# Proceedings of the CADE-17 ATP System Competition


**Abstract**

In order to stimulate ATP system development, and to expose ATP systems to interested researchers, the CADE-17 ATP System Competition (CASC-17) will be held on 17 June 2000. CASC-17 will evaluate the performance of sound, fully automatic, 1st order ATP systems. The evaluation will be in terms of:

- the number of problems solved, and
- the average runtime for successful solutions;

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-17 ATP System Competition (CASC-17) will be held at CADE-17, Pittsburgh, USA, on Saturday 17th June 2000. CASC-17 is the fifth CADE ATP system competition, following the successful competitions at CADEs-13 to -16 [SS97a,SS98d,SS99,Sut00]. CASC-17 will evaluate the performance of sound, fully automatic, 1st order ATP systems. The evaluation will be in terms of the number of problems solved and the average runtime for successful solutions, 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.

Sixteen ATP systems, listed in Table 1, have been entered into the various competition and demonstration divisions. The winners of the CASC-16 divisions have been automatically entered into those divisions, to provide benchmarks.

The design and procedures of CASC-17 evolved from those of CASCs-13 to -16 [SS97b, SS98a, SS98b, Sut99]. The rules, deadlines, and specifications given here, are absolute. Only the competition panel has the right to make exceptions. The design improves on the previous competitions' designs:

- A new SEM division has been added to the competition divisions. The problems in the SEM division are based on a specified encoding of a chosen semantic domain. In CASC-17, FOF theorems based on the TPTP set theory axiomatization `SET005+0.ax` will be used.
- The system installation requirements have changed. In particular, entrants have to produce an installation package containing the system source code, any other files required for installation, and a ReadMe file with instructions for installation. 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 competition organizers are Geoff Sutcliffe and Christian Suttner. The competition will be overseen by a panel of knowledgeable researchers who are not participating in the event; the panel members are Claude Kirchner, Don Loveland, and Jeff Pelletier. The competition will be run on computers provided by Carnegie Mellon University. The CASC-17 WWW site provides access to resources used before, during, and after the event:

    http://www.cs.jcu.edu.au/~tptp/CASC-17/

# 2. Divisions

CASC-17 is divided into divisions according to problem and system characteristics. There are five competition divisions in which the systems are explicitly ranked, and one demonstration division in which systems can demonstrate their abilities without being formally ranked.

## 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 is divided into five categories:

| System | Divisions | Entrant | Institution |
|---|---|---|---|
| Bliksem 1.10 | MIX UEQ SAT FOF SEM | Hans de Nivelle | Max-Planck-Institut für Informatik |
| E 0.6 | MIX UEQ | Stephan Schulz | TU München |
| E-SETHEO 2000csp | MIX UEQ FOF SEM | Stephan Schulz (Reinhold Letz, Gernot Stenz) | TU München |
| Gandalf c-2.1 | MIX UEQ SEM | Tanel Tammet | Tallinn Technical University |
| GandalfSat 1.0 | SAT | Tanel Tammet | Tallinn Technical University |
| HOTTER 0.1 | MIX SAT | Hantao Zhang | University of Iowa |
| MACE 1.4 | SAT | William McCune (Larry Wos, Ross Overbeek, Rusty Lusk) | Argonne National Laboratory |
| MUSCADET 2.2 | FOF | Dominique Pastre | Universite Rene Descartes |
| Otter-MACE 437 | SAT | CASC-16 | CASC-16 |
| Otter 3.1 | MIX UEQ FOF SEM | William McCune (Larry Wos, Ross Overbeek, Rusty Lusk) | Argonne National Laboratory |
| SCOTT V | MIX UEQ FOF SEM | Kahlil Hodgson (John Slaney) | Australian National University |
| SPASS 1.0.0T | FOF | CASC-16 | CASC-16 |
| Vampire 0.0 | MIX | CASC-16 | CASC-16 |
| Vampire 1.0 | MIX UEQ FOF SEM | Alexandre Riazanov (Andrei Voronkov) | University of Manchester |
| Waldmeister 799 | UEQ | CASC-16 | CASC-16 |
| Waldmeister 600 | UEQ | Thomas Hillenbrand (Bernd Loechner, Andreas Jaeger, Arnim Buch) | Universität Kaiserslautern |

Table 1: The CASC-17 Entrants

  - 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 **UEQ Division**: Unit Equality CNF Non-Propositional Theorems
- The **SAT Division**: Mixed CNF Non-Propositional Non-theorems
- The **FOF Division**: Mixed FOF Realy-Non-Propositional Theorems
  The FOF Division is divided into two categories:
  - The **FNE Category**: No Equality

    - The **FEQ Category**: With Equality
- The **SEM Division**: Problems based on a specified encoding of a chosen semantic domain.
  In CASC-17 FOF theorems based on the TPTP set theory axiomatization `SET005+0.ax` will be used.

