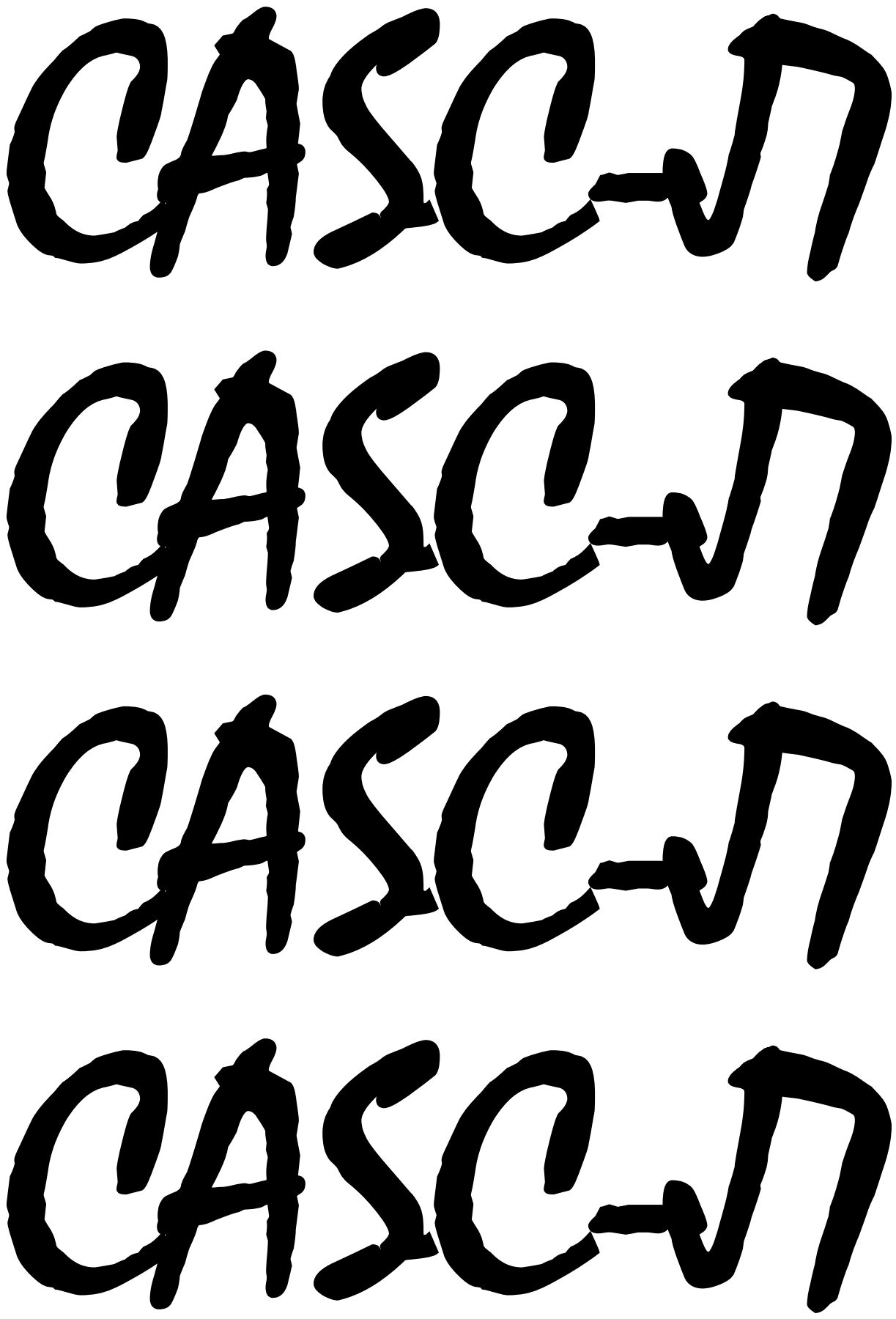
CASC-N
CASC-N
CASC-N
CASC-N

Proceedings of the 7th IJCAR ATP System Competition (CASC-J7)

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 7th IJCAR ATP System Competition (CASC-J7) was held on 20th July 2014. The design of the competition and its rules, and information regarding the competing systems, are provided in this report.

1 Introduction

The CADE and IJCAR 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 and IJCAR conference. CASC-J7 was held on 20th July 2014, as part of the 7th International Joint Conference on Automated Reasoning (IJCAR 2014)[1], which in turn was part of the Vienna Summer of Logic, in Vienna, Austria. It was the nineteenth competition in the CASC series [109, 114, 112, 77, 79, 108, 106, 107, 84, 86, 88, 90, 93, 96, 98, 100, 102, 103].

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

Twenty-five ATP system versions, listed in Table 1, entered into the various competition and demonstration divisions. The winners of the CASC-24 (the previous CASC) divisions were automatically entered into those divisions, to provide benchmarks against which progress can be judged (the competition archive provides access to the systems' executables and source code).[2]

[1] CADE was a constituent conference of IJCAR, hence CASC-"J7".

[2] In the THF division the CASC-24 runner-up, Satallax 2.7, was entered, because it was too hard to get the CASC-24 winner, Satallax-MaLeS 1.2, installed on StarExec. This additionally has the advantage of providing a comparison between the raw Satallax and the MaLeS enhanced version. As the UEQ division had been suspended since CASC-23 in 2011, the CASC-23 UEQ winner was entered.

ATP System	Divisions	Entrant (Associates)	Entrant's Affiliation
agsyHOL 1.0	THF	Fredrik Lindblad	University of Gothenburg
Beagle 0.9	TFA	Peter Baumgartner (Joshua Bax)	NICTA and ANU
cocATP 0.2.0	THF	Cristobal Camarero	University of Cantabria
CVC4 1.4	TFA FOF FNT	Andrew Reynolds (Clark Barrett, Cesare Tinelli, Morgan Deters)	EPFL
E 1.9	FOF FNT EPR UEQ	Stephan Schulz	DHBW Stuttgart
E.T. 0.1	FOF	Josef Urban	Radboud University Nijmegen
HOLyHammer 140616	THF	Cezary Kaliszyk (Stephan Schulz, Josef Urban)	University of Innsbruck
iProver 0.9	EPR	CASC	CASC-24 EPR winner
iProver 1.0	FNT	CASC	CASC-24 FNT winner
Isabelle 2013	THF	Jasmin Blanchette (Lawrence Paulson, Tobias Nipkow, Makarius Wenzel)	Technische Universität München
leanCoP 2.2	FOF	Jens Otten	University of Potsdam
LEO-II 1.6.2	THF	Christoph Benzmüller	Freie Universität Berlin
Muscadet 4.4	FOF	Dominique Pastre	University Paris Descartes
Princess 140704	TFA	Philipp Rümmer (Peter Backeman)	Uppsala University
Prover9 2009-11A	FOF	CASC (William McCune, Bob Veroff)	CASC fixed point
Satallax 2.7	THF	CASC	CASC-24 THF runner-up
Satallax-MaLeS 1.3	THF	Daniel Kuehlwein (Sil van de Leemput, Frank Dorssers, Wouter Geraedts)	Radboud University Nijmegen
SPASS+T 2.2.19	TFA	CASC	CASC-24 TFA winner
SPASS+T 2.2.20	TFA	Uwe Waldmann	Max-Planck-Institut für Informatik
Vampire 2.6	FOF	CASC	CASC-24 FOF winner
VanHElsing 1.0	FOF	Daniel Kuehlwein (Sil van de Leemput, Frank Dorssers, Wouter Geraedts)	Radboud University Nijmegen
Waldmeister 710	UEQ	CASC	CASC-23 UEQ winner
Zipperposition 0.4	TFA FOF	Simon Cruanes	INRIA

