CASC-18
CASC-18
CASC-18
CASC-18

The CADE-18 ATP System Competition (CASC-18)

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-18 ATP System Competition (CASC-18) will be held on 28th July 2002. 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 proof objects 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-18 ATP System Competition (CASC-18) will be held at CADE-18 in Copenhagen, Denmark, on 28th July 2002. 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 proof objects 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-18 is the seventh such ATP system competition [SS97a, SS98d, SS99, Sut00a, Sut01a, SSP02].

Twenty four ATP systems, listed in Table 1, have been entered into the various competition and demonstration divisions. The winners of the CASC-JC 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-18 evolved from those of CASCs-13 to -17 [SS97b, SS98a, SS98b, Sut99, Sut00b, Sut01b]. Important changes for CASC-18 are:

- The rules regarding limitations on tuning for TPTP problems have been clarified.
- The sections and required information in the system description have changed.
- The retrospective disqualification rule for systems that are found to be unsound after the competition has been changed.
- The required system properties have changed, with respect to the UNIX system signals used to control execution.

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 Alan Bundy, John Harrison, 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-18* WWW site provides access to resources used before, during, and after the event:

```
http://www.tptp.org/CASC/18
```

System	Divisions	Entrant	Institution
Bliksem 1.12a	MIX* UEQ SAT FOF	Hans de Nivelle	Max-Planck-Institut für Informatik
CiME 2	UEQ	Benjamin Monate (Evelyne Contejean)	LRI Universite Paris-Sud
DCTP 1.2	MIX EPR	Gernot Stenz	Technische Universität München
DCTP 1.2-SAT	SAT	Gernot Stenz	Technische Universität München
DCTP 10.1p	MIX FOF EPR	Gernot Stenz	Technische Universität München
DCTP 10.1p-SAT	SAT	Gernot Stenz	Technische Universität München
E 0.7	MIX UEQ	Stephan Schulz	Tallinn Technical University and Safelogic AB
EP 0.7	MIX*	Stephan Schulz	Tallinn Technical University and Safelogic AB
E-SETHEO csp01	MIX FOF EPR	CASC-JC	CASC
E-SETHEO csp02	MIX FOF EPR	Gernot Stenz (Reinhold Letz, Stephan Schulz)	Technische Universität München
E-SETHEO csp02–SAT	SAT	Gernot Stenz (Reinhold Letz, Stephan Schulz)	Technische Universität München
EXLOG 2	MIX SAT FOF EPR (Demo)	Ivan Kossey	Uzhgorod State University
Gandalf c-2.5	MIX8 UEQ	Tanel Tammet	Tallinn Technical University and Safelogic AB
Gandalf c-2.5-SAT	SAT EPR	Tanel Tammet	Tallinn Technical University and Safelogic AB
GandalfSat 1.0	SAT	CASC-JC	CASC
GrAnDe 1.1	EPR	Geoff Sutcliffe, Stephan Schulz	University of Miami Technische Universität München
ICGNS 2002	SAT	William McCune (Olga Shumsky Matlin, Michael Rose)	Argonne National Laboratory
Otter 3.2	MIX* UEQ FOF	CASC-18 (William McCune)	CASC (Argonne National Laboratory)
SCOTT 6.1	MIX* UEQ SAT FOF EPR	John Slaney (Kahlil Hodgson)	Australian National University
Vampire 2.0-CASC	MIX*	CASC-JC	CASC
Vampire 5.0	MIX* UEQ FOF EPR	Andrei Voronkov (Alexandre Riazanov)	University of Manchester
Vampire 5.0-CASC	MIX*	Andrei Voronkov (Alexandre Riazanov)	University of Manchester
Waldmeister 601	UEQ	CASC-JC	CASC
Waldmeister 702	UEQ	Thomas Hillenbrand (Bernd Löchner)	Max-Planck-Institut für Informatik (Universität Kaiserslautern)

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

Table 1: The CASC-18 Entrants

2. Divisions

CASC-18 is divided into divisions according to problem and system characteristics. There are five competition divisions in which systems are explicitly ranked, and one 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. The competition panel judges whether or not each system's proof format is *acceptable*.

