The CADE ATP System Competition (CASC-J2)

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

June 5, 2004

Abstract

The 2nd IJCAR ATP System Competition (CASC-J2) will be held on 6th July 2004. CASC evaluates the performance of sound, fully automatic, classical first-order logic, ATP systems. The evaluation is in terms of:

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

in the context of:

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

1 Introduction

The CADE conferences are the major forum for the presentation of new research in all aspects of automated deduction. In order to stimulate ATP research and system development, and to expose ATP systems within and beyond the ATP community, the CADE ATP System Competition (CASC) is held at each CADE conference. CASC-J2 will be held at the 2nd International Joint Conference on Automated Reasoning (IJCAR, which incorporates CADE), on 6th July 2004. CASC-J2 is the ninth such ATP system competition [51, 58, 55, 42, 44, 50, 48, 49].

CASC evaluates the performance of sound, fully automatic, classical first-order logic, ATP systems. The evaluation is in terms of the number of problems solved, the number of acceptable proofs and models produced, and the average runtime for problems solved, in the context of a bounded number of eligible problems chosen from the TPTP problem library [54], and a specified time limit for each solution attempt.

Twenty-two ATP systems and variants, listed in Table 1, have been entered into the various competition and demonstration divisions. The winners of the CASC-19 divisions have been automatically entered into those divisions, to provide benchmarks against which progress can be judged (the competition archive provides access to the systems' executables and source code).

The design and procedures of this CASC evolved from those of previous CASCs [57, 52, 47, 53, 40, 41, 43, 45, 46]. Important changes for this CASC are:

- The FOF division has been divided into two ranking classes, one ranked by the number of problems solved, and one ranked by the number of problems solved with acceptable proof output.

The competition organizers are Geoff Sutcliffe and Christian Suttner. The competition is overseen by a panel of knowledgeable researchers who are not participating in the event; the panel members are Alan Bundy, Uli Furbach, and Jeff Pelletier. The rules, specifications, and deadlines given here are absolute. Only the panel has the right to make exceptions. The competition will be run on computers provided by the Department of Computer Science at the University of Manchester. The CASC-J2 WWW site provides access to resources used before, during, and after the event:

`http://www.tptp.org/CASC/J2`

2 Divisions

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

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

2.1 The 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

Table 1: The ATP systems and entrants

ATP System	Divisions	Entrants	Affiliation
Darwin CASC-J2	MIX SAT* EPR	Alexander Fuchs, Peter Baumgartner, Cesare Tinelli	Universität Koblenz-Lindau, Max-Planck-Institut für Informatik, The University of Iowa
DCTP 1.3-EPR	EPR	CASC	*CASC-19 EPR winner*
DCTP 1.31	MIX SAT EPR	Gernot Stenz, Reinhold Letz	Technische Universität München
DCTP 10.21p	MIX FOF SAT EPR	Gernot Stenz, Reinhold Letz	Technische Universität München
Dilemma 0.1	FOF (demo)	Magnus Björk	Chalmers University of Technology
E 0.82	MIX FOF UEQ	Stephan Schulz	Technische Universität München
EP 0.82	MIX* FOF*		*E 0.82 variant*
E-SETHEO csp04	MIX FOF SAT EPR UEQ	Gernot Stenz, Reinhold Letz	Technische Universität München
Gandalf c-2.6-SAT	SAT	CASC	*CASC-19 SAT winner*
Mace2 2.2	SAT*	William McCune	Argonne National Laboratory
Mace4 2004-D	SAT*	William McCune	Argonne National Laboratory
Octopus 2004	MIX (demo)	Monty Newborn, Zongyan Wang	McGill University
Otter 3.3	MIX* FOF UEQ	William McCune	Argonne National Laboratory
Paradox 1.0	SAT*	CASC	*CASC-19 SAT winner*
Paradox 1.1-casc	SAT* EPR	Koen Claessen, Niklas Sörensson	Chalmers University of Technology
SOS 1.0	MIX* UEQ	John Slaney	Australian National University
THEO J2004	MIX	Monty Newborn, Zongyan Wang	McGill University
Vampire 5.0	FOF	CASC	*CASC-19 FOF winner*
Vampire 6.0	MIX*	CASC	*CASC-19 MIX winner*
Vampire 7.0	MIX* FOF* EPR UEQ	Andrei Voronkov, Alexandre Riazanov	The University of Manchester
Waldmeister 702	UEQ	CASC	*CASC-18 UEQ winner*
Waldmeister 704	UEQ	Thomas Hillenbrand, Bernd Löchener, Jean-Marie Gaillourdet	Max-Planck-Institut für Informatik, Technische Universität Kaiserslautern

MIX* indicates participation in the MIX division proof class,
FOF* indicates participation in the FOF division proof class, and
SAT* indicates participation in the SAT division model class - see Section 2.