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

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 particular 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).
- ATP systems that cannot run on the general hardware can enter into the demonstration division (see Section 2.2).

### 2.2. Demonstration Divisions

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

## 3. Infrastructure

### 3.1. Hardware and Software

The competition will be run on 20 SUN Ultra1 workstations, each having:

- A SUN Ultra I CPU
- 128MB memory
- Solaris 2.6 operating system

### 3.2. Problems

#### 3.2.1. Problem Selection

The problems will be selected from the TPTP Problem Library v2.3.0. The problems have to meet certain criteria to be eligible for selection:

- The TPTP uses system performance data to classify problems as one of:
    - Easy: Solvable by all state-of-the-art contributor ATP systems
    - Difficult: Solvable by some state-of-the-art contributor 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 will be eligible. Note: Entrants were encouraged to submit their latest system performance data to the collection, to help ensure that problems have appropriate ratings. Entrants benefit from contributing their latest system performance data, as it ensures that problems that their system can solve are eligible (unless they are easy problems). The deadline for submitting data to be taken into account for CASC-17 was 8th May 2000.
- The TPTP distinguishes versions of problems as one of standard, incomplete, augmented,

especial, or biased. Standard and especial problems will be eligible.

- No division or category will contain an excessive number of very similar problems.

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

#### 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, will be determined from the numbers of eligible problems in each division and category (the competition organizers have to ensure that there is sufficient CPU time available to run the ATP systems on this minimal number of problems). This minimal numbers of problems will then be used in determining the time limit imposed on each solution attempt.

A lower bound on the total number of problems that will be used will be 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 time limit, according to the following relationship:

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

It is a lower bound on the total number of problems because it assumes that every system will use all the time limit for each problem. Since some solution attempts will obviously succeed before the time limit is reached, more problems can actually be used. The actual numbers used in each division and category will be 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, will be used to:

- replace all predicate and function symbols by meaningless symbols (except in the SEM division),
- randomly reorder the clauses and literals in CNF problems,
- randomly reorder the formulae in FOF problems (except in the SEM division),
- randomly reverse the unit equalities in UEQ problems (except in the SEM division),
- 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 will be 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 will be specified to the ATP systems using the symbolic link names.

In the demonstration division the same problems will be used as for the general hardware, with the same tptp2X transformations applied. However, the original file names will be retained.

### 3.3. Time Limits and Timing

A time limit will be imposed on each solution attempt. On the general hardware a CPU time limit will be imposed, while in the demonstration division the entrant can choose to use either a CPU or a wall clock time limit.

A minimal time limit of 180 seconds will be used. The maximal time limit will be determined using the relationship used for determining the number of problems, with the minimal number of problems as the "Number of problems". The time limit will be chosen as a reasonable value within the range allowed.

The timing will be done by the UNIX `/bin/time` command, which returns times in units of 0.1 second. If an ATP system cannot solve a problem, the runtime will be set to the time limit.

In the competition divisions a wall clock time limit will be imposed in addition to the CPU time limit, to prevent very high memory usage that causes swapping. The wall clock time limit will be double the CPU time limit.

## 4. Entry Requirements and Procedures

To be entered into CASC-17, systems had to be registered using the CASC-17 system registration form by the registration deadline of 15th May. No registrations were 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-17, using the CADE-17 registration form. 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 CASC-16 will automatically be entered into their division, to provide benchmarks against which progress can be judged. After the competition all systems' source code will be made publically available on the CASC-17 WWW site.

The precomputation and storage of any information for individual TPTP problems for usage during the competition is contrary to the spirit of the competition, and is not allowed. The precomputation and storage of information that is reasonably likely to be useful in some future application is permitted. For every problem solved, the system's solution process has to be reproducible by running the system again.

It is assumed that each entrant has read all 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.

### 4.1. System Description

A system description had to be provided for each ATP system entered. The system descriptions, along with information regarding the competition design and procedures, form the proceedings for the competition. The system descriptions had to fit onto a single page, using a provided HTML schema. The schema has the following sections:

- Architecture. This section introduces the ATP system, and describes the calculus, inference rules, and search control 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.
- Expected competition performance. This section should make some prediction about the performance of the ATP system in each of the divisions and categories the system is competing in. Any tuning of the system for the TPTP problems must be clearly described.

- References.