Table 1: The ATP systems and entrants

The design and procedures of this CASC evolved from those of previous CASCs [109, 110, 105, 111, 75, 76, 78, 80, 81, 82, 83, 85, 87, 89, 92, 95, 97, 99, 101]. Important changes for this CASC were:

- The competition was run on StarExec [74]. Systems had to be delivered as StarExec installation packages.
- Systems had to use the SZS ontology and standards [91] for reporting their results.
- The LTB division took a one year hiatus.
- The UEQ division returned from its three year hiatus.

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-J7 panel members were Bernhard Beckert, Maria Paola Bonacina, and Aart Middeldorp. 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 StarExec at the University of Iowa. The CASC-J7 web site provides access to resources used before, during, and after the event: `http://www.tptp.org/CASC/J7`

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 divided into divisions according to problem and system characteristics. There are *competition divisions* in which systems are explicitly ranked, and a *demonstration division* in which systems demonstrate their abilities without being ranked. Some divisions are further divided into problem categories, which makes 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 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 TH0 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 TF0 syntax. The TFA division has three problem categories:

- The **TFI** category: TFA with only Integer arithmetic
- The **TFR** category: TFA with only Rational arithmetic
- The **TFE** category: TFA with only Real arithmetic

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

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

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

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

The **EPR** division: Effectively PRopositional (but syntactically non-propositional) clause normal form theorems and non-theorems (clause sets). *Effectively propositional* means that the problem is known to be reducible to a propositional problem, e.g., a problem that has no functions with arity greater than zero. 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 not effectively propositional clause normal form theorems (unsatisfiable clause sets).

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

2.2 The Demonstration Division

ATP systems that cannot run 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.

3 Infrastructure

3.1 Computers

The computers had

- Two quad-core Intel(R) Xeon(R) E5-2609, 2.40GHz CPUs
- 256GB memory
- The Red Hat Enterprise Linux Workstation release 6.3 (Santiago) operating system, kernel 2.6.32-431.1.2.el6.x86_64

Each ATP system ran one job on one computer at a time. Systems could use all the cores on the computers (although this did not necessarily help, because a CPU time limit was imposed).

3.2 Problems

3.2.1 Problem Selection

The problems were taken from the TPTP problem library, version v6.1.0. The TPTP version used for CASC is released after the competition has started, 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 [113]. 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 if there are not enough problems with the desired ratings. Performance data from systems submitted by the system submission deadline is used for computing the problem ratings for the TPTP version used for the competition.
- The TPTP distinguishes versions of problems as one of standard, incomplete, augmented, especial, or biased. All except biased problems are eligible.

The problems used are randomly selected from the eligible problems at the start of the competition, based on a seed supplied by the competition panel.

- The selection is constrained so that no division or category contains an excessive number of very similar problems.
- The selection mechanism is biased to select problems that are new in the TPTP version used, until 50% of the problems in each category have been selected, after which random selection (from old and new problems) continues. The actual percentage of new problems used depends on how many new problems are eligible and the limitation on very similar problems.

3.2.2 Number of Problems

The minimal numbers of problems that 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 [24] (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, 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. The problems in each division are given in increasing order of TPTP difficulty rating.

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, type declarations and definitions are kept before the symbols' uses)
- randomly swap the arguments of associative connectives, and randomly reverse implications
- randomly reverse equalities

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

3.3 Resource Limits

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. An additional memory limit is imposed, depending on the computers' memory. 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.

4 System Evaluation

For each ATP system, for each problem, four items of data are recorded: whether or not the problem was solved, the CPU time taken, the wall clock time taken, and whether or not a solution (proof or model) was output.

The systems are ranked in the competitions division, from the performance data. The THF, TFA, EPR, and UEQ, 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. In the competition divisions winners were announced and prizes are awarded.

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

- Derivations must be complete, starting at formulae from the problem, and ending at the conjecture (for axiomatic proofs) or a $false$ formula (for proofs by contradiction, including CNF refutations).

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

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

In addition to the ranking criteria, 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 (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.
- The *core usage* is the average of the ratios of CPU time to wall clock time used, over the problems solved. This measures the extent to which the systems take advantage the multiple cores.

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

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.

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 finalize their installation packages and 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, EPR, and UEQ 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. 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 StarExec (the *competition computers* - see Section 3.1). ATP systems that cannot run on the competition computers can be entered into the demonstration division.
- Systems must be fully automatic, i.e., all command line switches have to be the same for all problems in each division.

6.1.3 Output

- For each problem, the system must output a distinguished string indicating what solution has been found or that no conclusion has been reached. Systems must use the SZS ontology and standards [91] for this. For example

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

  or

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

- When outputting proofs/models, the start and end of the proof/model must be delimited by distinguished strings. Systems must use the SZS ontology and standards for this. For example

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

The string specifying the problem status must be output before the start of a proof/model. Use of the TPTP format for proofs and finite interpretations is encouraged [104].

6.1.4 Resource Usage

- 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`.
- 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.

6.2 System Delivery

For systems running on the competition computers, entrants must email a StarExec installation package to the competition organizers by the system delivery deadline. The entrants must also email a `.tgz` file containing the source code and any files required for building the StarExec installation package to the competition organizers by the system delivery deadline.

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 StarExec. The jobs are queued onto the computers so that each computer is running one job at a time. In non-batch divisions, all attempts at the Nth problems in all the divisions and categories are started before any attempts at the (N+1)th problems.

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 agsyHOL 1.0

Fredrik Lindblad
University of Gothenburg, Sweden

Architecture
AgsyHOL 1.0 is based on intuitionistic sequent calculus extended with rules for classical and equality reasoning. There is an introduction and elimination rule for each logical construct, plus rules for RAA, AC and extensionality. There are special inference modes for elimination and equality reasoning.

A backward search is applied to find proof derivations in the calculus. Search is controlled by locally assigning priorities to different kind of constraints, and refining that part of the half-finished proof which is blocking a constraint with the highest priority.

