Proceedings of the CADE-23 ATP System Competition CASC-23

Geoff Sutcliffe
University of Miami, USA

Abstract

The CADE ATP System Competition (CASC) evaluates the performance of sound, fully automatic, classical logic, ATP systems. The evaluation is in terms of the number of problems solved, the number of acceptable proofs and models produced, and the average runtime for problems solved, in the context of a bounded number of eligible problems chosen from the TPTP problem library, and specified time limits on solution attempts. The CADE-23 ATP System Competition (CASC-23) was held on 3rd August 2011. The design of the competition and its 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-23 was held on 3rd August 2011, as part of the 23rd International Conference on Automated Deduction (CADE-23), in Wroclaw, Poland. It is the sixteenth competition in the CASC series [117, 122, 120, 89, 91, 116, 114, 115, 96, 98, 100, 102, 105, 108, 109].

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

Thirty-five ATP systems and variants, listed in Table 1, entered into the various competition and demonstration divisions. The winners of the CASC-J5 (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).[1]

The design and procedures of this CASC evolved from those of previous CASCs [117, 118, 113, 119, 87, 88, 90, 92, 93, 94, 95, 97, 99, 101, 104, 107]. Important changes for this CASC were:

- The TFA division was a full competition division, with two problem categories.
- The TNT (Typed higher-order form Non-Theorem) demonstration division was added.

[1] The CASC-J5 LTB winner, Vampire-LTB 0.6, was unable to run in CASC-23 due to changes in the batch specification files' format

ATP System	Divisions	Entrants (Associates)	Entrant's Affiliation
CVC3 2.4	TFA	Cesare Tinelli (Clark Barrett)	University of Iowa
E(P/LTB) 1.4pre	FOF FNT CNF EPR UEQ LTB	Stephan Schulz	Technische Universität München
E-Darwin 1.4	FOF FNT CNF EPR UEQ	Björn Pelzer	Universität Koblenz-Landau
E-KRHyper 1.1.4	FOF FNT CNF EPR UEQ LTB	Björn Pelzer	Universität Koblenz-Landau
E-MaLeS 1.0	FOF	Daniel Kuehlwein (Josef Urban, Stephan Schulz)	Radboud Universiteit Nijmegen
FIMO 0.2	FNT EPR	Orkunt Sabuncu	University of Potsdam
H2WO4 11.07	TFA (demo)	David Stanovsky	Charles University in Prague
iProver 0.8	EPR	CASC	CASC-J5 EPR winner
iProver(-SInE) 0.9	FOF FNT CNF EPR UEQ LTB	Konstantin Korovin	University of Manchester
iProver-Eq(-SInE) 0.7	FOF FNT CNF EPR UEQ LTB	Christoph Sticksel (Konstantin Korovin)	University of Manchester
Isabelle/HOL 2011	THF	Jasmin Blanchette (Larry Paulson, Tobias Nipkow, Makarius Wenzel)	Technische Universität München
leanCoP 2.2	FOF	Jens Otten	University of Potsdam
LEO-II 1.2	THF	CASC	CASC-J5 THF winner
LEO-II 1.2.8	THF FOF CNF	Christoph Benzmüller	Freie Universität Berlin
MELIA 0.1	TFA	Peter Baumgartner	NICTA and ANU
Metis 2.3	FOF CNF EPR UEQ	Joe Hurd	Galois, Inc.
MetiTarski 1.8	TFA (demo)	Larry Paulson	University of Cambridge
Muscadet 4.1	FOF	Dominique Pastre	University Paris Descartes
Nitpick 2011	TNT	Jasmin Blanchette	Technische Universität München
Nitrox 0.2	FNT	Jasmin Blanchette (Emia Torlak)	Technische Universität München
Otter 3.3	FOF CNF UEQ	CASC (William McCune)	CASC
Paradox 3.0	FNT	CASC	CASC-J5 FNT winner
Refute 2011	TNT	Jasmin Blanchette (Tjark Weber)	Technische Universität München
Satallax 2.1	THF TNT	Chad E. Brown	Saarland University
SPASS+T 2.2.14	TFA	Uwe Waldmann (Stephan Zimmer)	Max Planck Institut für Informatik
SPASS-XDB 0.8	TFA (demo)	Geoff Sutcliffe (Martin Suda, David Stanovsky)	University of Miami
TPS 3.110228S1a	THF	Chad E. Brown	Saarland University
Vampire 0.6	FOF CNF	CASC	CASC-J5 FOF and CNF winner
Vampire 1.8	FOF CNF EPR UEQ LTB	Andrei Voronkov (Kryštof Hoder)	University of Manchester
Waldmeister 710	UEQ	CASC	CASC-J5 UEQ winner
Z3 2.20	TFA (demo)	Nikolaj Bjorner (Leonardo de Moura)	Microsoft Research

Table 1: The ATP systems and entrants

- The FOF division had only a proof ranking class. The FNT division had only a model ranking class. Systems that do not output proofs/models could still enter the FOF/FNT divisions, and the number of problems solved were shown in the results. Such systems just could not win the division trophy. The CNF division had only an assurance ranking class. The LTB division had only an assurance ranking class.
- In the LTB division:
  - The ISA problem category was added to the LTB division.
  - A batch specification file could have multiple batches, each consisting of a configuration section, an includes section, and a problems section. Each batch is independent of the other batches in the file.
  - The batch configuration sections specified what output was required and desired from the ATP systems.
  - The overall time limit for each problem category was available only as a command line parameter.
  - The ISA problem category had an additional performance measure, measuring the number and accuracy of lists of axioms sufficient for a proof - these are useful for replaying proofs within Isabelle.
  - The SMO problem category had an additional performance measure, counting the number of problems solved with the bindings for outermost existentially quantified variables reported – these are answers for query conjectures.

The competition organizer was Geoff Sutcliffe. The competition was overseen by a panel of knowledgeable researchers who were not participating in the event; the CASC-23 panel members were Franz Baader, Koen Claessen, 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-23 web site provides access to resources used before, during, and after the event: `http://www.tptp.org/CASC/23`

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

2 Divisions

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

2.1 The Competition Divisions

The competition divisions are open to ATP systems that meet the required system properties described in Section 6.1. Each competition division uses problems that have certain logical,

language, and syntactic characteristics, so that the ATP systems that compete in the division are, in principle, able to attempt all the problems in the division.

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

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

The **TFA** division: Typed First-order with Arithmetic theorems (axioms with a provable conjecture, using the TFF0 syntax. The TFA division has two problem categories:

- The **TFI** category: TFA with only Integer arithmetic
- The **TFR** category: TFA with only Rational and only Real arithmetic (no mixed rational and real arithmetic)

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

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

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

- The **FNN** category: FNT with No equality
- The **FNQ** category: FNT with eQuality

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

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

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

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

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

The **LTB** division: First-order form non-propositional theorems (axioms with a provable conjecture) from Large Theories, presented in Batches. The LTB division has four 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 **ISA** category: Problems taken from an Isabelle contribution to the `SWW` domain of the TPTP. These problems are `SWW104` to `SWW396`.
- The **MZR** category: Problems taken from the Mizar Problems for Theorem Proving (MPTP) contribution to the TPTP. These are problems `ALG214` to `ALG234`, `CAT021` to `CAT037`, `GRP618` to `GRP653`, `LAT282` to `LAT380`, `SEU406` to `SEU451`, and `TOP023` to `TOP048`.
- The **SMO** category: Problems taken from the Suggested Upper Merged Ontology (SUMO) contribution to the `CSR` domain of the TPTP. These are problems `CSR075` to `CSR109`, and `CSR118`.

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

2.2 The Demonstration Division

ATP systems that cannot run in the competition divisions for any reason (e.g., the system requires special hardware, or the entrant is an organizer) can be entered into the demonstration division. Demonstration division systems can run on the competition computers, or the computers can be supplied by the entrant. Computers supplied by the entrant may be brought to CASC, or may be accessed via the internet. The demonstration division results are presented along with the competition divisions' results, but might not be comparable with those results. The systems are not ranked and no prizes are awarded. For CASC-23 there was an additional demonstration division:

The **TNT** division: Typed higher-order form Non-Theorems (axioms with a countersatisfiable conjecture, and satisfiable axiom sets), using the THF0 syntax. The TNT division has two problem categories:

- The **TTN** category: TNT with No equality
- The **TTE** category: TNT with Equality

3 Infrastructure

3.1 Computers

The computers were Dell PowerEdge blade computers, each having:

- Two Intel Xeon E5620, quad-core, 2.40GHz CPUs
- 48GB memory
- GNU Linux cl5-001 2.6.32.22.1.amd64-smp operating system

In the non-LTB division systems could use only one core, and were limited to a fraction of the memory, as explained in Section 3.3 (multiple jobs were run on each node). In the LTB division each system was allocated one node, and could use all the cores and memory.

3.2 Problems

3.2.1 Problem Selection

The problems were taken from the TPTP problem library, version v5.2.0. 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 [121], and from the ratings classifies problems as one of:
  - 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 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 must 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 [45] (the competition organizers have to ensure that there are sufficient computers available to run the ATP systems on this minimal number of problems). The minimal numbers of problems is used in determining the time limits imposed on each solution attempt - see Section 3.3.