The system description had to be emailed to the competition organizers before the registration deadline.

### 4.2. System Properties

Entrants had to ensure that their systems execute in the competition environment, according to the checks listed below. Entrants were advised to perform these checks well in advance of the installation period. 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.

- 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.
prompt>
```
- 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.
prompt>
```

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. The default action on receiving this signal is to exit (thus complying with the time limit, as required), but systems may catch the signal and exit of their own accord. Both approaches are acceptable for the competition. If a system runs past the time limit this is noticed in the timing data and the system is considered to have not solved that problem. Wall clock time are imposed using `SIGALRM`.

- Check: The ATP system's CPU time can be limited using the `CPULimitedRun.c` program. For example:
```
prompt> which CPULimitedRun
/home/tptp/bin/CPULimitedRun
prompt> /home/tptp/bin/CPULimitedRun 10 /home/tptp/bin/MyATPSystem /home/tptp/
tmp/CCC001-1.p
CPULimitedRun: ----------------------------------------------------------
CPULimitedRun: /home/tptp/bin/MyATPSystem
CPULimitedRun: Time limit is 10s
CPULimitedRun: PID is 5827
CPULimitedRun: ----------------------------------------------------------
CPU time limit exceeded (core dumped)
prompt>
```

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

- A proof exists (for CNF problems, the clause set is unsatisfiable),
- No proof exists (for CNF problems, the clause set is satisfiable), or

- No conclusion reached.

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

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`.

- Check: The system outputs distinguished strings. The entrant has specified the distinguished string `Proof found` to indicate that a proof exists. If appropriate, similar checks should be made for the cases where no proof exists and where no conclusion is reached.

```
prompt> /home/tptp/bin/CPULimitedRun 200 /home/tptp/bin/MyATPSystem /home/tptp
/tmp/CCC001-1.p
CPULimitedRun: ----------------------------------------------------------
CPULimitedRun: /home/tptp/bin/MyATPSystem
CPULimitedRun: Time limit is 200s
CPULimitedRun: PID is 13526
CPULimitedRun: ----------------------------------------------------------
Proof found in 147 seconds.
```

- 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/CPULimitedRun 200 /home/tptp/bin/MyATPSystem /home/tptp
/tmp/CCC001-1.p
CPULimitedRun: ----------------------------------------------------------
CPULimitedRun: /home/tptp/bin/MyATPSystem
CPULimitedRun: Time limit is 200s
CPULimitedRun: ----------------------------------------------------------
Proof found in 147 seconds.
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/GRP
... no temporary or other files left here ...
```

The CPU time taken by the ATP systems on the general hardware must be measurable by the UNIX `/bin/time` command.

- Check: The CPU time taken by a CPU limited run can be measured by /bin/time. For example:

```
prompt> /bin/time /home/tptp/bin/CPULimitedRun 200 /home/tptp/bin/MyATPSystem
/home/tptp/tmp/CCC001-1.p
CPULimitedRun: ----------------------------------------------------------
CPULimitedRun: /home/tptp/bin/MyATPSystem
CPULimitedRun: Time limit is 200s
CPULimitedRun: PID is 13526
CPULimitedRun: ----------------------------------------------------------
Proof found in 147 seconds.
```