- The **PEQ** category: Pure Equality

The MIX division has two ranking classes:

- The Assurance class: Ranked according to the number of problems solved (a "yes" output, giving an assurance of the existence of a proof).
- The Proof class: Ranked according to the number of problems solved with an acceptable proof output on `stdout`. The competition panel judges whether or not each system's proof format is acceptable.

The **FOF** division: Mixed FOF non-propositional theorems.
The FOF Division has two problem categories:

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

The FOF division has two ranking classes:

- The Assurance class: Ranked according to the number of problems solved (a "yes" output, giving an assurance of the existence of a proof).
- The Proof class: Ranked according to the number of problems solved with an acceptable proof output on `stdout`. The competition panel judges whether or not each system's proof format is acceptable.

The **SAT** division: Mixed CNF really-non-propositional non-theorems.
The SAT Division has two problem categories:

- The SNE category: SAT with No Equality
- The SEQ category: SAT with Equality

The SAT division has two ranking classes:

- The Assurance class: Ranked according to the number of problems solved (a "yes" output, giving an assurance of the existence of a model).
- The Model class: Ranked according to the number of problems solved with an acceptable model output on `stdout`. The competition panel judges whether or not each system's model format is acceptable.

The **EPR** division: CNF effectively propositional theorems and non-theorems.
Effectively propositional means non-propositional with a finite Herbrand Universe. The EPR Division has two problem categories:

- The EPT category: Effectively Propositional Theorems (unsatisfiable clauses)
- The EPS category: Effectively Propositional non-theorems (Satisfiable clauses)

The **UEQ** division: Unit equality CNF really non-propositional theorems.

Section 3.2 explains what problems are eligible for use in each division and category.

2.2 The Demonstration Division

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

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

3 Infrastructure

3.1 Hardware and Software

The general hardware is 64 Dell computers, each having:

- AMD Athlon XP 2200+, 1797MHz CPU
- 512MB memory
- Linux 2.4.20-30.9 operating system

3.2 Problems

3.2.1 Problem Selection

The problems are from the TPTP problem library, version v2.7.0. The TPTP version used for the competition is not released until after the system installation deadline, so that new problems have not seen by the entrants.

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

- The TPTP uses system performance data to compute problem difficulty ratings, and from the ratings classifies problems as one of [56]:
  - 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: Theoremhood unknown

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

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

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

3.2.2 Number of Problems

The minimal numbers of problems that have to be used in each division and category, to ensure sufficient confidence in the competition results, are determined from the numbers of eligible problems in each division and category [9] (the competition organizers have to ensure that there is sufficient CPU time available to run the ATP systems on this minimal number of problems). The minimal numbers of problems is used in determining the CPU time limit imposed on each solution attempt - see Section 3.3.

A lower bound on the total number of problems to be used is determined from the number of computers 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:

$$NumberOfProblems = \frac{NumberOfComputers * TimeAllocated}{NumberOfATPSystems * CPUTimeLimit}$$

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

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

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

3.2.3 Problem Preparation

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

- rename all predicate and function symbols to meaningless symbols
- randomly reorder the clauses and literals in CNF problems
- randomly reorder the formulae in FOF problems
- randomly reverse the unit equalities in UEQ problems
- remove equality axioms that are not needed by the ATP systems

- add equality axioms that are needed by the ATP systems
- output the problems in the formats required by the ATP systems. (The clause type information, one of `axiom`, `hypothesis`, or `conjecture`, may be included in the final output of each formula.)

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

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

3.3 Time Limits

In the competition divisions, CPU and wall clock time limits are imposed on each solution attempt. A minimal CPU time limit of 240 seconds is used. The maximal CPU time limit is determined using the relationship used for determining the number of problems, with the minimal number of problems as the $NumberOfRoblems$. The CPU time limit is chosen as a reasonable value within the range allowed, and is announced at the competition. The wall clock time limit is imposed in addition to the CPU time limit, to prevent very high memory usage that causes swapping. The wall clock time limit is double the CPU time limit.

