CASC-

CASC-

CASC-

CASC-

CASC-

The IJCAR ATP System Competition (CASC-JC)

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

Abstract

In order to stimulate ATP system development, and to expose ATP systems to interested researchers, the IJCAR ATP System Competition (CASC-JC) will be held on 21st June 2001. CASC evaluates the performance of sound, fully automatic, 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 IJCAR ATP System Competition (CASC-JC) will be held at IJCAR[1], Siena, Italy, on 21st June 2001. 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-JC is the sixth such ATP system competition, following the successful competitions at CADEs-13 to -17 [SS97a, SS98d, SS99, Sut00a, Sut01].

Nineteen ATP systems, listed in Table 1, have been entered into the various competition and demonstration divisions. The winners of the CASC-17 divisions have been automatically entered into those divisions, to provide benchmarks against which progress can be judged (the competition archive provides access to the CASC-17 systems' executables and source code).

The design and procedures of CASC-JC evolved from those of CASCs-13 to -17 [SS97b, SS98a, SS98b, Sut99, Sut00b]. Important changes for CASC-JC are:

- The SEM division has not been continued.
- A new EPR division has been added to the competition divisions.
- The MIX division has been divided into two classes, one ranked by the number of problems solved, and one ranked by the number of acceptable proof objects output. To be eligible to win the Proof class, sample proof objects have to be supplied in advance, and the winner's proof objects are checked after the competition.
- The SAT division is divided into two categories, one with equality and one without.
- The problems are taken from an unreleased version of the TPTP.
- Incomplete and augmented TPTP problems are eligible.

[1] IJCAR incorporates CADE in 2001

- The required system properties have changed.

The full motivations for these changes are given in Section 8. The rules, deadlines, and specifications given here, are absolute. Only the competition panel has the right to make exceptions.

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 Maria Paola Bonacina, Claude Kirchner, and Jeff Pelletier. The competition will be run on computers provided by Facultät für Informatik, Technische Universität München. The CASC-JC WWW site provides access to resources used before, during, and after the event:

`http://www.cs.miami.edu/~tptp/CASC/JC`

System	Divisions	Entrant	Institution
Bliksem 1.12	TBA	Hans de Nivelle	Max-Planck-Institut für Informatik
DCTP 0.1	MIX SAT EPR	Gernot Stenz (Reinhold Letz)	TU München
E 0.6	MIX	CASC-17	CASC
E 0.62	MIX UEQ EPR	Stephan Schulz	TU München
EP 0.62	MIX*	Stephan Schulz	TU München
E-SETHEO csp01	MIX SAT FOF EPR	Gernot Stenz (Reinhold Letz, Stephan Schulz)	TU München
Gandalf c-2.3	MIX UEQ EPR	Tanel Tammet	Tallinn Technical University
GandalfFOF c-2.3	FOF	Tanel Tammet	Tallinn Technical University
GandalfSat 1.0	SAT	CASC-17	CASC
GandalfSat 1.1	SAT	Tanel Tammet	Tallinn Technical University
MUSCADET 2.3	FOF	Dominique Pastre	Université René Descartes
Otter-MACE 3.2-2.0	MIX* UEQ SAT FOF EPR	William McCune (Larry Wos, Bob Veroff)	Argonne National Laboratory
PizEAndSATO 0.1	EPR	Geoff Sutcliffe (Stephan Schulz)	University of Miami (TU München)
SCOTT 6	MIX* UEQ SAT FOF EPR	John Slaney (Kahlil Hodgson)	Australian National University
Vampire 1.0	FOF	CASC-17	CASC
Vampire CASC-JC	MIX* SAT FOF EPR	Alexandre Riazanov (Andrei Voronkov)	University of Manchester
Vampire 2.0	MIX*	Alexandre Riazanov (Andrei Voronkov)	University of Manchester
Waldmeister 600	UEQ	CASC-17	CASC
Waldmeister 600	UEQ	Thomas Hillenbrand (Bernd Loechner, Andreas Jaeger, Arnim Buch)	Max-Planck-Institut für Informatik (Universität Kaiserslautern)

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

2. Divisions

CASC-JC 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 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 assurances of the existence of a proof (the only class used up to CASC-17).
  - The Proof class: Ranked according to the number of acceptable proof objects produced.
- 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 Non-propositional Theorems and Non-theorems, 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.