```
real      3:04.2
user      2:19.3
sys          7.6
prompt>
```

### 4.3. System Installation

Access to the general hardware (or equivalent) was made available from the 29th May 2000. Entrants had to install their systems on the general hardware, and ensure that their systems execute on the general hardware according to the required system properties.

For systems entered in the competition divisions, entrants had to deliver an installation package to the competition organizers by 11th June 2000. The installation package had to be a `.tar.gz` file containing the system source code, any other files required for installation, and a `ReadMe` file with instructions for installation. 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 was reinstalled onto the general hardware by the competition organizers, following the instructions in the `ReadMe` file. Installation failures before the installation deadline were passed back to the entrant. After the installation deadline access to the general hardware was denied, and no further changes or late systems were accepted.

After the installation deadline the organizers tested the ATP systems, first to check that the systems execute correctly (according to the above checks), and secondly to check for soundness. For the soundness testing, non-theorems (satisfiable variants of the eligible problems, e.g., without the conjecture clause, and satisfiable problems selected from the TPTP) were submitted to the systems participating in the MIX, UEQ, FOF, and SEM Divisions, and theorems (selected from the TPTP) were submitted to the systems participating in the SAT Division. Finding a proof of a non-theorem or a model for a theorem indicates that the system is unsound. If an ATP system fails the soundness testing it 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. Further soundness testing will be performed after the competition, as described in the section on performance evaluation.

In the demonstration division the systems were installed on the respective hardware by the entrants, and no soundness testing needed to be performed.

### 4.4. System Execution

Execution of the ATP systems on the general hardware will be controlled by a `perl` script, provided by the competition organizers. The jobs will be 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 will be started before any attempts at the (N+1)th problems.

During the competition a `perl` script will parse the systems' output. If an ATP system's success string is found then the timing information from the time command will be extracted. The CPU time taken, or the time limit if no solution was found, will be recorded. This data will be used to generate an HTML-file, and a WWW-browser will be used to display the results. The execution of the demonstration division systems will be supervised by their entrants.

## 5. Performance Evaluation

The systems will be ranked within each competition division and category according to the number of problems solved. If several systems solve the same number of problems, then those systems will be ranked according to their average runtimes over solutions found. If any division is won by the system that won that division in the previous CASC, then no winner will be announced in that division.

Otherwise winners will be announced in each division and category, and prizes will be awarded.

If only one ATP system registers for a particular competition division, no winner can be announced for that division, but the results for that system will still be presented.

At some time after the competition, all high ranking systems in each division will be tested over the entire TPTP. This testing will provide a final check for soundness, and any system found to be unsound will be retrospectively disqualified.

# 6. The ATP Systems

These system descriptions were written by the entrants.

## 6.1. Bliksem 1.10

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

### 6.1.1. Architecture

Bliksem is a resolution and paramodulation based theorem prover. It implements many different orders, including many orders that define decision procedures. The automatic selection of strategies is poor. A special feature of Bliksem is that it is able to output totally specified proofs. After processing, these proofs can be checked by Coq.

### 6.1.2. Implementation

Bliksem is written in portable C. High priority has been given to portability. Bliksem has been compiled succcessfully using cc, gcc, and djpcc. The data structures have been carefully chosen after benchmark tests on the basic operations. Bliksem can be obtained from:

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

### 6.1.3. Expected Competition Performance

The system does not differ from last year's entry. So it is reasonable to expect the same competition performance as last year.

## 6.2. E 0.6

S. Schulz
Institut für Informatik, Technische Universität München
schulz@informatik.tu-muenchen.de

### 6.2.1. Architecture

E 0.6 [Sch99] is a purely equational theorem prover. The calculus used by E combines superposition (with selection of negative literals) and rewriting. For the special case that only unit clauses are present, it degenerates into unfailing completion. For the Horn case E can simulate various positive unit strategies. No special rules for non-equational literals have been implemented, i.e., resolution is simulated via paramodulation and equality resolution.

Proof search in E is controlled by a literal selection strategy and an evaluation heuristic. A variety of selection strategies (including e.g., no selection, or select all negative literals) is available. Evaluation heuristics can be constructed on the fly by combining various parameterized primitive evaluation functions. A simple automatic mode can select strategy, term ordering, and heuristic based on simple problem characteristics.