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

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

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

The numbers of problems used in the categories in the various divisions are (roughly) proportional to the numbers of eligible problems than can be used in the categories, after taking into account the limitation on very similar problems. The numbers of problems used in each division and category are determined according to the judgement of the competition organizers.

3.2.3 Problem Preparation

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

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

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

In order to prevent systems from recognizing problems from their file names, symbolic links are made to the selected problems, using names of the form `CCCNNN-1.p` for the symbolic links. `CCC` is the division or problem category name, and `NNN` runs from `001` to the number of problems in the respective division or problem category. The problems are specified to the ATP systems using the symbolic link names.

In the demonstration division the same problems are used as for the competition divisions, with the same preprocessing applied. However, the original fille names can be retained for systems running on computers provided by the entrant.

3.2.4 LTB Batch Specification Files

In the LTB division, the problems for each category are listed in a batch specification file, containing one or more *batch specifications*. Each batch specification consists of:

- A header line `% SZS start BatchConfiguration`
- A problem category line of the form
  `division.category LTB.`*category_mnemonic*
- A specification of what output is required from the ATP systems for each problem, in a line of the form
  `output.required` *space_separated_list*
  where the available list values are the SZS values `Assurance`, `Proof`, `Model`, and `Answer`. For CASC-23 it was
  `output.required Assurance`.
- A specification of what output is desired from the ATP systems for each problem, in a line of the form
  `output.desired` *space_separated_list*
  where the list values are as for the required output. For the CASC-23 CYC and MZR problem categories it was
  `output.desired Proof`
  For the CASC-23 ISA problem category it was
  `output.desired Proof ListOfFOF`
  where the `ListOfFOF` is a list of axioms sufficient for a proof (if a proof is output then the list of axioms is not necessary). For the CASC-23 SMO problem category it was

`output.desired Proof Answer`
where the answer is a definite binding for the outermost existentially quantified variables of the conjecture.
- The wall clock time limit per problem, in a line of the form
`limit.time.problem.wc` *limit_in_seconds*
- A terminator line `% SZS end BatchConfiguration`
- A header line `% SZS start BatchIncludes`
- `include` directives that are used in every problem. Problems in the batch have all these `include` directives, and can also have other `include` directives that are not listed here.
- A terminator 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. The problems must be attempted in the given order. Systems may not start any attempt on a problem, including reading the problem file, before ending the attempt on the preceding problem.
- A terminator line `% SZS end BatchProblems`

3.3 Resource Limits

3.3.1 Non-LTB divisions

CPU and wall clock time limits are imposed. The minimal CPU time limit per problem is 240s. 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 per problem is double the CPU time limit. For CASC-23, where multiple jobs were run on each node, an additional memory limit of 6GB was imposed. The time limits are imposed individually on each solution attempt.

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

3.3.2 LTB division

For each batch there is a wall clock time limit per problem, which is provided in the configuration section at the start of each batch. The minimal wall clock time limit per problem is 30s. For each problem category there is an overall wall clock time limit, which is available as a command line parameter. The overall limit is the sum over the batches of the batch's per-problem limit multiplied by the number of problems in the batch. Time spent before starting the first problem of a batch (e.g., preloading and analysing the batch axioms), and times spent between ending a problem and starting the next (e.g., learning from a proof just found), are not part of the times taken on the individual problems, but are part of the overall time taken. There are no CPU time limits.

4 System Evaluation

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

The systems are ranked in the competitions division, from the performance data. The THF, TFA, CNF, EPR, UEQ, and LTB divisions have an *assurance* ranking class, ranked according to the number of problems solved, but not necessarily accompanied by a proof or model (thus giving only an assurance of the existence of a proof/model). The FOF and FNT divisions have a *proof/model* ranking class, ranked according to the number of problems solved with an acceptable proof/model output. Ties are broken according to the average time over problems solved (CPU time for the non-LTB divisions, wall clock time for the LTB division). In the competition divisions, class winners are announced and prizes are awarded.

- The Isabelle group at the Technische Universität München provided a travel prize for the ISA problem category of the LTB division. The prize was awarded according to the axiom accuracy measure described below. The winner was invited to visit the group at the university for up to one week. The travel and hotel expenses were covered.
- Rearden Commerce provided $3000 of prize money for the SMO category of the LTB division. Prizes were awarded for the assurance ranking class, and also according to the question answering measure described below. In each case the winner received $750, the second place $500, and the third place $250.

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

- Derivations must be complete, starting at formulae from the problem, and ending at the conjecture (for axiomatic proofs) or a $false$ formula (for proofs by contradiction, including CNF refutations).
- For proofs of FOF problems by CNF refutation, the conversion from FOF to CNF must be adequately documented.
- Derivations must show only relevant inference steps.
- Inference steps must document the parent formulae, the inference rule used, and the inferred formula.
- Inference steps must be reasonably fine-grained.
- An unsatisfiable set of ground instances of clauses is acceptable for establishing the unsatisfiability of a set of clauses.
- Models must be complete, documenting the domain, function maps, and predicate maps. The domain, function maps, and predicate maps may be specified by explicit ground lists (of mappings), or by any clear, terminating algorithm.

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

In addition to the ranking criteria, other measures are made and presented in the results:

- The *state-of-the-art contribution* (SOTAC) quantifies the unique abilities of each system. For each problem solved by a system, its SOTAC for the problem is the inverse of the

number of systems that solved the problem. A system's overall SOTAC is its average SOTAC over the problems it solves.
- The *efficiency measure* balances the number of problems solved with the CPU time taken. It is the average of the inverses of the times for problems solved (CPU times for the non-LTB divisions, wall clock times for the LTB division, with times less than the timing granularity rounded up to the granularity, to avoid skewing caused by very low times) multiplied by the fraction of problems solved. This can be interpreted intuitively as the average of the solution rates for problems solved, multiplied by the fraction of problems solved.
- In the ISA problem category of the LTB division, the *axiom accuracy* measures the number and accuracy of the lists of axioms sufficient for a proof that the system outputs (see Section 6.1.3). For each problem solved by a system, its axiom accuracy for the problem is the size of the smallest sufficient axiom set reported by any system, divided by the size of this system's axiom set (or 0 if this system does not report an axiom set). A system's overall axiom accuracy is the average of its problem axiom accuracies over the problems it solves, multiplied by the fraction of problems solved. This is the basis for the ISA category prize.
- In the SMO problem category of the LTB division, the number of *questions answered* (output of the bindings for the outermost existentially quantified variables) is counted (see Section 6.1.3). This is the basis for the SMO category prize.

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

Systems can be entered at only the division level, and can be entered into more than one division (a system that is not entered into a competition division is assumed to perform worse than the entered systems, for that type of problem - wimping out is not an option). Entering many similar versions of the same system is deprecated, and entrants may be required to limit the number of system versions that they enter. Systems that rely essentially on running other ATP systems without adding value are deprecated; the competition panel may disallow or move such systems to the demonstration division. The division winners from the previous CASC are automatically entered into their divisions, to provide benchmarks against which progress can be judged.

5.1 System Description

A system description has to be provided for each ATP system entered, using the HTML schema supplied on the CASC web site. (See Section 7 for these descriptions.) The schema has the following sections:

- Architecture. This section introduces the ATP system, and describes the calculus and inference rules used.
- Strategies. This section describes the search strategies used, why they are effective, and how they are selected for given problems. Any strategy tuning that is based on specific problems' characteristics must be clearly described (and justified in light of the tuning restrictions described in Section 6.1).
- Implementation. This section describes the implementation of the ATP system, including the programming language used, important internal data structures, and any special code libraries used. The availability of system is described here.
- Expected competition performance. This section makes some predictions about the performance of the ATP system in each of the divisions and categories in which it is competing.
- References.

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

5.2 Sample Solutions

For systems in the proof/model classes, representative sample solutions must be emailed to the competition organizers by the sample solutions deadline. Use of the TPTP format for proofs and finite interpretations is encouraged. Proof samples for the FOF proof class must include a proof for `SEU140+2`. Model samples for the FNT model class must include models for `NLP042+1` and `SWV017+1`. The sample solutions must illustrate the use of all inference rules. An explanation must be provided for any non-obvious features.

For systems competing for the ISA problem category prize in the LTB division, representative sample proofs or lists of axioms must be emailed to the competition organizers by the sample solutions deadline. Use of the SZS standards is required. Samples must include a proof or list for `SEU140+2`. For systems competing for the SMO problem category prize in the LTB division, representative sample answers must be emailed to the competition organizers by the sample solutions deadline. Samples must include an answer for `CSR082+1`.

6 System Requirements

6.1 System Properties

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

6.1.1 Soundness and Completeness

- Systems must be sound. At some time before the competition all the systems in the competition divisions are tested for soundness. Non-theorems are submitted to the systems

in the THF, TFA, FOF, CNF, EPR, UEQ, and LTB divisions, and theorems are submitted to the systems in the FNT and EPR divisions. Finding a proof of a non-theorem or a disproof of a theorem indicates unsoundness. If a system fails the soundness testing it must be repaired by the unsoundness repair deadline or be withdrawn. The soundness testing eliminates the possibility of a system simply delaying for some amount of time and then claiming to have found a solution. For systems running on entrant supplied computers in the demonstration division, the entrant must perform the soundness testing and report the results to the competition organizers.