- The **UEQ** division: Unit Equality CNF Really-Non-Propositional Theorems

- 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 **FOF** division: Mixed FOF Non-Propositional Theorems

The FOF Division has two categories:

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

- 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)

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 30 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 v2.5.0. 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 classify 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. Problem ratings are computed from systems' performance data. Data from systems submitted by the system submission deadline are used for computing the problem ratings for the TPTP version used for the competition.
- The TPTP distinguishes versions of problems as one of standard, incomplete, augmented, especial, or biased. All except biased problems are eligible.

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

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

3.2.2. Number of Problems

The minimal numbers of problems that 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.

A lower bound on the total number of problems that is 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 each division and category are determined according to the judgement of the competition organizers.

3.2.3. Problem Preparation

It is necessary to ensure that no system receives an advantage or disadvantage due to the specific presentation of the problems in the TPTP. To this end the tptp2X utility, distributed with the TPTP, is used to:

- replace all predicate and function symbols with 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 and Timing

In the competition divisions, CPU and wall clock time limits are imposed on each solution attempt. A minimal CPU time limit of 180 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, UEQ, FOF, and EPR 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 or is disqualified. 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. In the demonstration division no soundness testing has to be performed.

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. In the MIX division proof class, the systems are ranked according to the number of problems solved with a proof output. In the MIX division assurance class and all other divisions, 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. If any division is won by the system that won that division in the previous CASC, then no winner is be announced in that division. Otherwise winners are announced in each division (two class winners in the MIX division), 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 are checked by the panel. If any of the proofs are unacceptable, i.e., they are significantly worse than the samples provided, then that system is retrospectively disqualified from the proof class. In all cases, the unsoundness will be reported in the journal paper about the competition.

5. Entry Requirements and 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. 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.

5.1. System Description and Proof Objects

A system description has to be provided for each ATP system entered, using this 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 the system is competing in.
- 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.

For systems in the MIX division proof class, representative sample proofs must be emailed to the competition organizers before the sample solutions deadline. The sample proofs 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 proof objects are acceptable.

5.2. 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, 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.

5.3. System Installation

Access to the general hardware (or equivalent) is available from the general hardware access deadline. 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.

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. Both approaches are acceptable for the competition. 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-18* 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, the start and end of the proof must be identified by distinguished strings (specified by the entrant).

- 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 proof object. 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 proof.

```
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
```

5.4. System Delivery

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 with instructions for installation , instructions for executing the system, and the distinguished strings output to indicate the result.

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.

6. 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 iff it outputs its "end of proof" 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.

7. The ATP Systems

These system descriptions were written by the entrants.

7.1. Bliksem 1.12a

H. de Nivelle
Max-Planck-Institut für Informatik, Germany
nivelle@mpi-sb.mpg.de

7.1.1. Architecture

Bliksem is a theorem prover implementing resolution and paramodulation. It implements many different strategies, including several orders that define decision procedures. It does not implement basicness or constraints. The automatic selection of strategies is poor, which partially explains the bad performance. A special feature of Bliksem is that it is able to output total proofs that can be easily verified [BHN00]. Additional programs that can check the proofs, or convert them into Coq-format [COQ02], are available from the home page.

Bliksem 1.12a is a reduced version of Bliksem 1.12. It was planned to replace the CNF-transformer by a Java program that is able to generate correctness proofs. In order to prepare for this, the CNF-transformer of Bliksem has been reduced to a minimum. Unfortunately, the new CNF-transformer will not be completed before the competition.

7.1.2. Implementation

Bliksem is written in portable C. High priority has been given to portability. Bliksem has been compiled succcessfully using cc, gcc, and djpcc. As said above, the CNF-transformer is to be replaced by a Java program. Bliksem (together with the additional proof checking programs) can be downloaded from:

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

7.1.3. Strategies

Stategy selection is primitive. Bliksem 1.12a checks if the problem is Horn or non-Horn, and it checks if the problem contains equality. If the problem is Horn, then hyperresolution is selected. If the problem contains equality, then paramodulation and equality factoring are switched on.