Strategies
The system has no set of strategies it chooses from by pre-analysing the problem. What it does do before starting the proof search is to try to minimise the need for classical reasoning by transforming the problem using double negation elimination and the de Morgan laws.

Implementation
The system is implemented in Haskell. It consists of a proof checker for the calculus. The proof checker is annotated with search control information controlling priorities of constraints and costs of choices. The system also consists of an implementation of lazy narrowing, a general purpose search mechanism, which is applied to the proof checker in order to achieve proof search. The lazy narrowing search has been extended in order to deal with customizable priorities and costs. agsyHOL is available from:

`https://github.com/frelindb/agsyHOL/`

Expected Competition Performance
On SystemOnTPTP it solves 1722 of the THF problems (TPTP version 6.0.0).

7.2 Beagle 0.9

Peter Baumgartner, Josh Bax
NICTA and Australian National University, Australia

Architecture
Beagle is an automated theorem prover for sorted first-order logic with equality over built-in theories. The theories currently supported are integer arithmetic, linear rational arithmetic and linear real arithmetic. It accepts formulas in the FOF and TFF formats of the TPTP syntax, and formulas in the SMT-LIB version 2 format.

Beagle first converts the input formulas into clause normal form. Pure arithmetic (sub-)formulas are treated by eager application of quantifier elimination. The core reasoning component implements the Hierarchic Superposition Calculus with Weak Abstraction (HSPWA) [6].

Extensions are a splitting rule for clauses that can be divided into variable disjoint parts, and a chaining inference rule for reasoning with inequalities.

The HSPWA calculus generalizes the superposition calculus by integrating theory reasoning in a black-box style. For the theories mentioned above, Beagle combines quantifier elimination procedures and other solvers to dispatch proof obligations over these theories. The default solvers are an improved version of Cooper's algorithm for linear integer arithmetic, and the Fourier-Motzkin algorithm for linear real/rational arithmetic. Non-linear integer arithmetic is treated by partial instantiation.

Strategies
Beagles uses the Discount loop for saturating a clause set under the calculus' inference rules. Simplification techniques include standard ones, such as subsumption deletion, demodulation by ordered unit equations, and tautology deletion. It also includes theory specific simplification rules for evaluating ground (sub)terms, and for exploiting cancellation laws and properties of neutral elements, among others. In the competition an aggressive form of arithmetic simplification is used, which seems to perform best in practice.

Beagle uses strategy scheduling by trying (at most) two flag settings sequentially.

Implementation
Beagle is implemented in Scala. It is a full implementation of the HSPWA calculus. It uses a simple form of indexing, essentially top-symbol hashes, stored with each term and computed in a lazy way. Fairness is achieved through a combination of measuring clause weights and their derivation-age. It can be fine-tuned with a weight-age ratio parameter, as in other provers.

Beagle's web site is

`http://users.cecs.anu.edu.au/~baumgart/systems/beagle/`

Expected Competition Performance
Beagle should perform reasonably well.

7.3 cocATP 0.2.0

Cristóbal Camarero
University of Cantabria, Spain

Architecture
cocATP is a Coq-inspired [7] automated theorem prover made on the free time of the author. It implements (the non-inductive) part of Coq's logic (calculus of constructions) and syntax. The proof terms it creates are accepted as proofs by Coq by the addition of a few definitions (substituting the respective inductive definitions of Coq). As in Coq, equality and logical connectives other than implication (which is a dependent product of the logic) are defined adding the proper axioms. The reasoning is tried to be done the more general possible, avoiding to give any special treatment to both equality and logical connectives. ¡p¿ At difference of most of the other provers, cocATP does not rely on SAT or first-order solver. All reasoning is done in the high-order logic that is the calculus of constructions.

Strategies
The first of the rules is: for a node with conjecture (forall x:T, P x) create a new node with hypothesis (x:T) and conecture (P x). Other of the major rules consist in: when having a node with conjecture (C) and some hypothesis (H:forall (x1:T1) (x2:T2) ... (xn:Tn), P x1 x2 ... xn), to unificate P with C and prove the Ti necessary. The unification is a subset of the possible high-order unifications. And in this case is a one-sided unification. There is a lot of other ad-hoc rules and conditions to apply them. Some of the rules can create existential terms (x:=??:T) alike to Coq's tactic evar. With these rules resulting proofs are very similar to human-made proofs.

There is a partially implemented second strategy more similar to the ones found in other theorem provers. It consist a saturation algorithm with the high-order modus ponens. As example, given a hypothesis (P) and another (or the same) hypothesis (forall x:T, Q x), unficate P with T generating a new hypothesis Q. This generalizes the typical saturation rules, as resolution and superposition. The strategy yet lacks a good conditions for subsumption and simplifcation. Probably it will not be used in competition.

Implementation
cocATP is implemented in Python-2.7 and C. It uses the Ply-3.4 library to build the parsers for both the Coq and TPTP syntaxes. Recently, the processing core has been reimplemented in C as a Python module, and a Python variable controls if using a pure Python or the C version. Using the C module can be an order of magnitude faster, but it is less tested and introduces the possibility of SEGFAULTS.

Includes a type-verifier of Calculus of Constructions without inductive constructions, which must be defined with axioms. That is, a buggy partial clone of Coq. There is support for most of the TPTP syntax, problems are translated to a set of calculus of constructions terms. cocATP has been specially prepared for THF, but all the other TPTP formulae without numbers should be accepted. However cocATP does NOT include a SAT solver, thus it will probably not solve your trivial CNF problems.

More info at:

`http://www.alumnos.unican.es/ccc66/cocATP.htm`

Expected Competition Performance
With the reimplementation of the kernel in C it is expected a great reduction in the execution time, but not a lot of newly solved problems.

7.4 Crossbow 0.1

Radek Micek
Charles University in Prague, Czech Republic

Architecture
Crossbow 0.1 is a MACE2-style finite model finder similar to Paradox. It is an implementation of techniques from [19] plus improved flattening and built-in support for commutativity.

Strategies
The same strategy is used for all problems:

1. First order formulas are clausified by another prover.

2. Additional lemmas are generated by another prover, small lemmas are added to the clauses of the problem.
3. Clauses of the problem are preprocessed (simplification, flattening, splitting, term definitions, commutativity detection).
4. Clauses are instantiated for increasing domain sizes until some model is found. Instantition takes into account commutativity of functions and symmetry of predicates.

Implementation
Crossbow is implemented in OCaml. Clausification and lemma generation is done by E. MiniSat is used for incremental SAT solving. Source code is available from:

```
https://github.com/radekm/crossbow
```

Expected Competition Performance
It should perform slightly better than Paradox.

