Lisp that is also used as its extension and scripting language. SNARK is available from `http://www.ai.sri.com/~stickel/snark.html`.

## 7.25 SPASS+T 2.2.12

Uwe Waldmann[1], Stephan Zimmer[2]
[1]Max-Planck-Institut für Informatik, Germany, [2]AbsInt GmbH, Germany

**Architecture**
SPASS+T is an extension of the superposition-based theorem prover SPASS that integrates algebraic knowledge into SPASS in three complementary ways: by passing derived formulae to an external SMT procedure (currently Yices or CVC3), by adding standard axioms, and by built-in arithmetic simplification and inference rules. A first version of the system has been described

in [78]. In the current version, a much more sophisticated coupling of the SMT procedure has been added [126].

**Strategies**
Standard axioms and built-in arithmetic simplification and inference rules are integrated into the standard main loop of SPASS. The external SMT procedure runs in parallel in a separate process, leading occasionally to non-deterministic behaviour.

**Implementation**
SPASS+T is implemented in C. The system is available from `http://www.mpi-inf.mpg.de/~uwe/software/#TSPASS`.

### 7.26 SPASS-XDB 3.01X0.5

Geoff Sutcliffe[1], Martin Suda[2]
[1]University of Miami, USA, [2]Charles University in Prague, Czech Republic

**Architecture**
SPASS-XDB [84, 108] is an extended version of the well-known, state-of-the-art, SPASS automated theorem proving system [122]. The original SPASS reads a problem, consisting of axioms and a conjecture, in TPTP format from a file, and searches for a proof by refutation for the conjecture. SPASS-XDB adds the capability of retrieving extra positive unit axioms (facts) from external sources during the proof search (hence the "XDB", standing for eXternal DataBases). The axioms are retrieved asynchronously, on-demand, based on an expectation that they will contribute to completing the proof. The axioms are retrieved from a range of external sources, including SQL databases, SPARQL endpoints, WWW services, computation sources (e.g., computer algebra systems), etc., using a TPTP standard protocol.

For the TFA division, the TFF formulae are converted to FOF using the standard approach [121], but without predicates to check the types of numeric variables. This means SPASS-XDB is unsound for certain types of conjectures that mix different types of numbers (e.g., integers with reals), but those problems will not arise in CASC. Until recently a simple computation system implemented in Prolog was used (and it's still available) to obtain ground arithmetic facts, as required to resolve against arithmetic relations (i.e., all arithmetic function evaluation had to be done inside an `$evaleq/2` relation). More recently this has been replaced by internal evaluation of ground arithmetic relations and functions. The external mechanism is now being used to develop more sophisticated arithmetic capabilities, e.g., solving for a variable in an equation involving arithmetic functions, e.g., solving for `X` in `$sum(2,X) = 5`. It is hoped this will be ready in time for CASC. SPASS-XDB supports integer, rational, and finite precision real arithmetic.

**Strategies**
Generally, SPASS-XDB follows SPASS' strategies. However, SPASS, like most (all?) ATP systems, was designed under the assumption that all formulae are in the problem file, i.e., it is ignorant that external axioms might be delivered. To regain completeness, constraints on SPASS' search had to be relaxed in SPASS-XDB. This increases the search space, i.e., so the constraints were relaxed in a controlled, incremental fashion. [108] The search space is also affected by the number of external axioms that can be delivered, and mechanisms to control the delivery and focus the consequent search have been implemented. [108]

**Implementation**
SPASS-XDB, as an extension of SPASS, is written in C. The internal arithmetic is done using the GMP multiple precision arithmetic library. Reals are converted to rationals for computation, but results are presented in real format. The external arithmetic computation system is written in SWI Prolog, and compiled to an executable. SPASS-XDB is available for use online in the SystemOnTPTP interface `http://www.tptp.org/cgi-bin/SystemOnTPTP`.