7.1.4. Expected Competition Performance

It can be expected that Bliksem 1.12a will perform worse than Bliksem 1.12, because of the fact that part of the system was removed.

7.2. CiME 2

E. Contejean, B. Monate
LRI, Universite Paris-Sud, France
{contejea,monate}@lri.fr

7.2.1. Architecture

CiME2 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
- 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 [CM96] will be used during the competion to attempt to solve unification problems, that is problems in the UEQ division.

7.2.2. Implementation

CiME 2 [CM+02] is fully written in Objective CAML, a functional language of the ML family developed in the CRISTAL project at INRIA Rocquencourt. CiME 2 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. Strategies

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 arbitrarily fixed to be an RPO ordering based on a precedence where the symbols are ordered according to their arities.
- 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. Some tuning may occur at this point by choosing the ratio between the coefficients for the size and the age. During the competition, this ratio will be fixed.

7.2.4. Expected Competition Performance

This will be the first participation of CiME 2 in CASC, in the UEQ division.

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

7.3. DCTP 1.2 and DCTP 10.1p

G. Stenz
Technische Universität München, Germany
stenz@informatik.tu-muenchen.de

7.3.1. Architecture

DCTP 1.2 [Ste02] is an automated theorem prover for first order clause logic. It is an implementation of the disconnection calculus described in [Bil96] and [LS01c]. 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 1.2 features several methods of builtin equality treatment [LS02], based on ordered paramodulation and on a variant of lazy root paramodulation that is also featured in the current version of SETHEO. Also, a formula preprocessor has been integrated making use of clause factoring, full subsumption, demodulation and a number of other techniques.

DCTP 10.1p is a strategy parallel version using the technology of E-SETHEO [SW99] to combine several different strategies based on DCTP 1.2.

7.3.2. Implementation

DCTP 1.2 has been implemented as a monolithic system in the Bigloo dialect of the Scheme language. The most important data structures are perfect discrimination trees, of which heavy use is made. DCTP 10.1p has been derived from the Perl implementation of E-SETHEO and includes DCTP 1.2 as well as additional components written in Prolog and Shell tools. Both versions run under Solaris and Linux.

7.3.3. Strategies

DCTP 1.2 is a single strategy prover. Individual strategies are started by DCTP 10.1p 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 1.2 and DCTP 10.1p to perform reasonably well, in particular in the SAT and EPR categories.

7.4. E 0.7 and EP 0.7

S. Schulz
Technische Universität München, Germany, and Safelogic A.B., Sweden
schulz@informatik.tu-muenchen.de

7.4.1. Architecture

E 0.7 [Sch01,Sch02b] 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.

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 0.7 is just a combination of E 0.7 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 versions of E, which destructively changed all shared instances of a term, the latest version adds only 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, and various versions of Linux. Sources of the latest released version and a current snapshot are available freely from:

```
http://wwwjessen.informatik.tu–muenchen.de/~schulz/WORK/eprover.html
```

EP 0.7 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 2.4.0. 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 300 second (or roughly equivalent configurations).

E distinguishes four primary problem classes: Problems where all non-negative clauses are unit clauses, where all non-negative clauses are Horn clauses (and at least one is not a unit), and non-Horn problems without and with equality. For each of these classes a separate set of candidate heuristics is created and run over all problems in this class. To generate the automatic mode for any of the four primary classes, the class is partitioned into subclasses defined by (some) of the following nine features:

- Are all negative clauses unit clauses?
- Are all literals equality literals, are some literas 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. Typically, most selected heuristics are assigned to more than one class. As an example, the current intermediate version uses a total of eight heuristics to cover all 532 problems with unit axioms.

7.4.4. Expected Competition Performance

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 by only Waldmeister.

EP 0.7 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 csp01

R. Letz, S. Schulz, G. 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 [Sch99] 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. E-SETHEO additionally includes the program Flotter [WGR96] as a preprocessing module for transforming non-clausal formulae to clausal form.

