The CADE-19 ATP System Competition (CASC-19)

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

Abstract

In order to stimulate ATP system development, and to expose ATP systems to interested researchers, the CADE-19 ATP System Competition (CASC-19) will be held on 31st July 2003. CASC evaluates the performance of sound, fully automatic, classical 1st order ATP systems. The evaluation is in terms of:

- the number of problems solved, and
- the number of acceptable proofs and models produced, and
- the average runtime for problems solved;

in the context of:

- a bounded number of eligible problems, chosen from the TPTP Problem Library, and
- a specified time limit for each solution attempt.

1. Introduction

The CADE-19 ATP System Competition (CASC-19) will be held at CADE-19 in Miami, USA, on 31st July 2003. CASC evaluates the performance of sound, fully automatic, 1st order ATP systems. The evaluation is in terms of the number of problems solved, the number of acceptable proofs and models produced, and the average runtime for problems solved, in the context of a bounded number of eligible problems chosen from the TPTP Problem Library [SS98c], and a specified time limit for each solution attempt. CASC-19 is the eighth such ATP system competition [SS97a, SS98d, SS99, Sut00a, Sut01a, SSP02, SS03].

Twenty-three ATP systems and variants, listed in Table 1, have been entered into the various competition and demonstration divisions. The winners of the CASC-18 divisions have been automatically entered into those divisions, to provide benchmarks against which progress can be judged (the competition archive provides access to the systems' executables and source code).

The design and procedures of CASC-19 evolved from those of CASCs-13 to -18 [SS97b, SS98a, SS98b, Sut99, Sut00b, Sut01b, Sut02]. Important changes for CASC-19 are:

- The SAT division has been divided into two classes, one ranked by the number of problems solved (the Assurance class), and one ranked by the number of problems solved with an acceptable model output (the Model class).
- The numbers of problems in the categories in the various divisions is (roughly) proportional to the numbers of eligible problems than can be used in the categories, after taking into account the limitation on very similar problems.
- The rule disallowing the previous year's winner from being declared the winner again has been removed.

System	Divisions	Entrant	Institution
CARINE 0.7	MIX*	Paul Haroun	McGill University
CiME 2.01	UEQ	Evelyne Contejean (Benjamin Monate)	LRI Universite Paris-Sud
DCTP 1.3	MIX EPR	Gernot Stenz	Max-Planck-Institut für Informatik
DCTP 1.3-SAT	SAT		*DCTP 1.3 variant*
DCTP 10.2p	MIX FOF EPR	Gernot Stenz	Max-Planck-Institut für Informatik
DCTP 10.2p-SAT	SAT		*DCTP 10.2p variant*
E 0.8	MIX EPR UEQ	Stephan Schulz	Technische Universität München and RISC Linz
EP 0.8	MIX*		*E 0.8 variant*
E-SETHEO csp02	EPR		*CASC-18 EPR winner*
E-SETHEO csp03	MIX FOF EPR UEQ	Gernot Stenz (Reinhold Letz, Stephan Schulz)	Max-Planck-Institut für Informatik (Technische Universität München)
E-S'O csp03–SAT	SAT		*E-SETHEO csp03 variant*
Gandalf c-2.5-SAT	SAT		*CASC-18 SAT winner*
Gandalf c-2.6	MIX UEQ	Tanel Tammet	Tallinn Technical University
Gandalf c-2.6-PRF	MIX*		*Gandalf c-2.6 variant*
Gandalf c-2.6-SAT	SAT* EPR		*Gandalf c-2.6 variant*
MUSCADET 2.4	FOF	Dominique Pastre	Université René Descartes
Octopus N	MIX (demonstration)	Monty Newborn (Zongyan Wang)	McGill University
Otter 3.2	MIX* FOF UEQ	William McCune	Argonne National Laboratory
Paradox 1.0	SAT* EPR	Koen Claessen (Niklas Sörensson)	Chalmers University of Technology
THEO J2003	MIX	Monty Newborn (Zongyan Wang)	McGill University
Vampire 5.0	MIX* FOF		*CASC-18 MIX and FOF winner*
Vampire 6.0	MIX* FOF EPR UEQ	Andrei Voronkov (Alexandre Riazanov)	University of Manchester
Waldmeister 702	UEQ		*CASC-18 UEQ winner*
Waldmeister 703	UEQ	Thomas Hillenbrand (J-M. Gaillourdet, Bernd Löchner)	Max-Planck-Institut für Informatik, Universität Kaiserslautern

MIX* indicates participation in the MIX division Proof class. SAT* indicates participation in the SAT division Model class. See Section 2.1 for details.

Table 1: The CASC-19 Entrants

- In the MIX and SAT divisions, if a system wins both the Assurance and Proof/Model class, then a single unannotated trophy will be awarded.

The competition organizers are Geoff Sutcliffe and Christian Suttner. The competition is overseen by a panel of knowledgeable researchers who are not participating in the event; the panel members are Uli Furbach, Don Loveland, and Jeff Pelletier. The rules, deadlines, and specifications given here are absolute. Only the competition panel has the right to make exceptions. The competition will be run on computers provided by the Department of Computer Science at the University of Manchester. The CASC-19 WWW site provides access to resources used before, during, and after the event:

```
http://www.tptp.org/CASC/19
```

2. Divisions

CASC-19 is divided into divisions according to problem and system characteristics. There are five competition divisions in which systems are explicitly ranked, and a demonstration division in which systems demonstrate their abilities without being formally ranked. Entry into the competition divisions is subject to the following rules:

- ATP systems can be entered at only the division level.
- ATP systems can be entered into more than one division. A system that is not entered into a division is assumed to perform worse than the entered systems, for that type of problem.
- The ATP systems have to run on a single locally provided standard UNIX workstation (the *general hardware* - see Section 3.1). ATP systems that cannot run on the general hardware can be entered into the demonstration division (see Section 2.2).

2.1. Competition Divisions

- The **MIX** division: Mixed CNF Really-Non-Propositional Theorems
  Mixed means Horn and non-Horn problems, with or without equality, but not unit equality problems (see the UEQ Division below). *Really-Non-Propositional* means with an infinite Herbrand universe. The MIX Division has five problem categories:
  - The **HNE** category: Horn with No Equality
  - The **HEQ** category: Horn with some (but not pure) Equality
  - The **NNE** category: Non-Horn with No Equality
  - The **NEQ** category: Non-Horn with some (but not pure) Equality
  - The **PEQ** category: Pure Equality

  The MIX division has two classes:
  - The **Assurance** class: Ranked according to the number of problems solved (a "yes" output, giving an *assurance* of the existence of a proof).
  - The **Proof** class: Ranked according to the number of problems solved with an acceptable proof output on `stdout`. The competition panel judges whether or not each system's proof format is *acceptable*.
- The **FOF** division: Mixed FOF Non-Propositional Theorems
  The FOF Division has two problem categories:
  - The **FNE** category: FOF with No Equality
  - The **FEQ** category: FOF with Equality
- The **SAT** division: Mixed CNF Really-Non-Propositional Non-theorems
  The SAT Division has two problem categories:
  - The **SNE** category: SAT with No Equality
  - The **SEQ** category: SAT with Equality

  The SAT division has two classes:
  - The **Assurance** class: Ranked according to the number of problems solved (a "yes" output, giving an *assurance* of the existence of a model).
  - The **Model** class: Ranked according to the number of problems solved with an acceptable model output on `stdout`. The competition panel judges whether or not each system's model format is *acceptable*.
- The **EPR** division: CNF Effectively Propositional Theorems and Non-theorems.
  Effectively propositional means non-propositional with a finite Herbrand Universe. The EPR Division has two problem categories:

    - The **EPT** category: Effectively Propositional Theorems (unsatisfiable clauses)
    - The **EPS** category: Effectively Propositional non-theorems (Satisfiable clauses)
- The **UEQ** division: Unit Equality CNF Really-Non-Propositional Theorems

Section 3.2.1 explains what problems are eligible for use in each division and category.

2.2. Demonstration Division

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

The entry specifies which competition divisions' problems are to be used. The results are presented along with the competition divisions' results, but may not be comparable with those results.

3. Infrastructure

3.1. Hardware and Software

The general hardware is 45 P4 Dell Precision 330 workstations, each having:

- Intel P4 993MHz CPU
- 512MB memory
- Linux 2.4.9-34 operating system

3.2. Problems

3.2.1. Problem Selection

The problems are from the TPTP Problem Library, version v2.6.0. The TPTP version used for the competition is not released until after the system installation deadline.

The problems have to meet certain criteria to be eligible for selection:

- The TPTP uses system performance data to compute problem difficulty ratings, and from the ratings classifies problems as one of [SS01]:
    - Easy: Solvable by all state-of-the-art ATP systems
    - Difficult: Solvable by some state-of-the-art ATP systems
    - Unsolved: Solvable by no ATP systems
    - Open: Theorem-hood unknown

  Difficult problems with a rating in the range 0.21 to 0.99 are eligible. Performance data from systems submitted by the system submission deadline is used for computing the problem ratings for the TPTP version used for the competition.
- The TPTP distinguishes versions of problems as one of standard, incomplete, augmented, especial, or biased. All except biased problems are eligible.