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

6.1.2 Execution

- Systems must run on a single locally provided standard UNIX computer (the *competition computers* - see Section 3.1). ATP systems that cannot run on the competition computers can be entered into the demonstration division.
- Systems must be executable by a single command line, using an absolute path name for the executable, which might not be in the current directory. In the non-LTB divisions the command line arguments are the absolute path name of a symbolic link as the problem file name, the time limit (if required by the entrant), and entrant specified system switches. In the LTB division the command line arguments are the absolute path name of the batch specification file, the overall category 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 file names.
- Systems must be fully automatic, i.e., all command line switches have to be the same for all problems in each division.
- In the LTB division the systems must attempt the problems in the order given in the batch specification file. Systems may not start any attempt on a problem, including reading the problem file, before ending the attempt on the preceding problem.

6.1.3 Output

- In the non-LTB divisions all solution output must be to `stdout`. In the LTB division all solution output must be to the named output file for each problem.
- In the LTB division the systems must print SZS notification lines to `stdout` when starting and ending work on a problem (including any cleanup work, such as deleting temporary files). It is recommended that the result for the problem be output as the last thing before the ending notification line (note, the result must also be output to the solution file anyway). For example

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

```
% SZS status Theorem for /home/graph/tptp/TPTP/Problems/CSR/CSR075+2.p
% SZS status Ended for /home/graph/tptp/TPTP/Problems/CSR/CSR075+2.p
```

- For each problem, the systems must 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 [103]. For example

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

  or

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

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

  The first distinguished string output is accepted as the system's result.
- When outputting proofs/models, the start and end of the proof/model must be delimited by distinguished strings (specified by the entrant). The distinguished strings should use the SZS ontology and standards. For example

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

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

  The string specifying the problem status must be output before the start of a proof/model. Use of the TPTP format for proofs and finite interpretations is encouraged [111].
- When outputting lists of axioms sufficient for a proof for the ISA problem category of the LTB division, the start and end of the list must be delimited by distinguished strings. The distinguished strings should use the SZS ontology and standards. For example

  ```
  % SZS output start ListOfFOF for SEU104+2
    ...
  % SZS output end ListOfFOF for SEU140+2
  ```

- When outputting answers for the SMO problem category of the LTB division, the answers must be output using the Tuple or Instantiated answer form of the proposed TPTP standard for answer reporting.

6.1.4 Resource Usage

- The systems that run on the competition computers must be interruptible by a `SIGXCPU` signal, so that the CPU time limit can be imposed, and interruptible by a `SIGALRM` signal, so

that the wall clock time limit can be imposed. For systems that create multiple processes, the signal is sent first to the process at the top of the hierarchy, then one second later to all processes in the hierarchy. The default action on receiving these signals is to exit (thus complying with the time limit, as required), but systems may catch the signals and exit of their own accord. If a system runs past a time limit this is noticed in the timing data, and the system is considered to have not solved that problem.

- If an ATP system terminates of its own accord, it may not leave any temporary or intermediate output files. If an ATP system is terminated by a `SIGXCPU` or `SIGALRM`, it may not leave any temporary or intermediate files anywhere other than in `/tmp`. Multiple copies of the ATP systems must be executable concurrently, in the same (NFS cross mounted) directory. It is therefore necessary that temporary files have unique names.
- For practical reasons excessive output from an ATP system is not allowed. A limit, dependent on the disk space available, is imposed on the amount of output that can be produced. The limit is at least 10MB per system.

6.2 System Delivery

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

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

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

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

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

6.3 System Execution

Execution of the ATP systems on the competition computers is controlled by a `perl` script, provided by the competition organizers. The jobs are queued onto the computers so that each

computer is running one job at a time. In the non-LTB divisions, all attempts at the Nth problems in all the divisions and categories are started before any attempts at the (N+1)th problems. In the LTB division all attempts in each category in the division are started before any attempts at the next category.

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

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

7 The ATP Systems

These system descriptions were written by the entrants.

7.1 CVC3 2.4

Clark Barrett[1], Cesare Tinelli[2]
[1]New York University, [2]University of Iowa

Architecture
CVC3 [10] is a DPLL-based theorem prover for Satisfiability Modulo Theories (SMT) problems. It can be used to prove the validity (or, dually, the satisfiability) of first-order formulas in a large number of built-in logical theories and their combination. CVC3 is the last offspring of a series of popular SMT provers, which originated at Stanford University with the SVC system. In particular, it builds on the code base of CVC Lite, its most recent predecessor. Its high level design follows that of the Sammy prover.

CVC3 works with a version of first-order logic with polymorphic types and has a wide variety of features including:

- several built-in base theories: rational and integer linear arithmetic, arrays, tuples, records, inductive data types, bit vectors, and equality over uninterpreted function symbols;
- support for quantifiers;
- an interactive text-based interface;
- a rich C and C++ API for embedding in other systems;
- proof and model generation abilities;
- predicate subtyping;
- essentially no limit on its use for research or commercial purposes.

Strategies
CVC3 uses congruence closure for equality and uninterpreted functions, and Fourier-Motzkin for arithmetic. Perhaps most relevant to CASC are the strategies for quantifiers. CVC3 uses E-matching and instantiation heuristics to search for quantifier instantiations that can close search branches. In addition, some heuristics for *complete* instantiation are available.

Implementation
CVC3 is implemented in C++. For details of the implementation, downloads, additional publications, and a user's guide, please refer to the CVC3 web site

`http://www.cs.nyu.edu/acsys/cvc3`

Expected Competition Performance
CVC3 is being entered as an experimental joint venture of the SMT and ATP community. We expect its performance to be somewhere in the middle of the pack as we have made no specific effort to tune it for CASC. We hope that its performance will shed light on areas where SMT solvers are strong as opposed to ATP systems.

7.2 E(P/LTB) 1.4pre

Stephan Schulz
Technische Universität München, Germany

Architecture
E 1.4pre [83, 85] is described in this section. E is a purely equational theorem prover for full first-order logic with equality. It consists of an (optional) clausifier for pre-processing full first-order formulae into clausal from, and a saturation algorithm implementing an instance of the superposition calculus with negative literal selection and a number of redundancy elimination techniques. E is based on the DISCOUNT-loop variant of the *given-clause* algorithm, i.e., a strict separation of active and passive facts. No special rules for non-equational literals have been implemented. Resolution is effectively simulated by paramodulation and equality resolution.

EP 1.4pre is just a combination of E 1.4pre in verbose mode and a proof analysis tool extracting the used inference steps. For the LTB division, a control program uses a SInE-like analysis to extract reduced axiomatizations that are handed to several instances of E.

Strategies
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 pre-programmed literal selection strategies. Clause evaluation heuristics can be constructed on the fly by combining various parametrized primitive evaluation functions, or can be selected from a set of predefined heuristics. Clause evaluation heuristics are based on symbol-counting, but also take other clause properties into account. In particular, the search can prefer clauses from the set of support, or containing many symbols also present in the goal. Supported term orderings are several parametrized instances of Knuth-Bendix-Ordering (KBO) and Lexicographic Path Ordering (LPO).

The automatic mode is based on a static partition of the set of all clausal problems based on a number of simple features (number of clauses, maximal symbol arity, presence of equality, presence of non-unit and non-Horn clauses,...). Each class of clauses is automatically assigned a heuristic that performs well on problems from this class in test runs. About 100 different strategies have been evaluated on all untyped first-order problems from TPTP 4.1.0.

Implementation
E is build around perfectly shared terms, i.e. each distinct term is only represented once in a term bank. The whole set of terms thus consists of a number of interconnected directed acyclic graphs. Term memory is managed by a simple mark-and-sweep garbage collector. Unconditional (forward) rewriting using unit clauses is implemented using perfect discrimination trees with size and age constraints. Whenever a possible simplification is detected, it is added as a rewrite link

in the term bank. As a result, not only terms, but also rewrite steps are shared. Subsumption and contextual literal cutting (also known as subsumption resolution) is supported using feature vector indexing [84]. Superposition and backward rewriting use fingerprint indexing, a new technique combining ideas from feature vector indexing and path indexing. Finally, LPO and KBO are implemented using the elegant and efficient algorithms developed by Bernd Löchner in [57, 58]. The prover and additional information are available at

`http://www.eprover.org`

Expected Competition Performance
E 1.4pre is relatively little changed from last years entry. The system is expected to perform well in most proof classes, but will at best complement top systems in the disproof classes.

7.3 E-Darwin 1.4

Björn Pelzer
University Koblenz-Landau, Germany

Architecture
E-Darwin 1.4 [11, 14] is an automated theorem prover for first order clausal logic with equality. It is a modified version of the Darwin prover [11], intended as a testbed for variants of the Model Evolution calculus [15]. Among other things it implements several different approaches [16, 14] to incorporating equality reasoning into Model Evolution. Three principal data structures are used: the context (a set of rewrite literals), the set of constrained clauses, and the set of derived candidates. The prover always selects one candidate, which may be a new clause or a new context literal, and exhaustively computes inferences with this candidate and the context and clause set, moving the results to the candidate set. Afterwards the candidate is inserted into one of the context or the clause set, respectively, and the next candidate is selected. The inferences are superposition-based. Demodulation and various means of redundancy detection are used as well.