7.5 CVC4 1.4

Andrew Reynolds
EPFL, Switzerland

Architecture
CVC4 [4] is an SMT solver based on the DPLL(T) architecture [46] that includes built-in support for many theories including linear arithmetic, arrays, bit vectors, datatypes and strings. It incorporates various approaches for handling universally quantified formulas. In particular, CVC4 uses primarily uses heuristic approaches based on E-matching for answering "unsatisfiable", and finite model finding approaches for answering "satisfiable".

Like other SMT solvers, CVC4 treats quantified formulas using a two-tiered approach. First, quantified formulas are replaced by fresh boolean predicates and the ground theory solver(s) are used in conjunction with the underlying SAT solver to determine satisfiability. If the problem is unsatisfiable at the ground level, then the solver answers "unsatisfiable". Otherwise, the quantifier instantiation module is invoked, and will either add instances of quantified formulas to the problem, answer "satisfiable", or return unknown.

The finite model finding has been developed to target problems containing background theories, whose quantification is limited to finite and uninterpreted sorts. In finite model finding mode, CVC4 uses a ground theory of finite cardinality constraints that minimizes the number of ground equivalence classes, as described in [65]. When the problem is satisfiable at the ground level, a candidate model is constructed that contains complete interpretations for all predicate and function symbols. Quantifier instantiation strategies are then invoked to add instances of quantified formulas that are in conflict with the candidate model, as described in [66]. If no instances are added, then the solver reports "satisfiable".

Strategies
For handling theorems, CVC4 primarily uses various configurations of E-matching. This year, CVC4 incorporates new methods for finding conflicting instances of quantified formulas [64], which have been shown to lead to improved performance on unsatisfiable TPTP benchmarks. CVC4 also incorporates a model-based heuristic for handling quantified formulas containing only pure arithmetic, which will be used in the TFA division.

For handling non-theorems, the finite model finding feature of CVC4 will use a number of orthogonal quantifier instantiation strategies. This year, it will incorporate several new features, including an optimized implementation of model-based quantifier instantiation which improves upon [66], as well as techniques for sort inference. Since CVC4 with finite model finding is also capable of answering "unsatisfiable", it will be used as a strategy for theorems as well.

Implementation
CVC4 is implemented in C++. The code is available from

```
https://github.com/CVC4
```

Expected Competition Performance
In the FOF division, CVC4 should perform better than last year, primarily due to the incorporation of methods from [64]. In the FNT division, CVC4 should also perform better than last year, due to its incorporation of sort inference techniques and general improvements to the implementation. This is the first year CVC4 has entered the TFA division, where it should be fairly competitive due to its efficient handling of ground arithmetic constraints.

7.6 E 1.9

Stephan Schulz
DHBW Stuttgart, Germany

Architecture
E [70, 73] 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 form, 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.

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 parameterized 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 parameterized instances of Knuth-Bendix-Ordering (KBO) and Lexicographic Path Ordering (LPO).

For CASC-J7, E implements a strategy-scheduling automatic mode. The total CPU time available is broken into 8 (unequal) time slices. For each time slice, the problem is classified into one of several classes, based on a number of simple features (number of clauses, maximal symbol arity, presence of equality, presence of non-unit and non-Horn clauses, ...). For each class, a schedule of strategies is greedily constructed from experimental data as follows: The first strategy assigned to a schedule is the the one that solves the most problems from this class

in the first time slice. Each subsequent strategy is selected based on the number of solutions on problems not already solved by a preceding strategy.

About 210 different strategies have been evaluated on all untyped first-order problems from TPTP 6.0.0, and about 180 of these strategies are used in the automatic mode.

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 [71]. Superposition and backward rewriting use fingerprint indexing [72], 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 L`ochner in [42, 42]. The prover and additional information are available at

`http://www.eprover.org`

Expected Competition Performance
E 1.9 has slightly better strategies than previous versions, and can produce proof objects quite efficiently. The system is expected to perform well in most proof classes, but will at best complement top systems in the disproof classes.

7.7 E.T. 0.1

Josef Urban[1], Cezary Kaliszyk[2], Stephan Schulz[3], Jiri Vyskocil[4]
[1]Radboud University Nijmegen, The Netherlands, [2]University of Innsbruck, Switzerland, [3]DHBW Stuttgart, Germany, [4]Czech Technical University, Czech Republic

Architecture
E.T. 0.1 is a metasystem using E prover with specific strategies [115, 35] and preprocessing tools [33, 31, 32] that are targeted mainly at problems with many redundant axioms. Its design is motivated by the recent experiments in the Large-Theory Batch division [37] and on the Flyspeck, Mizar and Isabelle datasets, however, E.T. does no learning from related proofs.

Strategies
We characterize formulas by the symbols and terms that they contain, normalized in various ways. Then we run various algorithms that try to remove the redundant axioms and use special strategies on such problems.

Implementation
The metasystem is implemented in ca. 1000 lines of Perl. It uses a number of external programs, some of them based on E's code base, some of them independently implemented in C++.

Expected Competition Performance
E.T. can solve some problems that E 1.8 cannot prove, and even some TPTP problems with rating 1. The CASC performance should not be much worse than that of E, possibly better, depending on problem selection.

7.8 HOLyHammer 140616

Cezary Kaliszyk[1], Josef Urban[2], Stephan Schulz[3]
[1]University of Innsbruck, Switzerland, [2]Radboud University Nijmegen, The Netherlands, [3]DHBW Stuttgart, Germany

Architecture
HOLyHammer [36, 34] is a metasystem for premise selection and proof translation from polymorphic HOL to FOF ATP provers. This is a version restricted to THF0 and to using E prover with specific strategies.

Strategies
The translation uses polymorphic type tags, the apply functor, lambda lifting and predicate abstraction. The premise selection [36] uses k-NN, naive Bayes and a modified implementation of the Meng-Paulson relevance filter. We characterize formulas by the symbols and terms that they contain, normalized in various ways.

Implementation
The HH metasystem consists of about 2000 lines of OCaml code on top of about 5000 lines of HOL Light code. The predictors are about 400 lines of C++ code. The system produces higher-order logic proofs internally that match the HOL Light kernel, however no TSTP proofs are produced [34].

Expected Competition Performance
We expect HOLyHammer to solve some large problems with many redundant axioms, however we have not tested it on the whole set of THF0 TPTP problems.

7.9 iProver 0.9

Konstantin Korovin
University of Manchester, United Kingdom

Architecture
iProver is an automated theorem prover based on an instantiation calculus Inst-Gen [22, 39] 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 [21] 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 [38] 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 [23, 38]; global subsumption [38]; 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 [30] as implemented in Vampire [28], 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 [28] 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 the CASC-24 EPR division winner.