New features since last year include a wider variety of literal selection strategies, and an improved automatic mode that now also switches between two different Knuth-Bendix orderings. A proof checker is now available as well.

#### 6.2.2. Implementation

E is implemented in ANSI C, using the GNU C compiler. The most unique feature of the implementation is the maximally shared term representation. This includes parallel rewriting for all instances of a particular subterm. 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 are available freely from:

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

#### 6.2.3. Expected Competition Performance

As in the last year, E is expected to perform pretty well in all categories it participates in, with particularly strong showing for Horn problems. Very little effort has been spent on the performance for unit equality problems, however, and it is unlikely that E can come close to Waldmeister's performance in this category.

E's auto mode is optimized for performance on the complete TPTP 2.3.0. 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.

### 6.3. E-SETHEO 2000csp

R. Letz, S. Schulz, G. Stenz
Institut für Informatik, Technische Universität München
{letz,schulz,stenzg}@informatik.tu-muenchen.de

#### 6.3.1. 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-FLOTTER additionally includes the program FLOTTER [WGR96] as a preprocessing module for transforming non-clausal formulae to clausal form.

In contrast to the former P-SETHEO [Wol98], 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. Version 2000csp of E-SETHEO employs a new strategy interweaving the lemma mechanisms of E and SETHEO.

#### 6.3.2. 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.

#### 6.3.3. Expected Competition Performance

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

## 6.4. Gandalf c-2.1

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

### 6.4.1. Architecture

Gandalf version "c" stands for the Gandalf for classical logic. Special versions for intuitionistic logic and type theory exist but are not covered here.

Gandalf is a reolsution 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.

When compared to earlier versions, the new version adds special strategies for a limited kind of clause demodulation and combining a propositional prover with a first-order prover.

The basic idea of the automatic mode in Gandalf is time-slicing: Gandalf selects a set of different search strategies, allocates time to these and finally runs the strategies one after another. The motivation is the following: since it is very hard to determine a suitable strategy by some heuristic procedure, it pays off to try a number of possibly suitable strategies. The selection of strategies and the percentage of time they receive is in a some what ad hoc way based on the characteristics of the problem: percentage of Horn and almost-Horn clauses, the number of clauses, percentage of clauses where literals can be ordered using term and variable depths.

Some relevant publications are: [Tam97, TS98, Tam98].

### 6.4.2. 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.

### 6.4.3. Expected Competition Performance

Gandalf is likely to do well in the non-Horn-with-equality subclass of problems. It is likely to have mediocre performance in the general mixed class, and poor performance in the pure equality class.

## 6.5. GandalfSat 1.0

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

### 6.5.1. 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: [Tam97, TS98].

### 6.5.2. 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.

### 6.5.3. Expected Competition Performance

Since GandalfSat has been developed recently, we cannot reliably predict its performance. However, we expect it to be among the better provers for satisfiability checking.

## 6.6. MACE 1.4

W. McCune
Mathematics and Computer Science, Argonne National Laboratory
mccune@mcs.anl.gov

### 6.6.1. Architecture

MACE [McC00a] searches for small finite models. The program applies to statements in first-order (unsorted) logic with equality. Roughly speaking, MACE works as follows. For a given domain size, an equivalent propositional problem is constructed and a Davis-Putnam-Loveland decision procedure is applied. MACE typically cannot handle clauses with many variables or deeply nested terms.

### 6.6.2. Implementation

MACE is written in C. Its Davis-Putnam-Loveland procedure uses a counting algorithm to do the unit resolutions, and forward subsumption is not applied.

### 6.6.3. Expected Competition Performance

MACE unexpectedly won the SAT division at CASC-16, in part because the difficulty ratings were biased towards MACE. MACE is expected to do well at CASC-17.

**Acknowledgments**: MACE was motivated by previous work of John Slaney, Mark Stickel, and Hantao Zhang.

## 6.7. MUSCADET v2.2

D. Pastre
Université René Descartes (Paris 5), UFR de Mathématiques et Informatique
pastre@math-info.univ-paris5.fr

### 6.7.1. Architecture

MUSCADET [Pas99] is a theorem prover based upon Natural Deduction, in the sense of [Ble71] and [Pas78]. It is built as a knowledged-based system, that is, methods are expressed by rules interpreted by an inference engine. Some rules are universal mathematical knowledge and are given to the system. Other rules are specific to some concepts and are built by metarules from the definitions of these concepts. The notions of hypothesis and conclusion to be proved are privileged. The first conclusion is the first order statement of the conjecture.