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

4 Performance Evaluation

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

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

At some time after the competition, all high ranking systems in each division are tested over the entire TPTP. This provides a final check for soundness. If a system is found to be unsound, and it cannot be shown that the unsoundness did not manifest itself in the competition, then the system is retrospectively disqualified. At some time after the competition, the proofs from the winners of the MIX and FOF division proof classes, and the models from the winner of the SAT division model class, are checked by the panel. If any of the proofs or models are unacceptable, i.e., they are significantly worse than the samples provided, then that system is retrospectively disqualified. All disqualifications are reported in the journal paper about the competition.

5 System Properties

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

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

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

5.1 System Checks

The ATP systems have to be executable by a single command line, using an absolute path name for the executable, which may not be in the current directory. The command line arguments are the absolute path name of 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 name for 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 of 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> ln -s /home/tptp/TPTP/Problems/GRP/GRP001-1.p \_foo-Blah
prompt> /home/tptp/bin/MyATPSystem \_foo-Blah
Proof found in 147 seconds.
```

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

- Check: The ATP system can run under the `TreeLimitedRun` program (sources are available from the CASC-J2 WWW site). For example:

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

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

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

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

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

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

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

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

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

```
prompt> /home/tptp/bin/TreeLimitedRun -q0 200 400 /home/tptp/bin/MyATPSystem
        /home/tptp/tmp/CCC001-1.p
TreeLimitedRun: -----------------------------------------------
```

```
TreeLimitedRun: /home/tptp/bin/MyATPSystem
TreeLimitedRun: CPU time limit is 200s
TreeLimitedRun: WC  time limit is 400s
TreeLimitedRun: PID is 5827
TreeLimitedRun: ----------------------------------------------
Proof found in 147 seconds.
FINAL WATCH: 147.8 CPU 150.0 WC
```

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

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

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

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

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

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