Since version 99csp of E-SETHEO, the different strategies are run sequentially, one after the other, 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. E-SETHEO csp01 incorporates the disconnection prover DCTP [LS01a] as a new strategy as well as a new version of the E prover.

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 2.6. Sources are available freely.

7.6. E-SETHEO csp02

R. Letz, S. Schulz, G. Stenz
Technische Universität München, Germany
{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 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].

7.6.2. 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, with a little luck, under Linux, too. Sources are available from the authors.

7.6.3. Strategies

Individual strategies are started be 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. EXLOG 2

I. Kossey
Uzhgorod State University, Ukraine
Ivan.Kossey@t-online.de

7.7.1. Architecture

EXLOG, developed since 1999, is a theorem prover in first order logic. The basic features are binary resolution with factors [Kos82], forward and backward subsumption, unit preference, a weak variant of set of support, and exclusion of predicates with satisfiable propositional abstraction. Features for equality are

rewrite rules (created from unit equality clauses with ordering of terms on length/alphabetical), and commutative unification. A somewhat modified Knuth-Bendix algorithm is used for equations with commutative unification for symmetric predicates and commutative functions. Paramodulation and AC-unification are not implemented. EXLOG accepts FOF, clausal TPTP format and his own input format.

7.7.2. Implementation

Assembler (TASM32) under Windows32. EXLOG runs on PC under Win95, Win98, WinNT, WinME, Win2000, WinXP. Developed and tested under Win2000. The subsumption algorithm [Kos84] and its implementation seems to be fast.

7.7.3. Strategies

Different strategies are implemented: some heuristic techniques for simplifying, unit preference, a weak variant of set of support (the conjecture clauses and their decsendants have preference). Many "small" heuristics are implemented, e.g., clause sorting on literal number/length. The program is self-tuning and has no run parameters.

7.7.4. Expected Competition Performance

Top half for problems with equality, bottom half for problems without equality.

7.8. Gandalf c-2.5

T. Tammet
Tallinn Technical University, Estonia, and Safelogic AB, Sweden
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-2.5 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.

7.8.2. 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.

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`

7.8.3. 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 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.

7.8.4. Expected Competition Performance

In the MIX and UEQ divisions, Gandalf will probably be among the better contestants but is not likely to win, except for (possibly) some categories of MIX.

In the SAT division, Gandalf version 2.5 should perform significantly better than older versions of Gandalf or any other satisfiability checker that participated at CASC during 2001. However, no prediction can be made about the relative performance at 2002.

In the EPR division, Gandalf now uses suitable strategies, differently from Gandalf in 2001, and is expected to be among the better provers in this class.

7.9. GandalfSat 1.0

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

7.9.1. Architecture

GandalfSat is a special version of the Gandalf theorem prover, optimised for satisifiability checking. Essentially it contains the same code as Gandalf, but uses different default search strategies to Gandalf.

Ordinary Gandalf is a resolution prover which implements a number of different basic strategies: binary ordered resolution for several orderings, versions of set-of-support resolution, binary unit resolution, hyperresolution. Enhancements are used for cutting off literals and combining different strategies, in particular forward and backward reasoning, into one run. Equality is handled by ordered paramodulation and demodulation.

GandalfSat uses time-slicing similarly to ordinary Gandalf. It attempts hyperresolution, ordered resolution based on [FL+93] results, and two kinds of finite model building algorithms (Falcon-like and MACE-like) for attempting to show satisfiability.

Some relevant publications are: [FL+93, Tam97, TS98].

7.9.2. Implementation

Gandalf is implemented in Scheme, using the scm interpreter developed by A. Jaffer and the Scheme-to-C compiler Hobbit developed by T. Tammet for the scm system.