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

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

3.2.2. Number of Problems

The minimal numbers of problems that have to be used in each division and category, to ensure sufficient confidence in the competition results, are determined from the numbers of eligible problems in each division and category [GS96] (the competition organizers have to ensure that there is sufficient CPU time available to run the ATP systems on this minimal number of problems). The minimal numbers of problems is used in determining the CPU time limit imposed on each solution attempt - see Section 3.3.

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

$$\text{Number of problems} = \frac{\text{Number of workstations} \times \text{Time allocated}}{\text{Number of ATP systems} \times \text{CPU time limit}}$$

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

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

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

3.2.3. Problem Preparation

In order to ensure that no system receives an advantage or disadvantage due to the specific presentation of the problems in the TPTP, the tptp2X utility (distributed with the TPTP) is used to:

- rename all predicate and function symbols to meaningless symbols
- randomly reorder the clauses and literals in CNF problems
- randomly reorder the formulae in FOF problems
- randomly reverse the unit equalities in UEQ problems
- remove equality axioms that are not needed by some of the ATP systems
- add equality axioms that are needed by some of the ATP systems
- output the problems in the formats required by the ATP systems. (The clause type information, one of `axiom`, `hypothesis`, or `conjecture`, may be included in the final output of each formula.)

Further, to prevent systems from recognizing problems from their file names, symbolic links are made to the selected problems, using names of the form `CCCNNN-1.p` for the symbolic links, with `NNN` running from `001` to the number of problems in the respective division or category. The problems are specified to the ATP systems using the symbolic link names.

In the demonstration division the same problems are used as for the competition divisions, with the same tptp2X transformations applied. However, the original file names are retained.

3.3. Time Limits

In the competition divisions, CPU and wall clock time limits are imposed on each solution attempt. A minimal CPU time limit of 240 seconds is used. The maximal CPU time limit is determined using the relationship used for determining the number of problems, with the minimal number of problems as the "Number of problems". 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 prevent very high memory usage that causes swapping. The wall clock time limit is double the CPU time limit.

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

4. Performance Evaluation

At some time before the competition, all systems in the competition divisions are tested for soundness. Non-theorems (satisfiable variants of the eligible problems, e.g., without the conjecture clause, and satisfiable problems selected from the TPTP) are submitted to the systems in the MIX, FOF, EPR, and UEQ divisions, and theorems (selected from the TPTP) are submitted to the systems in the SAT and EPR divisions. Finding a proof of a non-theorem or a disproof of a theorem indicates unsoundness. If an ATP system fails the soundness testing it must be repaired by unsoundness repair deadline or be withdrawn. The soundness testing has a secondary aim of eliminating the possibility of an ATP system simply delaying for some amount of time and then claiming to have found a solution. For systems running on entrant supplied hardware in the demonstration division the entrant must perform the soundness testing and report the results to the competition organizers.

During the competition, for each ATP system, for each problem attempted, three items of data are recorded: whether or not a solution was found, the CPU time taken, and whether or not a solution (proof or model) was output on `stdout`. In the MIX division proof class and the SAT division model class, the systems are ranked according to the number of problems solved with an acceptable solution output. In all other cases the systems are ranked according to the numbers of problems solved. If there is a tie according to these rankings then the tied systems are ranked according to their average CPU times over problems solved. Division and class winners are announced and prizes are awarded.

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. If a system is found to be unsound, 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 from the winner of the MIX division proof class, and the models from the winner of the SAT division model class, 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 reported in the journal paper about the competition.

5. System Properties

The precomputation and storage of any information specifically about TPTP problems is not allowed. Strategies and strategy selection based on the characteristics of a few specific TPTP problems is not allowed, i.e., strategies and strategy selection must be general purpose and expected to extend usefully to new unseen problems. If automatic strategy learning procedures are used, the learning must ensure that sufficient generalization is obtained, and that no learning at the individual problem level is performed.

For every problem solved, the system's solution process has to be reproducible by running the system again.

With the exception of the MIX division proof class and SAT division model class, the ATP systems are not required to output solutions (proofs or models). However, systems that do output solutions to `stdout` are highlighted in the presentation of results.

Entrants must install their systems on the general hardware, and ensure that their systems execute in the competition environment, according to the checks listed below. Entrants are advised to perform these checks well in advance of the system installation deadline. This gives the competition organizers time to help resolve any difficulties encountered.

5.1. System Checks

The ATP systems have to be executable by a single command line, using an absolute path to the executable that may not be in the current directory. The command line arguments are the absolute path name for a symbolic link as the problem file name, the time limit (if required by the entrant), and entrant specified system switches (the same for all problems). No shell features, such as input or output redirection, may be used in the command line. No assumptions may be made about the format of the problem file name.

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

```
prompt> pwd
/home/tptp
prompt> which MyATPSystem
/home/tptp/bin/MyATPSystem
prompt> /home/tptp/bin/MyATPSystem /home/tptp/TPTP/Problems/GRP/GRP001-1.p
Proof found in 147 seconds.
```

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

```
prompt> cd /home/tptp/tmp
prompt> ln -s /home/tptp/TPTP/Problems/GRP/GRP001-1.p CCC001-1.p
prompt> cd /home/tptp
prompt> /home/tptp/bin/MyATPSystem /home/tptp/tmp/CCC001-1.p
Proof found in 147 seconds.
```

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

```
prompt> cp /home/tptp/TPTP/Problems/PUZ/PUZ031-1.p _foo-Blah
prompt> /home/tptp/bin/MyATPSystem _foo-Blah
Proof found in 147 seconds.
```

The ATP systems that run on the general hardware have to be interruptable by a `SIGXCPU` signal, so that the CPU time limit can be imposed on each solution attempt, and interruptable by a `SIGALRM` signal, so that the wall clock time limit can be imposed on each solution attempt. For systems that create multiple processes, the signal is sent first to the process at the top of the hierarchy, then one second later to all processes in the hierarchy. Any orphan processes are killed after that, using `SIGKILL`.The default action on receiving these signals is to exit (thus complying with the time limit, as required), but systems may catch the signals and exit of their own accord. If a system runs past a time limit this is noticed in the timing data, and the system is considered to have not solved that problem.

- Check: The ATP system can run under the `TreeLimitedRun` program (sources are available from the CASC-19 WWW site) For example:

```
prompt> which TreeLimitedRun
/home/tptp/bin/TreeLimitedRun
prompt> /home/tptp/bin/TreeLimitedRun –
q0 200 400 /home/tptp/bin/MyATPSystem /home/tptp/tmp/CCC001-1.p
TreeLimitedRun: ----------------------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 200s
TreeLimitedRun: WC  time limit is 400s
TreeLimitedRun: PID is 4867
TreeLimitedRun: ----------------------------------------------------------
Proof found in 147 seconds.
FINAL WATCH: 147.8 CPU 150.0 WC
```

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

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

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

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

When terminating of their own accord, the ATP systems have to output a distinguished string (specified by the entrant) to `stdout` indicating the result, one of:

- A solution exists (for CNF problems, the clause set is unsatisfiable, for FOF problems, the conjecture is a theorem)
- No solution exists (for CNF problems, the clause set is satisfiable, for FOF problems, the conjecture is a non-theorem)
- No conclusion reached

When outputing proofs for MIX division's proof class, and models for the SAT division's model class, the start and end of the solution must be identified by distinguished strings (specified by the entrant). These pairs of strings must be different for proofs and models.

- Check: The system outputs a distinguished string when terminating of its own accord. For example, here the entrant has specified that the distinguished string `Proof found` indicates that a solution exists. If appropriate, similar checks should be made for the cases where no solution exists and where no conclusion is reached.

```
prompt> /home/tptp/bin/TreeLimitedRun –
q0 200 400 /home/tptp/bin/MyATPSystem /home/tptp/tmp/CCC001-1.p
TreeLimitedRun: ----------------------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 200s
TreeLimitedRun: WC  time limit is 400s
TreeLimitedRun: PID is 5827
TreeLimitedRun: ----------------------------------------------------------
Proof found in 147 seconds.
FINAL WATCH: 147.8 CPU 150.0 WC
```

- Check: The system outputs distinguished strings at the start and end of its solution. For example, here the entrant has specified that the distinguished strings `START OF PROOF` and `END OF PROOF` identify the start and end of the solution.

```
prompt> /home/tptp/bin/TreeLimitedRun –q0 200 400 /home/tptp/bin/MyATPSystem –
output_proof /home/tptp/tmp/CCC001-1.p
TreeLimitedRun: ----------------------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 200s
TreeLimitedRun: WC  time limit is 400s
TreeLimitedRun: PID is 5827
TreeLimitedRun: ----------------------------------------------------------
Proof found in 147 seconds.
START OF PROOF
     ... acceptable proof here ...
END OF PROOF
FINAL WATCH: 147.8 CPU 150.0 WC
```

If an ATP system terminates of its own accord, it may not leave any temporary or other output files. If an ATP system is terminated by a `SIGXCPU` or `SIGALRM`, it may not leave any temporary or other output files anywhere other than in `/tmp`.