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).

2.2. 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. 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

For CASC-JC, the general hardware is 28 SUN UltraSparc IIi workstations, each having:

- 440MHz UltraSparc-II CPU
- 256MB memory
- SunOS 5.8 operating system

3.2. Problems

3.2.1. Problem Selection

The problems are taken from the TPTP Problem Library. For CASC-JC, the TPTP version is v2.4.0. The TPTP version used for the competition is not released until after the system installation deadline, so that some problems have not been previously seen by the entrants.

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.
- 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 contrained so that no division or category contains an excessive number of very similar problems.
- The selection is biased to ensure that up to 50% of the problems in each category are unseen. (The percentage is dependent on how many new problems are eligible in the category. It also is limited by the "similar problems" constraint.)

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:

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

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 actually be used. The actual numbers used in each division and category is 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 by 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. 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.

The timing is 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 is set to 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

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 (a "yes" output).. 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 announced in that division. Otherwise winners are announced in each division (two class winners in the MIX division), and prizes are awarded.

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

At some time after the competition, all high ranking systems in each division are tested over the entire TPTP. This testing provides a final check for soundness, and any system found to be unsound is retrospectively disqualified. At some time after the competition, the proof objects from the winner of the MIX Proof class are checked by the panel. If any of the proof objects are unacceptable, i.e., they are significantly worse than the samples provided, then that system is retrospectively disqualified from the Proof class.

5. Entry Requirements and Procedures

To be entered into CASC, systems have to be registered using the CASC system registration form by the registration deadline. 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, using the conference 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 the previous CASC are automatically be entered into their divisions, to provide benchmarks against which progress can be judged. After the competition all systems' source code is made publically available on the CASC WWW site.

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.

5.1. System Description and Proof Objects

A system description has to be provided for each ATP system entered. The system description must fit onto a single A4 page, using 12pt times font. 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 makes predictions 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 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.

In the MIX division, the system that outputs the most acceptable proof objects (maximally one per problem solved) to `stdout` is the winner of the Proof class. Entrants who wish to be eligible for winning the Proof class have to email representative samples of their proof objects to the competition organizers before the proof objects deadline. The competition panel decides whether or not each system's proof objects are acceptable.

5.2. System Properties

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. 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.

Entrants have to 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 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. 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. 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's CPU time can be limited using the `TimeLimitedRun.c` program. For example:
```
prompt> which TimeLimitedRun
/home/tptp/bin/TimeLimitedRun
prompt> /home/tptp/bin/TimeLimitedRun -q0 10 20 /home/tptp/bin/MyATPSystem /hom
e/tptp/tmp/CCC001-1.p
TimeLimitedRun: ----------------------------------------------------------
TimeLimitedRun: /home/tptp/bin/MyATPSystem
TimeLimitedRun: CPU time limit is 10s
TimeLimitedRun: WC  time limit is 20s
TimeLimitedRun: PID is 5827
TimeLimitedRun: ----------------------------------------------------------
CPU time limit exceeded
```
- Check: The ATP system's wall clock time can be limited using the `TimeLimitedRun.c` program. For example:
```
prompt> /home/tptp/bin/TimeLimitedRun -q0 20 10 /home/tptp/bin/MyATPSystem /hom
e/tptp/tmp/CCC001-1.p
TimeLimitedRun: ----------------------------------------------------------
TimeLimitedRun: /home/tptp/bin/MyATPSystem
TimeLimitedRun: CPU time limit is 20s
TimeLimitedRun: WC  time limit is 10s
TimeLimitedRun: PID is 5827
TimeLimitedRun: ----------------------------------------------------------
Alarm clock
```