7.10. GrAnDe 1.1

G. Sutcliffe[1], S. Schulz[2]
[1]University of Miami, USA, [2]Technische Universität München, Germany
geoff@cs.miami.edu, schulz@informatik.tu-muenchen.de

7.10.1. Architecture

GrAnDe (Ground And Decide) [SS02] is a decision procedure for CNF problems with a finite Herbrand universe. GrAnDe uses the grounding procedure `eground` [Sch02a] to generate ground instances of such a problem, and the propositional decision system ZChaff [MM+01] to determine the satisfiability of the resulting propositional problem. The set of all ground instances of a set of clauses is often too large to compute with reasonable resources. Therefore `eground` tries to find a smaller set of ground instances that is still equiconsistent with the original set of clauses. Three techniques are combined: clause splitting, structural constraints on variable instantiation, and propositional simplification. Clause splitting separates a clause into variable disjoint parts, and replaces the orginal clause by new clauses formed from the parts and link literals. Structural constraints are used to prohibit variable instantiations that would lead to the generation of pure literals in the resulting ground clauses. Propositional simplification removes clauses and literals from the generated ground clause set, while maintaining satisfiability.

For details about ZChaff, see [MM+01].

Despite the optimizations in `eground`, there are problems where `eground` runs out of time or memory. In these cases `eground` outputs an incomplete ground clause set. If `eground` has output an incomplete clause set and ZChaff reports that it is satisfiable, then no result is reported by GrAnDe. If `eground` has generated an incomplete clause set and ZChaff reports that it is unsatisfiable, or if `eground` has generated a complete clause set, then ZChaff's result is reported by GrAnDe as the overall result.

7.10.2. Implementation

`eground` is implemented in ANSI-C and based on the E [Sch01] libraries. It uses the basic infrastructure up to the shared term representation, and adds a new, compact clause data type for the generated propositional clauses. Propositional unit-clauses, used for simplification, are represented by a simple array entry, making unit subsumption and unit simplification into O(1) operations (in the number of unit clauses). Clause splitting is implemented naively, using a flood-fill algorithm to find connected sub-clauses. Finally, variable constraints are directly computed in conjunctive normal form, and are used to directly construct possible instantiations.

For details about ZChaff, see [MM+01].

A `perl` script called `And` is used to combine eground and ZChaff. The script invokes eground on the EPR problem, allowing it maximally 66% of the CPU time limit. The propositional clauses written by eground are captured into a temporary file, which is used as the input for ZChaff. ZChaff is allowed whatever CPU time has not been used by eground.

GrAnDe, including the separate components, is available from:

```
    http://www.cs.miami.edu/~tptp/ATPSystems/GrAnDe/
```

It should be widely portable between recent UNIX variants.

7.10.3. Strategies

GrAnDe has only one mode of operation, i.e., there are no different strategies for different types of problems. The overhead for the different techniques in eground is small enough that all optimizations can be applied in all cases without serious degradation of performance.

7.10.4. Expected Competition Performance

GrAnDe is expected to outperform all systems that use only first-order techniques.

7.11. ICGNS 2002

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

7.11.1. Architecture

ICGNS 2002 [McC02b] searches for finite models of first-order (unsorted, with equality) statements. Given input clauses, it generates ground instances over a finite domain, then uses a decision procedure try to determine satisfiability. If there is no model of that size, it increments the domain size and tries again.

The ICGNS method is SEM-like [ZZ95] rather than MACE-like [McC01]; that is, the ground problem is propositional/equality rather than flattened and purely propositional. The two main parts of the ICGNS method are (1) selecting the next empty cell in the tables of functions being constructed and deciding which values need to be considered for that cell, and (2) propagating assignments. ICGNS uses the basic least number heuristic (LNH) to reduce isomorphism. The LNH was introduced in Falcon [Zha96] and is also used in SEM. Effective use of the LNH requires careful cell selection. Propagation is by ground rewriting and inference rules to derive negated equalities.

