CASC-22

CASC-22

CASC-22

CASC-22

The CADE-22 ATP System Competition (CASC-22)

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 a specified time limit for each solution attempt. The CADE-22 ATP System Competition (CASC-22) was held on 5th August 2009. 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-22 was held on 5th August 2009, as part of the 22nd International Conference on Automated Deduction (CADE-22), in Montreal, Canada. It is the fourteenth competition in the CASC series [SS97a, SS98c, SS99, Sut00b, Sut01b, SSP02, SS03, SS04, Sut05b, Sut06b, Sut07b, Sut08b, Sut09a].

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 [Sut09b], and
- specified time limits on solution attempts.

Twenty-six ATP systems and variants, listed in Table 1, entered into the various competition and demonstration divisions. The winners of the CASC-J4 (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). *For CASC-22 the CASC-J4 SAT winner, MetaProver 1.0, could not be entered because some of its component systems would not compile on the CASC-22 computers. Also, the CASC-J4 LTB winner, SInE 0.3, required some slight modifications because the LTB division rules had changed, thus the modified version SInE 0.4 was entered.*

The design and procedures of this CASC evolved from those of previous CASCs [SS97c, SS97b, SS98a, SS98b, Sut99, Sut00a, Sut01a, Sut02, Sut03, Sut04, Sut05a, Sut06a, Sut07a, Sut08a]. Important changes for this CASC were:

- All problem files were in TPTP format, with `include` directives. Systems had to add equality axioms themselves if necessary.

Table 1: The ATP systems and entrants

ATP System	Divisions	Entrants	Affiliation
Darwin 1.4.4	EPR	Peter Baumgartner	NICTA
		(Alexander Fuchs, Cesare Tinelli)	(University of Iowa)
E 1.1	FOF FNT CNF SAT EPR UEQ LTB	Stephan Schulz	Technische Universität München
EP 1.1	FOF* LTB*		*E 1.1 variant*
E-Darwin 1.2	CNF EPR	Björn Pelzer	University Koblenz-Landau
E-KRHyper 1.1.3	FOF FNT CNF SAT EPR LTB	Björn Pelzer	University Koblenz-Landau
Equinox 1.0	FOF CNF UEQ	Koen Claessen	Chalmers University of Technology
Infinox 1.0	FNT demo SAT demo	Ann Lillieström (Koen Claessen)	Chalmers University of Technology
iProver 0.5	EPR	CASC	*CASC-J4 EPR winner*
iProver 0.7	FOF FNT CNF SAT EPR	Konstantin Korovin	The University of Manchester
iProver-SInE 0.7	LTB		*iProver 0.7 variant*
iProver-Eq 0.5	FOF CNF UEQ	Konstantin Korovin (Christoph Sticksel)	The University of Manchester
Isabelle/HOL 2009	THF	Jasmin Blanchette	Technische Universität München
		(Larry Paulson, Tobias Nipkow,	(Paulson: University of Cambridge)
		Makarius Wenzel, Stefan Berghofer)	
leanCoP 2.1	FOF*	Jens Otten	University of Potsdam
leanCoP-SInE 2.1	LTB	Jens Otten (Thomas Raths)	*leanCoP 2.1 variant*
LEO-II 1.0	THF	Christoph Benzmüller (Frank Theiss)	International University in Germany
matita 0.5.7	UEQ	Wilmer Ricciotti	University of Bologna
Metis 2.2	FOF* FNT* CNF SAT EPR UEQ	Joe Hurd	Galois, Inc.
OSHL-S 0.6	FOF CNF EPR	Hao Xu (David Plaisted)	University of N. Carolina at Chapel Hill
Otter 3.3	FOF CNF UEQ	CASC (William McCune)	CASC (Argonne National Laboratory)
Paradox 3.0	FNT* SAT	CASC	*CASC-J4 FNT* winner, SAT runner-up*
SInE 0.4	LTB*	CASC	*CASC-J4 LTB* winner*
TPS 3.20080227G1d	THF	Peter B. Andrews	Carnegie Mellon University
		(Chad E. Brown)	(Saarland University)
Vampire 10.0	FOF* CNF	CASC	*CASC-J4 FOF* and CNF winner*
Vampire 11.0	FOF* CNF UEQ LTB*	Andrei Voronkov (Krystof Hoder)	The University of Manchester
Waldmeister 806	UEQ	CASC	*CASC-J4 UEQ winner*
Waldmeister C09a	UEQ	Thomas Hillenbrand	Max Planck Institut für Informatik

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

- The obfuscation of problem files changed. Predicate and function symbols names were not renamed to meaningless symbols. The arguments of associative connectives were randomly swapped, and implications were randomly reversed.
- An efficiency measure was been added to the results.
- Systems were explicitly encouraged to use the SZS standards for result status and output, and to produce solutions in TPTP format.
- In the LTB division:
  - Each batch specification file listed the `include` files that were used in every problem, to facilitate preloading and analysis of those files.
  - The problems had to be attempted in the batch order, and systems had to report when they start and end their attempt on each problem.
  - Prelearning of generally useful strategies was acceptable.
  - The +6 versions of the CYC problems were eligible for use.
- The THF (Typed Higher-order Form) demonstration division was added.