7.10 iProver 1.0

Konstantin Korovin, Christoph Sticksel
University of Manchester, United Kingdom

Architecture
iProver is an automated theorem prover based on an instantiation calculus Inst-Gen [22, 40] which is complete for first-order logic. iProver combines first-order reasoning with ground reasoning for which it uses MiniSat [21] and was recently extended with PicoSAT [9] and Lingeling [10] (only MiniSat will be used at this CASC). iProver also combines instantiation with ordered resolution; see [38] for the implementation details. The proof search is implemented using a saturation process based on the 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 [23, 38]; global subsumption [38]; 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 uses axiom selection based on the SInE algorithm [30] as implemented in Vampire [29], i.e., axiom selection is done by Vampire and proof attempts are done by iProver.

Some of iProver features are summarised below.

- proof extraction for both instantiation and resolution,
- model representation, using first-order definitions in term algebra,
- answer substitutions,
- semantic filtering,
- type inference, monotonic [18] and non-cyclic types,
- Brand's transformation.

Type inference is targeted at improving finite model finding and symmetry breaking. Semantic filtering is used in preprocessing to eliminated irrelevant clauses. Proof extraction is challenging due to simplifications such global subsumption which involve global reasoning with the whole clause set and can be computationally expensive.

Strategies
iProver has around 60 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. The strategy for satisfiable problems (FNT division) includes finite model finding.

Implementation
iProver is implemented in OCaml and for the ground reasoning uses MiniSat [21]. iProver accepts FOF and CNF formats. Vampire [29, 27] is used for proof-producing clausification of FOF problems as well as for axiom selection [30] in the LTB division.

iProver is available from

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

Expected Competition Performance
iProver 1.0 is the CASC-24 FNT division winner.

7.11 Isabelle 2013

Jasmin C. Blanchette[1], Lawrence C. Paulson[2],
Tobias Nipkow[1], Makarius Wenzel[3]¡BR¿ [1]Technische Universität München, Germany, [2]University of Cambridge, United Kingdom, [3]Université Paris Sud, France

Architecture
Isabelle/HOL 2013 [48] is the higher-order logic incarnation of the generic proof assistant Isabelle2013. Isabelle/HOL provides several automatic proof tactics, notably an equational reasoner [47], a classical reasoner [62], and a tableau prover [60]. It also integrates external first- and higher-order provers via its subsystem Sledgehammer [61, 11].

Isabelle includes a parser for the TPTP syntaxes CNF, FOF, TFF0, and THF0, due to Nik Sultana. It also includes TPTP versions of its popular tools, invokable on the command line as `isabelle tptp_`*`tool max_secs file.p`*. For example:

```
isabelle tptp_isabelle_hot 100 SEU/SEU824^3.p
```

Strategies
The *Isabelle* tactic submitted to the competition simply tries the following tactics sequentially:

- `sledgehammer` : Invokes the following sequence of provers as oracles via Sledgehammer:
  - `spass` - SPASS 3.8ds [12];
  - `vampire` - Vampire 1.8 (revision 1435) [67];
  - `e` - E 1.4 [71];
  - `z3_tptp` - Z3 3.2 with TPTP syntax [20].
- `nitpick` : For problems involving only the type `$o` of Booleans, checks whether a finite model exists using Nitpick [14].
- `simp` : Performs equational reasoning using rewrite rules [47].
- `blast` : Searches for a proof using a fast untyped tableau prover and then attempts to reconstruct the proof using Isabelle tactics [60].
- `auto+spass` : Combines simplification and classical reasoning [62] under one roof; then invoke Sledgehammer with SPASS on any subgoals that emerge.
- `z3` : Invokes the SMT solver Z3 3.2 [20].
- `cvc3` : Invokes the SMT solver CVC3 2.2 [5].
- `fast` : Searches for a proof using sequent-style reasoning, performing a depth-first search [62]. Unlike `blast`, it construct proofs directly in Isabelle. That makes it slower but enables it to work in the presence of the more unusual features of HOL, such as type classes and function unknowns.
- `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 [43]. Constructs proofs directly in Isabelle.
- `fastforce` : Combines `fast` and `force`.

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. 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
Third place, after Satallax and Satallax-MaLeS but before AgsyHOL and LEO-II.

7.12 leanCoP 2.2

7.13 LEO-II 1.6.2

Jens Otten
University of Potsdam, Germany

Architecture
leanCoP [53, 49] is an automated theorem prover for classical first-order logic with equality. It is a very compact implementation of the connection (tableau) calculus [8, 41].

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 performed in order to achieve completeness. Additional inference rules and techniques include regularity, lemmata, and restricted backtracking [50]. leanCoP uses an optimized structure-preserving transformation into clausal form [50] and a fixed strategy scheduling, which is controlled by a shell script.

Implementation
leanCoP is implemented in Prolog. The source code of the core prover consists only of a few lines of code. Prolog's built-in indexing mechanism is used to quickly find connections when the extension rule is applied. leanCoP can read formulae in leanCoP syntax and in TPTP first-order syntax. Equality axioms and axioms to support distinct objects are automatically added if required. The leanCoP core prover returns a very compact connection proof, which is then translated into a more comprehensive output format, e.g., into a lean (TPTP-style) connection proof or into a readable text proof.

The source code of leanCoP 2.2 is available under the GNU general public license. It can be downloaded from the leanCoP website at: available from

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

The website also contains information about ileanCoP [49] and MleanCoP [51, 52], two versions of leanCoP for first-order intuitionistic logic and first-order modal logic, respectively.

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

7.14 Muscadet 4.4

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 [13] and [54], 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 [55] and [56]. These two possibilities are not used while working with the TPTP Library.

Implementation
Muscadet [57] 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 [58, 59]. 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. Its originality is that proofs are given in natural style. Changes since last year are only minor corrections.

7.15 Princess—140704

Philipp Rümmer, Peter Backeman¡BR¿ Uppsala University, Sweden

Architecture
Princess [68, 69] is a theorem prover for first-order logic modulo linear integer arithmetic. The prover uses a combination of techniques from the areas of first-order reasoning and SMT solving. The main underlying calculus is a free-variable tableau calculus, which is extended with constraints to enable backtracking-free proof expansion, and positive unit hyper-resolution for lightweight instantiation of quantified formulae. Linear integer arithmetic is handled using a set of built-in proof rules resembling the Omega test, which altogether yields a calculus that is complete for full Presburger arithmetic, for first-order logic, and for a number of further fragments. In addition, some built-in procedures for nonlinear integer arithmetic are available.

The internal calculus of Princess only supports uninterpreted predicates; uninterpreted functions are encoded as predicates, together with the usual axioms. Through appropriate translation of quantified formulae with functions, the e-matching technique common in SMT solvers can be simulated; triggers in quantified formulae are chosen based on heuristics similar to those in the Simplify prover.

Strategies
For CASC, Princess will run a fixed schedule of configurations for each problem (portfolio method). Configurations determine, among others, the mode of proof expansion (depth-first, breadth-first), selection of triggers in quantified formulae, clausification, and the handling of functions. The portfolio was chosen based on training with a random sample of problems from the TPTP library.