Multiple copies of the ATP systems have to be executable concurrently on different machines but in the same (NFS cross mounted) directory. It is therefore necessary to avoid producing temporary files that do not have unique names, with respect to the machines and other processes. An adequate solution is a file name including the host machine name and the process id.

For practical reasons excessive output from the ATP systems is not allowed. A limit, dependent on the disk space available, is imposed on the amount of `stdout` and `stderr` output that can be produced. The limit is at least 10KB per problem (averaged over all problems so that it is possible to produce *some* long proofs).

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

```
prompt> pwd
/home/tptp
prompt> /home/tptp/bin/TreeLimitedRun –
q0 200 400 /home/tptp/bin/MyATPSystem /home/tptp/tmp/CCC001-1.p
TreeLimitedRun: ----------------------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 200s
TreeLimitedRun: WC  time limit is 400s
TreeLimitedRun: PID is 13526
TreeLimitedRun: ----------------------------------------------------------
Proof found in 147 seconds.
FINAL WATCH: 147.8 CPU 150.0 WC
prompt> ls /home/tptp
     ... no temporary or other files left here ...
prompt> ls /home/tptp/bin
     ... no temporary or other files left here ...
prompt> ls /home/tptp/tmp
     ... no temporary or other files left here ...
prompt> ls /home/tptp/TPTP/Problems/PUZ
     ... no temporary or other files left here ...
prompt> ls /tmp
     ... no temporary or other files left here by decent systems ...
```

- Check: Multiple concurrent executions do not clash. For example:

```
prompt> (/bin/time /home/tptp/bin/TreeLimitedRun –
q0 200 400 /home/tptp/bin/MyATPSystem /home/tptp/tmp/CCC001–
1.p) & (/bin/time /home/tptp/bin/TreeLimitedRun –
q0 200 400 /home/tptp/bin/MyATPSystem /home/tptp/tmp/CCC001–1.p)
TreeLimitedRun: ----------------------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 200s
TreeLimitedRun: WC  time limit is 400s
TreeLimitedRun: PID is 5827
TreeLimitedRun: ----------------------------------------------------------
TreeLimitedRun: ----------------------------------------------------------
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 200s
TreeLimitedRun: WC  time limit is 400s
TreeLimitedRun: PID is 5829
TreeLimitedRun: ----------------------------------------------------------
Proof found in 147 seconds.
FINAL WATCH: 147.8 CPU 150.0 WC

Proof found in 147 seconds.
FINAL WATCH: 147.8 CPU 150.0 WC
```

5.2. System Delivery

Access to the general hardware (or equivalent) is available from the general hardware access deadline. For systems running on the general hardware, entrants have to deliver an installation package to the competition organizers by the installation deadline. The installation package must be a `.tar.gz` file containing the system source code, any other files required for installation, and a `ReadMe` file. The `ReadMe` file must contain:

- Instructions for installation
- Instructions for executing the system
    - Format of problem files, in the form of `tptp2X` format and transformation parameters.
    - Command line, using `%s` and `%d` to indicate where the problem file name and CPU time limit must appear.
- The distinguished strings output to indicate
    - The result
    - The start of solution output
    - The end of solution output

The installation procedure may require changing path variables, invoking `make` or something similar, etc, but nothing unreasonably complicated. All system binaries must be created in the installation process; they cannot be delivered as part of the installation package. The system is reinstalled onto the general hardware by the competition organizers, following the instructions in the `ReadMe` file. Installation failures before the installation deadline are passed back to the entrant. After the installation deadline access to the general hardware is denied, and no further changes or late systems are accepted.

For systems running on entrant supplied hardware in the demonstration division, the systems are installed on the respective hardware by the entrants.

5.3. System Execution

Execution of the ATP systems on the general hardware is controlled by a `perl` script, provided by the competition organizers. The jobs are queued onto the workstations so that each workstation is running one job at a time. All attempts at the Nth problems in all the divisions and categories are started before any attempts at the (N+1)th problems.

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

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

6. Entry Procedures

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

Entering many similar versions of the same system is deprecated. Entrants may be required to limit the number of system versions that they enter. The division winners from the previous CASC are automatically entered into their divisions, to provide benchmarks against which progress can be judged. Systems entered in the MIX division are automatically ranked in the assurance class, and are ranked in the proof class if they output acceptable proofs. Systems entered in the SAT division are automatically ranked in the assurance class, and are ranked in the model class if they output acceptable models. After the competition all systems' source code is made publically available on the CASC WWW site.

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

6.1. System Description

A system description has to be provided for each ATP system entered, using the supplied HTML schema. The system description must fit onto two pages, using 12pt times font. The schema has the following sections:

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

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

6.2. Sample Solutions

For systems in the MIX division proof class and the SAT division model class, representative sample solutions must be emailed to the competition organizers before the sample solutions deadline. Proof samples must include a proof for `SYN075-1`, and model samples must include a model for `MGT031-1`. The sample solutions must illustrate the use of all inference rules. A key must be provided if any non-obvious abbreviations for inference rules or other information are used.

The competition panel decides whether or not the proofs and models are acceptable.

7. The ATP Systems

These system descriptions were written by the entrants.

7.1. CARINE 0.7

Paul Haroun, Monty Newborn
McGill University Canada
`pharoun@cs.mcgill.ca`

7.1.1. Architecture

CARINE is a first-order classical logic ATP system intended for experimental purposes. It is based on ideas from THEO [New01]. The inference rules implemented so far are binary resolution and binary factoring.

7.1.2. Implementation

CARINE is implemented in ANSI-C and currently runs under SunOS and Microsoft Windows. It has not yet been tested on Linux but it works in the Linux emulation layer: Cygwin. CARINE relies heavily on tables that are either 1) representations of graphs (i.e. dependency/adjacency matrices) or 2) lookup tables resulting from memoization and dynamic programming techniques. The graphs are relations that are formed from the information gathered from the base clauses. These relations are mainly groupings of the base clauses or their literals according to certain common characteristics. The tables are allocated dynamically based on the input and remain the same size throughout the search. A finite automaton, also implemented as a table, is used to store unit clauses.

The system will be available at:

`http://www.cs.mcgill.ca/~pharoun/atp_carine_site`

7.1.3. Strategies

CARINE is based on a semi-linear resolution search. It performs the implemented inference rules in an iteratively deepening depth first search until a unit clause is obtained. The unit clause is then evaluated and if it passes the tests it is resolved with all the stored unit clauses and if none yields the empty clause then it is added to the unit clauses table. The main implemented strategies are delayed clause construction, time slicing, extended depth search and memoization.

Delayed clause construction is very useful when clauses contain many literals and/or many terms. Only an "interesting" clause is actually constructed. This strategy is the heart of the system.

Input parameters are used to control the search. Time slicing, if turned on, would calculate time slices based on the evaluation of the input clauses at the beginning of the search. Each time slice uses different parameters to control the search. Currently, only two time slices are applicable. In the first slice an aggressive search is performed and in the second time slice a more conservative search is performed.

Extended depth search is performed based on certain heuristics. Certain paths are explored deeper than the iteration depth when the heuristics apply.

7.1.4. Expected Competition Performance

CARINE is a system built mainly for experimental purposes. It is still in its elementary stage.

7.2. CiME 2.01

Evelyne Contejean, Benjamin Monate
Université Paris-Sud, France
`contejea@lri.fr monate@lix.polytechnique.fr`

7.2.1. Architecture

CiME [CM+00] is intended to be a toolkit, which contains nowadays the following features:

- An interactive toplevel to allow naming of objects and call to various functions.
- Solving Diophantine constraints over finite intervals
- Solving Presburger constraints
- Unification modulo disjoint classical equational theories, such as the free theory, C, AC, ACU, Abelian Groups, Boolean Rings
- String Rewriting Systems, KB completion.
- Parameterized String Rewriting Systems confluence checking
- Term Rewriting Systems, possibly with commutative or associative-commutative symbols, KB or ordered completion.
- Termination of TRSs using standard or dependency pairs criteria, automatic generation of termination orderings based on polynomial interpretations, including weak orderings for dependency pairs criteria.

The ordered completion of term rewriting systems will be used during the competion to attempt to solve unification problems, that is problems in the UEQ division [CM96].

7.2.2. Implementation

CiME2 is fully written in Objective CAML, a functional language of the ML family developed in the CRISTAL project at INRIA Rocquencourt. CiME2 is available at:

`http://cime.lri.fr/`

as binaries for SPARC workstations running Solaris (at least version 2.6) and for pentium PCs running Linux, and its sources are available by read-only anynomous CVS.

7.2.3. Strategies

There are two distinct kinds of strategies to perform completion:

- The first one is, given an equation, how to choose its orientation when it becomes a rule? The choice is made thanks to an ordering which has usually to be provided by the user. During the competition, this ordering is chosen among RPO, KBO or a polynomial ordering according to an analysis of the input axioms, where the known axioms are among classical ones.
- The second one is which inference rule has to be applied to the system, among orienting an equation into a rule and computing critical pairs. Each of these choices is given a weight, and the lowest weighted choice is made. The weight depends on the size of the involved equations/rules and on how "old" they are.