The competition organizer was Geoff Sutcliffe, assisted by Martin Suda. The competition was overseen by a panel of knowledgeable researchers who were not participating in the event; the CASC-22 panel members were Uli Furbach, William McCune, 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. The CASC-22 web site provides access to resources used before, during, and after the event: `http://www.tptp.org/CASC/22`

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

2 Divisions

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

2.1 The Competition Divisions

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

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

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

- The **FNE** category: FOF with No Equality

- The **FEQ** category: FOF with Equality

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

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

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

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

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

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

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

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

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

The **LTB** division: First-order form non-propositional theorems (axioms with a provable conjecture) from Large Theories, presented 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`.

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

2.2 The Demonstration Division

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

The entry specifies which competition divisions' problems are to be used. In addition to the competition divisions, for CASC-22 there is an additional demonstration 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 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.

3 Infrastructure

3.1 Computers

The competition computers were Dual-Opteron computers, each having:

- AMD Opteron(tm) Processor 250, 2390MHz CPU
- 4GB memory
- Linux 2.6.27.10.1.amd64-smp operating system

3.2 Problems

3.2.1 Problem Selection

The problems were 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 [SS01]:
  - Easy: Solvable by all state-of-the-art ATP systems
  - Difficult: Solvable by some state-of-the-art ATP systems
  - Unsolved: Solvable by no ATP systems
  - Open: Theoremhood unknown

  Difficult problems with a rating in the range 0.21 to 0.99 are eligible. Problems of lesser and greater difficulty ratings might also be eligible in some divisions (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 [GS96] (the competition organizers have to ensure that there is sufficient CPU time available to run the ATP systems on this minimal number of problems). The minimal numbers of problems is used in determining the CPU time limit imposed on each solution attempt - see Section 3.3.

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

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

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

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

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

3.2.3 Problem Preparation

The problems are in TPTP format, with `include` directives (included files are found relative to the `TPTP` environment variable). 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' use)
- 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, with `NNN` running from `001` to the number of problems in the respective division or category. The problems are specified to the ATP systems using the symbolic link names.

In the demonstration division the same problems are used as for the competition divisions, with the same preprocessing applied. However, the original file 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 BatchIncludes`
- `include` directives that are used in every problem (problems in the batch have all these include directives, and can also have other ¡TT¿include¡/TT¿ 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

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. In the non-LTB competition divisions the time limits are imposed individually on each solution attempt. In the LTB division the aggregated time limits (the individual time limits multiplied by the number of problems) are imposed on each category.

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

4 System Evaluation

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

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

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

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

- In the LTB division the proofs must be in TPTP format.
- An unsatisfiable set of ground instances of clauses is acceptable for establishing the unsatisfiability of a set of clauses.
- Models must be complete, documenting the domain, function maps, and predicate maps. The domain, function maps, and predicate maps may be specified by explicit ground lists (of mappings), or by any clear, terminating algorithm.

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

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 fraction of problems solved divided by the average CPU time for problems solved. This can be interpreted intuitively as the solution rate (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. However, it is not necessary for entrants to physically attend the competition.

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

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 before the sample solutions deadline. Use of the TPTP format for proofs and finite interpretations is encouraged. Proof samples for the FOF and LTB proof classes 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. A key must be provided if any non-obvious abbreviations for inference rules or other information are used.

6 System Requirements

6.1 System Properties

Systems are required to have the following properties:

- Systems have to run on a single locally provided standard UNIX computer (the *competition computers* - see Section 3.1). ATP systems that cannot run on the competition computers can be entered into the demonstration division.
- Systems must be fully automatic, i.e., any command line switches have to be the same for all problems in each division.
- Systems must be sound. At some time before the competition all the systems in the competition divisions are tested for soundness. Non-theorems are submitted to the systems in the FOF, CNF, EPR, UEQ, and LTB divisions, and theorems are submitted to the systems in the FNT, SAT and EPR divisions. Finding a proof of a non-theorem or a disproof of a theorem indicates unsoundness. If a system fails the soundness testing it must be repaired by the unsoundness repair deadline or be withdrawn. The soundness testing eliminates the possibility of a system simply delaying for some amount of time and then claiming to have found a solution. At some time after the competition, all high ranking systems in the competition divisions are tested over the entire TPTP. This provides a final check for soundness. For systems running on entrant supplied computers in the demonstration division, the entrant must perform the soundness testing and report the results to the competition organizers.
- Systems do not have to be complete in any sense, including calculus, search control, implementation, or resource requirements.

- Systems must be executable by a single command line, using an absolute path name for the executable, which might not be in the current directory. In the non-LTB divisions the command line arguments are the absolute path name of a symbolic link as the problem file name, the time limit (if required by the entrant), and entrant specified system switches. 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.
- The systems that run on the competition computers have to be interruptable by a `SIGXCPU` signal, so that the CPU time limit can be imposed, and interruptable 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.
- 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 [Sut08c]. For example

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

  or

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

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

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

- 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 before ending the attempt on the preceding problem. The systems must print SZS notification lines to `stdout` when starting and ending all work (including cleanup work at the end) on each problem. 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, 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.
- 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.
- 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 automatic strategy learning procedures are used, the learning must ensure that sufficient generalization is obtained so that no there is no specialization to individual problems. In the LTB division, prelearning of generally useful strategies that extend usefully to new unseen problems, based on the problems specified for each category, is acceptable.
- The system's performance must be reproducible by running the system again.