```
prompt> pwd
/home/tptp
prompt> /home/tptp/bin/TreeLimitedRun -q0 200 400 /home/tptp/bin/MyATPSystem
```

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

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

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

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

5.2 System Delivery

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

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

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

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

5.3 System Execution

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

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

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

6 Entry Procedures

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

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

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

6.1 System Description

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

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

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

6.2 Sample Solutions

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

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

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

7 The ATP Systems

These system descriptions were written by the entrants.

7.1 Darwin CASC-J2

Peter Baumgartner[1], Alexander Fuchs[2], Cesare Tinelli[3]
[1]Max-Planck-Institut für Informatik Saarbrücken, Germany,
`baumgart@mpi-sb.mpg.de`
[2]Universität Koblenz-Landau, Germany
`alexander.fuchs@uni-koblenz.de`
[3]The University of Iowa, USA
`tinelli@cs.uiowa.edu`

Architecture

Darwin [4] is an automated theorem prover for first order clausal logic. It is the first implementation of the Model Evolution Calculus [5]. The Model Evolution Calculus lifts the propositional DPLL procedure to first-order logic. One of the main motivations for this approach was the possibility of migrating to the first-order level some of those very effective search techniques developed by the SAT community for the DPLL procedure.

The current version of Darwin implements first-order versions of unit propagation inference rules analogously to a restricted form of unit resolution and subsumption by unit clauses. To retain completeness, it includes a first-order version of the (binary) propositional splitting inference rule.

Proof search in Darwin starts with a default interpretation for a given clause set, which is evolved towards a model or until a refutation is found.

Implementation

The central data structure is the *context*. A context represents an interpretation as a set of first-order literals. The context is grown by using instances of literals from the input clauses. The implementation of Darwin is intended to support basic operations on contexts in an efficient way. This involves the handling of large sets of candidate literals for modifying the current context. The candidate literals are computed via simultaneous unification between given clauses and context literals. This process is sped up by storing partial unifiers for each given clause and merging them for different combinations of context literals, instead of redoing the whole unifier computations. For efficient filtering of unneeded candidates against context literals, discrimination tree or substitution tree indexing is employed. The splitting rule generates choice points in the derivation which are backtracked using a form of backjumping similar to the one used in DPLL-based SAT solvers. A generalized version of Backjumping called Dynamic Backtracking is implemented by representing the derivation tree by a graph of states and their dependencies.

Darwin is implemented in OCaml and has been tested under Linux. It is available from:

`http://www.mpi-sb.mpg.de/ baumgart/DARWIN`

Strategies

Darwin traverses the search space by iterative deepening over the term depth of candidate literals. Darwin employs a uniform search strategy for all problem classes.

Expected Competition Performance

Darwin is a first prototype implementation for the Model Evolution calculus. We expect its performance to be strong in the EPR division; we anticipate performance below average in the MIX division, and weak performance in the SAT division.

7.2 DCTP 1.3-EPR

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

Architecture

DCTP 1.3 [37] is an automated theorem prover for first order clause logic. It is an implementation of the disconnection calculus described in [6, 13, 38]. The disconnection calculus is a proof confluent and inherently cut-free tableau calculus with a weak connectedness condition. The inherently depth-first proof search is guided by a literal selection based on literal instantiatedness or literal complexity and a heavily parameterised link selection. The pruning mechanisms mostly rely on different forms of *variant deletion* and *unit based strategies*. Additionally the calculus has been augmented by full tableau pruning.

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

Implementation

DCTP 1.3 has been implemented as a monolithic system in the Bigloo dialect of the Scheme language. The most important data structures are perfect discrimination trees, which are used in many variations. It runs under Solaris and Linux.

Strategies

DCTP 1.3 is a single strategy prover.

Expected Competition Performance

DCTP 1.3-EPR is the CASC-19 EPR division winner.

7.3 DCTP 1.31 and 10.21p

Gernot Stenz
Technische Universität München, Germany
`stenzg@informatik.tu-muenchen.de`

Architecture

DCTP 1.31 [37] is an automated theorem prover for first order clause logic. It is an implementation of the disconnection calculus described in [6, 13, 38]. The disconnection calculus is a proof confluent and inherently cut-free tableau calculus with a weak connectedness condition. The inherently depth-first proof search is guided by a literal selection based on literal instantiatedness or literal complexity and a heavily parameterised link selection. The pruning mechanisms mostly rely on different forms of *variant deletion* and *unit based strategies.* Additionally the calculus has been augmented by full tableau pruning.

DCTP 10.21p is a strategy parallel version using the technology of E-SETHEO [39] to combine several different strategies based on DCTP 1.31.

Implementation

DCTP 1.31 has been implemented as a monolithic system in the Bigloo dialect of the Scheme language. The most important data structures are perfect discrimination trees, which are used in many variations. DCTP 10.21p has been derived of the Perl implementation of E-SETHEO and includes DCTP 1.31 as well as the E prover as its CNF converter. Both versions run under Solaris and Linux.