Implementation
Princess is entirely written in Scala and runs on any recent Java virtual machine; besides the standard Scala and Java libraries, only the Cup parser library is used. Princess is available from:

`http://www.philipp.ruemmer.org/princess.shtml`

Expected Competition Performance
Princess is mainly designed for integer problems (TFI), and should perform reasonably well here. Reasoning about rationals or reals is not the main focus of the work, and results will be accordingly.

7.16 Prover9 2009-11A

Bob Veroff on behalf of William McCune
University of New Mexico, USA

Architecture
Prover9, Version 2009-11A, is a resolution/paramodulation prover for first-order logic with equality. Its overall architecture is very similar to that of Otter-3.3 [45]. It uses the "given clause algorithm", in which not-yet-given clauses are available for rewriting and for other inference operations (sometimes called the "Otter loop").

Prover9 has available positive ordered (and nonordered) resolution and paramodulation, negative ordered (and nonordered) resolution, factoring, positive and negative hyperresolution, UR-resolution, and demodulation (term rewriting). Terms can be ordered with LPO, RPO, or KBO. Selection of the "given clause" is by an age-weight ratio.

Proofs can be given at two levels of detail: (1) standard, in which each line of the proof is a stored clause with detailed justification, and (2) expanded, with a separate line for each operation. When FOF problems are input, proof of transformation to clauses is not given.

Completeness is not guaranteed, so termination does not indicate satisfiability.

Strategies
Like Otter, Prover9 has available many strategies; the following statements apply to CASC-2012.

Given a problem, Prover9 adjusts its inference rules and strategy according to syntactic properties of the input clauses such as the presence of equality and non-Horn clauses. Prover9 also does some preprocessing, for example, to eliminate predicates.

In previous CASC competitions, Prover9 has used LPO to order terms for demodulation and for the inference rules, with a simple rule for determining symbol precedence. For CASC 2012, we are going to use KBO instead.

For the FOF problems, a preprocessing step attempts to reduce the problem to independent subproblems by a miniscope transformation; if the problem reduction succeeds, each subproblem is clausified and given to the ordinary search procedure; if the problem reduction fails, the original problem is clausified and given to the search procedure.

Implementation
Prover9 is coded in C, and it uses the LADR libraries. Some of the code descended from EQP [44]. (LADR has some AC functions, but Prover9 does not use them). Term data structures are not shared (as they are in Otter). Term indexing is used extensively, with discrimination tree indexing for finding rewrite rules and subsuming units, FPA/Path indexing for finding subsumed units, rewritable terms, and resolvable literals. Feature vector indexing [71] is used for forward and backward nonunit subsumption. Prover9 is available from

`http://www.cs.unm.edu/~mccune/prover9/`

Expected Competition Performance
Some of the strategy development for CASC was done by experimentation with the CASC-2004 competition "selected" problems. (Prover9 has not yet been run on other TPTP problems.) Prover9 is unlikely to challenge the CASC leaders, because (1) extensive testing and tuning over TPTP problems has not been done, (2) theories (e.g., ring, combinatory logic, set theory) are not recognized, (3) term orderings and symbol precedences are not fine-tuned, and (4) multiple searches with differing strategies are not run.

Finishes in the middle of the pack are anticipated in all categories in which Prover9 competes.

7.17 Satallax 2.7

Chad E. Brown
Saarland University, Germany

Architecture
Satallax 2.7 [15] 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 [21] 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 [17]

with a choice operator [3]. 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 [16]. 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.

Additionally, Satallax may optionally generate first-order formulas in addition to the propositional clauses. If this option is used, then Satallax peroidically calls the first-order theorem prover E to test for first-order unsatisfiability. If the set of first-order formulas is unsatisfiable, then the original branch is unsatisfiable.

Strategies
There are about a hundred 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 (such as whether or not to call the theorem prover E). A collection of flag settings is called a mode. Approximately 500 modes have been defined and tested so far. 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 2.7 has strategy schedule consisting of 68 modes. Each mode is tried for time limits ranging from 0.1 seconds to 54.9 seconds. The strategy schedule was determined through experimentation using the THF problems in version 5.4.0 of the TPTP library.

Implementation
Satallax 2.7 is implemented in OCaml. A foreign function interface is used to in teract with MiniSat 2.2.0. Satallax is available from

```
http://satallax.com
```

Expected Competition Performance
Satallax 2.7 is the CASC-24 THF division runner-up.

7.18 Satallax-MaLeS 1.3

Daniel Kuehlwein
Radboud University Nijmegen, The Netherlands

Architecture
Satallax-MaLeS 1.3 improves Satallax's automatic mode with machine learning techniques to predict which search strategy is most likely to find a proof.

Strategies
Satallax-MaLeS 1.3 relies on Geoff Sutcliffe's MakeListStat to classify problems. It uses the same data as Satallax-MaLeS 1.2 as basis for the learning algorithms.

Implementation
Satallax-MaLeS is based on Python, in particular the Sklearn library that contains many machine learning algorithms. After CASC Satallax-MaLeS will be available from

`http://www.cs.ru.nl/~kuehlwein/`

Expected Competition Performance
We expect Satallax-MaLeS 1.3 to perform slightly better than Satallax-MaLeS 1.2.

7.19 SPASS+T 2.2.19

Uwe Waldmann
Max-Planck-Insitut für Informatik, 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 [63]; later a much more sophisticated coupling of the SMT procedure has been added [117]. The latest version provides improved support for isint/1, israt/1, floor/1 and ceiling/1 and adds partial input abstraction and history-dependent weights for numerical constants.

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. SPASS+T is available from

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

Expected Competition Performance
SPASS+T 2.2.16 came second in the TFA division of last CASC; we expect a similar performance in CASC 2013.

7.20 SPASS+T 2.2.20

Uwe Waldmann
Max-Planck-Insitut für Informatik, 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 [63]; later a much more sophisticated coupling of the SMT procedure has been added [117]. The latest version provides improved support for isint/1, israt/1, floor/1 and ceiling/1 and adds partial input abstraction and history-dependent weights for numerical constants.

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. SPASS+T is available from

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

Expected Competition Performance
SPASS+T 2.2.19 came first in the TFA division of last CASC; we expect a similar performance in CASC 2014.

7.21 Vampire 2.6

Krystof Hoder, Andrei Voronkov
University of Manchester, England

Architecture
Vampire 2.6 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. It also implements the Inst-gen calculus. The splitting rule in kernel adds propositional parts to clauses, which are being 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 [30] can be enabled as part of the preprocessing.

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

Strategies
The Vampire 2.6 kernel provides a fairly large number of options for strategy se lection. The most important ones are:

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

Implementation
Vampire 2.6 is implemented in C++.

Expected Competition Performance
Vampire 2.6 is the CASC-24 FOF division winner.