Methods are those used by humans, and are executed according to a plan decided by metarules and heuristics. Methods include splitting theorems into two or more subtheorems (conjunctive conclusion or disjunctive hypotheses), treatment of universal, existential or disjunctive conclusion, treatment of existential or disjunctive hypotheses (one at a time when it is decided by heuristic rules), replacement of the conclusion by its definition. Hypotheses are never replaced by their definition. Instead, rules are built by metarules from definitions and used to deduce new hypotheses from old ones. Universal hypotheses are not added. Instead, rules are built as from the definitions. 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. Positive properties are privileged, in hypotheses and in conclusions. So, a special treatment of negation is done. 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).

#### 6.7.2. Implementation

The first version of MUSCADET was a real rule-based system [Pas89]. Rules and metarules were written in a declarative language defined for this and interpreted by an inference engine written in Pascal. The competition version is entirely written in SWI-Prolog. Rules are written as declarative Prolog clauses. Metarules are written as sets of Prolog clauses, more or less declarative. The Pascal inference engine is replaced by the Prolog interpreter and some procedural Prolog clauses.

#### 6.7.3. 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.

### 6.8. Otter-MACE 437

W. McCune
Mathematics and Computer Science, Argonne National Laboratory
mccune@mcs.anl.gov

#### 6.8.1. Architecture

Otter [McC94, McC00b, MW97] searches for proofs, and MACE [McC00a] searches for small finite models. Both programs apply to statements in first-order (unsorted) logic with equality. The two programs are independent and work by completely different methods. (In practice, the programs are complementary.) Version 3.0.5 of Otter has been used. 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. Version 1.3.2 of MACE has been used. Roughly speaking, MACE works as follows. For a given domain size, an equivalent propositional problem is constructed and a Davis-Putnam-Loveland decision procedure is applied. MACE typically cannot handle clauses with many variables or deeply nested terms. Otter-MACE 437 (= Otter 3.0.5 + MACE 1.3.2) is a simple combination of the two programs.

#### 6.8.2. Implementation

Both programs are 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. MACE is much simpler than Otter; its Davis-Putnam-Loveland procedure uses a counting algorithm to do the unit resolutions, and forward subsumption is not applied. Otter-MACE is a shell script that runs Otter for a few seconds, then (if Otter has not found a refutation) MACE for a few seconds, then (if MACE has not found a model) Otter again for the remaining time.

#### 6.8.3. Expected Competition Performance

These versions of Otter and MACE are essentially the same as the versions entered in the previous two CASC competitions. Both programs generally perform well at CASC, but no top prizes have been won since CASC-13. We do not expect any winning performances in CASC-16. We see our entry as a benchmark for measuring improvements in other systems.

### 6.9. Otter 3.1

W. McCune
Mathematics and Computer Science, Argonne National Laboratory
mccune@mcs.anl.gov

#### 6.9.1. Architecture

Otter [McC94, McC00b, MW97] searches for proofs. Otter applies to 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.

#### 6.9.2. 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.

#### 6.9.3. Expected Competition Performance

Otter's original automatic mode was used for CASC-13 (the first CASC). A secondary automatic mode, derived from some tuning with respect to TPTP, was used For CASC-14, -15, and -16. The secondary automatic mode is less complete for nonunit equality problems but it proves a few more TPTP theorems. For CASC-17, we return to the original automatic mode, which reflects no tuning to TPTP. Otter did very well in CASC-13, but it has changed very little in automatic capability since then. Powerful new programs have been introduced in the past four years, and old competitors have been greatly improved, so no winning performances are expected from Otter.

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

### 6.10. SCOTT V

K. Hodgson and J. Slaney
Automated Reasoning Project, Australian National University
{kahlil, jks}@arp.anu.edu.au

#### 6.10.1. Architecture

SCOTT (Semantically Constrained OTTER) [HS99, HS00] is an extension of OTTER [McC94] and uses the same inference rules and overall proof search structure. This extension incorporates FINDER [Sla94] (a finite model generator) as a subsystem of OTTER in order to determine consistent subsets of the clause space. These consistent subsets are used to guide the proof search by increasing the likelihood that certain necessary inferences are made [HS00]. (This can be seen to loosely generalise the set of support strategy). Not all consistent subsets are created equal, with different collections providing different levels of guidance, and so a number of strategies for determining which collections provide the best guidance have been developed. These incorporate theoretical estimates of the guidance provided and the corresponding cost incurred, enabling us to manage the trade-off much more effectively than in previous versions of SCOTT [HS99].

#### 6.10.2. Implementation

SCOTT is written in C. It uses the program OTTER to perform the proof search, the progam FINDER to witness the consistency of particular sets of clauses, and additional code to determine which clauses should be chosen for inference. The system may be obtained from:

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

#### 6.10.3. Expected Competition Performance

Preliminary results show that, SCOTT-V out performs SCOTT-IV, so we expect an improvement on the performance in CASC-16. However, searching for models and testing clauses against them can be costly, and so SCOTT may perform poorly if the time-limit is too severe.

### 6.11. SPASS 1.0.0T

C. Weidenbach
Max-Planck-Institut für Informatik
weidenb@mpi-sb.mpg.de