## 7.27 TPS 3.080227G1d

Peter B. Andrews[1], Chad E. Brown[2]
[1]Carnegie Mellon University, USA, [2]Saarland University, Germany

**Architecture**
TPS is a higher-order theorem proving system that has been developed over several decades under the supervision of Peter B. Andrews with substantial work by Eve Longini Cohen, Dale A. Miller, Frank Pfenning, Sunil Issar, Carl Klapper, Dan Nesmith, Hongwei Xi, Matthew Bishop, Chad E. Brown, and Mark Kaminski. TPS can be used to prove theorems of Church's type theory automatically, interactively, or semi-automatically [4, 5].

When searching for a proof, TPS first searches for an expansion proof [61] or an extensional expansion proof [31] of the theorem. Part of this process involves searching for acceptable matings [2]. Using higher-order unification, a pair of occurrences of subformulae (which are usually literals) is mated appropriately on each vertical path through an expanded form of the theorem to be proved. The expansion proof thus obtained is then translated [77] without further search into a proof of the theorem in natural deduction style. The translation process provides an effective soundness check, but it is not actually used during the competition.

**Strategies**
Strategies used by TPS in the search process include:

- Re-ordering conjunctions and disjunctions to alter the way paths through the formula are enumerated.
- The use of primitive substitutions and gensubs [3].
- Path-focused duplication [47].
- Dual instantiation of definitions, and generating substitutions for higher-order variables which contain abbreviations already present in the theorem to be proved [25].
- Component search [24].
- Generating and solving set constraints [29].
- Generating connections using extensional and equational reasoning [31].

**Implementation**
TPS has been developed as a research tool for developing, investigating, and refining a variety of methods of searching for expansion proofs, and variations of these methods. Its behavior is controlled by hundreds of flags. A set of flags, with values for them, is called a mode. 52 modes have been found which collectively suffice for automatically proving virtually all the theorems which TPS has proved automatically thus far. When searching for a proof in automatic mode, TPS tries each of these modes in turn for a specified amount of time.

TPS is implemented in Common Lisp, and is available from `http://gtps.math.cmu.edu/tps.html`

**Expected Competition Performance**
TPS 3.080227G1d is the CASC-22 THF division winner.

### 7.28 Vampire 10.0

Andrei Voronkov
University of Manchester, United Kingdom

No system description supplied.

**Expected Competition Performance**
Vampire 10.0 is the CASC-22 CNF division winner.

### 7.29 Vampire 11.0 and Vampire-LTB 11.0

Andrei Voronkov, Kryštof Hoder
University of Manchester, United Kingdom

**Architecture**
Vampire 11.0 is an automatic theorem prover for first-order classical logic. It consists of a shell and a kernel. The kernel implements the calculi of ordered binary resolution and superposition for handling equality. The splitting rule in kernel adds propositional parts to clauses, which are manipulated using binary decision diagrams (BDDs). A number of standard redundancy criteria and simplification techniques are used for pruning the search space: subsumption, tautology deletion, subsumption resolution and rewriting by ordered unit equalities. The reduction ordering is the Knuth-Bendix Ordering.

Substitution tree indexing is used to implement all major operations on sets of terms and literals. Although the kernel of the system works only with clausal normal forms, the shell accepts a problem in the full first-order logic syntax, clausifies it and performs a number of useful transformations before passing the result to the kernel. Also the axiom selection algorithm of SInE can be enabled as part of the preprocessing.

When a theorem is proved, the system produces a verifiable proof, which validates both the clausification phase and the refutation of the CNF.

**Strategies**
The Vampire 11.0 kernel provides a fairly large number of features for strategy selection. The most important ones are:

- Choice of the main saturation procedure :
  - Limited Resource Strategy
  - DISCOUNT loop
  - Otter loop
- A variety of optional simplifications.
- Parameterised reduction orderings.
- A number of built-in literal selection functions and different modes of comparing literals.
- Age-weight ratio that specifies how strongly lighter clauses are preferred for inference selection.
- Set-of-support strategy.

The Vampire 11.0 core is a completely new Vampire, and it shares virtually no code with the previous versions. The automatic mode of Vampire 11.0, however, uses both the Vampire 11.0 core and Vampire 10.0 to solve problems, based on input problem classification that takes into account simple syntactic properties, such as being Horn or non-Horn, presence of equality, etc.