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/TimeLimitedRun -q0 200 400 /home/tptp/bin/MyATPSystem /h
ome/tptp/tmp/CCC001-1.p
TimeLimitedRun: ----------------------------------------------------------
TimeLimitedRun: /home/tptp/bin/MyATPSystem
TimeLimitedRun: CPU time limit is 200s
TimeLimitedRun: WC  time limit is 400s
TimeLimitedRun: PID is 5827
TimeLimitedRun: ----------------------------------------------------------
Proof found in 147 seconds.
```

- 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/TimeLimitedRun -q0 200 400 /home/tptp/bin/MyATPSystem -o
utput_proof /home/tptp/tmp/CCC001-1.p
TimeLimitedRun: ----------------------------------------------------------
TimeLimitedRun: /home/tptp/bin/MyATPSystem
TimeLimitedRun: CPU time limit is 200s
TimeLimitedRun: WC  time limit is 400s
TimeLimitedRun: PID is 5827
TimeLimitedRun: ----------------------------------------------------------
Proof found in 147 seconds.
START OF PROOF
     ... acceptable proof here ...
END OF PROOF
```

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/TimeLimitedRun -q0 200 400 /home/tptp/bin/MyAT
PSystem  /home/tptp/tmp/CCC001-1.p
TimeLimitedRun: ----------------------------------------------------------
TimeLimitedRun: /home/tptp/bin/MyATPSystem
TimeLimitedRun: CPU time limit is 200s
TimeLimitedRun: WC  time limit is 400s
TimeLimitedRun: PID is 5827
TimeLimitedRun: ----------------------------------------------------------
Proof found in 147 seconds.
 real     3:04.2
 user     2:19.3
 sys         7.6
```

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/TimeLimitedRun -q0 200 400 /home/tptp/bin/MyATPSystem /h
ome/tptp/tmp/CCC001-1.p
TimeLimitedRun: ----------------------------------------------------------
TimeLimitedRun: /home/tptp/bin/MyATPSystem
TimeLimitedRun: CPU time limit is 200s
TimeLimitedRun: WC  time limit is 400s
TimeLimitedRun: PID is 13526
TimeLimitedRun: ----------------------------------------------------------
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/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/TimeLimitedRun -q0 200 400 /home/tptp/bin/MyA
TPSystem /home/tptp/tmp/CCC001-1.p) & (/bin/time /home/tptp/bin/TimeLimitedRun
-q0 200 400 /home/tptp/bin/MyATPSystem /home/tptp/tmp/CCC001-1.p)
TimeLimitedRun: ----------------------------------------------------------
TimeLimitedRun: /home/tptp/bin/MyATPSystem
TimeLimitedRun: CPU time limit is 200s
TimeLimitedRun: WC  time limit is 400s
TimeLimitedRun: PID is 5827
TimeLimitedRun: ----------------------------------------------------------
TimeLimitedRun: ----------------------------------------------------------
TimeLimitedRun: /home/tptp/bin/MyATPSystem
TimeLimitedRun: CPU time limit is 200s
TimeLimitedRun: WC  time limit is 400s
TimeLimitedRun: PID is 5829
TimeLimitedRun: ----------------------------------------------------------
Proof found in 147 seconds.
Proof found in 147 seconds.
 real     3:04.2
 user     2:19.3
 sys         7.6
 real     3:04.2
 user     2:19.3
 sys         7.6
```

5.3. System Installation

Access to the general hardware (or equivalent) is available from the general hardware access deadline. Entrants have 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 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. 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 (i.e., deliver your installation package before the installation deadline so if the installation fails you have a chance to fix it!). If you are in doubt about your installation package or procedure, please email the competition organizers.