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

6.2 System Delivery

For systems running on the competition computers, entrants must email an installation package to the competition organizers by the system delivery deadline. The installation package must be a `.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 CPU 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 CPU time used to that point is noted. A system has solved a problem iff it outputs its termination string within the CPU time limit, and a system has produced a proof/model iff it outputs its end-of-proof/model string within the CPU time limit. The result and timing data is used to generate an HTML file, and a web browser is used to display the results.

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

6.4 System Checks

- Check: The ATP system can run on a computer that has the same configuration as the competition computers.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
     ... no temporary or intermediate files left here ...
prompt> ls /home/tptp/TPTP/Problems/GRP
     ... no temporary or intermediate files left here ...
prompt> ls /tmp
     ... no temporary or intermediate files left here by decent systems ...
```

- Check: Multiple concurrent executions do not clash.

```
prompt> (/bin/time /home/tptp/bin/TreeLimitedRun -q0 200 400
        /home/tptp/bin/MyATPSystem /home/tptp/tmp/CCC001-1.p) &
        (/bin/time /home/tptp/bin/TreeLimitedRun -q0 200 400
        /home/tptp/bin/MyATPSystem /home/tptp/tmp/CCC001-1.p)
TreeLimitedRun: ----------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 200s
TreeLimitedRun: WC  time limit is 400s
TreeLimitedRun: PID is 5827
TreeLimitedRun: ----------------------------------------------
TreeLimitedRun: ----------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 200s
TreeLimitedRun: WC  time limit is 400s
TreeLimitedRun: PID is 5829
TreeLimitedRun: ----------------------------------------------
SZS status Unsatisfiable for CCC001-1
FINAL WATCH: 147.8 CPU 150.0 WC

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

7 The ATP Systems

These system descriptions were written by the entrants.

7.1 Darwin 1.4.4

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

Architecture

Darwin [BFT04, BFT06] is an automated theorem prover for first order clausal logic. It is an implementation of the Model Evolution calculus [BT03]. The Model Evolution calculus lifts the propositional DPLL procedure to first-order logic. One of the main motivations for this approach was the possibility of migrating to the first-order level some of those very effective search techniques developed by the SAT community for the DPLL procedure.

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

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

Strategies

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

Implementation

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

Darwin is implemented in OCaml and has been tested under various Linux distributions. It is available from `http://goedel.cs.uiowa.edu/Darwin/`.

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.

7.2 E 1.1pre and EP 1.1pre

Stephan Schulz
Technische Universität München, Germany,

Architecture

E 1.1pre [Sch02, Sch04b] is a purely equational theorem prover. The core proof procedure operates on formulas in clause normal form, using a calculus that combines superposition (with selection of negative literals) and rewriting. No special rules for non-equational literals have been implemented, i.e., resolution is simulated via paramodulation and equality resolution. The basic calculus is extended with rules for AC redundancy elimination, some contextual simplification, and pseudo-splitting with definition caching. The latest versions of E also supports simultaneous paramodulation, either for all inferences or for selected inferences.

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

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

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

EP 1.1pre is just a combination of E 1.1pre in verbose mode and a proof analysis tool extracting the used inference steps.

Strategies

E has been optimized for performance over the TPTP. The automatic mode of E 1.1pre is partially inherited from the previous version and is based on about 150 test runs over TPTP 3.5.0 and additional tests on large proof problems. 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 suceeds or runs out of time.

Implementation

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

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

EP 1.1pre 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 1.1p will be hampered by the fact that it has to analyse the inference step listing, an operation that typically is about as expensive as the proof search itself. Nevertheless, it should be competitive among the FOF and LTB proof class systems.

7.3 E-Darwin 1.2

Björn Pelzer
University 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 [BFT04], and it implements the Model Evolution with Equality calculus [BT05]. The general operation of the original Darwin has been extended by inferences for handling equality, specifically a paramodulation and a reflexivity inference. Whereas the original Darwin derived units only, the new inferences in E-Darwin allow the derivation of multi-literal clauses. A more extensive detection and handling of redundancy has been added as well.

Strategies

The uniform search strategy is identical to the one employed in the original Darwin.

Implementation

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

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 the E-Darwin website at `http://www.uni-koblenz.de/~bpelzer/edarwin`.

Expected Competition Performance

E-Darwin development forked from the original Darwin at version 1.3 and is still very much work in progress, serving as a testbed for modifications to the model evolution calculus. The performance on problems without equality remains on the level of the by now outdated Darwin 1.3.

7.4 E-KRHyper 1.1.3

Björn Pelzer
University Koblenz-Landau, Germany

Architecture

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

An E-hyper tableau is a tree whose nodes are labeled with clauses and which is built up by the application of the inference rules of the E-hyper tableau calculus. The calculus rules are designed such

that most of the reasoning is performed using positive unit clauses. A branch can be extended with new clauses that have been derived from the clauses of that branch.

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

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

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

Strategies

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

Implementation

E-KRHyper is implemented in the functional/imperative language OCaml. 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 [McC03b]. 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.

The system runs on Unix and MS-Windows platforms and is available under the GNU Public License from the E-KRHyper website at `http://www.uni-koblenz.de/~bpelzer/ekrhyper`.