**Implementation**
Vampire 11.0 is implemented in C++. The supported compilers are gcc 4.1.x, gcc 4.3.x.

**Expected Competition Performance**
Vampire 11.0 is the CASC-22 FOF division winner. Vampire-LTB 11.0 is the CASC-22 LTB division winner.

### 7.30 Vampire 0.6

Andrei Voronkov, Kryštof Hoder
University of Manchester, United Kingdom

No system description supplied.

### 7.31 Waldmeister C09a

Thomas Hillenbrand
Max-Planck-Institut für Informatik, Germany

**Architecture**
Waldmeister C09a [44] is a system for unit equational deduction. Its theoretical basis is unfailing completion in the sense of [8] with refinements towards ordered completion (cf. [6]). The system saturates the input axiomatization, distinguishing active facts, which induce a rewrite relation, and passive facts, which are the one-step conclusions of the active ones up to redundancy. The saturation process is parameterized by a reduction ordering and a heuristic assessment of passive facts [45].

Waldmeister C09a is a minor upgrade of Waldmeister 806, the best feature of which is TPTP format output.

**Strategies**
The approach taken to control the proof search is to choose the search parameters according to the algebraic structure given in the problem specification [45]. This is based on the observation that proof tasks sharing major parts of their axiomatization often behave similar. Hence, for a number of domains, the influence of different reduction orderings and heuristic assessments has been analyzed experimentally; and in most cases it has been possible to distinguish a strategy uniformly superior on the whole domain. In essence, every such strategy consists of an instantiation of the first parameter to a Knuth-Bendix ordering or to a lexicographic path ordering, and an instantiation of the second parameter to one of the weighting functions *addweight*, *gtweight*, or *mixweight*, which, if called on an equation $s = t$, return $|s| + |t|$, $|max_>(s,t)|$, or $|max_>(s,t)| \cdot (|s| + |t| + 1) + |s| + |t|$, respectively, where $|s|$ denotes the number of symbols in $s$.

**Implementation**
The prover is coded in ANSI-C. It runs on Solaris, Linux, and newly also on MacOS X. In addition, it is now available for Windows users via the Cygwin platform. The central data structures are: perfect discrimination trees for the active facts; group-wise compressions for the passive ones; and sets of rewrite successors for the conjectures. Visit the Waldmeister web pages at `http://www.waldmeister.org`.

**Expected Competition Performance**
Waldmeister C09a is the CASC-22 UEQ division winner.

## 7.32 Waldmeister 710

Thomas Hillenbrand
Max-Planck-Institut für Informatik, Germany

**Architecture**
Waldmeister 710 [44] is a system for unit equational deduction. Its theoretical basis is unfailing completion in the sense of [8] with refinements towards ordered completion (cf. [6]). The system saturates the input axiomatization, distinguishing active facts, which induce a rewrite relation, and passive facts, which are the one-step conclusions of the active ones up to redundancy. The saturation process is parameterized by a reduction ordering and a heuristic assessment of passive facts [45]. This year's version is the result of polishing and fixing a few things in last year's.

**Implementation**
The approach taken to control the proof search is to choose the search parameters – reduction ordering and heuristic assessment – according to the algebraic structure given in the problem specification [45]. This is based on the observation that proof tasks sharing major parts of their axiomatization often behave similarly.

**Strategies**
The prover is coded in ANSI-C. It runs on Solaris, Linux, MacOS X, and Windows/Cygwin. The central data structures are: perfect discrimination trees for the active facts; group-wise compressions for the passive ones; and sets of rewrite successors for the conjectures. Visit the Waldmeister web pages at `http://www.waldmeister.org`.

**Expected Competition Performance**
The system should again be rather strong, and close to the previous version.

## 7.33 Zenon 0.6.3

Damien Doligez
INRIA, France

**Architecture**
Zenon 0.6.3 [28] is a theorem prover based on a proof-confluent version of analytic tableaux. It uses all the usual tableau rules for first-order logic with a rule-based handling of equality.