#### 6.11.1. Architecture

SPASS [WA+99] is an automated theorem prover for first-order logic with equality. It is a saturation based prover employing superposition, sorts and splitting [Wei99]. The current version 1.0.0 offers a variety of further inference and reduction rules including hyper resolution, unit resulting resolution, various variants of paramodulation and a terminator [AO83].

The prover is available for many platforms, in particular, we now also provide a Windows 95/98/NT version including a neat graphical user interface.

#### 6.11.2. Implementation

SPASS is implemented in ANSI C and relies on our internal library supporting specific data structures and algorithms like, e.g., indexing or orderings (KBO, RPOS). The code is documented, flexible and was designed to be reusable. The code should compile on any platform supporting an ANSI C compiler.

#### 6.11.3. Expected Competition Performance

The SPASS competition version 1.0.0TPTP is tuned towards TPTP problems. In particular, the standard automatic configuration module is replaced by a module adapted to the TPTP. We hope that the new version can improve over the version 0.95TPTP that entered the previous competition.

### 6.12. Vampire 0.0

A. Riazanov, A. Voronkov
Computer Science Department, The University of Manchester
{riazanoa,voronkov}@cs.man.ac.uk

#### 6.12.1. Architecture

Vampire is a resolution-based theorem prover for first-order classical logic. The current version implements ordered binary resolution and hyperresolution, superposition and some simplifications. For proof search Vampire uses a saturation-based procedure similar to that of Otter [McC94]. One interesting feature of Vampire is the so-called limited resource strategy. The idea is as follows. When a time limit is given, Vampire makes estimations on what clauses (called currently eligible clauses) can be processed by the end of the time limit. For each newly generated clause it is checked whether it is currently eligible. If it is not, Vampire discards the clause. The estimation principle takes into consideration the current strategy and tries to calculate how the process of clause generation will develop by the time limit.

#### 6.12.2. Implementation

To gain efficiency we implement the most costly set-at-a-time operations, such as resolution, superposition and subsumption, using combination of indexing and compilation. We use a variant of code trees [Vor95] for forward subsumption and forward demodulation, path indexing [Sti89] for backward subsumption and a variant of discrimination trees [McC92] for resolution and superposition. All used indexing techniques provide so-called "perfect filtering" [Gra96, RSV99]. For storing persistent clauses we use tree-like representation of terms. The sharing is agressive. For other operations we use different versions of flatterms. Vampire is implemented in C++ and compiled by gcc.

#### 6.12.3. Expected Competition Performance

We expect Vampire to perform well, especially on the problems without equality.

### 6.13. Vampire 1.0

A. Riazanov, A. Voronkov
Computer Science Department, The University of Manchester
{riazanoa,voronkov}@cs.man.ac.uk

#### 6.13.1. Architecture

Vampire 1.0 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 is 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. The competition version will exploit a number of simple preprocessing techniques, such as elimination of simple predicate and function definitions. A number of efficient indexing techniques is used to implement all major operations on sets of terms and clauses, such as an improved version of code trees [Vor95] for forward subsumption and a combination of path indexing [Sti89] and database joins for backward subsumption.

#### 6.13.2. Implementation

Vampire 1.0 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`

#### 6.13.3. Expected Competition Performance

We cannot make reliable predictions of Vampire's performance at CASC-17.

### 6.14. Waldmeister 799

T. Hillenbrand, A. Jaeger, B. Löchner, and A. Buch
FB Informatik, Universität Kaiserslautern
{hillen,jaeger,loechner}@informatik.uni-kl.de,arnim.buch@sdm.de

This year's version of our unit equality theorem prover is in most parts identical to Waldmeister 798 that won the CASC-15 UEQ competition with an enormous lead [SS99]. Since last year's description is included in this collection we will only report on the differences here.

#### 6.14.1. Architecture

Waldmeister is an implementation of unfailing Knuth-Bendix completion [BDP89]. Our approach to control the proof search is to choose selection strategy and reduction ordering according to the algebraic structure given in the problem specification [HJL99]. Since Waldmeister 798, the corresponding part of the system has been developed further into a robust component that is easy to maintain. In addition we have systematized the derivation of useful control knowledge, which also gave rise to progress on some problem domains.

Recently, Waldmeister has successfully been integrated into the Omega proof development environment [BC+97]. Moreover, we have established a convenient WWW interface that is reachable via:

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

From this URL further documents and the complete Waldmeister distribution containing an easy-to-use version together with problems and a manual are available.

#### 6.14.2. Implementation

No major changes took place. Some portions of code have been slightly revised, e.g., a number of time-consuming inner loops have carefully been reworked.

#### 6.14.3. Expected Competition Performance

We expect some improvements in comparison with last year's version, depending on problem domains and proof tasks.

### 6.15. Waldmeister 600

T. Hillenbrand, A. Jaeger, B. Löchner, and A. Buch
FB Informatik, Universität Kaiserslautern
{hillen,jaeger,loechner}@informatik.uni-kl.de,arnim.buch@sdm.de