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

Implementation
E-Darwin is implemented in the functional/imperative language OCaml. Darwin's method of storing partial unifiers has been adapted to equations and subterm positions for the superposition inferences in E-Darwin. A combination of perfect and non-perfect discrimination tree indexes is used to store the context and the clauses. 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
There have been some calculus changes since last year, but the performance should remain similar. While the original Darwin performs strongly in EPR, E-Darwin is more of a generalist, less effective in EPR, yet stronger in the other divisions.

7.4 E-KRHyper 1.2

Björn Pelzer
University Koblenz-Landau, Germany

Architecture
E-KRHyper [79] is a theorem proving and model generation system for first-order logic with equality. It is an implementation of the E-hyper tableau calculus [13], which integrates a superposition-based handling of equality [8] into the hyper tableau calculus [12]. The system is an extension of the KRHyper theorem prover [128], 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. Splitting is done without rigid variables. Instead, variables which would be shared between branches are prevented by ground substitutions, which are guessed from the Herbrand universe and constrained by rewrite rules. 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 [124] 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.

In the LTB division E-KRHyper sequentially tries three axiom selection strategies: an implementation of Krystof Hoder's SInE algorithm, another incomplete selection based on the CNF representations of the axioms, and finally the complete axiom set.

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 similar to the one used by Otter [61], with some additional connector literals to prevent explosive clause growth when dealing with DNF-like structures. 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
There have been minor improvements since the last version, but overall E-KRHyper will remain in the middle ground.

7.5 E-MaLeS 1.0

Daniel Kuehlwein[1], Josef Urban[1], Stephan Schulz[2]
[1]Radboud Universiteit Nijmegen, The Netherlands, [2]Technische Universität München, Germany

Architecture
E-MaLeS is a meta system for E 1.3. It uses kernel methods to learn which of E's strategies are most likely to solve a problem. Furthermore E-MaLeS runs several strategies for a shorter time instead of one strategy for the whole time.

Strategies
Since E-MaLeS is based on E, please refer to E's description for its internal procedures. The performance of E's strategies was evaluated over the TPTP problems. Each problem was characterised by E's problem features. Then kernel methods were used to learn which strategy is most likely to solve a problem given its features.

Implementation
E-MaLeS is implemented in Python using the Numpy/Scipy library.

Expected Competition Performance
Since E-MaLeS is based on E we expect it to perform at least as good as E.

7.6 FIMO 0.2

Orkunt Sabuncu
University of Potsdam, Germany

Architecture
FIMO is a system for computing finite models of first-order formulas by incremental Answer Set Programming (iASP). The input theory is transformed to an incremental logic program. If any, answer sets of this program represent finite models of the input theory. iClingo is used for computing answer sets of iASP programs.

Strategies
FIMO is the successor of the system fmc2iasp [44]. Unlike fmc2iasp, FIMO does not rely on flattening for translating the input theory to a satisfiablity problem. FIMO features symmetry breaking and incremental answer set solving provided by the underlying iASP system iClingo.

Implementation
FIMO is developed in Python. Being the successor of fmc2iasp, it will be available from

```
http://potassco.sourceforge.net
```

Expected Competition Performance
fmc2iasp performed well against Paradox in FNT division of 2009 (not in competition but published in [44]). However, FIMO is based on a different strategy.

7.7 H2WO4 11.07

David Stanovsky
Charles University in Prague, Czech Republic

Architecture
Tungstic acid is a substance reacting with problems in the first order logic wit h special arithmetical functions (as defined in the TPTP library), producing a code in Wolfram's Mathematica that attempts to find a solution. The first version, H_2WO_4 11.07, is a simple script transla ting between the two languages and calling built-in functions of Mathematica to solve the problem.

Strategies
The current version is using various combinations of three functions:

- the `Reduce` and `FullSimplify` functions, for simplifying expressions - a problem is solved if simplified to `True` or `False`
- the `FindInstance` function, for solving systems of equations over specified domains - an existential problem is solved if a solution is found or if no solutions exists

Implementation
This is a pair of Perl scripts: one for parsing TPTP problems into Mathematica, the other for processing Mathematica's output. The scripts are available at

`http://www.karlin.mff.cuni.cz/~stanovsk/h2wo4/`

They were tested with the text-based interface of Mathematica 7.0.

Expected Competition Performance
My motivation is, to compare the other entrants with a commercial, state-of-the- art computer algebra system (Mathematica, in my case). I wish it did not do well, compared to specially developed systems :-) Mathematica is strong in pure arithmetic and weak in logical reasoning. It does well on the last year set of problems in TPTP.

7.8 iProver 0.8

Konstantin Korovin
University of Manchester, United Kingdom

Architecture
iProver is an automated theorem prover based on an instantiation calculus Inst-Gen [42, 53] 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 [41] 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 [52] 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 [43, 52]; global subsumption [52]; 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. Major additions in the current version are:

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

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

Implementation
iProver is implemented in OCaml and for the ground reasoning uses MiniSat. iProver accepts FOF and CNF formats, where either Vampire [82] or E prover [85] is used for clausification of FOF problems. iProver is available from:

`http://www.cs.man.ac.uk/~korovink/iprover/`

Expected Competition Performance
iProver 0.8 is the CASC-J5 EPR division winner.

7.9 iProver(-SInE) 0.9

Konstantin Korovin
The University of Manchester, United Kingdom

Architecture
iProver is an automated theorem prover based on an instantiation calculus Inst-Gen [42, 53] 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 [41] 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 [52] for the implementation details. The saturation process is implemented as a modification of a given clause algorithm. iProver uses 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 [43, 52]; global subsumption [52]; resolution-based simplifications and propositional-based simplifications. A compressed feature vector index is used for efficient forward/backward subsumption and subsumption resolution. Equality is dealt with (internally) by adding the necessary axioms of equality with an option of using Brand's transformation. In the LTB division, iProver-SInE uses axiom selection based on the SInE algorithm [49] as implemented in Vampire [48], i.e., axiom selection is done by Vampire and proof attempts are done by iProver. Major additions in the current version are:

- answer computation,
- several modes for model output using first-order definitions in term algebra,
- Brand's transformation.

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

Implementation
iProver is implemented in OCaml and for the ground reasoning uses MiniSat. iProver accepts FOF and CNF formats, where Vampire [48] is used for clausification of FOF problems.

iProver is available from:

`http://www.cs.man.ac.uk/~korovink/iprover/`

Expected Competition Performance
iProver 0.9 is expected to perform slightly better than the previous version.

7.10 iProver-Eq(-SInE) 0.7

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

Architecture
iProver-Eq [54] extends the iProver system [52] with built-in equational reasoning, along the lines of [43]. As in the iProver system, first-order reasoning is combined with ground satisfiability checking where the latter is delegated to an off-the-shelf ground solver.

iProver-Eq consists of three core components: i) ground reasoning by an SMT solver, ii) first-order equational reasoning on literals in a candidate model by a labelled unit superposition calculus [54, 55] and iii) instantiation of clauses with substitutions obtained by ii). Given a set of first-order clauses, iProver-Eq first abstracts it to a set of ground clauses which are then passed to the ground solver. If the ground abstraction is unsatisfiable, then the set of first-order clauses is also unsatisfiable. Otherwise, literals are selected from the first-order clauses based on the model of the ground solver. The labelled unit superposition calculus checks whether selected literals are conflicting. If they are conflicting, then clauses are instantiated such that the ground solver has to refine its model in order to resolve the conflict. Otherwise, satisfiability of the initial first-order clause set is shown.

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

For the LTB division, iProver-Eq-SInE uses axiom selection based on the SInE algorithm [HV11] as implemented in Vampire [HKV10], i.e., axiom selection is done by Vampire and proof attempts are done by iProver-Eq.

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

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

Implementation
iProver-Eq is implemented in OCaml and uses CVC3 [10] for the ground reasoning in the equational case and MiniSat [41] in the non-equational case. iProver-Eq accepts FOF and CNF formats, where Vampire [48] is used for clausification of FOF problems. iProver-Eq is available at

`http://www.cs.man.ac.uk/~sticksec/iprover-eq`

Expected Competition Performance
iProver-Eq has seen many optimisations from the version in the previous CASCs. We expect reasonably good performance in all divisions, including the EPR divisions where instantiation-based methods are particularly strong.

7.11 Isabelle/HOL (a.k.a. IsabelleP) 2011

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

Architecture
Isabelle/HOL 2011 [65] is the higher-order logic incarnation of the generic proof assistant Isabelle2011. Isabelle/HOL provides several automatic proof tactics, notably an equational reasoner [64], a classical reasoner [78], a tableau prover [76], and a first-order resolution-based prover [50].

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

TPTP2X's Isabelle format output, and then run the command line isabelle-process on the resulting theory file.

Strategies
The *IsabelleP* tactic 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 [59]. Constructs proofs directly in Isabelle.
- smt – Invokes the Z3 SMT solver [39] developed at Microsoft Research and optionally reconstructs the proofs in Isabelle [30].
- sledgehammer – Invokes Sledgehammer as an oracle with the sound fully typed translation [77].

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 datatype. Assuming the kernel is correct, all values of type theorem are correct by construction.