7.2.4. Expected Competition Performance

This will be the second participation of CiME2 in CASC, in the UEQ division.

Acknowledgments: Claude Marché and Xavier Urbain contributed to the development of CiME 2.

7.3. DCTP 1.3 and 10.2p

Gernot Stenz
Max-Planck-Institut für Informatik, Germany
`stenz@mpi-sb.mpg.de`

7.3.1. Architecture

DCTP 1.3 [Ste02a] is an automated theorem prover for first order clause logic. It is an implementation of the disconnection calculus described in [Bil96,LS01a,Ste02b]. The disconnection calculus is a proof confluent and inherently cut-free tableau calculus with a weak connectedness condition. The inherently depth-first proof search is guided by a literal selection based on literal instantiatedness or literal complexity and a heavily parameterised link selection. The pruning mechanisms mostly rely on different forms of variant deletion and unit based strategies. Additionally the calculus has been augmented by full tableau pruning.

The new DCTP 1.3 has been enhanced with respect to clause preprocessing, selection functions and closure heuristics. Most prominent among the improvements is the introduction of a unification index for finding connections, which also replaces the connection graph hitherto used.

DCTP 10.2p is a strategy parallel version using the technology of E-SETHEO [SW99] to combine several different strategies based on DCTP 1.3.

7.3.2. Implementation

DCTP 1.3 has been implemented as a monolithic system in the Bigloo dialect of the Scheme language. The most important data structures are perfect discrimination trees, which are used in many variations. DCTP 10.2p has been derived of the Perl implementation of E-SETHEO and includes DCTP 1.3 as well as additional components written in Prolog and Shell tools. Both versions run under Solaris and Linux.

7.3.3. Strategies

DCTP 1.3 is a single strategy prover. Individual strategies are started by DCTP 10.2p using the schedule based resource allocation scheme known from the E-SETHEO system. Of course, different schedules have been precomputed for the syntactic problem classes. The problem classes are more or less identical with the sub-classes of the competition organisers. We have no idea whether or not this conflicts with the organisers' tuning restrictions.

7.3.4. Expected Competition Performance

We expect both DCTP 1.3 and DCTP 10.2p to perform reasonably well, in particular in the EPR (in any case) and SAT (depending on the selection of problems for the competition) categories.

7.4. E 0.8 and EP 0.8

Stephan Schulz
Technische Universität München, Germany, and RISC-Linz, Johannes Kepler Universität, Austria
`schulz@informatik.tu-muenchen.de`

7.4.1. Architecture

E 0.8 [Sch01,Sch02] is a purely equational theorem prover. The calculus used by E combines superposition (with selection of negative literals) and rewriting. No special rules for non-equational literals have been implemented, i.e. resolution is simulated via paramodulation and equality resolution. E also implements AC redundancy elimination and AC simplification for dynamically recognized associative and commutative equational theories, as well as pseudo-splitting for clauses. It now also unfolds equational definitions in a preprocessing stage.

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

An automatic mode can select literal selection strategy, term ordering (different versions of KBO and LPO), and search heuristic based on simple problem characteristics.

EP 0.8 is just a combination of E 0.8 in verbose mode and a proof analysis tool extracting the used inference steps.

7.4.2. Implementation

E is implemented in ANSI C, using the GNU C compiler. The most outstanding feature is the global sharing of rewrite steps. Contrary to earlier version of E, which destructively changed all shared instances of a term, the latest version only adds a rewrite link from the rewritten to the new term. In effect, E is caching rewrite operations as long as sufficient memory is available. A second important feature is the use of perfect discrimination trees with age and size constraints for rewriting and unit-subsumption.

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

```
http://www4.informatik.tu-muenchen.de/~schulz/WORK/eprover.html
```

EP 0.8 is a simple Bourne shell script calling E and the postprocessor in a pipeline.

7.4.3. Strategies

E's automatic mode is optimized for performance on TPTP 2.5.1. The optimization is based on a fairly large number of test runs and consists of the selection of one of about 50 different strategies for each problem. All test runs have been performed on SUN Ultra 60/300 machines with a time limit of 120 seconds (or roughly equivalent configurations). All individual strategies are general purpose, the worst one solves about 45% of TPTP 2.5.1, the best one 55%.

E distinguishes problem classes based on a number of features, all of which have between two and four possible values. These are:

- Is the most general non-negative clause unit, Horn, or Non-Horn?
- Is the most general negative clause unit or non-unit?
- Are all negative clauses unit clauses?
- Are all literals equality literals, are some literals equality literals, or is the problem non-equational?
- Are there only a few, some, or many positive non-ground unit clauses among the axioms?
- Are all goals (negative clauses) ground?
- Are there a few, some, or many clauses in the problem?
- Are there a few, some, or many literals?
- Are there a few, some, or many (sub)terms?
- Are there a few, some or many positive ground unit clauses among the axioms?
- Is the maximum arity of any function symbol 0, 1, 2, or greater?

Wherever there is a selection of few, some, and many of a certain entity, the limits are selected automatically with the aim of splitting the set of clauses into three sets of approximately equal size based on this one feature.

For each non-empty class, we assign the most general candidate heuristic that solves the same number of problems on this class as the best heuristic on this class does. Empty classes are assigned the globally best heuristic. Typically, most selected heuristics are assigned to more than one class.

7.4.4. Expected Competition Performance

In the last year, E performed well in the MIX category of CASC and came in third in the UEQ division. We believe that E will again be among the strongest provers in the MIX category, in particular due to its good performance for Horn problems. In UEQ, E will probably be beaten only by Waldmeister, and possibly, E-SETHEO (which incorporates E).

EP 0.8 will be hampered by the fact that it has to analyse the inference step listing, an operation that typically is about as expensive as the proof search itself. Nevertheless, it should be competitive among the MIX* systems.

7.5. E-SETHEO csp02

Reinhold Letz, Stephan Schulz, Gernot Stenz
Technische Universität München, Germany
`{letz,schulz,stenz}@informatik.tu-muenchen.de`

7.5.1. Architecture

E-SETHEO is a compositional theorem prover for formulae in first order clause logic, combining the systems E [Sch01], DCTP [Ste02] and SETHEO [MI+97]. It incorporates different calculi and proof procedures like superposition, model elimination and semantic trees (the DPLL procedure). Furthermore, the system contains transformation techniques which may split the formula into independent subparts or which may perform a ground instantiation. Finally, advanced methods for controlling and optimizing the combination of the subsystems are applied. The first-order variant of E-SETHEO no longer uses Flotter [WGR96] as a preprocessing module for transforming non-clausal formulae to clausal form. Instead, a more primitive normal form transformation is employed.

Since version 99csp of E-SETHEO, the different strategies are run sequentially, one after the other. E-SETHEO csp02 incorporates the new version of the disconnection prover DCTP with integrated equality handling as a new strategy as well as a new version of the E prover. The new Scheme version of SETHEO that is in use features local unit failure caching [LS01b] and lazy root paramodulation, an optimisation of lazy paramodulation which is complete in the Horn case [LS02].

7.5.2. Implementation

According to the diversity of the contained systems, the modules of E-SETHEO are implemented in different programming languages like C, Prolog, Scheme, and Shell tools.

The program runs under Solaris and, with a little luck, under Linux, too. Sources are available from the authors.

7.5.3. Strategies

Individual strategies are started by E-SETHEO depending on the allocation of resources to the different strategies, so-called schedules, which have been computed from experimental data using machine learning techniques as described in [SW99]. Schedule selection depends on syntactic characteristics of the input formula such as the Horn-ness of formulae, whether a problem contains equality literals or whether the formula is in the Bernays-Schönfinkel class. The problem classes are more or less identical with the sub-classes of the competition. We have no idea whether or not this conflicts with the organisers' tuning restrictions.

7.5.4. Expected Competition Performance

E-SETHEO csp02 was the CASC-18 EPR division winner.

7.6. E-SETHEO csp03

Reinhold Letz, Stephan Schulz, Gernot Stenz
Technische Universität München, Germany, Max-Planck-Institut für Informatik, Germany, and RISC-Linz, Johannes Kepler Universität, Austria
`{letz,schulz,stenz}@informatik.tu-muenchen.de`

7.6.1. Architecture

E-SETHEO is a compositional theorem prover for formulae in first order clause logic, combining the systems E [Sch01], DCTP [Ste02] and SETHEO [MI+97]. It incorporates different calculi and proof procedures like superposition, model elimination and semantic trees (the DPLL procedure). Furthermore, the system contains transformation techniques which may split the formula into independent subparts or which may perform a ground instantiation. Finally, advanced methods for controlling and optimizing the combination of the subsystems are applied. The first-order variant of E-SETHEO no longer uses Flotter [WGR96] as a preprocessing module for transforming non-clausal formulae to clausal form. Instead, a more primitive normal form transformation is employed.