Zenon outputs formal proofs that can be checked by Coq or Isabelle.

**Strategies**
Zenon is a fully automatic black-box design with no user-selectable strategies.

**Implementation**
Zenon is implemented in OCaml. Its most interesting data structure is the representation of first-order formulae and terms: they are hash-consed modulo alpha conversion.

Zenon is available from `http://zenon-prover.org`.

**Expected Competition Performance**
This version is only slightly better than the version of 2008, and the handling of equality is still really bad, so Zenon is again expected to rank in the lower part of the field.

## 8 Conclusion

The CADE-J5 ATP System Competition was the fifteenth large scale competition for classical logic ATP systems. The organizer believes 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 provides an overview of the implementation state of running, fully automatic, classical logic, ATP systems.

## References

[1] B. Akbarpour and L. Paulson. MetiTarski: An Automatic Theorem Prover for Real-Valued Special Functions. *Journal of Automated Reasoning*, 44(3):175–205, 2010.

[2] P. B. Andrews. Theorem Proving via General Matings. *Journal of the ACM*, 28(2):193–214, 1981.

[3] P. B. Andrews. On Connections and Higher-Order Logic. *Journal of Automated Reasoning*, 5(3):257–291, 1989.

[4] P. B. Andrews, M. Bishop, S. Issar, Nesmith. D., F. Pfenning, and H. Xi. TPS: A Theorem-Proving System for Classical Type Theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

[5] P. B. Andrews and C.E. Brown. TPS: A Hybrid Automatic-Interactive System for Developing Proofs. *Journal of Applied Logic*, 4(4):367–395, 2006.

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

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

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

[9] 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.

[10] J. Backes and C. Brown. Analytic Tableaux for Higher-Order Logic with Choice. In J. Giesl and R. Haehnle, editors, *Proceedings of the 5th International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, page To appear, 2010.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] P. Baumgartner and C. Tinelli. The Model Evolution Calculus with Equality. In R. Nieuwenhuis, editor, *Proceedings of the 20th International Conference on Automated Deduction*, number 3632 in Lecture Notes in Artificial Intelligence, pages 392–408. Springer-Verlag, 2005.

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

[18] C. Benzmüller, B. Fischer, and G. Sutcliffe, editors. *Term Indexing for the LEO-II Prover*, number 212 in CEUR Workshop Proceedings, 2006.

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

[20] C. Benzmüller, L. Paulson, F. Theiss, and A. Fietzke. LEO-II - A Cooperative Automatic Theorem Prover for Higher-Order Logic. In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 162–170. Springer-Verlag, 2008.

[21] C. Benzmüller, F. Rabe, and G. Sutcliffe. THF0 - The Core TPTP Language for Classical Higher-Order Logic. In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 491–506. Springer-Verlag, 2008.

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

[23] W. Bibel. *Automated Theorem Proving.* Vieweg and Sohn, 1987.

[24] M. Bishop. A Breadth-First Strategy for Mating Search. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 359–373. Springer-Verlag, 1999.

[25] M. Bishop and P.B. Andrews. Selectively Instantiating Definitions. In C. Kirchner and H. Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction*, number 1421 in Lecture Notes in Artificial Intelligence, pages 365–380. Springer-Verlag, 1998.

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

[27] S. Böhme. Proof Reconstruction for Z3 in Isabelle/HOL. In B. Duterte and O. Strichman, editors, *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories*, 2009.

[28] 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.

[29] C.E. Brown. Solving for Set Variables in Higher-Order Theorem Proving. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, number 2392 in Lecture Notes in Artificial Intelligence, pages 408–422. Springer-Verlag, 2002.

[30] C.E. Brown. QEPCAD B - A Program for Computing with Semi-algebraic sets using CADs. *ACM SIGSAM Bulletin*, 37(4):97–108, 2003.

[31] C.E. Brown. *Automated Reasoning in Higher-Order Logic: Set Comprehension and Extensionality in Church's Type Theory*. Number 10 in Studies in Logic: Logic and Cognitive Systems. College Publications, 2007.