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

Isabelle/HOL is available for all major platforms under a BSD-style license from

http://www.cl.cam.ac.uk/research/hvg/Isabelle

Expected Competition Performance
Results from last year would suggest that Isabelle will finish third in the THF category, after Satallax and LEO-II. However, since last year, we have added the sledgehammer proof methods, which we expect will improve our chances.

7.12 leanCoP 2.2

Jens Otten
University of Potsdam, Germany

Architecture
leanCoP [68, 66] is an automated theorem prover for classical first-order logic with equality. It is a very compact implementation of the connection (tableau) calculus [24, 56].

Strategies
The reduction rule of the connection calculus is applied before the extension rule. Open branches are selected in a depth-first way. Iterative deepening on the proof depth is used to achieve completeness. Additional inference rules and strategies include regularity, lemmata, and restricted backtracking [67]. leanCoP uses an optimized structure-preserving transformation into clausal form [67] and a fixed strategy scheduling.

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

leanCoP can read formulae using the leanCoP syntax as well as the (raw) TPTP syntax format. Equality axioms are automatically added if required. The core leanCoP prover returns a very compact connection proof, which is translated into a readable proof. Several output formats are available.

As the main enhancement leanCoP 2.2 now supports the output of proofs in an unofficial TPTP syntax format for representing derivations in connection (tableau) calculi [69]. Furthermore, besides the Linux/Unix and MacOS platforms, most Windows platforms are now supported as well.

The source code of leanCoP 2.2 is available under the GNU general public license. Together with more information it can be found on the leanCoP website at

`http://www.leancop.de`

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

7.13 LEO-II 1.2

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

Architecture
LEO-II [21], the successor of LEO [20], is a higher-order ATP system based on extensional higher-order resolution. More precisely, LEO-II employs a refinement of extensional higher-order RUE resolution [18]. LEO-II is designed to cooperate with specialist systems for fragments of higher-order logic. By default, LEO-II cooperates with the first-order ATP systems E [83]. LEO-II is often too weak to find a refutation amongst the steadily growing set of clauses on its own.

However, some of the clauses in LEO-II's search space attain a special status: they are first-order clauses modulo the application of an appropriate transformation function. The default transformation is Hurd's fully typed translation [50]. Therefore, LEO-II launches a cooperating first-order ATP system every n iterations of its (standard) resolution proof search loop (e.g., n = 10). If the first-order ATP system finds a refutation, it communicates its success to LEO-II in the standard SZS format. Communication between LEO-II and the cooperating first-order ATP system uses the TPTP language and standards.

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

Implementation
LEO-II is implemented in Objective Caml version 3.10, and its problem representation language is the new TPTP THF language [22]. In fact, the development of LEO-II has largely paralleled the development of the TPTP THF language and related infrastructure [110].

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 [19], support for primitive equality reasoning (extensional higher-order RUE resolution), and improved heuristics at the calculus level. One recent development is LEO-II's new parser: in addition to the TPTP THF language, this parser now also supports the TPTP FOF and CNF languages. Hence, LEO-II can now also be used for FOF and CNF problems. Unfortunately the LEO-II system still uses only a very simple sequential collaboration model with first-order ATPs instead of using the more advanced, concurrent and resource-adaptive OANTS architecture [23] as exploited by its predecessor LEO.

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

`http://leoprover.org`

Expected Competition Performance
LEO-II 1.2 is the CASC-J5 THF division winner.

7.14 LEO-II 1.2.8

Christoph Benzmüller[1], Frank Theiss[2]
[1]Freie Universität Berlin, Germany, [2]Saarland University, Germany

Architecture
LEO-II [21], the successor of LEO [20], is a higher-order ATP system based on extensional higher-order resolution. More precisely, LEO-II employs a refinement of extensional higher-order RUE resolution [18]. LEO-II is designed to cooperate with specialist systems for fragments of higher-order logic. By default, LEO-II cooperates with the first-order ATP systems E [83]. LEO-II is often too weak to find a refutation amongst the steadily growing set of clauses on its own. However, some of the clauses in LEO-II's search space attain a special status: they are first-order clauses modulo the application of an appropriate transformation function. The default

transformation is Hurd's fully typed translation [50]. Therefore, LEO-II launches a cooperating first-order ATP system every n iterations of its (standard) resolution proof search loop (e.g., 10). If the first-order ATP system finds a refutation, it communicates its success to LEO-II in the standard SZS format. Communication between LEO-II and the cooperating first-order ATP system uses the TPTP language and standards.

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

Implementation
LEO-II is implemented in Objective Caml version 3.12, and its problem representation language is the TPTP THF language [22]. In fact, the development of LEO-II has largely paralleled the development of the TPTP THF language and related infrastructure [110].

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 [19], support for primitive equality reasoning (extensional higher-order RUE resolution), and improved heuristics at the calculus level. LEO-II's parser supports the TPTP THF0 language and also the TPTP languages FOF and CNF. Unfortunately the LEO-II system still uses only a very simple sequential collaboration model with first-order ATPs instead of using the more advanced, concurrent and resource-adaptive OANTS architecture [23] as exploited by its predecessor LEO.

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

`http://leoprover.org`

Expected Competition Performance
LEO-II has not improved much over the last year. The main modifications concern proof output in order to enable proof reconstruction/verification of LEO-II proofs in the Isabelle system. I doubt that LEO-II will be able to defend its championship at this years CASC since some of its competitor systems, such as Satallax, have significantly changed over the last year. However, it is great to have such a strong dynamics in the THF category.

LEO-II will again participate in the FOF and CNF categories in order to evaluate its performance for these fragments. For this, note that LEO-II still employs its own input processing and normalization techniques, and that calls to prover E are applied only modulo Hurd's fully typed translation.

7.15 MELIA 0.1

Peter Baumgartner
NICTA and ANU, Australia

Architecture
MELIA is a theorem prover for the Model Evolution Calculus with Equality and Linear Integer Arithmetic [17, 14]. It also integrates most of the theoretical developments of the Model Evolution calculus, in particular superposition-like inference rules for equality handling and built-in inference rules for linear integer arithmetic.

MELIA accepts formulas in the TFF format (typed TPTP formulas, see the TPTP technical report). It includes a pre-processor for transforming such formulas into (sorted) clausal logic over foreground sorts specified in the input files and built-in linear integer arithmetic.

Strategies
MELIA features a variety of flag settings to control its search, e.g., for selecting literals to focus inferences on. In the competition, MELIA uses the same search strategy for all problems. It includes a heuristics to select the next literal to split on and to select the next superposition inference.

Implementation
MELIA has been written (from scratch) in Scala, and runs on the Java virtual machine.

Expected Competition Performance
MELIA's participates in the TFF division only. It is in a very early stage and has not been tuned for performance (or for the competition). It lacks term indexing techniques to make it more efficient. When integer arithmetic is involved, MELIA is incomplete even if (theoretically) unneccessary, and it is likely to miss proving some theorems because of that.

Expectations are not high.

7.16 Metis 2.3

Joe Hurd
Galois Inc., USA

Architecture
Metis 2.3 [50] 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 [50] have shown a *given clause algorithm* and ordered resolution to best suit this application, and that is what Metis 2.3 implements. Since equality often appears in interactive theorem prover goals, Metis 2.3 also implements the ordered paramodulation calculus.

Strategies
Metis 2.3 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 2.3 is written in Standard ML, for ease of integration with HOL4. It uses indexes for resolution, paramodulation, (forward) subsumption and demodulation. It keeps the *Active* clause set reduced with respect to all the unit equalities so far derived.

In addition to standard age and size measures, Metis 2.3 uses finite models to weight clauses in the *Passive* set. When integrated with higher order logic, an interpretation of known functions and relations is manually constructed to make many of their standard properties valid in the finite model. For example, the domain of the model is the set *0,...,7*, and the higher order logic

arithmetic functions are interpreted in the model modulo *8*. Unknown functions and relations are interpreted randomly, but with a bias towards making supporting theorems valid in the model. The finite model strategy carries over to TPTP problems, by manually interpreting a collection of functions and relations that appear in TPTP axiom files in such a way as to make the axioms valid in the model.

Metis 2.3 reads problems in TPTP format and outputs detailed proofs in TSTP format, where each refutation step is one of 6 simple inference rules. Metis 2.3 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 2.3 is free software, released under the MIT license. It can be downloaded from

`http://www.gilith.com/software/metis`

Expected Competition Performance
There have been only minor changes to Metis 2.3 since CASC J5, so it is expected to perform at approximately the same level in CASC 23 and end up in the lower third of the table.

7.17 MetiTarski 1.8

Lawrence C. Paulson
University of Cambridge, United Kingdom

Architecture
MetiTarski [1, 40] is an automatic theorem prover based on a combination of resolution and QEPCAD-B [32], a decision procedure for the theory of real closed fields. It is designed to prove theorems involving real-valued special functions such as log, exp, sin, cos, atan and sqrt. In particular, it is designed to prove universally quantified inequalities involving such functions. Support for existentially quantified inequalities is very limited. MetiTarski is a modified version of Joe Hurd's theorem prover, Metis [50].

Strategies
MetiTarski employs resolution, augmented with axiom files that specify upper and lower bounds of the special functions mentioned in the problem. MetiTarski also has code to simplify polynomials and put them into canonical form. The resolution calculus is extended with a literal deletion rule: if the decision procedure finds a literal to be inconsistent with its context (which consists of known facts and the negation of the other literals in the clause), then it is deleted. From 2011, MetiTarski also implements case-splitting with backtracking. MetiTarski is incomplete, and nothing can be inferred if it fails to prove a conjecture.