7.22 Waldmeister 710

Thomas Hillenbrand
Max-Planck-Institut für Informatik, Germany

Architecture
Waldmeister 710 [25] is a system for unit equational deduction. Its theoretical basis is unfailing completion in the sense of [2] with refinements towards ordered completion (cf. [1]). 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 [26]. This year's version is the result of polishing and fixing a few things in last year's.

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`

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 [26]. This is based on the observation that proof tasks sharing major parts of their axiomatization often behave similarly.

Expected Competition Performance
Waldmeister 710 is the CASC-23 UEQ division winner.

7.23 VanHElsing 1.0

Daniel Kuehlwein
Radboud University Nijmegen, The Netherlands

Architecture
VanHElsing improves E's automatic mode with machine learning techniques to predict which search strategy is most likely to find a proof.

Strategies
VanHElsing relies on E's features to classify problems. It uses the same data as E 1.8 as basis for the learning algorithms.

Implementation
VanHElsing is based on Python, in particular the Sklearn library that contains many machine learning algorithms. After CASC VanHElsing will be availble from

```
http://www.cs.ru.nl/~kuehlwein/
```

Expected Competition Performance
We expect VanHElsing to perform slightly better than E 1.8.

7.24 Zipperposition 0.4

Simon Cruanes
INRIA, France

Architecture
Zipperposition is a typed first-order superposition theorem prover written in OCaml for prototyping. It deals with hashconsed polymorphic terms, has a Hindley-Milner-like type-inference algorithm and accepts TFF1 input. Its core superposition calculus is strongly inspired from E (discount loop, mostly same inferences including condensation and contextual literal cutting).

The new version has experimental support for linear integer arithmetic. It features a new calculus (not published yet), inspired from the work of Waldmann on superposition modulo rational arithmetic [116]. The calculus is an extension of superposition that doesn't use a black-box solver, but has special inferences to deal with arithmetic equations, inequations and divisibility literals. Additional redundancy criteria are used to make the search space more tractable. As we don't have completeness results yet, the solver will fail on satisfiable problems, even if it finds a saturated set of clauses.

Strategies
Zipperposition uses several clause queues, but unlike E it has no heuristic to choose them based on the input. The strategy is therefore a simple alternance of old clauses, and "small" clauses, in one run.

Implementation
Zipperposition is in pure OCaml, with a mix of imperative and functional style, including iterators, zippers, etc. It uses Non-perfect Discrimination Trees and Feature Vectors [71] for indexing. Numbers are handled by Zarith, a frontend to GMP.

Expected Competition Performance
In FOF, we expect decent performance. The results should be quite similar to last year as the superposition core isn't the focus of our research.

In TFA, since we feature a new calculus, performance is hard to predict. We expect the prover to perform well on problems SMT solvers will struggle with, and conversely; best performance should be reached on unsatisfiable problems with few inequations but possibly function symbols and quantifiers; worst will be on combinatorial ground problems that require little symbolic reasoning.

8 Conclusion

The 7th IJCAR ATP System Competition was the nineteenth 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] 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.

[2] 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.

[3] J. Backes and C.E. Brown. Analytic Tableaux for Higher-Order Logic with Choice. *Journal of Automated Reasoning*, 47(4):451–479, 2011.

[4] C. Barrett, C. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. In G. Gopalakrishnan and S. Qadeer, editors, *Proceedings of the 23rd International Conference on Computer Aided Verification*, number 6806 in Lecture Notes in Computer Science, pages 171–177. Springer-Verlag, 2011.

[5] 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.

[6] P. Baumgartner and U. Waldmann. Hierarchic Superposition With Weak Abstraction. In M.P. Bonacina, editor, *Proceedings of the 24th International Conference on Automated Deduction*, number 7898 in Lecture Notes in Artificial Intelligence, pages 39–57. Springer-Verlag, 2013.

[7] Y. Bertot and P. Casteran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer-Verlag, 2004.

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

[9] A. Biere. PicoSAT Essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008.

[10] A. Biere. Lingeling and Friends Entering the SAT Challenge 2012. In A. Balint, A. Belov, A. Diepold, S. Gerber, M. Järvisalo, and C. Sinz, editors, *Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions*, number B-2012-2 in Department of Computer Science Series of Publications B, pages 15–16. University of Helsinki, 2012.

[11] J. Blanchette, S. Boehme, and L. Paulson. Extending Sledgehammer with SMT Solvers. 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 116–130. Springer-Verlag, 2011.

[12] J. Blanchette, A. Popescu, D. Wand, and C. Weidenbach. More SPASS with Isabelle. In L. Beringer and A. Felty, editors, *Proceedings of Interactive Theorem Proving 2012*, number 7406 in Lecture Notes in Computer Science, pages 345–360. Springer-Verlag, 2012.

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

[14] S. Böhme and T. Nipkow. Sledgehammer: Judgement Day. 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 107–121, 2010.

[15] C.E. Brown. Satallax: An Automated Higher-Order Prover (System Description). In B. Gramlich, D. Miller, and U. Sattler, editors, *Proceedings of the 6th International Joint Conference on Automated Reasoning*, number 7364 in Lecture Notes in Artificial Intelligence, pages 111–117, 2012.

[16] C.E. Brown. Reducing Higher-Order Theorem Proving to a Sequence of SAT Problems. *Journal of Automated Reasoning*, 51(1):57–77, 2013.

[17] C.E. Brown and G. Smolka. Analytic Tableaux for Simple Type Theory and its First-Order Fragment. *Logical Methods in Computer Science*, 6(2), 2010.

[18] K. Claessen, A. Lilliestrom, and N. Smallbone. Sort It Out with Monotonicity - Translating between Many-Sorted and Unsorted First-Order Logic. 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 207–221. Springer-Verlag, 2011.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] K. Hoder, Z. Khasidashvili, K. Korovin, and A. Voronkov. Preprocessing Techniques for First-Order Clausification. In G. Cabodi and S. Singh, editors, *Proceedings of the Formal Methods in Computer-Aided Design 2012*, pages 44–51. IEEE Press, 2012.

[28] 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.

[29] K. Hoder, L. Kovacs, and A. Voronkov. Invariant Generation in Vampire. In P. Abdulla and R. Leino, editors, *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 6605 in Lecture Notes in Computer Science, pages 60–64. Springer-Verlag, 2011.

[30] 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.

[31] C. Kaliszyk and J. Urban. Automated Reasoning Service for HOL Light. In J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger, editors, *Intelligent Computer Mathematics*, number 7961 in Lecture Notes in Computer Science, pages 120–135. Springer-Verlag, 2013.

[32] C. Kaliszyk and J. Urban. Lemma Mining over HOL Light. In K. McMillan, A. Middeldorp, and A. Voronkov, editors, *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 8312 in Lecture Notes in Computer Science, pages 503–517. Springer-Verlag, 2013.