[32] C.E. Brown and G. Smolka. Terminating Tableaux for the Basic Fragment of Simple Type Theory. In M. Giese and A. Waaler, editors, *Proceedings of the 18th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, number 5697 in Lecture Notes in Artificial Intelligence, pages 138–151. Springer-Verlag, 2009.

[33] C.E. Brown and G. Smolka. Analytic Tableaux for Simple Type Theory and its First-Order Fragment. *Logical Methods in Computer Science*, page To appear, 2010.

[34] K. Claessen and N. Sörensson. 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.

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

[36] 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.

[37] H. de Nivelle. Classical Logic with Partial Functions. In J. Giesl and R. Haehnle, editors, *Proceedings of the 5th International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, page To appear, 2010.

[38] 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.

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

[40] U. Furbach, I. Glöckner, and B. Pelzer. An Application of Automated Reasoning in Natural Language Question Answering. *AI Communications*, 23(2-3):241–265, 2010.

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

[42] 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. Springer-Verlag, 2004.

[43] 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.

[44] T. Hillenbrand. Citius altius fortius: Lessons Learned from the Theorem Prover Waldmeister. In I. Dahn and L. Vigneron, editors, *Proceedings of the 4th International Workshop on First-Order Theorem Proving*, number 86.1 in Electronic Notes in Theoretical Computer Science, pages 1–13, 2003.

[45] T. Hillenbrand, A. Jaeger, and B. Löchner. Waldmeister - Improvements in Performance and Ease

of Use. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 232–236. Springer-Verlag, 1999.

[46] 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.

[47] S. Issar. Path-Focused Duplication: A Search Procedure for General Matings. In Swartout W. Dietterich T., editor, *Proceedings of the 8th National Conference on Artificial Intelligence* , pages 221–226. American Association for Artificial Intelligence / MIT Press, 1990.

[48] 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*, number 5195 in Lecture Notes in Artificial Intelligence, pages 292–298, 2008.

[49] 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*, number 5663 in Lecture Notes in Computer Science, pages 163–166. Springer-Verlag, 2009.

[50] K. Korovin and C. Sticksel. iProver-Eq - An Instantiation-Based Theorem Prover with Equality. In J. Giesl and R. Haehnle, editors, *Proceedings of the 5th International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, page To appear, 2010.

[51] M. Korp, C. Sternagel, H. Zankl, and A. Middledorp. Tyrolean Termination Tool 2. In R. Treinen, editor, *Proceedings of the 20th International Conference on Rewriting Techniques and Applications*, number 5595 in Lecture Notes in Computer Science, pages 295–304, 2009.

[52] M. Kurihara and H. Kondo. Completion for Multiple Reduction Orderings. *Journal of Automated Reasoning*, 23(1):25–42, 1999.

[53] 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.

[54] 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.

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

[56] D.W. Loveland. *Automated Theorem Proving : A Logical Basis.* Elsevier Science, 1978.

[57] W.W. McCune. Mace4 Reference Manual and Guide. Technical Report ANL/MCS-TM-264, Argonne National Laboratory, Argonne, USA, 2003.

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

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

[60] J. Meng and L. Paulson. Lightweight Relevance Filtering for Machine-generated Resolution Problems. *Journal of Applied Logic*, 7(1):41–57, 2009.

[61] D. Miller. A Compact Representation of Proofs. *Studia Logica*, 46(4):347–370, 1987.

[62] T. Nipkow. Equational Reasoning in Isabelle. *Science of Computer Programming*, 12(2):123–149, 1989.

[63] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic.* Number 2283 in Lecture Notes in Computer Science. Springer-Verlag, 2002.

[64] 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*, number 5195 in Lecture Notes in

Artificial Intelligence, pages 283–291, 2008.

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

[66] J. Otten and W. Bibel. leanCoP: Lean Connection-Based Theorem Proving. *Journal of Symbolic Computation*, 36(1-2):139–161, 2003.