After the installation deadline the organizers test the ATP systems, first to check that the systems execute correctly (according to the checks described in Section 5.2), and secondly to test 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) are submitted to the systems participating in the MIX, UEQ, FOF, and EPR divisions, and theorems (selected from the TPTP) are submitted to the systems participating 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 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 is performed after the competition, as described in Section 4.

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

5.4. 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 an ATP system's success string is found then the timing information from the `/bin/time` command is extracted. The CPU time taken, or the CPU time limit if no solution was found, is recorded. This data is used to generate an HTML file, and a WWW browser is used to display the results.

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

6. The ATP Systems

These system descriptions were written by the entrants.

6.1. Bliksem 1.12

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

6.1.1. Architecture

Bliksem is a theorem prover implementing resolution and paramodulation. It implements many different orderings, 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. We plan to rewrite Bliksem, but at this moment we are not certain about the direction in which Bliksem should develop. Bliksem is available from:

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

6.1.3. Expected Competition Performance

Bliksem did not change significantly in the last 1.5 years. But the competition has changed, and the other systems as well. We will see ...

6.2. DCTP 0.1

R. Letz, G. Stenz
Institut für Informatik, Technische Universität München, Germany
{letz,stenz}@informatik.tu-muenchen.de

6.2.1. Architecture

DCTP 0.1 [LS01] is an automated theorem prover for first order clause logic. It is an implementation of the disconnection calculus described in [Bil96]. The disconnection calculus is a proof confluent and inherently cut-free tableau calculus with a weak connectedness condition. Initially, an initial active path is selected through the input clause set, which is then expanded by repeated application of the single inference rule, the linking rule: Two clauses that are linked via the active path are unified and appended to the bottom of the active path. The new path containing the linked literals is closed, The remaining new paths are solved in the same manner, until no open path remains in the tableau, in which case the input set is considered proven, or no further links can be applied, then a model for the input set can be derived from the open path. The concept of branch closure is different from most other tableau calculi. A branch is considered closed iff it is ground closed, i.e. it contains a pair of contradicting ground literals when all variables on the branch are instantiated to the same new constant symbol.

The inherently depth-first proof search is guided by a literal selection based on literal instantiatedness and a heavily parameterized link selection. The pruning mechanisms indispensable for a proof confluent calculus mostly rely on different forms variant deletion and unit based strategies. Additionally the calculus has been augmented by relevancy based backtracking.

The current version has no builtin equality treatment, that is, equality problems must be augmented by the full set of equality axioms to preserve completeness. Currently a version of ordered paramodulation, different from the proposals in Billon's paper, is being integrated into DCTP, but will not be ready for the competition.

6.2.2. Implementation

DCTP has been implemented in the Bigloo dialect of the Scheme language. Major emphasis has been placed on memory preserving techniques, as a proof confluent calculus otherwise easily gobbles up all available resources. All parts of the tableau no longer needed are immediately discarded. DCTP uses a kind of weak term sharing via bound variables to prevent excessive replication of terms during unification.Expected Competition Performance

6.2.3. Expected Competition Performance

As DCTP is a very new system and little experience could be gathered on the effects of the existing guidance heuristics, generally DCTP is not expected to perform on par with the current state of the art systems. Slightly better performance might possibly be expected in the new EPR division.

6.3. E 0.6

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

6.3.1. 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.