We are currently integrating a range of new techniques into DCTP which are mostly based on the results described in [15], as well as a certain form of unit propagation. We are hopeful that these improvements will be ready in time for CASC-J2.

Strategies

DCTP 1.31 is a single strategy prover. Individual strategies are started by DCTP 10.21p using the schedule based resource allocation scheme known from the E-SETHEO system. Of course, different schedules have been precomputed for the syntactic problem classes. The problem classes are more or less identical with the sub-classes of the competition organisers. Again, we have no idea whether or not this conflicts with the organisers' tuning restrictions.

Expected Competition Performance

We expect both DCTP 1.31 and DCTP 10.21p to perform reasonably well, in particular in the EPR (in any case) and SAT (depending on the selection of problems for the competition) categories.

7.4 Dilemma 0.1

Magnus Björk
Chalmers University of Technology, Sweden
mab@cs.chalmers.se

Architecture

Dilemma 0.1 is a theorem prover based on an extension of Stålmarck's method to first order classical logic [17, 18, 19]. This calculus is proof confluent and does not use backtracking. It incorporates all non-branching rules of both KE and KI, and also the dilemma rule, that is a branch-and-merge rule. It works like the analytic cut rule, except that the branches can be recombined to one branch again, keeping the intersection of all new consequences. Dilemma formulas may contain free variables (called *rigid*), that may not be substituted in the branches, but are turned into universal variables after a merge. Rigid variables are not instantiated destructively, but possible instantiations are detected and stored for later, when they will be used to find more specialized dilemma formulas.

Implementation

The system is currently being developed in C++, and if a working prototype exists at the time of the CASC-J2 then it will enter the demonstration division.

Strategies

The prototype will have no special strategies.

Expected Competition Performance

The prototype will probably not be very competitive. We enter mostly for the possibility of getting feedback.

7.5 E and EP 0.82

Stephan Schulz
Technische Universität München, Germany, and ITC/irst, Italy
`schulz@informatik.tu-muenchen.de`

Architecture

E 0.82 [32, 34] is a purely equational theorem prover. The core proof procedure operates on formulas in clause normal form, using a calculus that 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. The basic calculus is extended with rules for AC redundancy elemination, some contextual simplification, and pseudo-splitting.

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).

E uses a preprocessing step to convert formulas in full first order format to clause normal form. Preprocessing also unfolds equational definitions and performs some simplifications on the clause level.

The automatic mode can selects literal selection strategy, term ordering, and search heuristic based on simple problem characteristics of the preprocessed clausal problem.

EP 0.82 is just a combination of E 0.82 in verbose mode and a proof analysis tool extracting the used inference steps.

Implementation

E is implemented in ANSI C, using the GNU C compiler. The most outstanding feature is the global sharing of rewrite steps. Current versions of E add *rewrite links* from rewritten to new terms. In effect, E is caching rewrite operations as long as sufficient memory is available. Other important features are the use of *perfect discrimination trees* with age and size constraints for rewriting and unit-subsumption, *feature vector indexing* [33] for forward and backward subsumption and contextual literal cutting, and a new polynomial implementation of LPO [16].

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

`http://www.eprover.org`

EP 0.82 is a simple Bourne shell script calling E and the postprocessor in a pipeline.

Strategies

E's automatic mode is optimized for performance on TPTP 2.6.0. The optimization is based on about 90 test runs over the library (and previous experience) and consists of the selection of one of about 50 different strategies for each problem. All test runs have been performed on SUN Ultra 60/300 machines with a time limit of 300 seconds (or roughly equivalent configurations). All individual strategies are general purpose, the worst one solves about 49% of TPTP 2.6.0, the best one about 60%.

E distinguishes problem classes based on a number of features, all of which have between two and 4 possible values. The most important ones are:

- Is the most general non-negative clause unit, Horn, or Non-Horn?
- Is the most general negative clauce unit or non-unit?
- Are all negative clauses unit clauses?
- Are all literals equality literals, are some literas equality literals, or is the problem non-equational?
- Are there a few, some, or many clauses in the problem?
- Is the maximum arity of any function symbol 0, 1, 2, or greater?
- Is the sum of function symbol arities in the signature small, medium, or large?

Wherever there is a three-way split on a numerical feature value, the limits are selected automatically with the aim of splitting the set of problems into approximately equal sized parts based on this one feature.

For classes above a threshold size, we assign the absolute best heuristic to the class. For smaller, non-empty classes, we assign the globally best heuristic that solves the same number of problems on this class as the best heuristic on this class does. Empty classes are assigned the globally best heuristic. Typically, most selected heuristics are assigned to more than one class.

Expected Competition Performance

In the last year, E performed well in the MIX category of CASC and came in third in the UEQ division. We believe that E will again be among the strongest provers in the MIX category, in particular due to its good performance for Horn problems. In UEQ, E will probably be beaten only by Waldmeister. We cannot predict performance on FOF problems yet, but hope that E will be competitive.

EP 0.82 will be hampered by the fact that it has to analyse the inference step listing, an operation that typically is about as expensive as the proof search itself. Nevertheless, it should be competitive among the MIX* and FOF* systems.

7.6 E-SETHEO csp04

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

Architecture