Since version 99csp of E-SETHEO, the different strategies are run sequentially, one after the other. E-SETHEO csp03 incorporates the new version of the disconnection prover DCTP with new preprocessing and heuristics as a new strategy, as well as a new version of the E prover. The new Scheme version of SETHEO that is in use features local unit failure caching [LS01] and lazy root paramodulation, an optimisation of lazy paramodulation which is complete in the Horn case [LS02]. Other than that (and a new resource distribution scheme), E-SETHEO csp03 is identical to E-SETHEO csp02.

7.6.2. Implementation

According to the diversity of the contained systems, the modules of E-SETHEO are implemented in different programming languages like C, Prolog, Scheme, and Shell tools.

The program runs under Solaris and Linux. Sources are available from the authors.

7.6.3. Strategies

Individual strategies are started by E-SETHEO depending on the allocation of resources to the different strategies, so-called schedules, which have been computed from experimental data using machine learning techniques as described in [SW99]. Schedule selection depends on syntactic characteristics of the input formula such as the Horn-ness of formulae, whether a problem contains equality literals, or whether the formula is in the Bernays-Schönfinkel class. The problem classes are more or less identical with the sub-classes of the competition. We have no idea whether or not this conflicts with the organisers' tuning restrictions.

7.6.4. Expected Competition Performance

We expect E-SETHEO to perform well in all categories it participates in.

7.7. Gandalf c-2.5

Tanel Tammet
Tallinn Technical University, Estonia, and Safelogic AB, Sweden
`tammet@cc.ttu.ee`

7.7.1. Architecture

Gandalf [Tam97,Tam98] is a family of automated theorem provers, including classical, type theory, intuitionistic and linear logic provers, plus finite a model builder. The version c-2.5 contains the classical logic prover for clause form input and the finite model builder. One distinguishing feature of Gandalf is that it contains a large number of different search strategies and is capable of automatically selecting suitable strategies and experimenting with these strategies.

Gandalf is available under GPL. There exists a separate commercial version of Gandalf, called G, developed and distributed by Safelogic AB (`www.safelogic.se`), which contains numerous additions, strategies, and optimisations, aimed specifically at verification of large systems.

The finite model building component of Gandalf uses the Zchaff propositional logic solver by L.Zhang [MM+01] as an external program called by Gandalf. Zchaff is not free, although it can be used freely for research purposes. Gandalf is not optimised for Zchaff or linked together with it: Zchaff can be freely replaced by other satisfiability checkers.

7.7.2. Implementation

Gandalf is implemented in Scheme and compiled to C using the Hobbit Scheme-to-C compiler. Version scm5d6 of the Scheme interpreter scm by A.Jaffer is used as the underlying Scheme system.

Gandalf has been tested on Linux, Solaris and MS Windows under Cygwin.

The propositional satisifiability checker Zchaff used by Gandalf during finite model building is implemented by L.Zhang in C++.

Gandalf should be publicly available at:

`http://www.ttu.ee/it/gandalf`

7.7.3. Strategies

One of the basic ideas used in Gandalf is time-slicing: Gandalf typically runs a number of searches with different strategies one after another, until either the proof is found or time runs out. Also, during each specific run Gandalf typically modifies its strategy as the time limit for this run starts coming closer. Selected clauses from unsuccessful runs are sometimes used in later runs.

In the normal mode Gandalf attempts to find only unsatisfiability. It has to be called with a -sat flag to find satisfiability. Gandalf selects the strategy list according to the following criteria:

- **Unsatisfiability checking**. Gandalf selects the basic strategies from the following list: hyperresolution, binary sos resolution, unit resolution, ordered resolution (term-depth based, literal size based and polarity plus literal size and structure based).

  Each strategy may be iterated over a limit on term depth. For clause sets containing equality, some strategies are tried with both the Knuth-Bendix ordering and recursive path ordering, as well as with several different ordering principles of function symbols for these orderings.

  Typically Gandalf selects one or two strategies to iterate over the term depth limit and one or two strategies to iterate over the selection of equality orderings. At the second half of each strategy run Gandalf imposes additional restrictions, like introducing unit restriction and switching over to strict best-first clause selection.

  The strategy list selection criteria for a particular problem is based on following:

- Problem class from TPTP: UEQ, PEQ, HNE, HEQ, NEQ, NNE. This strictly determines the list of basic strategies. The following criteria determine relative amount of time given to each strategy.
- Problem size. A problem is classified either as small, medium, or big, according to the number of clauses in the problem. For bigger problems, the set of support strategy gets relatively more time than other strategies.
- Percentage of clauses which can be ordered by term depth: small, medium, or all. For bigger percentages, the term depth ordering gets relatively more time than other strategies.

- **Satisfiability checking**. The following strategies are run:
  - Finite model building by incremental search through function symbol interpretations.
  - Ordered binary resolution (term depth): only for problems not containing equality.
  - Finite model building using MACE-style flattening and external propositional prover.
- **Satisfiability/unsatisfiability checking** for essentially propositional problems. The following strategies are run:
  - Unsatisfiability search by resolution.
  - Satisfiability/unsatisfiability search by propositonal saturation.
  - Satisfiability search for small models by propositonal saturation.

7.7.4. Expected Competition Performance

Gandalf c-2.5-SAT was the CASC-18 SAT division winner.

7.8. Gandalf c-2.6

Tanel Tammet
Tallinn Technical University, Estonia
`tammet@cc.ttu.ee`

7.8.1. Architecture

Gandalf [Tam97,Tam98] is a family of automated theorem provers, including classical, type theory, intuitionistic and linear logic provers, plus finite a model builder. The version c-2.6 contains the classical logic prover for clause form input and the finite model builder. One distinguishing feature of Gandalf is that it contains a large number of different search strategies and is capable of automatically selecting suitable strategies and experimenting with these strategies.

The finite model building component of Gandalf uses the Zchaff propositional logic solver by L.Zhang [MM+01] as an external program called by Gandalf. Zchaff is not free, although it can be used freely for research purposes. Gandalf is not optimised for Zchaff or linked together with it: Zchaff can be freely replaced by other satisfiability checkers.

7.8.2. Implementation

Gandalf is implemented in Scheme and compiled to C using the Hobbit Scheme-to-C compiler. Version scm5d6 of the Scheme interpreter scm by A.Jaffer is used as the underlying Scheme system. Zchaff is implemented in C++.

Gandalf has been tested on Linux, Solaris, and MS Windows under Cygwin.

Gandalf is available under GPL from:

    http://www.ttu.ee/it/gandalf

7.8.3. Strategies

One of the basic ideas used in Gandalf is time-slicing: Gandalf typically runs a number of searches with different strategies one after another, until either the proof is found or time runs out. Also, during each specific run Gandalf typically modifies its strategy as the time limit for this run starts coming closer. Selected clauses from unsuccessful runs are sometimes used in later runs.

In the normal mode Gandalf attempts to find only unsatisfiability. It has to be called with a -sat flag to find satisfiability. Gandalf selects the strategy list according to the following criteria:

- **Unsatisfiability checking**. Gandalf selects the basic strategies from the following list: hyperresolution, binary sos resolution, unit resolution, ordered resolution (term-depth based, literal size based and polarity plus literal size and structure based).

  Each strategy may be iterated over a limit on term depth. For clause sets containing equality, some strategies are tried with both the Knuth-Bendix ordering and recursive path ordering, as well as with several different ordering principles of function symbols for these orderings.

  Typically Gandalf selects one or two strategies to iterate over the term depth limit and one or two strategies to iterate over the selection of equality orderings. At the second half of each strategy run Gandalf imposes additional restrictions, like introducing unit restriction and switching over to strict best-first clause selection.

  The strategy list selection criteria for a particular problem is based on the following:
  - Problem class from TPTP: UEQ, PEQ, HNE, HEQ, NEQ, NNE. This strictly determines the list of basic strategies. The following criteria determine relative amount of time given to each strategy.
  - Problem size. A problem is classified either as small, medium, or big, according to the number of clauses in the problem. For bigger problems, the set of support strategy gets relatively more time than other strategies.
  - Percentage of clauses which can be ordered by term depth: small, medium, or all. For bigger percentages, the term depth ordering gets relatively more time than other strategies.
- **Satisfiability checking**. The following strategies are run:
  - Finite model building by incremental search through function symbol interpretations.
  - Ordered binary resolution (term depth): only for problems not containing equality.
  - Finite model building using MACE-style flattening and the external propositional prover.
- Satisfiability/unsatisfiability checking for essentially propositional problems. The following strategies are run:
  - Unsatisfiability search by resolution.
  - Satisfiability/unsatisfiability search by propositonal saturation.
  - Satisfiability search for small models by propositonal saturation.

7.8.4. Expected Competition Performance

We expect Gandalf to be among the best provers in most of the main categories it competes in.

7.9. MUSCADET 2.4

Dominique Pastre
Université René Descartes (Paris 5), France
`pastre@math-info.univ-paris5.fr`

7.9.1. Architecture

The MUSCADET theorem prover is a knowledge-based system. It is based on Natural Deduction, following the terminology of [Ble71] and [Pas78], and uses methods that resemble those used by humans. It is composed of an inference engine, which interprets and executes rules, and of one or several bases of facts, which are the internal representation of "theorems to be proved". Rules are either universal and put into the system, or built by the system itself by metarules 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.