6.3.2. 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
```

6.4. E 0.62 and EP 0.62

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

6.4.1. Architecture

E 0.62 [Sch01] 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. New additions to the calculus since E 0.6 include AC redundancy elimination and AC simplification for dynamically recognized associative and commutative equational theories, as well as simulated clause splitting (inspired by Vampire).

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 0.62 is just a combination of E 0.62 in full verbose mode and a proof generation tool. Proof object generation is handled by complex postprocessing of the full search protocol.

6.4.2. 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 of the latest released version and a current snapshot are available freely from:

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

EP 0.62 is a simple Bourne shell script calling E and the postprocessor in a pipeline.

6.4.3. Expected Competition Performance

In the last year, E won the MIX category of CASC and came in fourth 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. We also hope to improve in the UEQ category, although we probably will not nearly reach the performance of Waldmeister.

EP 0.62 will be hampered by the fact that it has to reconstruct proofs in a post-processing step that typically usually uses at least the same amount of time as the proof search itself. This is particularly grave because E is able to find a relatively large number of proofs late in the proof search.

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

6.5. E-SETHEO csp01

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

6.5.1. 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 [LS01] as a new strategy as well as a new version of the E prover.

6.5.2. 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.

6.5.3. Expected Competition Performance

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

6.6. Gandalf and GandalfFOF c-2.3

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

6.6.1. 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.

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.

GandalfFOF c-2.3 is a pipeline consisting of Otter [McC94], which converts the problem to CNF, and Gandalf c-2.3.

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

6.6.2. 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.

6.6.3. 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.

6.7. GandalfSat 1.0

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

6.7.1. 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].

6.7.2. 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.

6.8. GandalfSat 1.1

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

6.8.1. 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].

6.8.2. 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.

6.8.3. Expected Competition Performance

We expect it to be among the better provers for satisfiability checking.

6.9. MUSCADET v2.3

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

6.9.1. Architecture

The MUSCADET theorem prover is a knowledge-based system. It is based on Natural Deduction, following the terminology of [Ble71] and [Pas78], and uses methods which resembles those used by humans. It is composed of an inference engine, which interprets and executes rules, and of one or several bases of facts, which are the internal representation of "theorems to be proved". Rules are either universal and put into the system, or built by the system itself by metarules from data (definitions and lemmas). Rules may add new hypotheses, modify the conclusion, create objects, split theorems into two or more subtheorems or build new rules which are local for a (sub-)theorem.

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

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

6.9.2. Implementation

MUSCADET 2 [Pas01] is implemented in SWI-Prolog. Rules are written as declarative Prolog clauses. Metarules are written as sets of Prolog clauses, more or less declarative. The inference engine includes the Prolog interpreter and some procedural Prolog clauses. MUSCADET 2.3 can be obtained from:

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

6.9.3. 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.

6.10. Otter-MACE 3.2-2.0

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

6.10.1. Architecture

Otter 3.2 [McC94, McC00b, MW97] and MACE 2.0 [McC01, McC00a] are introduced in this section. It describes the calculus, inference rules, and search control used. 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.) 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. Roughly speaking, MACE works as follows. For a given domain size, an equivalent propositional problem is constructed and a Davis-Putnam-Loveland-Logeman decision procedure is applied. MACE typically cannot handle clauses with many variables or deeply nested terms.

6.10.2. Implementation

The programs are written in C and accept the same input files. 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-Logeman procedure uses a counting algorithm to do the unit resolutions, and forward subsumption is not applied.

6.10.3. Expected Competition Performance

Otter's original automatic mode, which reflects no tuning to the TPTP problems, will be used. Otter is especially ineffective on non-Horn problems, so it is not expected to do well on the NNE or NEQ categories. Otter's performance in the other categories should be acceptable. MACE won the SAT division at the 1999 CASC and should do well on the SAT problems this year.

Acknowledgments: Ross Overbeek, Larry Wos, Bob Veroff, and Rusty Lusk contributed to the development of Otter. MACE was motivated by previous work of John Slaney, Mark Stickel, and Hantao Zhang.

6.11. PizEAndSATO 0.1

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

6.11.1. Architecture

PizEAndSATO is a system for CNF first order problems with a finite Herbrand universe. It is implemented by combining the ground-instance generating program *PizE* with the propositional prover *SATO*.

PizE is currently a very simple program. It systematically enumerates all ground instances of a clause set with a finite Herbrand universe. The program is not yet very much optimized. There are only two techniques implemented to cut down on the number of instances generated. First, PizE will split clause input clauses that can be partitioned into variable-disjoint subsets. Secondly, it performs forward unit subsumption and forward unit resolution on the generated ground instances to simplify the formula and to reduce the amount of memory used.

SATO [ZS00], is a highly optimized implementation of the Davis-Putnam-Loveland-Logemann procedure. SATO is well known and has been sucessfully applied in a variety of challenging propositional domains.

PizE and SATO are combined with a shell script called *And*.

6.11.2. Implementation

PizE is implemented in ANSI C. It is based on the E [Sch01] libraries, shares a large amount of code with E, and will be bundled with future releases of E. Since E is optimized for the first order case, clause representation in PizE is far from optimal for propositional clauses. SATO is implemented in C. It implements its own highly efficient data structures and algorithms.

6.11.3. Expected Competition Performance

We expect a reasonable performance in the EPR division. Testing indicates that the only problems that have very many clauses instances are beyond PizEAndSATO's reach, because PizE runs out of memory.

6.12. SCOTT 6

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

6.12.1. Architecture

SCOTT (Semantically Constrained OTTER) [HS99] 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. (This can be seen to loosely generalize 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.

6.12.2. Implementation

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

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

6.12.3. Expected Competition Performance

In principle SCOTT should perform at least as well as OTTER across all parts of the MIX and UEQ divisions, if only because it has OTTER as a subroutine. 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. Preliminary results show that, SCOTT-6 out performs SCOTT-5, so we expect an improvement on the performance in CASC-17. Overall, we expect good performance in comparison with cognate systems that employ relatively naive guidance.

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`

6.14. Vampire 2.0 and Vampire CASC-JC

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

6.14.1. 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.

VampireCASC-JC is a version based on Vampire 2.0, specialized 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.

6.14.2. 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
```