Expected Competition Performance

Since last year only minor changes have been done which could affect the performance within CASC, so the results are expected to be at or below the level of Otter.

7.5 Equinox 4.0

Koen Claessen
Chalmers University of Technology, Sweden

Architecture

Equinox is an experimental theorem prover for pure first-order logic with equality. It finds ground proofs of the input theory, by solving successive ground instantiations of the theory using an incremental SAT-solver. Equinox is based on a "stack" of increasingly powerful logics, starting at propositional logic all

the way up the first-order logic with equality. The implementation consists of a corresponding stack of theorem provers.

This is the first version of Equinox that is complete.

Strategies

There is only one strategy in Equinox:

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

Implementation

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

Expected Competition Performance

Equinox should perform better than previous years. There should be problems that it can solve that few other provers can handle.

7.6 Infinox 1.0

Ann Lilliestrom, Koen Claessen
Chalmers University of Technology, Sweden

Architecture

Infinox is a tool that (for some problems) can show the absence of finite models. It works by showing the existence of functions or relations with certain properties that imply infinity, such as:

1. for an injective function $f : D \to D$ that is not surjective, D must be infinite
2. for a surjective function $f : D \to D$ that is not injective, D must be infinite
3. for a relation $r : D \times D$ that is anti-symmetric, transitive, and serial, D must be infinite (or empty)

All methods in Infinox are based on one of these three principles.

Strategies

Infinox runs a number of fixed methods in sequence, and terminates if one of them succeeds.

Implementation

Infinox is implemented in Haskell using the GHC compiler, and it uses Stephan Schulz' E-prover as a back-end.

Expected Competition Performance

Infinox participates in only the demonstration division.

7.7 iProver 0.5

Konstantin Korovin
The University of Manchester, United Kingdom

Architecture

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

Strategies

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

For the LTB division a small script partitions the time available to the given problem set and runs iProver with the allocated time limit, and time is repartitioned after each solved problem.

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, either Vampire or E prover is used for clausification.

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

Expected Competition Performance

iProver 0.5 is the CASC-J4 EPR division winner.

7.8 iProver 0.7 and iProver-SInE 0.7

Konstantin Korovin
The University of Manchester, United Kingdom

Architecture

iProver is an automated theorem prover based on an instantiation calculus Inst-Gen [GK03, Kor08a] 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 [ES04] for reasoning with ground abstractions of first-order clauses. In addition to instantiation, iProver implements ordered resolution calculus and a combination of instantiation and ordered resolution, see [Kor08b] for the implementation details. The saturation process is implemented as a modification of a given clause algorithm. We use non-perfect discrimination trees for the unification indexes, priority queues for passive clauses and a compressed vector index for subsumption and subsumption resolution (both forward and backward). The following redundancy eliminations are implemented: blocking non-proper instantiations; dismatching constraints [GK04, Kor08b]; global subsumption [Kor08b]; resolution-based simplifications and propositional-based simplifications. 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.

For the LTB division, iProver-SInE runs iProver as the underlying inference engine of SInE 0.4 (described in Section 7.18), i.e., axiom selection is done by SInE, and proof attempts are done by iProver.

Strategies

iProver has around 40 options to control the proof search including options for literal selections, passive clause selections, frequency of calling the SAT solver, simplifications and options for combination of instantiation with resolution. At 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 at: `http://www.cs.man.ac.uk/~korovink/iprover/`.

Expected Competition Performance

We expect strong performance in the EPR division and reasonably good performance in FOF, CNF, FNT, SAT, and LTB divisions.

7.9 iProver-Eq 0.5

Konstantin Korovin, Christoph Sticksel
The University of Manchester, United Kingdom

Architecture

iProver-Eq extends the iProver system [Kor08a] with built-in equational reasoning, along the lines of [GK04]. 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. The main differences with iProver are that the ground reasoning is done modulo the theory of equality and the first-order reasoning is done by a variant of ordered paramodulation calculus.

The first-order reasoning in iProver-Eq consists of three components, each based on a given clause algorithm: selection of clauses for instantiation, unit paramodulation on selected literals and resolution for simplifications. For non-equational input, the system behaves almost identically to the iProver system, employing the same calculus and simplifications. When equational literals are selected in clauses, a separate calculus based on ordered paramodulation and demodulation is used to find conflicts between selected literals. Literals participating in a conflict give rise to instantiations of the clauses they are selected in and these instances are in turn passed to the ground solver. The ground solver is required to reason modulo equality and we have integrated the CVC3 solver [BT07] for this task.

Strategies

The most influential options on the proof search in iProver-Eq control literal selection and passive clause selection in the individual given clause algorithms as well as the global distribution of time between the three parts and the ground solver. At CASC, iProver-Eq will execute a fixed schedule of selected options as well as simple heuristics for the initial distribution between the equational and non-equational reasoning.

Implementation

iProver-Eq is implemented in OCaml. For the ground reasoning iProver-Eq uses CVC3 in the equational case and MiniSat in the non-equational case. iProver-Eq accepts FOF and CNF formats. In the case of FOF format, the E prover is used for clausification.

Expected Competition Performance

iProver-Eq is still work in progress and has not yet been evaluated. We hope iProver-Eq will perform reasonably well in FOF and CNF divisions, in particular improving iProver's performance on equational problems.