[33] C. Kaliszyk and J. Urban. MizAR 40 for Mizar 40. arXiv:1310.2805, 2013.

[34] C. Kaliszyk and J. Urban. PRocH: Proof Reconstruction for HOL Light. In M.P. Bonacina, editor, *Proceedings of the 24th International Conference on Automated Deduction*, number 7898 in Lecture Notes in Artificial Intelligence, pages 267–274. Springer-Verlag, 2013.

[35] C. Kaliszyk and J. Urban. Stronger Automation for Flyspeck by Feature Weighting and Strategy Evolution. In *Proceedings of the 3rd International Workshop on Proof Exchange for Theorem Proving*, page To appear. EasyChair Proceedings in Computing, 2013.

[36] C. Kaliszyk and J. Urban. Learning-assisted Automated Reasoning with Flyspeck. *Journal of Automated Reasoning*, page To appear, 2014.

[37] C. Kaliszyk, J. Urban, and J. Vyskocil. Machine Learner for Automated Reasoning 0.4 and 0.5. arXiv:1402.2359, 2014.

[38] K. Korovin. iProver - An Instantiation-Based Theorem Prover for First-order Logic (System

Description). In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 292–298, 2008.

[39] K. Korovin. Instantiation-Based Reasoning: From Theory to Practice. In R. Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction*, number 5663 in Lecture Notes in Computer Science, pages 163–166. Springer-Verlag, 2009.

[40] K. Korovin. Inst-Gen - A Modular Approach to Instantiation-Based Automated Reasoning. In A. Voronkov and C. Weidenbach, editors, *Programming Logics, Essays in Memory of Harald Ganzinger*, number 7797 in Lecture Notes in Computer Science, pages 239–270. Springer-Verlag, 2013.

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

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

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

[44] W.W. McCune. Solution of the Robbins Problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.

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

[46] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.

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

[48] 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.

[49] 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.

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

[51] J. Otten. Implementing Connection Calculi for First-order Modal Logics. In K. Korovin and S. Schulz, editors, *Proceedings of the 9th International Workshop on the Implementation of Logics*, number 22 in EPiC, pages 18–32, 2012.

[52] J. Otten. MleanCoP: A Connection Prover for First-Order Modal Logic. In S. Demri, D. Kapur, and C. Weidenbach, editors, *Proceedings of the 7th International Joint Conference on Automated Reasoning*, number 8562 in Lecture Notes in Artificial Intelligence, pages 269–276, 2014.

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

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

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

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

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

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

[59] 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.

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

[61] 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*, number 2 in EPiC, pages 1–11, 2010.

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

[63] 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.

[64] A. Reynolds, C. Tinelli, and L. de Moura. Finding Conflicting Instances of Quantified Formulas in SMT. In H. Chen, F. Lonsing, and M. Seidl, editors, *Proceedings of the 1st International Workshop on Quantification*, page To appear, 2014.

[65] A. Reynolds, C. Tinelli, A. Goel, and S. Krstic. Finite Model Finding in SMT. In N. Sharygina and H. Veith, editors, *Proceedings of the 25th International Conference on Computer Aided Verification*, number 8044 in Lecture Notes in Computer Science, pages 640–655. Springer-Verlag, 2013.

[66] A. Reynolds, C. Tinelli, A. Goel, S. Krstic, M. Deters, and C. Barrett. Quantifier Instantiation Techniques for Finite Model Finding in SMT. In M.P. Bonacina, editor, *Proceedings of the 24th International Conference on Automated Deduction*, number 7898 in Lecture Notes in Artificial Intelligence, pages 377–391. Springer-Verlag, 2013.

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

[68] P. Rümmer. A Constraint Sequent Calculus for First-Order Logic with Linear Integer Arithmetic. In I. Cervesato, H. Veith, and A. Voronkov, editors, *Proceedings of the 15th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, number 5330 in Lecture Notes in Artificial Intelligence, pages 274–289. Springer-Verlag, 2008.

[69] P. Rümmer. E-Matching with Free Variables. In N. Bjorner and A. Voronkov, editors, *Proceedings of the 18th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, number 7180 in Lecture Notes in Artificial Intelligence, pages 359–374. Springer-Verlag, 2012.

[70] S. Schulz. A Comparison of Different Techniques for Grounding Near-Propositional CNF Formulae. In S. Haller and G. Simmons, editors, *Proceedings of the 15th International FLAIRS Conference*, pages 72–76. AAAI Press, 2002.

[71] 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. Springer-Verlag, 2004.

[72] S. Schulz. Fingerprint Indexing for Paramodulation and Rewriting. In B. Gramlich, D. Miller, and U. Sattler, editors, *Proceedings of the 6th International Joint Conference on Automated Reasoning*, number 7364 in Lecture Notes in Artificial Intelligence, pages 477–483. Springer-Verlag, 2012.

[73] S. Schulz. Simple and Efficient Clause Subsumption with Feature Vector Indexing. In M.P. Bonacina and M. Stickel, editors, *Automated Reasoning and Mathematics: Essays in Memory*

of William W. McCune, number 7788 in Lecture Notes in Artificial Intelligence, pages 45–67. Springer-Verlag, 2013.

[74] A. Stump, G. Sutcliffe, and C. Tinelli. StarExec: a Cross-Community Infrastructure for Logic Solving. In S. Demri, D. Kapur, and C. Weidenbach, editors, *Proceedings of the 7th International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, page To appear, 2014.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[91] 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.

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

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

[94] 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.

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

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

[97] G. Sutcliffe. Proceedings of the CADE-23 ATP System Competition. Wroclaw, Poland, 2011.

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

[99] G. Sutcliffe. Proceedings of the 6th IJCAR ATP System Competition. Manchester, England, 2012.

[100] G. Sutcliffe. The CADE-23 Automated Theorem Proving System Competition - CASC-23. *AI*

Communications, 25(1):49–63, 2012.

[101] G. Sutcliffe. Proceedings of the 24th CADE ATP System Competition. Lake Placid, USA, 2013.

[102] G. Sutcliffe. The 6th IJCAR Automated Theorem Proving System Competition - CASC-J6. *AI Communications*, 26(2):211–223, 2013.

[103] G. Sutcliffe. The CADE-24 Automated Theorem Proving System Competition - CASC-24. *AI Communications*, 27(4):To appear, 2014.

[104] 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.

[105] 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.

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

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

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

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

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

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

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

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

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

[115] J. Urban. BliStr: The Blind Strategymaker. arXiv:1301.2683, 2013.

[116] U. Waldmann. Superposition and Chaining for Totally Ordered Divisible Abelian Groups. In R. Gore, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning*, number 2083 in Lecture Notes in Artificial Intelligence, pages 226–241. Springer-Verlag, 2001.

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