6.14.3. Expected Competition Performance

We expect VampireCASC-JC to perform much better than Vampire 2.0.

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
```

6.16. Waldmeister 601

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

6.16.1. 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.

6.16.2. 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
```

6.16.3. Expected Competition Performance

Once again, the Waldmeister system being that powerful already, we hope to improve over last year's version in some problem domains.

7. Conclusion

The CADE-JC ATP System Competition is the sixth large scale competition for 1st order ATP systems. The organizers believe that CASC-JC fulfills its main motivations: stimulation of research, motivation for improving implementations, evaluation of relative capabilities of ATP systems, and providing an exciting event. The design changes for CASC-JC, particularly the introduction of the MIX Proof class, and the use of unseen problems, are expected to provide extra interest and further contribute to the evaluation of the systems.

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.

8. Appendix: Motivations for Design Changes

8.1. Discontinuance of the SEM Division

Although the SEM division was well motivated, and at CASC-16 many entrants expressed an interest and desire for such a division, at CASC-17 no systems were especially tuned to the chosen application domain of set theory. Therefore:

- The SEM division has not been continued.

8.2. Addition of the EPR Division

There are many CNF problems, often from 'real world' applications, that have a finite Herbrand Universe. These problems can be converted to propositional problems, by generating all ground instances of the clauses and assigning a proposition to each unique ground atom. Such problems are decidable, and may be tackled using 1st order techniques, or by conversion to propositional form and using propositional techniques. Some approaches are capable of deciding such problems, while others can only determine that the clause set is unsatisfiable, and others can only determine that the clause set is satisfiable. It would be interesting to evaluate and compare the 1st order and propositional approaches. The addition of propositional systems to CASC would also contribute to making CASC a stimulating event. Therefore:

- CASC-JC has an EPR division, using problems with a finite Herbrand Universe.
- The EPR division has two categories:
  - The **EPT** category: Effectively Propositional Theorems (unsatisfiable clauses)
  - The **EPS** category: Effectively Propositional non-theorems (Satisfiable clauses)

8.3. Two Classes in the MIX Division

ATP systems have three main types of successful output:

- A logical solution:
  - For theorems, proofs in which each formula is either an input formula, or deduced (following the semantics of the logic) from the input formulae and other deduced formulae.
  - For non-theorems, models.
- A meta-solution, that proves that there is a logical solution. Refutations are meta-solutions.
- An assurance ("yes") that there is a logical solution.