7.10 Isabelle/HOL 2009

Jasmin 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 [NPW02] is the higher-order logic incarnation of the generic proof assistant Isabelle2009. While designed for interactive proof development, Isabelle/HOL also provides several automatic proof tactics, notably an equational reasoner [Nip89], a classical reasoner [PN94], a tableau prover [Pau99], and a first-order resolution-based prover [Hur03].

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 nine 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 nine 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 simply 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 [Lov78]. Constructs proofs directly in Isabelle.

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` datatypes. 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

Early results with Isabelle/HOL 2008 suggest that Isabelle will finish third in the THF category, immediately after TPS. However, in the meantime, two unsound proof modes have been disabled in TPS. This could be enough to provide us with a second place.

7.11 leanCoP 2.1 and leanCoP-SInE 2.1

Jens Otten
University of Potsdam, Germany

Architecture

leanCoP [OB03, Ott08b] is an automated theorem prover for classical first-order logic. It is a very compact implementation of the connection (tableau) calculus [Bib87, LS01]. For the LTB division, leanCoP-SInE runs leanCoP as the underlying inference engine of SInE 0.4 (described in Section 7.18), i.e., axiom selection is done by SInE, and proof attempts are done by leanCoP.

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. leanCoP 2.1 includes all the additional strategies and search techniques of leanCoP 2.0, i.e., regularity, lemmata, restricted backtracking [Ott08a], and a fixed strategy scheduling.

Implementation

leanCoP is implemented in Prolog. The source code of the core prover is only a few lines long and fits on half a page. Prolog's built-in indexing mechanism is used to quickly find connections. While the core prover and the definitional clausal form transformation from leanCoP 2.0 have not changed, the following extensions and improvements have been integrated into leanCoP 2.1:

- Besides the leanCoP syntax the (raw) TPTP syntax is now accepted as input format; the equality axioms are automatically added if required.
- The leanCoP core prover returns a compact connection proof, which is translated into a readable proof.
- leanCoP 2.1 is now available for several Prolog systems, e.g., ECLiPSe, SICStus, and SWI Prolog.
- The fixed strategy scheduling from leanCoP 2.0 was slightly changed and is now completely integrated into the shell script used to invoke the prover.

The source code of leanCoP 2.1, which is available under the GNU general public license, together with more information can be found on the leanCoP web site at `http://www.leancop.de`.

Expected Competition Performance

We expect the performance of leanCoP 2.1 to be similar to that of leanCoP 2.0. On problems with many axioms, like those in the LTB division of CASC, the combined leanCoP-SInE prover should perform significantly better than leanCoP alone.

7.12 LEO-II 1.0

Christoph Benzmüller[1], Frank Theiss[1]
[1]International University of Germany, Germany, [1]Articulate Software, USA

Architecture

The higher-order theorem prover LEO-II is the successor of LEO [BK98], which was developed as part of the Saarbruecken OMEGA system [SBA06]. The development of the independent LEO-II prover started in 2006 at Cambridge University, UK, in a joint project with Larry Paulson that was funded by the EPSRC. Previous versions of LEO-II have been described in the literature [BPTF07, BPTF08].

The calculus currently employed by LEO-II is best characterized as a strategic refinement of extensional higher-order RUE resolution [Ben99]. Novel aspects in LEO-II version 1.0 over previous versions include splitting of the conjecture in several subproblems (if appropriate), improved clause normalization and improved strategies for the basic fragment of simple type theory [BS09] (all but one of the examples that were characterized as challenging for higher-order provers in [BS09] are now provable in less than 0.15 seconds in LEO-II). LEO-II offers also an interactive mode and, most importantly, it is capable of generating proof objects in TSTP format.

Strategies

LEO-II is designed to cooperate with specialist systems for fragments of higher-order logic. The idea is to combine the strengths of the different systems: LEO-II predominantly addresses higher-order aspects in its reasoning process, with the aim of quickly removing higher-order clauses from the search space, and turning them into first-order clauses that can be refuted with a first-order ATP system. Currently, LEO-II is capable of cooperating with the first-order ATP systems E, SPASS, and Vampire. By default LEO-II cooperates with E.

Implementation

LEO-II is implemented in Objective Caml version 3.10 and its problem representation language is the new TPTP THF language [BRS08].

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 [TB06], support for primitive equality reasoning (extensional higher-order RUE resolution), and improved heuristics at the calculus level.

Unfortunately the LEO-II system currently uses only a very simple sequential collaboration model with first-order ATPs instead of using the more advanced, concurrent and resource-adaptive OANTS architecture [BSJK08] as exploited by its predecessor LEO.

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

Expected Competition Performance

LEO-II's competition performance is hard to predict for different reasons: this is the first higher-order CASC competition ever held, the THF TPTP library [SBBT09] grew very fast over the last year (faster than the first-order TPTP at any point) and many novel problems have recently entered the THF TPTP, LEO-II version 1.0 has substantially changed compared to the previous versions, and the performance of the competitor systems is hard to predict since they appear to be improving fast. An exciting and very motivating situation!

7.13 Matita 1.0

The HELM Team and contributors
University of Bologna, Italy

Architecture

Matita (0.5.7) is an interactive theorem prover including a fully automatic module (prover) for problems falling in the first order unit equality case. An overall description of the system and its architecture can be found in [ACTZ07], but there is no paper devoted to the automatic prover built in the system.