Implementation
MetiTarski, like Metis, is implemented in Standard ML. QEPCAD is implemented in C and C++. The latest version of MetiTarski can be downloaded from

`http://www.cl.cam.ac.uk/~lp15/papers/Arith/`

Expected Competition Performance
No expectation provided.

7.18 Muscadet 4.1

Dominique Pastre
University Paris Descartes, France

Architecture
The Muscadet theorem prover is a knowledge-based system. It is based on Natural Deduction, following the terminology of [29] and [70], and uses methods which resembles those used by humans. It is composed of an inference engine, which interprets and executes rules, and of one or several bases of facts, which are the internal representation of "theorems to be proved". Rules are either universal and put into the system, or built by the system itself by metarules from data (definitions and lemmas). Rules may add new hypotheses, modify the conclusion, create objects, split theorems into two or more subtheorems or build new rules which are local for a (sub-)theorem.

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

The system is also able to work with second order statements. It may also receive knowledge and know-how for a specific domain from a human user; see [71] and [72]. These two possibilities are not used while working with the TPTP Library.

Implementation
Muscadet [73] is implemented in SWI-Prolog. Rules are written as more or less declarative Prolog clauses. Metarules are written as sets of Prolog clauses. The inference engine includes the Prolog interpreter and some procedural Prolog clauses. A theorem may be split into several subtheorems, structured as a tree with "and" and "or" nodes. All the proof search steps are memorized as facts including all the elements which will be necessary to extract later the useful steps (the name of the executed action or applied rule, the new facts added or rule dynamically built, the antecedents and a brief explanation).

Muscadet is available from:

`http://www.math-info.univ-paris5.fr/~pastre/muscadet/muscadet.html`

Expected Competition Performance
The best performances of Muscadet will be for problems manipulating many concepts in which all statements (conjectures, definitions, axioms) are expressed in a manner similar to the practice of humans, especially of mathematicians [74, 75]. It will have poor performances for problems using few concepts but large and deep formulas leading to many splittings. Its best results will be in set theory, especially for functions and relations. It's originality is that proofs are given in natural style.

7.19 Nitpick (a.k.a. IsabelleN) 2011

Jasmin C. Blanchette
Technische Universität München, Germany

Architecture
Nitpick [28] is an open source counterexample generator for Isabelle/HOL [65]. It builds on Kodkod [123], a highly optimized first-order relational model finder based on SAT. The name Nitpick is appropriated from a now retired Alloy precursor.

Strategies
Nitpick employs Kodkod to find a finite model of the negated conjecture. The translation from HOL to Kodkod's first-order relational logic (FORL) is parameterized by the cardinalities of the atomic types occurring in it. Nitpick enumerates the possible cardinalities for each atomic type, exploiting monotonicity to prune the search space [27]. If a formula has a finite counterexample, the tool eventually finds it, unless it runs out of resources.

SAT solvers are particularly sensitive to the encoding of problems, so special care is needed when translating HOL formulas. As a rule, HOL scalars are mapped to FORL singletons and functions are mapped to FORL relations accompanied by a constraint.

An n-ary first-order function (curried or not) can be coded as an $(n + 1)$-ary relation accompanied by a constraint. However, if the return type is the type of Booleans, the function is more efficiently coded as an unconstrained n-ary relation.

Higher-order quantification and functions bring complications of their own. A function from σ to τ cannot be directly passed as an argument in FORL; Nitpick's workaround is to pass $|\sigma|$ arguments of type τ that encode a function table.

Implementation
Nitpick, like most of Isabelle/HOL, is written in Standard ML. Unlike Isabelle itself, which adheres to the LCF small-kernel discipline, Nitpick does not certify its results and must be trusted.

Nitpick is available as part of Isabelle/HOL for all major platforms under a BSD-style license from

`http://www.cl.cam.ac.uk/research/hvg/Isabelle`

Expected Competition Performance
Thanks to Kodkod's amazing power, we expect that Nitpick will beat both Satallax and Refute with its hands tied behind its back in the TNT category.

7.20 Nitrox 0.2

Jasmin C. Blanchette[1], Emina Torlak[2]
[1]Technische Universität München, Germany
[2]IBM Research, USA

Architecture
Nitrox is the first-order version of Nitpick [28], an an open source counterexample generator for Isabelle/HOL [65]. It builds on Kodkod [123], a highly optimized first-order relational model finder based on SAT. The name Nitrox is a portmanteau of ***Nit***pick and Pa***r***ad***ox*** (clever, eh?).

Strategies
Nitrox employs Kodkod to find a finite model of the negated conjecture. It performs a few transformations on the input, such as pushing quantifiers inside, but 99

The translation from HOL to Kodkod's first-order relational logic (FORL) is parameterized by the cardinalities of the atomic types occurring in it. Nitrox enumerates the possible cardinalities for the universe. If a formula has a finite counterexample, the tool eventually finds it, unless it runs out of resources.

Nitpick is optimized to work with higher-order logic (HOL) and its definitional principles (e.g., (co)inductive predicates, (co)inductive datatypes, (co)recursive functions). When invoked on untyped first-order problem, few of its optimizations come into play, and the problem handed to Kodkod is essentially a first-order relational logic (FORL) rendering of the TPTP FOF problem. One exception is nested quantifiers, which Nitpick optimizes before Kodkod gets a chance to look at them [28].

Implementation
Nitrox, like most of Isabelle/HOL, is written in Standard ML. Unlike Isabelle itself, which adheres to the LCF small-kernel discipline, Nitrox does not certify its results and must be trusted. Kodkod is written in Java. MiniSat 1.14 is used as the SAT solver.

Expected Competition Performance
Since Nitpick was designed for HOL, it doesn't have any type inference à la Paradox. It also doesn't use the SAT solver incrementally, which penalizes it a bit (but not as much as the missing type inference). Kodkod itself is known to perform less well on FOF than Paradox, because it is designed and optimized for a somewhat different logic, FORL. On the other hand, Kodkod's symmetry breaking seems better calibrated than Paradox's. Hence, we expect Nitrox to end up in second place at best in the TNF category.

7.21 Otter 3.3

William McCune
Argonne National Laboratory, USA

Architecture
Otter 3.3 [61] is an ATP system for statements in first-order (unsorted) logic with equality. Otter is based on resolution and paramodulation applied to clauses. An Otter search uses the

"given clause algorithm", and typically involves a large database of clauses; subsumption and demodulation play an important role.

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

Implementation
Otter is written in C. Otter uses shared data structures for clauses and terms, and it uses indexing for resolution, paramodulation, forward and backward subsumption, forward and backward demodulation, and unit conflict. Otter is available from:

`http://www.cs.unm.edu/~mccune/otter/`

Expected Competition Performance
Otter has been entered into CASC as a stable benchmark against which progress can be judged (there have been only minor changes to Otter since 1996 [62], nothing that really affects its performance in CASC). This is not an ordinary entry, and we do not hope for Otter to do well in the competition.

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

7.22 Paradox 3.0

Koen Claessen, Niklas Sörensson
Chalmers University of Technology, Sweden

Architecture
Paradox [38] is a finite-domain model generator. It is based on a MACE-style [60] 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-J5 FNT division winner.

7.23 Refute (a.k.a. IsabelleM) 2011

Jasmin C. Blanchette[1], Tjark Weber[2]
[1]Technische Universität München, Germany
[2]University of Cambridge, United Kingdom

Architecture
Refute [126] is an open source counterexample generator for Isabelle/HOL [65] based on a SAT solver, and Nitpick's [28] precursor.

Strategies
Refute employs a SAT solver to find a finite model of the negated conjecture. The translation from HOL to propositional logic is parameterized by the cardinalities of the atomic types occurring in the conjecture. Refute enumerates the possible cardinalities for each atomic type. If a formula has a finite counterexample, the tool eventually finds it, unless it runs out of resources.

Implementation
Refute, like most of Isabelle/HOL, is written in Standard ML. Unlike Isabelle itself, which adheres to the LCF small-kernel discipline, Refute does not certify its results and must be trusted.

Refute is available as part of Isabelle/HOL for all major platforms under a BSD-style license from

`http://www.cl.cam.ac.uk/research/hvg/Isabelle`

Expected Competition Performance
We expect that Refute will solve about 75the TNT category, and perhaps a few problems that Nitpick cannot solve.

7.24 Satallax 2.1

Chad E. Brown
Saarland University, Germany

Architecture
Satallax [34] is an automated theorem prover for higher-order logic. The particular form of higher-order logic supported by Satallax is Church's simple type theory with extensionality and choice operators. The SAT solver MiniSat [41] is responsible for much of the search for a proof.

The theoretical basis of search is a complete ground tableau calculus for higher-order logic [37] with a choice operator [9]. A problem is given in the THF format. A branch is formed from the axioms of the problem and the negation of the conjecture (if any is given). From this point on, Satallax tries to determine unsatisfiability or satisfiability of this branch.