7.9.2. Implementation

MUSCADET 2 [Pas02b] is implemented in SWI-Prolog. Rules are written as declarative Prolog clauses. Metarules are written as sets of Prolog clauses, more or less declarative. The inference engine includes the Prolog interpreter and some procedural Prolog clauses. MUSCADET 2.4 is available from:

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

7.9.3. Strategies

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

The system is also able to work with second order statements. It may also receive knowledge and know-how for a specific domain from a human user; see [Pas89] and [Pas93]. These two possibilities are not used while working with the TPTP Library.

7.9.4. 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. It will have poor performances for problems using few concepts but large and deep formulas leading to many splittings.

7.10. Octopus N

Monty Newborn, Zongyan Wang
McGill University, Canada
`newborn@cs.mcgill.ca`

7.10.1. Architecture

Octopus is a parallel ATP system. It is an improved version of the single-processor ATP system Theo [New01]. Inference rules used by Octopus include binary resolution, binary factoring, instantiation, demodulation, and hash table resolutions. Octopus performs 3000-10000 inferences/second on each processor.

7.10.2. Implementation

Octopus is implemented in C and currently runs under Linux. It runs on a network of 20-40 PCs housed in a laboratory at McGill University's School of Computer Science. The processors communicate using PVM [GB+94].

7.10.3. Strategies

Octopus begins by determining a number of weakened versions of the given theorem, and then assigns one such version to each computer. Each computer then attempts to prove the weakened version of the theorem assigned to it. If successful, the computer then uses the proof found to weakened theorem to help prove the given theorem. In essence, Octopus combines learning and parallel theorem proving.

In the current version of Octopus, a weakened version of a theorem consists of the same clauses of the given theorem except for one. In that one clause, a constant is replaced by a variable that doesn't appear elsewhere in the clause. If a proof exists to the given theorem, a proof exists to the weakened version, and often, though far from always, the proof of the weakened version is easier to find.

When a proof is found to a weakened version, certain clauses in the proof are added to the base clauses of the given theorem. Octopus then tries to prove the given theorem with the augmented set of base clauses.

Each processor in the system also uses different values for the maximum number of literals and terms in inferences generated when looking for a proof. Thus while two computers may try to solve the same weakened version of the given theorem, they do it with different values for the maximum number of literals and terms in derived inferences.

7.10.4. Expected Competition Performance

Octopus should do somewhat better than its predecessors that competed in the competition in the late 1990s, though it is not likely to finish among the top perfomers.

7.11. Otter 3.2

William McCune
Argonne National Laboratory, USA
mccune@mcs.anl.gov

7.11.1. Architecture

Otter 3.2 [McC94] is an ATP system for statements in first-order (unsorted) logic with equality. Otter is based on resolution and paramodulation applied to clauses. An Otter search uses the "given clause algorithm", and typically involves a large database of clauses; subsumption and demodulation play an important role.

7.11.2. Implementation

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

7.11.3. Strategies

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

7.11.4. Expected Competition Performance

Otter has been entered into CASC-19 as a stable benchmark against which progress can be judged (there have been only minor changes to Otter since 1996 [MW97], nothing that really affects its performace in CASC). This is not an ordinary entry, and we do not hope for Otter to do well in the competition.

7.12. Paradox 1.0

Koen Claessen, Niklas Sörensson
Chalmers University of Technology and Gothenburg University, Sweden
{koen,nik}@cs.chalmers.se

7.12.1. Architecture

Paradox 1.0 [CS03] is a finite-domain model generator. It is based on a MACE-style [McC94] flattening and instantiating of the FO clauses into propositional clauses, and then the use of a SAT solver to solve the resulting problem.

Paradox incorporates the following novel features: New polynomial-time clause splitting heuristics, the use of incremental SAT, static symmetry reduction techniques, and the use of sort inference.

7.12.2. Implementation

The main part of Paradox is implemented in Haskell using the GHC compiler. Paradox also has a built-in incremental SAT solver which is written in C++. The two parts are linked together on the object level using Haskell's Foreign Function Interface. Paradox uses the following non-standard Haskell extensions: local universal type quantification and hash-consing.

7.12.3. Strategies

There is only one strategy in Paradox:

1. Analyze the problem, finding an upper bound N on the domain size of models, where N is possibly infinite. A finite such upper bound can for example be found for EPR problems.
2. Flatten the problem, and split clauses and simplify as much as possible.
3. Instantiate the problem for domain sizes 1 up to N, applying the SAT solver incrementally for each size. Report "SATISFIABLE" when a model is found.
4. When no model of sizes smaller or equal to N is found, report "CONTRADICTION".

In this way, Paradox can be used both as a model finder and as an EPR solver.

7.12.4. Expected Competition Performance

Paradox will enter the CASC competition in two categories: SAT and EPR. Paradox beats last year's winner (2002) in the SAT category, and has also solved a number of "unknown" problems from TPTP within a short time limit. So it should have some chance of winning this year's SAT competition. Paradox is not optimized at all for the EPR category, but should perform reasonably well.

7.13. THEO J2003

Monty Newborn
McGill University, Canada
`newborn@cs.mcgill.ca`

7.13.1. Architecture

THEO [New01] is a resolution-refutation theorem prover for first order clause logic. It uses binary resolution, binary factoring, instantiation, demodulation, and hash table resolutions.

7.13.2. Implementation

Theo is written in C and runs under both LINUX and FREEBSD. It contains about 35000 lines of source code. Originally it was called The Great Theorem Prover.

7.13.3. Strategies

THEO uses a large hash table (16 megaentries) to store clauses. This permits complex proofs to be found, some as long as 500 inferences. It uses what might be called a brute-force iteratively deepening depth-first search for a contradiction while storing information about clauses - unit clauses in particular - in its hash table.

7.13.4. Expected Competition Performance

THEO should perform slightly better than it has in the past.

7.14. Vampire 5.0

Alexandre Riazanov, Andrei Voronkov
University of Manchester, England
`{riazanoa,voronkov}@cs.man.ac.uk`

7.14.1. Architecture

Vampire [RV01, RV02] 5.0 is an automatic theorem prover for first-order classical logic. Its kernel implements the calculi of ordered binary resolution and superposition for handling equality. The splitting rule is simulated by introducing new predicate symbols. 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 only term ordering used in Vampire at the moment is a special non-recursive version of the Knuth-Bendix ordering that allows efficient approximation algorithms for solving ordering constraints. By the system installation deadline we may implement the standard Knuth-Bendix ordering. A number of efficient indexing techniques are used to implement all major operations on sets of terms and clauses. Although the kernel of the system works only with clausal normal forms, the preprocessor component 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.

7.14.2. Implementation

Vampire 5.0 is implemented in C++. The main supported compiler version is gcc 2.95.3, although in the nearest future we are going to move to gcc 3.x. The system has been successfully compiled for Linux and Solaris. It is available from:

`http://www.cs.man.ac.uk/~riazanoa/Vampire/`

7.14.3. Strategies

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

- Choice of the main saturation procedure : (i) OTTER loop, with or without the Limited Resource Strategy, (ii) DISCOUNT loop.
- A variety of optional simplifications.
- Parameterised simplification ordering.
- 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.

The standalone executables for Vampire 5.0 and Vampire 5.0-CASC use very simple time slicing to make sure that several kernel strategies are tried on a given problem.

The automatic mode of Vampire 5.0 is primitive. Seven problem classes are distinguished corresponding to the competition divisions HNE, HEQ, NNE, NEQ, PEQ, UEQ and EPR. Every class is assigned a fixed schedule consisting of a number of kernel strategies called one by one with different time limits.

7.14.4. Expected Competition Performance

Vampire 5.0 is the CASC-18 MIX and FOF divisions winner.

7.15. Vampire 6.0

Alexandre Riazanov, Andrei Voronkov
University of Manchester, England
`{riazanoa,voronkov}@cs.man.ac.uk`

7.15.1. Architecture

Vampire [RV02] 6.0 is an automatic theorem prover for first-order classical logic. Its kernel implements the calculi of ordered binary resolution, superposition for handling equality and ordered chaining for transitive predicates. The splitting rule is simulated by introducing new predicate symbols. 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 orderings used are the standard Knuth-Bendix ordering and a special non-recursive version of the Knuth-Bendix ordering that allows efficient approximation algorithms for solving ordering constraints. A number of efficient indexing techniques are used to implement all major operations on sets of terms and clauses. Although the kernel of the system works only with clausal normal forms, the preprocessor component 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.

7.15.2. Implementation

Vampire 6.0 is implemented in C++. The supported compilers are gcc 2.95.3, 3.x and Microsoft Visual C++. The system has been successfully compiled for Linux, Solaris and Win32. It is available (conditions apply) from:

`http://www.cs.man.ac.uk/~riazanoa/Vampire/`

7.15.3. Strategies

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

- Choice of the main saturation procedure : (i) OTTER loop, with or without the Limited Resource Strategy, (ii) DISCOUNT loop.
- A variety of optional simplifications.
- Parameterised reduction orderings.
- A number of built-in literal selection functions and different modes of comparing literals.
- Age-weight ratio that specifies how strongly lighter clauses are preferred for inference selection.
- The standalone executable for Vampire 6.0 uses very simple time slicing to make sure that several kernel strategies are tried on a given problem.