Strategies

The prover is based on standard superposition calculus for unit equality clauses.

Implementation

Matita is written in Objective Caml, a functional language similar to ML developed at INRIA. The automatic prover participating to the CASC is a module of 4000 lines, designed with simplicity and ease of integration in mind. It uses the functor abstraction mechanism of the OCaml language to achieve a good degree of reusability (i.e., comparison over terms is an abstract notion, that can be instantiated with a reduction aware relation for higher order logics like the Calculus of Inductive Constructions on which Matita is based, or a simpler alpha equivalence relation for the TPTP first order language). Matita is based on the Curry-Howard Isomorphism, i.e., proofs are lambda-terms of the CIC calculus. The automatic prover is thus able to produce a trace suitable for proof term reconstruction [AT07].

The ATP system registered to the CASC competition will be part of the upcoming 1.0 release of the Matita interactive theorem prover, and is thus not part of any officially released versions, but can be downloaded using the SVN repository at `http://matita.cs.unibo.it`.

Expected Competition Performance

Since the system never participated to the CASC, there is no real expectation, but we hope not to finish last ;-).

7.14 Metis 2.2

Joe Hurd
Galois, Inc., USA

Architecture

Metis 2.2 [Hur03] is a proof tactic used in the HOL4 interactive theorem prover. It works by converting a higher order logic goal to a set of clauses in first order logic, with the property that a refutation of the clause set can be translated to a higher order logic proof of the original goal.

Experiments with various first order calculi [Hur03] have shown a given clause algorithm and ordered resolution to best suit this application, and that is what Metis 2.2 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 ¡em¿Active¡/em¿ 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, a finite model is manually constructed to interpret standard functions and relations in such a way as to make many standard axioms true and negated goals false. Non-standard functions and relations are interpreted randomly, but with a bias towards making negated goals false. Since it is part of the CASC competition rules that standard functions and relations are obfuscated, Metis 2.2 will back off to the biased random interpretation of all functions and relations (except equality), using a finite model with 4 elements.

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 can be downloaded from `http://www.gilith.com/software/metis`.

Expected Competition Performance

The major change between Metis 2.1, which was entered into CASC-J4, and Metis 2.2 is the representation of normalization steps in TSTP proof format. There were only minor changes to the core proof engine, so Metis 2.2 is expected to perform at approximately the same level and end up in the lower half of the table.

7.15 OSHL-S 0.6

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

Architecture

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

Strategies

OSHL-S employs a uniform strategy for all problems, which is similar to OSHL: it starts with a all negative or all positive model. In each iteration, it finds contradicting instances of the model using inferred types, and then modifies the model to make the instance satisfiable, if possible.

Implementation

OSHL-S is implemented in Java using Java SE 6 SDK and the Netbeans IDE. The implementation of OSHL-S combines a few strategies for improving the performance of the system, including relaxed ordering, context-free types, relevance ranking, and batched generation. The website for the prover and tools is `http://www.cs.unc.edu/~xuh/oshls`.

Expected Competition Performance

The performance should be improved over that of OSHL-S 0.1.

7.16 Otter 3.3

William McCune
Argonne National Laboratory, USA

Architecture

Otter 3.3 [McC03b] is an ATP system for statements in first-order (unsorted) logic with equality. Otter is based on resolution and paramodulation applied to clauses. An Otter search uses the "given clause algorithm", and typically involves a large database of clauses; subsumption and demodulation play an important role.

Strategies

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

Implementation

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

Expected Competition Performance

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

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

7.17 Paradox 3.0

Koen Claessen, Niklas Sörensson
Chalmers University of Technology, Sweden

Architecture

Paradox [CS03] is a finite-domain model generator. It is based on a MACE-style [McC03a] 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-J4 FNT division winner.

7.18 SInE 0.4

Krystof Hoder
The University of Manchester, United Kingdom

Architecture

SiNE 0.4 is an axiom selection system for first order theories. It uses a syntactic approach based on symbols' presence in axioms and the conjecture. (When we say symbols, we mean function and predicate symbols taken together.) A relation D (as in "Defines") is created between symbols and axioms, which represents the fact that for a symbol there are some axioms that "give it its meaning". After the relation has been constructed, the axiom selection starts. At the beginning only the conjecture is selected, in each iteration the selection is extended by all axioms that are D-related to any symbol used in already selected formulae. The iteration continues until no more axioms are selected. The selected axioms and the conjecture are then handed to an underlying inference engine.

Strategies

In order to construct the D-relation, we compute for each symbol the number of axioms in which it appears, which we call the *generality index* of the symbol. Then each axiom is put into the D-relation of the least general symbol it contains. (If there are more symbols with lowest generality index, the axiom is put in the relation of all of them.) This strategy selects about 2% of axioms on problems `CSR(075-109)+1` In order to get different sets of axioms for multiple proof attempts on each problem, we have introduced a *benevolence* parameter. Given benevolence $b \geq 1$, formula F, which has the least general symbol with generality index g_0, is put into the D-relations of all its symbols with generality index $g \leq b \times g_0$. Another way to modify the axiom set is to add axioms that were used in proofs of previously proved problems (which used the same axiom set).