[67] J. Otten and G. Sutcliffe. Using the TPTP Language for Representing Derivations in Tableau and Connection Calculi. In B. Konev, R. Schmidt, and S. Schulz, editors, *Proceedings of the Workshop on Practical Aspects of Automated Reasoning, 5th International Joint Conference on Automated Reasoning*, page To appear, 2010.

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

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

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

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

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

[73] 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.

[74] L. Paulson. A Generic Tableau Prover and its Integration with Isabelle. *Artificial Intelligence*, 5(3):73–87, 1999.

[75] L.C. Paulson and T. Nipkow. *Isabelle: A Generic Theorem Prover*. Number 828 in Lecture Notes in Computer Science. Springer-Verlag, 1994.

[76] 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.

[77] F. Pfenning. *Proof Transformations in Higher-Order Logic*. PhD thesis, Carnegie-Mellon University, Pittsburg, USA, 1987.

[78] V. Prevosto and U. Waldmann. SPASS+T. In G. Sutcliffe, R. Schmidt, and S. Schulz, editors, *Proceedings of the FLoC'06 Workshop on Empirically Successful Computerized Reasoning, 3rd International Joint Conference on Automated Reasoning*, number 192 in CEUR Workshop Proceedings, pages 19–33, 2006.

[79] W. Pugh. The Omega Test: A Fast and Practical Integer Programming Algorithm for Dependence Analysis. *Communications of the ACM*, 31(8):4–13, 1992.

[80] A. Riazanov and A. Voronkov. Splitting without Backtracking. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence* , pages 611–617. Morgan Kaufmann, 2001.

[81] A. Roederer, Y. Puzis, and G. Sutcliffe. Divvy: A ATP Meta-system based on Axiom Relevance Ordering. In R. Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction*, number 5663 in Lecture Notes in Artificial Intelligence, pages 157–162. Springer-Verlag, 2009.

[82] 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.

[83] 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.

[84] M. Suda, G. Sutcliffe, P. Wischnewski, M. Lamotte-Schubert, and G. de Melo. External Sources of

Axioms in Automated Theorem Proving. In B. Mertsching, editor, *Proceedings of the 32nd Annual Conference on Artificial Intelligence*, number 5803 in Lecture Notes in Artificial Intelligence, pages 281–288, 2009.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[104] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

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

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

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

[108] G. Sutcliffe, M. Suda, A. Teyssandier, N. Dellis, and G. de Melo. Progress Towards Effective Automated Reasoning with World Knowledge. In C. Murray and H. Guesgen, editors, *Proceedings of the 23rd International FLAIRS Conference*, pages 110–115. AAAI Press, 2010.

[109] 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.

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

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

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

[113] G. Sutcliffe and C.B. Suttner, editors. *Special Issue: The CADE-13 ATP System Competition*, volume 18, 1997.

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

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

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

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

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

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

[120] A' Voronkov. Algorithms, Datastructures, and Other Issues in Efficient Automated Deduction. 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 13–28. Springer-Verlag, 2001.

[121] C. Walther. A Many-Sorted Calculus Based on Resolution and Paramodulation. In Bundy A., editor, *Proceedings of the 8th International Joint Conference on Artificial Intelligence* , pages 882–891, 1983.

[122] C. Weidenbach, A. Fietzke, R. Kumar, M. Suda, P. Wischnewski, and D. Dimova. SPASS Version 3.5. In R. Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction*, number 5663 in Lecture Notes in Artificial Intelligence, pages 140–145. Springer-Verlag, 2009.

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

[124] S. Winkler and A. Middledorp. Termination Tools in Ordered Completion. In J. Giesl and R. Haehnle, editors, *Proceedings of the 5th International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, page To appear, 2010.

[125] S. Winkler, H. Sato, A. Middledorp, and M. Kurihara. Optimizing mkbTT. In C. Lynch, editor, *Proceedings of the 21st International Conference on Rewriting Techniques and Applications*, Leibniz International Proceedings in Informatics, page To appear, 2010.

[126] S. Zimmer. Intelligent Combination of a First Order Theorem Prover and SMT Procedures. Master's thesis, Saarland University, Saarbruecken, Germany, 2007.