The automatic mode of Vampire 6.0 is primitive. Seven problem classes are distinguished corresponding to the competition divisions HNE, HEQ, NNE, NEQ, PEQ, UEQ and EPR. Every class is assigned a fixed schedule consisting of a number of kernel strategies called one by one with different time limits.

7.15.4. Expected Competition Performance

We have made many, mostly cosmetic, changes since version 5.0, but they are unlikely to affect the performance drastically. We hope that Vampire 6.0 will perform at least as well as Vampire 5.0.

7.16. Waldmeister 702

Thomas Hillenbrand[1], Bernd Löchner[2]
[1]Max-Planck-Institut für Informatik Saarbrücken, Germany, [2]Universität Kaiserslautern, Germany
`waldmeister@informatik.uni-kl.de`

7.16.1. Architecture

Waldmeister 702 is an implementation of unfailing Knuth-Bendix completion [BDP89] with extensions towards ordered completion (see [AHL00]) and basicness [BG+92, NR92]. 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.

Only recently, we have designed a thorough refinement of the system architecture concerning the representation of passive facts [HL02]. The aim of that work - the next Waldmeister loop - is, besides gaining more structural clarity, to cut down memory consumption especially for long-lasting proof attempts, and hence less relevant in the CASC setting.

7.16.2. Implementation

The system is implemented in ANSI-C and runs under Solaris and Linux. The central data strucures are: perfect discrimination trees for the active facts; element-wise compressions for the passive ones; and sets of rewrite successors for the conjectures. Waldmeister can be found on the Web at

`http://www-avenhaus.informatik.uni-kl.de/waldmeister`

7.16.3. Strategies

Our approach to control the proof search is to choose the search parameters according to the algebraic structure given in the problem specification [HJL99]. This is based on the observation that proof tasks sharing major parts of their axiomatization often behave similar. Hence, for a number of domains, the influence of different reduction orderings and heuristic assessments has been analyzed experimentally; and in most cases it has been possible to distinguish a strategy uniformly superior on the whole domain. In essence, every such strategy consists of an instantiation of the first parameter to a Knuth-Bendix ordering or to a lexicographic path ordering, and an instantiation of the second parameter to one of the weighting functions *addweight*, *gtweight*, or *mixweight*, which, if called on an equation $s = t$, return $|s| + |t|$, $|\max_{>}(s,t)|$, or $|\max_{>}(s,t)| \cdot (|s| + |t| + 1) + |s| + |t|$, respectively, where $|s|$ denotes the number of symbols in s.

7.16.4. Expected Competition Performance

Waldmeister 702 is the CASC-18 UEQ division winner.

7.17. Waldmeister 703

Jean-Marie Gaillourdet[1], Thomas Hillenbrand[2], Bernd Löchner[1]
[1]Universität Kaiserslautern, Germany, [2]Max-Planck-Institut für Informatik Saarbrücken, Germany
`waldmeister@informatik.uni-kl.de`

7.17.1. Architecture

Waldmeister is a system for unit equational deduction. Its theoretical basis is unfailing completion in the sense of [BDP89] with refinements towards ordered completion (cf. [AHL03]). The system saturates the input axiomatization, distinguishing active facts, which induce a rewrite relation, and passive facts, which are the one-step conclusions of the active ones up to redundancy. The saturation process is parameterized by a reduction ordering and a heuristic assessment of passive facts [HJL99]. For an in-depth description of the system, see [Hil03].

Since last year's competition, the "new Waldmeister loop" has been implemented, and is now operational [GH+03]. This notion captures a novel organization of the saturation-based proof procedure into a system architecture [HL02], featuring a highly compact representation of the search state which exploits its inherent structure. The revealed structure also paves the way to an easily implemented parallelization of the prover with modest communication requirements and rewarding speed-ups. With the new architecture it is now possible to solve problems that were previously out of reach, for example deriving Boolean from the second Winker Lemma (`ROB007-1`). The proof is found with a standard strategy just in a parallel overnight run. Over 70,000 active facts and 500,000,000 passive ones are represented in no more than 200 MBytes. This foils the common belief that provers succeed either within five minutes or not all.

7.17.2. Implementation

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

`http://www.mpi-sb.mpg.de/~hillen/waldmeister/`

7.17.3. Strategies

The approach taken to control the proof search is to choose the search parameters according to the algebraic structure given in the problem specification [HJL99]. This is based on the observation that proof tasks sharing major parts of their axiomatization often behave similarly. Hence, for a number of domains, the influence of different reduction orderings and heuristic assessments has been analyzed experimentally; and in most cases it has been possible to distinguish a strategy uniformly superior on the whole domain. In essence, every such strategy consists of an instantiation of the first parameter to a Knuth-Bendix ordering or to a lexicographic path ordering, and an instantiation of the second parameter to one of the weighting functions *addweight*, *gtweight*, or *mixweight*, which, if called on an equation $s = t$, return $|s| + |t|$, $|\max_>(s,t)|$, or $|\max_>(s,t)| \cdot (|s| + |t| + 1) + |s| + |t|$, respectively, where $|s|$ denotes the number of symbols in s.

7.17.4. Expected Competition Performance

The focus of recent developments has been on coping with large search states; so we hope that this year's system version will be competitive with last year's.

8. Conclusion

The CADE-19 ATP System Competition is the eighth large scale competition for 1st order ATP systems. The organizers believe that CASC fulfills its main motivations: stimulation of research, motivation for improving implementations, evaluation of relative capabilities of ATP systems, and providing an exciting event. For the entrants, their research groups, and their systems, there is substantial publicity both within and outside the ATP community. The significant efforts that have gone into developing the ATP systems receive public recognition; publications, which adequately present theoretical work, have not been able to expose such practical efforts appropriately. The competition provides an overview of which researchers and research groups have decent, running, fully automatic ATP systems.

9. References

AHL00 Avenhaus J., Hillenbrand T., Löchner B. (2000), **On Using Ground Joinable Equations in Equational Theorem Proving**, Baumgartner P., Zhang H., *Proceedings of the 3rd International Workshop on First Order Theorem Proving* (St Andrews, Scotland), pp.33–43.

AHL03 Avenhaus J., Hillenbrand T., Löchner B. (2003), **On Using Ground Joinable Equations in Equational Theorem Proving**, *Journal of Symbolic Computation* 36(1-2), pp.217-233, Elsevier Science.

AL01 Avenhaus J., Löchner B. (2001), **System Description: CCE: Testing Ground Joinability**, Gore R., Leitsch A., Nipkow T., *Proceedings of the International Joint Conference on Automated Reasoning* (Siena, Italy), Lecture Notes in Artificial Intelligence, Springer-Verlag.

BDP89 Bachmair L., Dershowitz N., Plaisted D.A. (1989), **Completion Without Failure**, Ait-Kaci H., Nivat M., *Resolution of Equations in Algebraic Structures*, pp.1-30, Academic Press.

BG+92 Bachmair L., Ganzinger H., Lynch C., Snyder W. (1992), **Basic Paramodulation and Superposition**, Kapur D., *Proceedings of the 11th International Conference on Automated Deduction* (Saratoga Springs, USA), pp.462-476, Lecture Notes in Artificial Intelligence 607, Springer-Verlag.

Bil96 Billon J-P. (1996), **The Disconnection Method: A Confluent Integration of Unification in the Analytic Framework**, Miglioli P., Moscato U., Mundici D., Ornaghi M., *Proceedings of TABLEAUX'96: the 5th Workshop on Theorem Proving with Analytic Tableaux and Related Methods* (Palermo, Italy), pp.110-126, Lecture Notes in Artificial Intelligence 1071, Springer-Verlag.

Ble71 Bledsoe W.W. (1971), **Splitting and Reduction Heuristics in Automatic Theorem Proving**, *Artificial Intelligence* 2, pp.55-77.

CS03 Claessen K., Sorensson N. (2003), **New Techniques that Improve MACE-style Finite Model Finding**, Baumgartner P., Fermueller C., *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications* (Miami, USA).

CM96 Contejean E., March C. (1996), **CiME: Completion Modulo E**, Ganzinger H., *Proceedings of the 7th International Conference on Rewriting Techniques and Applications* (New Brunswick, USA), pp.416-419, Lecture Notes in Computer Science 1103, Springer-Verlag.

CM+02 Contejean E., March C., Monate B., Urbain X. (2000), **CiME version 2**, `http://cime.lri.fr`.

GH+03 Gaillourdet J-M., Hillenbrand T., Löchner B., Spies H. (2003), **The New Waldmeister Loop at Work**, Baader F., *Proceedings of the 19th International Conference on Automated Deduction* (Miami, USA), To appear, Lecture Notes in Artificial Intelligence, Springer-Verlag.