7.11.2. Implementation

ICGNS 2002 is a new program, started in the winter of 2002. It uses some pre-existing code from various sources for some of the low level operations such as parsing and term structure. ICGNS is coded in ANSI C and should be easily portable to recent variants of UNIX. The source code and test problems are available from:

```
    http://www.mcs.anl.gov/home/mccune/icgns/
```

7.11.3. Strategies

Several strategies are available for cell selection and for applying the negative inference rules. A single strategy is used for CASC-18: (1) the LNH, selecting cells with the fewest number of possible values, and (2) applying all negative propagation inference rules. The strategy was derived by experimenting on problems that arose in mathematics practice, mostly in abstract algebra. No tuning to the TPTP set has been done.

7.11.4. Expected Competition Performance

ICGNS seems to do well on equational problems, especially if the input set contains some simple equations. Satisfiable problems without reasonably sized finite models (including problems without finite models) are out of reach of ICGNS. Some of those problems can be done by satisfiability methods that work by saturation with a complete inference system. It is doubtful that ICGNS will do as well, overall, as programs that try various methods and various strategies.

Acknowledgments: Olga Shumsky Matlin and Michael Rose assisted in the development of ICNGS 2002.

7.12. Otter 3.2

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

7.12.1. 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.

7.12.2. 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.

7.12.3. Strategies

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

7.12.4. Expected Competition Performance

Otter has been entered into CASC-18 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.

7.13. SCOTT 6.1

J. Slaney
Australian National University, Australia
John.Slaney@anu.edu.au

7.13.1. Architecture

SCOTT [HS01, HS02] consists of two main components: a saturation-based theorem prover (Otter [McC02a] in fact) and a finite domain constraint solver (FINDER [Sla94]) that generates models for subsets of the clauses deduced. The models are used to guide selection of given clauses, clauses which are false in the guiding models being given preference. Because model generation is computationally expensive, the critical issue for performance is the tradeoff between search quality due to semantic guidance and the time used in securing and refining it.

Successive versions of SCOTT have competed regularly in CASC. Version 6.1 is little changed from version 6.0 which competed in 2001, except for some amendments aimed at reducing the number of times a useless model is generated, deleted and re-generated, and slight tuning of the default values for parameters.

7.13.2. Implementation

The system is written in C, and consists of three main modules: the theorem prover is Otter, the model generator is FINDER, and there is a relatively small module linking the two into the combined system. The sources are available from

```
http://arp.anu.edu.au/software/scott/
```

7.13.3. Strategies

The proof techniques are taken over directly from Otter, along with that system's weight reduction strategies, pick-given ratio and the like. The characteristic feature of SCOTT is semantic guidance, applied as a heuristic for selection of given clauses. This improves the quality of the choices overall, though the effect is far from uniform across problems and the overheads incurred are severe. Many problems from TPTP have been used in the development of the semantic guidance strategy, but there has been little attempt to tune it more specifically than that.

7.13.4. Expected Competition Performance

SCOTT 6.0 competed in 2001, and version 6.1 is essentially similar, so we expect similar performance. Much depends on the time limit, as there are over 1000 problems in TPTP that the system proves in more than one minute and less than ten minutes (taking timings on a Sun E250 UltraSPARC II, for instance) so it will perform significantly better with a 600 second limit than with a 300 second one. In the past it has done moderately well on UEQ problems and on essentially propositional ones. In the MIX division, however, it is hampered by the lack of a sufficiently adaptive autonomous mode and will not threaten the leading provers. We shall be interested to discover what it finds easy and what it finds hard, in comparison with other systems.

7.14. Vampire 2.0-CASC

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

7.14.1. Architecture

Vampire 2.0 is an automatic theorem prover for first-order classical logic. It implements the calculi of ordered binary resolution, hyperresolution, 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 version of the Knuth-Bendix ordering which 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, such as an improved version [RV00] of code trees [Vor95] for forward subsumption, and a combination of path indexing [Sti89] and database joins for backward subsumption.

Compared to Vampire 1.0 that participated in the previous competition, this version has many more literal selection functions, more flexible splitting without backtracking and improved memory management. The automatic mode of Vampire 2.0 is primitive, it recognises only very simple syntactic properties of the input problem, like the presence of equality or non-Horn clauses. In the preprocessing stage it exploits a number of primitive techniques, such as elimination of simple predicate and function definitions.