Implementation

The axiom selection is implemented in Python. Problems are accessed one by one as present in the batch file. First each problem's time limit is calculated. This calculation is "optimistic" in a way that it assumes that the success rate of problems to come will be the same as of the already encountered ones. (It gives a problem more than it's fair share of time, assuming that many of the problems won't use their whole time limit.) Each problem is attempted in five phases, each of them specifies different selection parameters (i.e., benevolence and whether the axioms from previous proofs are included). For each phase a D-relation is constructed (or reused if it was constructed for the same axiom set previously). Then axioms are selected and passed to an underlying prover. This version contains EP as an underlying prover, as it was used last year by the competition-winning version.

SInE is available from `http://www.cs.man.ac.uk/~hoderk/sine/sine.tgz`.

Expected Competition Performance

SInE 0.3 was the CASC-J4 LTB division winner. SInE 0.4 is a minimally updated version for the new LTB rules.

7.19 TPS 3.20080227G1d

Peter B. Andrews[1], Chad E. Brown[1]
[1]Carnegie Mellon University, USA, [1]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 [ABI+96, AB06].

When searching for a proof, TPS first searches for an expansion proof [Mil87] or an extensional expansion proof [Bro07] of the theorem. Part of this process involves searching for acceptable matings [And81]. Using higher-order unification, a pair of occurrences of subformulas (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 [Pfe87], 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 [And89].
- Path-focused duplication [Iss90].
- Dual instantiation of definitions, and generating substitutions for higher-order variables which contain abbreviations already present in the theorem to be proved [BA98].
- Component search [Bis99].
- Generating and solving set constraints [Bro02].
- Generating connections using extensional and equational reasoning [Bro07].

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 is quite versatile when searching for proofs of higher-order theorems, but it is effectively slow when it must try many modes. Therefore, the performance of TPS is likely to depend very heavily on the way the competition is set up. If the emphasis is on speed, with short time limits, TPS may not do very well.

On the other hand, if the emphasis is on the variety of theorems that can be proved, with very liberal time limits, TPS may do very well.

7.20 Vampire 10.0

Andrei Voronkov
University of Manchester, United Kingdom

No system description supplied.

Expected Competition Performance

Vampire 10.0 is the CASC-J4 FOF and CNF division winner.

7.21 Vampire 11.0

Krystof Hoder, Andrei Voronkov
The 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 last year's LTB divison winner 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

We expect Vampire 11.0 to perform comparably to the Vampire 10.0.

7.22 Waldmeister 806

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

No system description supplied.

Expected Competition Performance

Waldmeister 806 is the CASC-J4 UEQ division winner.

7.23 Waldmeister C09a

Thomas Hillenbrand
Max-Planck-Institut für Informatik, Germany

Architecture

Waldmeister C09a [Hil03] is a system for unit equational deduction. Its theoretical basis is unfailing completion in the sense of [BDP89] with refinements towards ordered completion (cf. [AHL03]). 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 [HJL99].

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

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 strucures 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`.

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 [HJL99]. 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.

Expected Competition Performance

Waldmeister C09a will output much nicer proofs than Waldmeister 806.

8 Conclusion

The CADE-22 ATP System Competition is the fourteenth large scale competition for classical logic ATP systems. The organizers believe that CASC fulfills its main motivations: stimulation of research, motivation for improving implementations, evaluation of relative capabilities of ATP systems, and providing an exciting event. Through the continuity of the event and consistency in the the reporting of the results, performance comparisons with previous and future years are easily possible. The competition provides exposure for system builders both within and outside of the community, and provides an overview of the implementation state of running, fully automatic, classical logic, ATP systems.

References

[AB06] 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.

[ABI+96] 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.

[ACTZ07] A. Asperti, C. Coen, E. Tassi, and S. Zacchiroli. User Interaction with the Matita Proof Assistant. *Journal of Automated Reasoning*, 39(2):109–139, 2007.

[AHL03] 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.

[And81] P. B. Andrews. Theorem Proving via General Matings. *Journal of the ACM*, 28(2):193–214, 1981.

[And89] P. B. Andrews. On Connections and Higher-Order Logic. *Journal of Automated Reasoning*, 5(3):257–291, 1989.

[AT07] A. Asperti and E. Tassi. Higher order Proof Reconstruction from Paramodulation-Based Refutations: The Unit Equality Case. In M. Kauers, M. Kerber, R. Miner, and W. Windsteiger, editors, *Proceedings of the 6th International Conference on Mathematical Knowledge Management*, volume 4573 of *Lecture Notes in Computer Science*, pages 146–160, 2007.

[BA98] 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.

[BDP89] 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.

[Ben99] 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.

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

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

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

[BFT06] P. Baumgartner, A. Fuchs, and C. Tinelli. Implementing the Model Evolution Calculus. *International Journal on Artificial Intelligence Tools*, 15(1):21–52, 2006.

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

[Bib87] W. Bibel. *Automated Theorem Proving*. Vieweg and Sohn, 1987.

[Bis99] 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.

[BK98] 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.

[BPTF07] C. Benzmüller, L. Paulson, F. Theiss, and A. Fietzke. Progress Report on LEO-II - An Automatic Theorem Prover for Higher-Order Logic. In K. Schneider and J. Brandt, editors, *Proceedings of the 20th International Conference on Theorem Proving in Higher Order Logics*, pages 33–48, 2007.