For theorems, logical solutions and meta-solutions are collectively called proof objects.

There is evidence that for some applications of ATP there is a need for the production of logical or meta-solutions, and that the output must be produced as part of the system run, i.e., the production of the solution cannot be deferred to a later run of the system. The need for output as part of the first system run appears to come mainly from applications whose problems are CNF theorems that would fall into the MIX division, i.e., there is a demand for the production of proof objects.

Current systems' proof objects vary in format and level of detail. For CNF theorems, many systems output at least a refutation, and the ATP community is experienced in examining these. For this type of problem, it therefore seems possible to say whether or not a system's proof objects are acceptable. For FOF theorems, most decent current systems convert the problem to CNF and work with the CNF. The output, although acceptable wrt the CNF, may not be meaningful in terms of the original FOF problem. For non-theorems, finite models can be easily presented, while for non-finite models the situation is less clear.

Therefore, CASC-JC ensures the appropriate recognition of the production of acceptable proof objects:

- The MIX division has two classes:
    - The **Assurance** class: Ranked according to the number of assurances of the existence of a proof (the only class used up to CASC-17).
    - The **Proof** class: Ranked according to the number of acceptable proof objects output.
- For the Proof class, only acceptable proof objects count towards winning. Entrants who wish to be eligible for winning the Proof class must supply representative samples of their proof objects before the competition. The competition panel decides whether or not a system's proof objects are acceptable.
- The proof objects from the winner of the Proof class are checked by the panel. If any of the proof objects are unacceptable, i.e., they are significantly worse than the samples provided, then that system is retrospectively disqualified from the Proof class.

8.4. Two categories in the SAT division

Various ATP systems and techniques have been observed to be particularly well suited to problems with certain syntactic characteristics. Due to this specialization, empirical evaluation of ATP systems must be done in the context of problems that are reasonably homogeneous with respect to the systems. This motivates the syntactically defined divisions and categories of CASC. Evaluation of ATP systems within such divisions and categories makes it possible to say which systems work well for what types of problems. This identifies specialist capabilities of ATP systems, while general capabilities can be inferred from the separate division and category capabilities.

Recent analysis of ATP systems' performances on problems in the SAT division has shown that there is specialization between problems with equality and problems without equality. It is therefore appropriate to separate these two types of problems in the SAT division, so that the specialist capabilities can be observed.

Therefore:

- The SAT divsion has two categories:
    - The **SNE** category: SAT with no Equality
    - The **SEQ** category: SAT with Equality

8.5. Non-availability of the TPTP version used

Over the last few CASCs there has been an increase in the extent to which systems have been tuned to the TPTP problems that have been likely to be eligible for use. This is problematic, as these sets of problems have been small in some categories/divisions, and thus tuning has become close to precomputation and storage of information for individual TPTP problems.

Various mechanisms have been proposed to limit the extent of the tuning for CASC, e.g., a limit on the number of strategies from which a system may select, based on problem characteristics. The most direct approach, one that corresponds directly to ensuring that "work done using the TPTP should also be useful in a broader context" (as discussed at the CASC panel at CADE-17), is to include unseen problems in CASC. Previously this was not possible, because the CASC organizers did not have the resources to independently evaluate the difficulty, and hence eligibility, of new problems. Rather they relied on performance data kindly submitted by developers. That in turn made it necessary to make the problems available in a TPTP release well in advance of CASC, and hence also available for tuning. This situation has now changed, as thanks to the Australian government the organizers now have almost dedicated use of a new 4 CPU SUN, so that new problems can be evaluated privately, without making the problems publically available.