Satallax progressively generates higher-order formulae and corresponding propositional clauses [34]. These formulae and propositional clauses correspond to instances of the tableau rules. Satallax uses the SAT solver MiniSat as an engine to test the current set of propositional clauses for unsatisfiability. If the clauses are unsatisfiable, then the original branch is unsatisfiable. If there are no quantifiers at function types, the generation of higher-order formulae and corresponding clauses may terminate [36, 35]. In such a case, if MiniSat reports the final set of clauses as satisfiable, then the original set of higher-order formulae is satisfiable (by a standard model in which all types are interpreted as finite sets).

Strategies

There are a number of flags that control the order in which formulas and instantiation terms are considered and propositional clauses are generated. Other flags activate some optional extensions to the basic proof procedure. A collection of flag settings is called a mode. Approximately 250 modes have been tried so far. Regardless of the mode, the search procedure is sound and complete for higher-order logic with choice. This implies that if search terminates with a particular mode, then we can conclude that the original set of formulae is unsatisfiable or satisfiable.

A strategy schedule is an ordered collection of modes with information about how much time the mode should be allotted. Satallax tries each of the modes for a certain amount of time sequentially. Satallax 2.1 has eight strategy schedules which were determined through experimentation using the THF problems in version 5.1.0 of the TPTP library. One of these eight strategy schedules is chosen based on the amount of time Satallax is given to solve the problem. For example, if Satallax is given 180 seconds to solve the problem, then a schedule with 38 modes is chosen.

Implementation

Satallax 2.1 is implemented in OCaml. A foreign function interface is used to interact with MiniSat. Satallax is available from

`http://satallax.com`

Expected Competition Performance

Satallax 1.4 proved to be competitive in the THF division of CASC last year, coming in second out of four systems. Since last year, Satallax has been reimplemented in OCaml (instead of Lisp) and the integration with MiniSat has been improved. In addition, a number of new heuristics and flags to control these heuristics have been added. Based on these improvements, Satallax is expected to perform well in the THF division of CASC this year. On the other hand, Satallax is typically weak on problems requiring equality reasoning or nontrivial higher-order instantiations. Since Satallax can sometimes be used to determine satisfiability of a set of formulas, it will also compete in the TNT demonstration division.

7.25 SPASS+T—2.2.14

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

Architecture
SPASS+T is an extension of the superposition-based theorem prover SPASS that integrates algebraic knowledge into SPASS in three complementary ways: by passing derived formulas to an external SMT procedure (currently Yices or CVC3), by adding standard axioms, and by built-in arithmetic simplification and inference rules. A first version of the system has been described in [81]. In the current version, a much more sophisticated coupling of the SMT procedure has been added [129].

Strategies
Standard axioms and built-in arithmetic simplification and inference rules are integrated into the standard main loop of SPASS. Inferences between standard axioms are excluded, so the user-supplied formulas are taken as set of support. The external SMT procedure runs in parallel in a separate process, leading occasionally to non-deterministic behaviour.

Implementation
SPASS+T is implemented in C. The system is available from

`http://www.mpi-inf.mpg.de/~uwe/software/#TSPASS`

Expected Competition Performance
SPASS+T came a close second in the TFA division of last CASC, and is has improved noticeably since then.

7.26 SPASS-XDB 3.01X0.6

Geoff Sutcliffe[1], Martin Suda[2,3]
[1]University of Miami, USA,
[2]Max-Planck-Institut für Informatik and Saarland University, Germany,
[3]Charles University in Prague, Czech Republic

Architecture
SPASS-XDB [86, 112] is an extended version of the well-known, state-of-the-art, SPASS automated theorem proving system [127]. The original SPASS reads a problem, consisting of axioms and a conjecture, in TPTP format from a file, and searches for a proof by refutation of the negated conjecture. SPASS-XDB adds the capability of retrieving extra positive unit axioms (facts) from external sources during the proof search (hence the "XDB", standing for eXternal DataBases). The axioms are retrieved asynchronously, on-demand, based on an expectation that they will contribute to completing the proof. The axioms are retrieved from a range of external sources, including SQL databases, SPARQL endpoints, WWW services, computation sources (e.g., computer algebra systems), etc., using a TPTP standard protocol.

For the TFA division, the TFF formulae are converted to FOF using the standard approach [125], with type predicates to check the types of numeric variables. Numbers are represented

internally as special constant symbols that carry both the type and the value. The basic mathematical functionality is provided by a new inference rule called ground arithmetic rewriting. Given a clause as a premise, it traverses the term structure of all its literals in a bottom up fashion, and performs the symbolically represented arithmetic operations whenever all arguments are numbers (thus even terms below uninterpreted symbols may get simplified). Arithmetic predicates (including equality) may get evaluated, which simplifies the clause, or might show its redundancy. The GMP arithmetic library is used for arbitrary precision computation.

To extend the mathematical capabilities beyond ground arithmetic rewriting, Mathematica is used as an external source of axioms. SPASS-XDB generates requests from negative arithmetic literals, taking advantage of the fact that Mathematica understands all arithmetic and logical symbols. When SPASS-XDB selects a negative literal that contains arithmetic symbols, all negative literals in the clause are scanned to check whether they can be conjoined with the selected literal into a single request. The S2M2S mediator translates the request into the Mathematica language, and calls the `FindInstance` function of Mathematica. `FindInstance` inputs an expression and a set of variables that occur in the expression, and finds instances of the variables that make the expression true. Axioms that are instances of the first literal in the request are returned. The mediator reads the answer, and creates new axioms that are passed back to SPASS-XDB.

Strategies
Generally, SPASS-XDB follows SPASS' strategies. However, SPASS, like most (all?) ATP systems, was designed under the assumption that all formulae are in the problem file, i.e., it is ignorant that external axioms might be delivered. To regain completeness, constraints on SPASS' search are relaxed in SPASS-XDB. This increases the search space, so the constraints are relaxed in a controlled, incremental fashion [112]. The search space is also affected by the number of external axioms that can be delivered, and mechanisms to control the delivery and focus the consequent search are used [112].

Implementation
SPASS-XDB, as an extension of SPASS, is written in C. The internal arithmetic is done using the GMP multiple precision arithmetic library. Reals are converted to rationals for computation, but results are presented in real format. SPASS-XDB is available for use online in the SystemOnTPTP interface:

`http://www.tptp.org/cgi-bin/SystemOnTPTP`

Expected Competition Performance
SPASS-XDB should do well, particularly on problems that use uncommon parts of the TPTP TFA syntax, e.g., the `evaleq` predicate.

7.27 TPS 3.110228S1a

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

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, Mark Kaminski, Rémy Chrétien and Cris Perdue. TPS can be used to prove theorems of Church's type theory automatically, interactively, or semi-automatically [4, 5].

When searching for a proof, TPS first searches for an expansion proof [63] or an extensional expansion proof [33] of the theorem. Part of this process involves searching for acceptable matings [2]. Using higher-order unification, a pair of occurrences of subformulae (which are usually literals) is mated appropriately on each vertical path through an expanded form of the theorem to be proved. The expansion proof thus obtained is then translated [80] without further search into a proof of the theorem in natural deduction style.

Strategies
Strategies used by TPS in the search process include:

- Re-ordering conjunctions and disjunctions to alter the way paths through the formula are enumerated.
- The use of primitive substitutions and gensubs [3].
- Path-focused duplication [51].
- Dual instantiation of definitions, and generating substitutions for higher-order variables which contain abbreviations already present in the theorem to be proved [26].
- Component search [25].
- Generating and solving set constraints [31].
- Generating connections using extensional and equational reasoning [33].

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. When searching for a proof during the competition, TPS tries each of 80 selected modes in turn for a specified amount of time. If TPS succeeds in finding an expansion proof, it translates the expansion proof to a natural deduction proof. This final step ensures that TPS will not incorrectly report that a formula has been proven.

TPS is implemented in Common Lisp, and is available from

`http://gtps.math.cmu.edu/tps.html`

Expected Competition Performance
TPS 3.080227G1d was the CASC-22 THF division winner. The main difference between the older version and the newer version is that more modes are included during the search and the final translation to natural deduction has been enabled.

7.28 Vampire 1.8

Krystof Hoder, Andrei Voronkov
The University of Manchester, United Kingdom

Architecture
Vampire 1.8 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 and a SAT solver. A number of standard redundancy criteria and simplification techniques are used for pruning the search space: subsumption, tautology deletion, subsumption resolution and rewriting by ordered unit equalities. The reduction ordering is the Knuth-Bendix Ordering.

Substitution tree and code tree indexes are used to implement all major operations on sets of terms, literals and clauses. Although the kernel of the system works only with clausal normal form, 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 Sine [49] 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.

Implementation
Vampire 1.8 is implemented in C++.

Strategies
The Vampire 1.8 kernel provides a fairly large number of options for strategy selection. The most important ones are:

1. Choice of the main procedure:
   - Limited Resource Strategy
   - DISCOUNT loop
   - Otter loop
   - Goal oriented mode based on tabulation
2. A variety of optional simplifications.
3. Parameterized reduction orderings.
4. A number of built-in literal selection functions and different modes of comparing literals.
5. Age-weight ratio that specifies how strongly lighter clauses are preferred for inference selection.
6. Set-of-support strategy.

Expected Competition Performance
We expect Vampire 1.8 to slightly outperform the last year's Vampire 0.6.