[BPTF08] 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.

[Bro02] 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.

[Bro07] 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.

[BRS08] 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.

[BS09] 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.

[BSJK08] C. Benzmüller, V. Sorge, M. Jamnik, and M. Kerber. Combined Reasoning by Automated Cooperation. *Journal of Applied Logic*, 6(3):318–342, 2008.

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

[BT05] 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.

[BT07] C. Barrett and C. Tinelli. CVC3. In W. Damm and H. Hermanns, editors, *Proceedings of the 19th International Conference on Computer Aided Verification*, number 4590 in Lecture Notes in Computer Science, pages 298–302. Springer-Verlag, 2007.

[CS03] 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.

[ES04] 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.

[GK03] H. Ganzinger and K. Korovin. New Directions in Instantiation-Based Theorem Proving. In P. Kolaitis, editor, *Proceedings of the 18th IEEE Symposium on Logic in Computer Science*, pages 55–64. IEEE Press, 2003.

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

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

[Hil03] 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, 2003.

[HJL99] 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.

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

[Iss90] 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.

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

[Kor08b] K. Korovin. iProver - An Instantiation-Based Theorem Prover for First-order Logic (System Description). In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 292–298, 2008.

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

[Lov78] D.W. Loveland. *Automated Theorem Proving : A Logical Basis*. Elsevier Science, 1978.

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

[McC03a] W.W. McCune. Mace4 Reference Manual and Guide. Technical Report ANL/MCS-TM-264, Argonne National Laboratory, Argonne, USA, 2003.

[McC03b] W.W. McCune. Otter 3.3 Reference Manual. Technical Report ANL/MSC-TM-263, Argonne National Laboratory, Argonne, USA, 2003.

[Mil87] D. Miller. A Compact Representation of Proofs. *Studia Logica*, 46(4):347–370, 1987.

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

[Nip89] T. Nipkow. Equational Reasoning in Isabelle. *Science of Computer Programming*, 12(2):123–149, 1989.

[NPW02] 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.

[OB03] J. Otten and W. Bibel. leanCoP: Lean Connection-Based Theorem Proving. *Journal of Symbolic Computation*, 36(1-2):139–161, 2003.

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

[Ott08b] J. Otten. leanCoP 2.0 and ileancop 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic. In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 283–291, 2008.

[Pau99] L. Paulson. A Generic Tableau Prover and its Integration with Isabelle. *Artificial Intelligence*, 5(3):73–87, 1999.

[Pfe87] F. Pfenning. *Proof Transformations in Higher-Order Logic*. PhD thesis, Carnegie-Mellon University, Pittsburg, USA, 1987.

[PN94] L.C. Paulson and T. Nipkow. *Isabelle: A Generic Theorem Prover*. Number 828 in Lecture Notes in Computer Science. Springer-Verlag, 1994.

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

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

[SBA06] J. Siekmann, C. Benzmüller, and S. Autexier. Computer Supported Mathematics with OMEGA. *Journal of Applied Logic*, 4(4):533–559, 2006.

[SBBT09] G. Sutcliffe, C. Benzmüller, C. E. Brown, and F. Theiss. Progress in the Development of Automated Theorem Proving for Higher-order Logic. In R. Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction*, number 5663 in Lecture Notes in Artificial Intelligence, pages 116–130. Springer-Verlag, 2009.

[Sch02] S. Schulz. E: A Brainiac Theorem Prover. *AI Communications*, 15(2-3):111–126, 2002.

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

[Sch04b] S. Schulz. System Abstract: E 0.81. In M. Rusinowitch and D. Basin, editors, *Proceedings of the 2nd International Joint Conference on Automated Reasoning*, number 3097 in Lecture Notes in Artificial Intelligence, pages 223–228, 2004.

[SS97a] G. Sutcliffe and C.B. Suttner, editors. *Special Issue: The CADE-13 ATP System Competition*, volume 18, 1997.

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

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

[SS98a] G. Sutcliffe and C. Suttner. The CADE-14 ATP System Competition. Technical Report 98/01, Depart-

ment of Computer Science, James Cook University, Townsville, Australia, 1998.

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

[SS98c] C.B. Suttner and G. Sutcliffe. The CADE-14 ATP System Competition. *Journal of Automated Reasoning*, 21(1):99–134, 1998.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[Sut08a] G. Sutcliffe. Proceedings of the 4th IJCAR ATP System Competition. Sydney, Australia, 2008.

[Sut08b] G. Sutcliffe. The CADE-21 Automated Theorem Proving System Competition. *AI Communications*, 21(1):71–82, 2008.

[Sut08c] 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.

[Sut09a] G. Sutcliffe. The 4th IJCAR Automated Theorem Proving System Competition - CASC. *AI Communications*, 22(1):59–72, 2009.

[Sut09b] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, To appear, 2009.

[TB06] F. Theiss and C. Benzmüller, editors. *Proceedings of the 6th International Workshop on the Implementation of Logics*, number 212 in CEUR Workshop Proceedings, 2006.

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

[Wer03] C. Wernhard. System Description: KRHyper. Technical Report Fachberichte Informatik 14–2003,

Universität Koblenz-Landau, Koblenz, Germany, 2003.