Vampire 2.0-CASC is a version based on Vampire 2.0, specialised to win the competition. To adjust the strategy it estimates the problem by checking some syntactic properties, such as presence of multiliteral, non-Horn and ground clauses, equations and nonequational literals. Additionally we consider such quantitative characteristics as the number of axioms, literals, small and large terms.

7.14.2. Implementation

Vampire 2.0 is implemented in C++ and can be compiled by 2.91 or newer versions of gcc. The binaries for Solaris and Linux are available from the authors, for details see:

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

7.15. Vampire 5.0 and Vampire 5.0-CASC

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

7.15.1. 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 which 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 them and performs a number of useful transformations before passing the result to the kernel.

7.15.2. 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/
```

7.15.3. 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 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. The automatic mode of Vampire 5.0-CASC is better tuned towards TPTP 2.4.0. It uses the same problem characteristics as E 0.7 to divide the problems into a large number of classes and may potentially assign a separate strategy to every such class.

7.15.4. Expected Competition Performance

We expect Vampire 5.0 to perform much better than Vampire 2.0 on problems with equality, especially on the unit equality problems.

7.16. Waldmeister 601

T. Hillenbrand[1], B. Löchner[2], A. Jaeger, and A. Buch
[1]Max-Planck-Institut für Informatik Saarbrücken, Germany, [2]Universität Kaiserslautern, Germany
waldmeister@informatik.uni-kl.de

7.16.1. 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. The prover saturates the input axiomatization in a repeated cycle that works on a set of active resp. passive facts. The selection of the reduction ordering and the heuristical guidance of the proof search are described in [HJL99].

Since last year's version, stronger redundancy criteria have been integrated, including ground joinability tests with ordering constraints on variables [AHL00]. In several problem domains this technique is helpful especially for harder proof tasks. Some restructuring of the prover is on the way to also include an implementation of confluence trees with full ordering constraints [AL01]; but further work is necessary to have them speed up the proof search.

7.16.2. Implementation

The prover is coded in ANSI-C and available for SunOS, Solaris and Linux. The set of active facts is represented by perfect discrimination trees. Stimulated by the evaluation of indexing techniques reported on in [NH+01], the implementation of these has somewhat been improved. The storage of passive facts, and the treatment of hypotheses have remained unchanged. The Waldmeister Web page is located at:

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

7.17. Waldmeister 702

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

7.17.1. 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.

7.17.2. 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`

7.17.3. 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.

7.17.4. Expected Competition Performance

We expect Waldmeister 702 to be quite competitive. But since our current restructuring of the prover to implement the concepts of [HL02] may affect the inference behaviour, the outcome in comparison to last year's version is difficult to predict.

8. Conclusion

The CADE-18 ATP System Competition is the seventh 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.

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.

BHN00 Bezem M., Hendriks D., de Nivelle H. (2000), **Automated Proof construction in Type Theory using Resolution**, McAllester D., *Proceedings of the 17th International Conference on Automated Deduction* (Pittsburg, USA), pp.148-163, Lecture Notes in Artificial Intelligence 1831, 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.

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`.

COQ02 (2002), **The Coq Proof Assistant**, `http://pauillac.inria.fr/coq`.

FL+93 Fermüller C., Leitsch A., Tammet T., Zamov N. (1993), **Resolution Methods for the Decision Problem**, Lecture Notes in Computer Science 679, Springer-Verlag.