#### 6.15.1. Architecture

Waldmeister 600 [HJL99] is an implementation of unfailing Knuth-Bendix completion [BDP89]. The system architecture is built on a stringent distinction of active facts (selected rewrite rules) and passive facts (critical pairs): Firstly, only the active facts induce the normalization relation. Secondly, the passive facts are normalized not permanently, but on generation only. The saturation cycle is parameterized by a reduction ordering and a heuristic assessment of passive facts. The control parameters are chosen according to the algebraic structure given in the problem specification [HJL99]. Recent developments include the integration of the kernel strategy [HL00] as well as extensions towards ordered completion, but are still in progress.

#### 6.15.2. Implementation

The system is implemented in ANSI-C and runs under SunOS, Solaris and Linux. Refined perfect discrimination trees are employed to index the active facts, whereas the passive facts are stored in a compressed representation. The hypotheses are blown up to sets of rewrite successors and sometimes predecessors. A step-by-step proof object is constructed at run-time. Waldmeister can be found on the Web at:

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

#### 6.15.3. Expected Competition Performance

The Waldmeister system being as powerful as it is today, we hope to improve over last year's version in some problem domains.

## 7. Conclusion

The CADE-17 ATP System Competition will be the fifth large scale competition fot 1st order ATP systems. The organizers believe that CASC-17 will fulfill 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 will be substantial publicity both within and outside the ATP community. The significant efforts that have gone into developing the ATP systems will receive public recognition; publications, which adequately present theoretical work, have not been able to expose such practical efforts appropriately. The competition will provide an overview of which researchers and research groups have decent, running, fully automatic ATP systems.

## 8. References


AO83 Antoniou G., Ohlbach H-J. (1983), **Terminator**, Bundy A., *Proceedings of the 8th International Joint Conference on Artificial Intelligence* (Karlsruhe, Germany), pp.916-919.

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.

BC+97 Benzmüller C., Cheikhrouhou L., Fehrer D., Fiedler A., Huang X., Kerber M., Kohlhase M., Konrad K., Meier A., Melis E., Schaarschmidt W., Siekmann J., Sorge V. (1997)**, OMEGA: Towards a Mathematical Assistant**, McCune W.W., *Proceedings of the 14th International Conference on Automated Deduction* (Townsville, Australia), pp.146-160, Lecture Notes in Artificial Intelligence 1249, Springer-Verlag.

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

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.

Gra96 Graf P. (1996), **Term Indexing**, Lecture Notes in Computer Science 1053, Springer-Verlag.

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.

HL00 Hillenbrand T., Lõchner B. (2000), **Shorter Fixed Point Combinators with Waldmeister and the Kernel Strategy**, *AAR Newsletter* 46.

HS00 Hodgson K., Slaney J.K. (2000), **Semantic Guidance for Saturation-based Theorem Proving**, Technical Report TR-ARP-04-2000, Automated Reasoning Project, Australian National University, Canberra, Australia.

HS99 Hodgson K., Slaney J.K. (1999), **Semantic Guidance with SCOTT**, Technical Report TR-ARP-01-99, Automated Reasoning Project, Australian National University, Canberra, Australia.

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

McC00a McCune W.W. (2000), **MACE: Models and Counterexamples** `http://www-unix.mcs.anl.gov/AR/mace/`.

McC00b McCune W.W. (2000), **Otter: An Automated Deduction System** `http://www-unix.mcs.anl.gov/AR/otter/`.

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

McC92 McCune W.W. (1992), **Experiments with Discrimination-Tree Indexing and Path Indexing for Term Retrieval**, Journal of Automated Reasoning 9(2), pp.147-167.

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.

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.

Pas99 Pastre D. (1999), **Le nouveau MUSCADET et la TPTP Problem Library**, Colloque sur la métaconnaissance, Berder, report LIP6 2000/002, pp.54-98, Universite Rene Descartes (Paris 5), UFR de Mathematiques et Informatique, Paris, France.

RSV99 Ramakrishnan I.V., Sekar R., Voronkov A. (1999), **Term Indexing**, Robinson A., Voronkov A., *Handbook of Automated Reasoning*, Elsevier Science.

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.

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.

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.

Sut00 Sutcliffe G. (2000), **The CADE-16 ATP System Competition**, *Journal of Automated Reasoning* 24(3), pp.371-396.

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.

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.

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.

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

WA+99 Weidenbach C., Afshordel B., Brahm U., Cohrs C., Engel T., Keen E., Theobalt C., Tpoic D. (1999), **System Description: SPASS Version 1.0.0**, Ganzinger H., *Proceedings of the 16th International Conference on Automated Deduction* (Trento, Italy), pp.378-382, Lecture Notes in Artificial Intelligence 1632, 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.

Wei99 Weidenbach C. (1999), **SPASS: Combining Superposition, Sorts and Splitting**, Robinson A., Voronkov A., *Handbook of Automated Reasoning*, Elsevier Science.

Wol98 Wolf A. (1998), **p-SETHEO: Strategy Parallelism in Automated Theorem Proving**, de Swart H., *Proceedings of International Conference TABLEAUX'98: Analytic Tableaux and Related Methods* (Oisterwijk, The Netherlands), pp.320-324, Lecture Notes in Artificial Intelligence 1397, Springer-Verlag.