7.29 Waldmeister 710

Thomas Hillenbrand
Max-Planck-Institut für Informatik, Germany

Architecture
Waldmeister 710 [46] is a system for unit equational deduction. Its theoretical basis is unfailing completion in the sense of [7] with refinements towards ordered completion (cf. [6]). The system saturates the input axiomatization, distinguishing active facts, which induce a rewrite relation, and passive facts, which are the one-step conclusions of the active ones up to redundancy. The saturation process is parameterized by a reduction ordering and a heuristic assessment of passive facts [47]. This year's version is the result of polishing and fixing a few things in last year's.

Implementation
The approach taken to control the proof search is to choose the search parameters - reduction ordering and heuristic assessment - according to the algebraic structure given in the problem specification [47]. This is based on the observation that proof tasks sharing major parts of their axiomatization often behave similarly.

Strategies
The prover is coded in ANSI-C. It runs on Solaris, Linux, MacOS X, and Windows/Cygwin. The central data 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`

Expected Competition Performance
Waldmeister 710 is the CASC-J5 UEQ division winner.

7.30 Z3 2./20

Nikolaj Bjorner, Leonardo de Moura
Microsoft Research, USA

System description not supplied.

8 Conclusion

The CADE-23 ATP System Competition was the sixteenth large scale competition for classical logic ATP systems. The organizer believes that CASC fulfills its main motivations: stimulation of research, motivation for improving implementations, evaluation of relative capabilities of ATP systems, and providing an exciting event. Through the continuity of the event and consistency in the the reporting of the results, performance comparisons with previous and future years are easily possible. The competition provides exposure for system builders both within and outside of the community, and provides an overview of the implementation state of running, fully automatic, classical logic, ATP systems.

References

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

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

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

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

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

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

[7] L. Bachmair, 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.

[8] 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.

[9] J. Backes and C.E. Brown. Analytic Tableaux for Higher-Order Logic with Choice. *Journal of Automated Reasoning*, page To appear, 2011.

[10] 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.

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

[12] P. Baumgartner, 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.

[13] 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.

[14] P. Baumgartner, B. Pelzer, and C. Tinelli. Model Evolution with Equality - Revised and Implemented. *Journal of Symbolic Computation*, page To appear, 2011.

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

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

[17] P. Baumgartner and C. Tinelli. Model Evolution with Equality Modulo Built-in Theories. In N. Bjorner and V. Sofronie-Stokkermans, editors, *Proceedings of the 23rd International Conference on Automated Deduction*, number 6803 in Lecture Notes in Artificial Intelligence, pages 85–100. Springer-Verlag, 2011.

[18] 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.

[19] C. Benzmüller, B. Fischer, and G. Sutcliffe, editors. *Term Indexing for the LEO-II Prover*, number

212 in CEUR Workshop Proceedings, 2006.

[20] 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.

[21] 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.

[22] 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.

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

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

[25] 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.

[26] 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.

[27] J. Blanchette and A. Kraus. Monotonicity Inference for Higher-Order Formulas. *Journal of Automated Reasoning*, page To appear, 2011.

[28] J. Blanchette and T. Nipkow. Nitpick: A Counterexample Generator for Higher-Order Logic Based on a Relational Model Finder. In Paulson L. Kaufmann M., editor, *Proceedings of the 1st International Conference on Interactive Theorem Proving*, number 6172 in Lecture Notes in Computer Science, pages 131–146. Springer-Verlag, 2010.

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

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

[31] 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.

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

[33] 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.

[34] C.E. Brown. Reducing Higher-Order Theorem Proving to a Sequence of SAT Problems. In N. Bjorner and V. Sofronie-Stokkermans, editors, *Proceedings of the 23rd International Conference on Automated Deduction*, number 6803 in Lecture Notes in Artificial Intelligence, pages 147–161. Springer-Verlag, 2011.

[35] C.E. Brown and G. Smolka. Extended First-Order Logic. In T. Nipkow and C. Urban, editors, *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, number 5674 in Lecture Notes in Computer Science, pages 164–179. Springer-Verlag, 2009.

[36] 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.

[37] C.E. Brown and G. Smolka. Analytic Tableaux for Simple Type Theory and its First-Order Frag-

ment. *Logical Methods in Computer Science*, 6(2), 2010.

[38] 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.

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

[40] W. Denman, B. Akbarpour, S. Tahar, M. Zaki, and L. Paulson. Formal Verification of Analog Designs using MetiTarski. In A. Biere and C. Pixley, editors, *Proceedings of Formal Methods in Computer Aided Design 2009*, pages 93–100. IEEE, 2009.

[41] 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.

[42] 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.

[43] 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.

[44] M. Gebser, O. Sabuncu, and T. Schaub. An Incremental Answer Set Programming Based System for Finite Model Computation. In T. Janhunen and I. Niemelä, editors, *Proceedings of the 12th European Conference on Logics in Artificial Intelligence*, number 6341 in Lecture Notes in Artificial Intelligence, pages 169–181. Springer-Verlag, 2010.

[45] 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.

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

[47] 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.

[48] K. Hoder, L. Kovacs, and A. Voronkov. Interpolation and Symbol Elimination in Vampire. In J. Giesl and R. Haehnle, editors, *Proceedings of the 5th International Joint Conference on Automated Reasoning*, number 6173 in Lecture Notes in Artificial Intelligence, pages 188–195, 2010.

[49] K. Hoder and A. Voronkov. Sine Qua Non for Large Theory Reasoning. In V. Sofronie-Stokkermans and N. Bjœrner, editors, *Proceedings of the 23rd International Conference on Automated Deduction*, number 6803 in Lecture Notes in Artificial Intelligence, pages 299–314. Springer-Verlag, 2011.

[50] 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.

[51] 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.

[52] 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 Inter-*

national Joint Conference on Automated Reasoning, number 5195 in Lecture Notes in Artificial Intelligence, pages 292–298, 2008.

[53] K. Korovin. An Invitation to Instantiation-Based Reasoning: From Theory to Practice. In A. Podelski, A. Voronkov, and R. Wilhelm, editors, *Volume in Memoriam of Harald Ganzinger*, number 5663 in Lecture Notes in Computer Science, pages 163–166. Springer-Verlag, 2009.

[54] K. Korovin and C. Sticksel. iProver-Eq - An Instantiation-Based Theorem Prover with Equality. In J. Giesl and R. Haehnle, editors, *Proceedings of the 5th International Joint Conference on Automated Reasoning*, number 6173 in Lecture Notes in Artificial Intelligence, pages 196–202, 2010.

[55] K. Korovin and C. Sticksel. Labelled Unit Superposition Calculi for Instantiation-based Reasoning. In C. Fermüller and A. Voronkov, editors, *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 6397 in Lecture Notes in Computer Science, pages 459–473. Springer-Verlag, 2010.

[56] R. Letz and G. Stenz. Model Elimination and Connection Tableau Procedures. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 2015–2114. Elsevier Science, 2001.

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

[58] B. Loechner. Things to Know When Implementing LBO. *Journal of Artificial Intelligence Tools*, 15(1):53–80, 2006.

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

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

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

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

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

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

[65] T. Nipkow, L. Paulson, and M. Wenzel. Isabelle/HOL: A Proof Assistant for Higher-Order Logic. http://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/Isabelle/doc/tutorial.pdf.

[66] 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.

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

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

[69] J. Otten and G. Sutcliffe. Using the TPTP Language for Representing Derivations in Tableau and Connection Calculi. In B. Konev, R. Schmidt, and S. Schulz, editors, *Proceedings of the Workshop on Practical Aspects of Automated Reasoning, 5th International Joint Conference on Automated Reasoning*, pages 90–100, 2010.

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

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

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

[73] D. Pastre. Muscadet version 2.3 : User's Manual. http://www.math-info.univ-

paris5.fr/ pastre/muscadet/manual-en.ps, 2001.

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

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

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

[77] L. Paulson and J. Blanchette. Three Years of Experience with Sledgehammer, a Practical Link between Automatic and Interactive Theorem Provers. In G. Sutcliffe, E. Ternovska, and S. Schulz, editors, *Proceedings of the 8th International Workshop on the Implementation of Logics*, page To appear, 2010.

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

[79] 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.

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

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

[82] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

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

[84] 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.

[85] 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.

[86] M. Suda, G. Sutcliffe, P. Wischnewski, M. Lamotte-Schubert, and G. de Melo. External Sources of Axioms in Automated Theorem Proving. In B. Mertsching, editor, *Proceedings of the 32nd Annual Conference on Artificial Intelligence*, number 5803 in Lecture Notes in Artificial Intelligence, pages 281–288, 2009.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[103] 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.

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

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

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

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

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

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

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

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

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

[113] G. Sutcliffe and C. Suttner. The CADE-14 ATP System Competition. Technical Report 98/01, Department of Computer Science, James Cook University, Townsville, Australia, 1998.

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

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

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

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

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

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

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

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

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

[123] E. Torlak and D. Jackson. Kodkod: A Relational Model Finder. In O. Grumberg and M. Huth, editors, *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 4424 in Lecture Notes in Computer Science, pages 632–647. Springer-Verlag, 2007.

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

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

[126] T. Weber. *SAT-based Finite Model Generation for Higher-Order Logic.* PhD thesis, Technische Universität München, Munich, Germany, 2008.

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

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

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