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), To appear, Lecture Notes in Artificial Intelligence, Springer-Verlag.

HS01 Hodgson K., Slaney J.K. (2001), **System Description: SCOTT-5**, Gore R., Leitsch A., Nipkow T., *Proceedings of the International Joint Conference on Automated Reasoning* (Siena, Italy), pp.443-447, Lecture Notes in Artificial Intelligence 2083, Springer-Verlag.

HS02 Hodgson K., Slaney J.K. (2002), **TPTP, CASC and the Development of a Semantically Guided Theorem Prover**, *AI Communications*, To appear.

Kos82 Kossey I. (1982), **Four Fast Algorithms in Resolution Method**, Messages of the Academy of Sciences of USSR, Technical Cybernetics Series, Nr 5, Academy of Sciences of USSR, Moscow, Russia.

Kos84 Kossey I. (1984), **Subsumption Algorithms in First and Finite Order Resolution**, *Mathematical Logic and its Applications*, Moscow Institute of Pedagogics, Moscow, Russia.

LS01a Letz R., Stenz G. (2001), **System Description: DCTP - A Disconnection Calculus Theorem Prover**, Gore R., Leitsch A., Nipkow T., *Proceedings of the International Joint Conference on Automated Reasoning* (Siena, Italy), Lecture Notes in Artificial Intelligence, Springer-Verlag.

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**, Ferm&uuml;ller C., Egly U., *Proceedings of TABLEAUX 2002: Automated Reasoning with Analytic Tableaux and Related Methods* (Copenhagen, Denmark), To appear.

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.

McC01 McCune W.W. (2001), **MACE 2.0 Reference Manual and Guide**, ANL/MCS-TM-249, Argonne National Laboratory, Argonne, USA.

McC02a McCune W.W. (2002), **Otter: An Automated Deduction System** `http://www-unix.mcs.anl.gov/AR/otter/`.

McC02b McCune W.W. (2002), **ICGNS**, `http://www.mcs.anl.gov/~mccune/icgns`.

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.
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.
RV00	Riazanov A., Voronkov A. (2000), **Partiallly Adaptive Code Trees**, Ojeda-Aciego M., de Guzman I., Brewka G., Pereira L., *Proceedings of the 7th European Workshop on Logics in Artificial Inteligence* (Malaga, Spain), pp.209-223, Lecture Notes in Artificial Intelligence 1919, Springer-Verlag.
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*, To appear.
Sch99	Schulz S. (1999), **System Abstract: E 0.3**, Ganzinger H., *Proceedings of the 16th International Conference on Automated Deduction* (Trento, Italy), pp.297-301, Lecture Notes in Artificial Intelligence 1632, Springer-Verlag.
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.
SS02	Schulz S., Sutcliffe G. (2002), **System Description: GrAnDe 1.0**, Voronkov A., *Proceedings of the 18th International Conference on Automated Deduction* (Copenhagen, Denmark), To appear, Lecture Notes in Artificial Intelligence, Springer-Verlag.
Sch02a	Schulz S. (2002), **A Comparison of Different Techniques for Grounding Near-Propositional CNF Formulae**, Haller S., Simmons G., *Proceedings of the 15th Florida Artificial Intelligence Research Symposium* (Pensecola, USA), pp.72-76, AAAI Press.
Sch02b	Schulz S. (2002), **E: A Brainiac Theorem Prover**, *AI Communications*, To appear.
Sla94	Slaney J.K. (1994), **Finite Domain Enumerator, System Description**, Bundy A., *Proceedings of the 12th International Conference on Automated Deduction* (Nancy, France), pp.764-768, Lecture Notes in Artificial Intelligence 814, Springer-Verlag.
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), To appear.
Sti89	Stickel M.E. (1989), **The Path Indexing Method for Indexing Terms**, Technical Note 473, SRI International, Menlo Park, USA.
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*, To appear.

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 ATP Progress**, 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.

TS98 Tammet T., Smith J. (1998), Optimised Encodings of Fragments of Type Theory in First Order Logic, Journal of Logic and Computation 8(6), pp.713-744.

Vor95 Voronkov A. (1995), **The Anatomy of Vampire**, *Journal of Automated Reasoning* 15(2), pp.237-265.

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.

ZZ95 Zhang J., Zhang H. (1995), **SEM: A System for Enumerating Models**, Mellish C.S., *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp.298-303, Morgan Kaufmann.

Zha96 Zhang J. (1996), **Constructing Finite Algebras with FALCON**, *Journal of Automated Reasoning* 17(1), pp.1-22.