GB+94 Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R., and Sunderam V. (1994), **PVM: Parallel Virtual Machine: A Users Guide and Tutorial for Network Parallel Computing**, MIT Press.

GS96 Greiner M., Schramm M. (1996), **A Probablistic Stopping Criterion for the Evaluation of Benchmarks**, I9638, Institut für Informatik, Technische Universität München, München, Germany.

HJL99 Hillenbrand T., Jaeger A., Lõchner B. (1999), **Waldmeister - Improvements in Performance and Ease of Use**, Ganzinger H., <EM>*Proceedings of the 16th International Conference on Automated Deduction* (Trento, Italy), pp.232-236, Lecture Notes in Artificial Intelligence 1632, Springer-Verlag.

HL02 Hillenbrand T., Löchner B. (2002), **The Next Waldmeister Loop**, Voronkov A., *Proceedings of the 18th International Conference on Automated Deduction* (Copenhagen, Denmark), pp.486-500, Lecture Notes in Artificial Intelligence 2392, Springer-Verlag.

Hil03 Hillenbrand T. (2003), **Citius altius fortius: Lessons Learned from the Theorem Prover Waldmeister**, Dahn I., Vigneron L., *Proceedings of the 4th International Workshop on First-*

Order Theorem Proving (Valencia, Spain), Electronic Notes in Theoretical Computer Science 86.1, Elsevier Science.

LS01b Letz R., Stenz G. (2001), **Model Elimination and Connection Tableau Procedures**, Robinson A., Voronkov A., *Handbook of Automated Reasoning*, pp.2015-2114, Elsevier Science.

LS01c Letz R., Stenz G. (2001), **Proof and Model Generation with Disconnection Tableaux**, Nieuwenhuis R., Voronkov A., *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning* (Havana, Cuba), pp.142-156, Lecture Notes in Artificial Intelligence 2250, Springer-Verlag.

LS02 Letz R., Stenz G. (2002), **Integration of Equality Reasoning into the Disconnection Calculus**, Fermüller C., Egly U., *Proceedings of TABLEAUX 2002: Automated Reasoning with Analytic Tableaux and Related Methods* (Copenhagen, Denmark), pp.176-190, Lecture Notes in Artificial Intelligence 2381, Springer-Verlag.

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

McC94 McCune W.W. (1994), **Otter 3.0 Reference Manual and Guide**, Technical Report ANL-94/6, Argonne National Laboratory, Argonne, USA.

MI+97 Moser M., Ibens O., Letz R., Steinbach J., Goller C., Schumann J., Mayr K. (1997), **SETHEO and E-SETHEO: The CADE-13 Systems**, *Journal of Automated Reasoning* 18(2), pp.237-246.

MM+01 Moskewicz M., Madigan C., Zhao Y., Zhang L., Malik S. (2001), **Chaff: Engineering an Efficient SAT Solver**, Blaauw D., Lavagno L., *Proceedings of the 39th Design Automation Conference* (Las Vegas, USA), pp.530-535.

New02 Newborn M. (2001), **Automated Theorem Proving: Theory and Practice**, Springer.

NR92 Nieuwenhuis R., Rivero J.M. (1992), **Basic Superposition is Complete**, Krieg-Brückner B., *Proceedings of the 4th European Symposium on Programming* (Rennes, France), pp.371-390, Lecture Notes in Computer Science 582, Springer-Verlag.

Pas78 Pastre D. (1978), **Automatic Theorem Proving in Set Theory**, *Artificial Intelligence* 10, pp.1-27.

Pas89 Pastre D. (1989), **MUSCADET : An Automatic Theorem Proving System using Knowledge and Metaknowledge in Mathematics**, *Artificial Intelligence* 38, pp.257-318.

Pas93 Pastre D. (1993), **Automated Theorem Proving in Mathematics**, *Annals of Mathematics and Artificial Intelligence* 8, pp.425-447.

Pas01a Pastre D. (2001), **Muscadet2.3 : A Knowledge-based Theorem Prover based on Natural Deduction**, Gore R., Leitsch A., Nipkow T., *Proceedings of the International Joint Conference on Automated Reasoning* (Siena, Italy), pp.685-689, Lecture Notes in Artificial Intelligence 2083, Springer-Verlag.

Pas01b Pastre D. (2001), **Implementation of Knowledge Bases for Natural Deduction**, Nieuwenhuis R., Voronkov A., Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (Havana, Cuba), pp.29-68, Lecture Notes in Artificial Intelligence 2250, Springer-Verlag.

Pas02a Pastre D. (2002), **Strong and Weak Points of the MUSCADET Theorem Prover**, *AI Communications* 15(2-3), pp.147-160.

Pas02b Pastre D. (2002), **MUSCADET version 2.4 : User's Manual**, http://www.math-info.univ-paris5.fr/~pastre/muscadet/manual-en.ps.

RV01 Riazanov A., Voronkov A. (2001), **Vampire 1.1 (System Description)**, Gore R., Leitsch A., Nipkow T., *Proceedings of the International Joint Conference on Automated Reasoning* (Siena, Italy), pp.376-380, Lecture Notes in Artificial Intelligence 2083, Springer-Verlag.

RV02 Riazanov A., Voronkov A. (2002), **The Design and Implementation of Vampire**, *AI Communications*, 15(2-3), pp.91-110.

Sch01 Schulz S. (2001), **System Abstract: E 0.61**, Gore R., Leitsch A., Nipkow T., *Proceedings of the International Joint Conference on Automated Reasoning* (Siena, Italy), Lecture Notes in Artificial Intelligence, Springer-Verlag.

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

SW99 Stenz G., Wolf A. (1999), **E-SETHEO: Design, Configuration and Use of a Parallel Automated Theorem Prover**, Foo N., *Proceedings of AI'99: The 12th Australian Joint Conference on Artificial Intelligence* (Sydney, Australia), pp.231-243, Lecture Notes in Artificial Intelligence 1747, Springer-Verlag.

Ste02 Stenz G. (2002), **DCTP 1.2 - System Abstract**, Fermüller C., Egly U., *Proceedings of TABLEAUX 2002: Automated Reasoning with Analytic Tableaux and Related Methods* (Copenhagen, Denmark), pp.335-340, Lecture Notes in Artificial Intelligence 2381, Springer-Verlag..

SS97a Sutcliffe G., Suttner C.B. (1997), **Special Issue: The CADE-13 ATP System Competition**, *Journal of Automated Reasoning* 18(2).

SS98a Sutcliffe G., Suttner C.B. (1998), **The CADE-14 ATP System Competition**, Technical Report 98/01, Department of Computer Science, James Cook University, Townsville, Australia.

SS98b Sutcliffe G., Suttner C.B. (1998), **Proceedings of the CADE-15 ATP System Competition**, Lindau, Germany.

SS98c Sutcliffe G., Suttner C.B. (1998), **The TPTP Problem Library: CNF Release v1.2.1**, *Journal of Automated Reasoning* 21(2), pp.177-203.

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

Sut99 Sutcliffe G. (1999), **Proceedings of the CADE-16 ATP System Competition**, Trento, Italy.

Sut00a Sutcliffe G. (2000), **The CADE-16 ATP System Competition**, *Journal of Automated Reasoning* 24(3), pp.371-396.

Sut00b Sutcliffe G. (2000), **Proceedings of the CADE-17 ATP System Competition**, Pittsburgh, USA.

Sut01a Sutcliffe G. (2001), **The CADE-17 ATP System Competition**, *Journal of Automated Reasoning* 27(3), pp. 227-250.

Sut01b Sutcliffe G. (2001), **Proceedings of the IJCAR ATP System Competition**, Siena, Italy.

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

SSP02 Sutcliffe G., Suttner C., Pelletier F.J. (2002), **The IJCAR ATP System Competition**, *Journal of Automated Reasoning*, 28(3), pp.307-320.

SS02 Sutcliffe G. (2002), **Proceedings of the CADE-18 ATP System Competition**, Copenhagen, Denmark.

SS97b Suttner C.B., Sutcliffe G. (1997), **The Design of the CADE-13 ATP System Competition**, *Journal of Automated Reasoning* 18(2), pp.139-162.

SS98d Suttner C.B., Sutcliffe G. (1998), **The CADE-14 ATP System Competition**, Journal of Automated Reasoning 21(1), pp.99-134.

Tam97 Tammet T. (1997), **Gandalf**, *Journal of Automated Reasoning* 18(2), pp.199-204.

Tam98 Tammet T. (1998), **Towards Efficient Subsumption**, Kirchner C., Kirchner H., *Proceedings of the 15th International Conference on Automated Deduction* (Lindau, Germany), pp.427-440, Lecture Notes in Artificial Intelligence 1421, Springer-Verlag.

WGR96 Weidenbach C., Gaede B., Rock G. (1996), **SPASS and FLOTTER**, McRobbie M., Slaney J.K., *Proceedings of the 13th International Conference on Automated Deduction* (New Brunswick, USA), pp.141-145, Lecture Notes in Artificial Intelligence 1104, Springer-Verlag.