Therefore:

- CASC-JC will use TPTP v2.4.0, or a patch thereof.
- The TPTP version used for the competition is not released until after the system installation deadline, so that some problems have not been previously seen by the entrants.
- The selection mechanism is biased to ensure that up to 50% of the problems in each category are unseen. (The percentage is dependent on how many new problems are eligible in the category. It is also limited by the mechanism that prevents having an excessive number of very similar problems in any category.)

This change means that it is disadvantagous to tune excessively to previously available TPTP problems. Rather, systems that are fundamentally strong, and whose strategies are useful beyond previously used test problems, should succeed. Currently there are over 1600 problems prepared for TPTP v2.4.0 that are not in TPTP v2.3.0. We hope that many more will be contributed before TPTP v2.4.0 is finalized. (But note, because of the random selection of problems in CASC and the mechanism for avoiding many similar problems, it is impossible to bias the competition by submitting large numbers of problems that are solved by only some particular system.)

8.6. Eligibility of Non-standard Problems

The TPTP distinguishes versions of problems as one of standard, incomplete, augmented, especial, or biased. Previously, only standard and especial problems were eligible for use in CASC. Incomplete and augmented problems were excluded, as there was a perceived danger that the modifications that make the problems incomplete or augmented were made with a particular ATP system in mind, i.e., the problems were considered to be potentially biased. Recent discussions have led to the conclusion that the modifications would generally be effective for all ATP systems, and that the problems are therefore not biased. Therefore:

- All except biased problems are eligible.

8.7. Changes to required System Properties

The ATP systems entered into CASC are run in various ways by the competition organizers, firstly to ensure that they have the required properties, and of course subsequently in the competition itself. In these runs problems with various strange names are used, and it is necessary that the ATP systems are robust to these. Therefore:

- No assumptions may be made about the format of the problem file name.

In the competition divisions a 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, and is imposed by sending a SIGALRM when the wall clock time limit is reached. Therefore:

- The ATP systems that run on the general hardware have to be interruptable by a SIGALRM signal, so that the wall clock time limit can be imposed on each solution attempt.

In the MIX division, the system that outputs the most acceptable proof objects (maximally one per problem solved) to `stdout` is the winner of the Proof class. It is necessary that the proofs be easily identifiable, so that their acceptability can be verified. Therefore:

- When outputing proofs for the MIX division proof class, the start and end of the proof must be identified by distinguished strings (specified by the entrant).

Execution of the ATP systems on the general hardware is controlled by a `perl` script. The jobs are queued onto the workstations so that each workstation is running one job at a time. This means that multiple copies of the ATP systems have to be executable concurrently on different machines but in the same (NFS cross mounted) directory. Previously the competition organizers have had to modify many of the ATP systems to make this possible. Now the responsibility has been transferred to the entrants.

- Multiple copies of the ATP systems have to be executable concurrently on different machines but in the same (NFS cross mounted) directory.

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.

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

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

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.

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

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.

LS01 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.

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.

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/.

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

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

NH+01 Nieuwenhuis R., Hillenbrand T., Riazanov A., Voronkov A. (2001), **On the Evaluation of Indexing Techniques for Theorem Proving**, Gore R., Leitsch A., Nipkow T., *Proceedings of the International Joint Conference on Automated Reasoning* (Siena, Italy), Lecture Notes in Artificial Intelligence, Springer-Verlag.

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

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

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

Pas01 Pastre D. (2001), **MUSCADET version 2.3 : User's Manual**, http://www.math-info.univ-paris5.fr/~pastre/muscadet/manual-en.ps.

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.

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.

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.

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.

Sut01 Sutcliffe G. (To appear), **The CADE-17 ATP System Competition**, *Journal of Automated Reasoning*.

SS01 Sutcliffe G., Suttner C.B. (To appear), **Evaluating General Purpose Automated Theorem Proving Systems**, *Artificial Intelligence*.

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 ATPProgress**, 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.

ZS00 Zhang H., Stickel M. (2000), **Implementing the Davis-Putnam Method**, *Journal of Automated Reasoning* 24(1/2), pp.277-296.