E-SETHEO is a compositional theorem prover for formulae in first order clause logic, combining the systems E [31] and SETHEO [25]. It incorporates different calculi and proof procedures like superposition, model elimination and semantic trees (the DPLL procedure). Furthermore, the system contains transformation techniques which may split the formula into independent subparts or which may perform a ground instantiation. Finally, advanced methods for controlling and optimizing the combination of the subsystems are applied. The first-order variant of E-SETHEO no longer uses Flotter [61] as a preprocessing module for transforming non-clausal formulae to clausal form. Instead, a more primitive normal form transformation is employed.

Since version 99csp of E-SETHEO, the different strategies are run sequentially, one after the other. E-SETHEO csp04 incorporates the new version of the disconnection prover DCTP [37] with modified unification and a form of unit propagation as well as a new version of the E prover [31]. The new E prover also replaces TPTP2X as the CNF conversion tool. The new Scheme version of SETHEO that is in use features local unit failure caching [13] and lazy root paramodulation, an optimisation of lazy paramodulation which is complete in the Horn case [14]. Other than that (and a new resource distribution scheme), E-SETHEO csp04 is identical to E-SETHEO csp03.

Implementation

According to the diversity of the contained systems, the modules of E-SETHEO are implemented in different programming languages like C, Scheme, and Shell tools. Due to the replacement of TPTP2X, E-SETHEO is finally Prolog-free.

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

Strategies

Individual strategies are started be E-SETHEO depending on the allocation of resources to the different strategies, so-called *schedules*, which have been computed from experimental data using machine learning techniques as described in [39]. Schedule selection depends on syntactic characteristics of the input formula such as the Horn-ness of formulae, whether a problem contains equality literals or whether the formula is in the Bernays-Schönfinkel class. The problem classes are more or less identical with the subclasses of the competition organisers. We have no idea whether or not this conflicts with the organisers' tuning restrictions.

Expected Competition Performance

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

7.7 Gandalf c-2.6-SAT

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

Architecture

Gandalf [59, 60] is a family of automated theorem provers, including classical, type theory, intuitionistic and linear logic provers, plus finite a model builder. The version c-2.6 contains the classical logic prover for clause form input and the finite model builder. One distinguishing feature of Gandalf is that it contains a large number of different search strategies and is capable of automatically selecting suitable strategies and experimenting with these strategies.

The finite model building component of Gandalf uses the Zchaff propositional logic solver by L.Zhang [26] as an external program called by Gandalf. Zchaff is not free, although it can be used freely for research purposes. Gandalf is not optimised for Zchaff or linked together with it: Zchaff can be freely replaced by other satisfiability checkers.

Implementation

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

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

Gandalf is available under GPL from:

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

Strategies

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

In the normal mode Gandalf attempts to find only unsatisfiability. It has to be called with a `-sat` flag to find satisfiability. The following strategies are run:

- Finite model building by incremental search through function symbol interpretations.
- Ordered binary resolution (term depth): only for problems not containing equality.
- Finite model building using MACE-style flattening and the external propositional prover.

Expected Competition Performance

Gandalf c-2.6-SAT is the CASC-19 SAT division, assurance class, winner.

7.8 Mace2 2.2

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

Architecture

Mace2 [21] searches for finite models of first-order (including equality) statements. Mace2 iterates through domain sizes, starting with 2. For a given domain size, a propositional satisfiability problem is constucted from the ground instances of the statements, and a DPLL procedure is applied.

Mace2 is an entirely different program from Mace4 [22], in which the ground problem for a given domain size contains equality and is decided by rewriting.

Implementation

Mace2 is coded in ANSI C. It uses the same code as Otter [23] for parsing input, and (for the most part) accepts the same intput files as Otter. Mace2 is packaged and distributed with Otter, which is available at the following URL:
`http://www.mcs.anl.gov/AR/otter`

Strategies

Mace2 has been evolving slowly for about ten years. Two important strategies have been added recently. In 2001, a method to reduce the number of isomorphic models was added; this method is similar in spirit to the least number optimization used in rewrite-based methods, but it applies only to the first five constants. In 2003, a clause-parting method (based on the variable occurrences in literals of flattened clauses) was added to improve performace on inputs with large clauses. Although Mace2 has several experimental features, it uses one fixed stragegy for CASC.

Expected Competition Performance

Mace2 is not expected to win any prizes, because it uses one fixed strategy, and no tuning has been done with respect to the TPTP problem library. Also, Mace2 does not accept function symbols with arity greater than 3 or predicate symbols with arity greater than 4. Overall performace, however, should be respectable.

7.9 Mace4 2004-D

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

Architecture

Mace4 [22] searches for finite models of first-order (unsorted, with equality) statements. Given input clauses, it generates ground instances over a finite domain, then it uses a decision procedure based on rewriting try to determine satisfiability. If there is no model of that size, it increments the domain size and tries again. Input clauses are not "flattened" as they are in procedures that reduce the problem to propositional satisfiability without equality, for example, Mace2 [21].

Mace4 is an entirely different program from Mace2, in which the problem for a given domain size is reduced to a purely propositional SAT problem that is decided by DPLL.

Implementation

Mace4 is coded in ANSI C and is available at the following URL:
`http://www.mcs.anl.gov/AR/mace4`

Strategies

The two main parts of the Mace4 method are (1) selecting the next empty cell in the tables of functions being constructed and deciding which values need to be considered for that cell, and (2) propagating assignments. Mace4 uses the basic least number heuristic (LNH) to reduce isomorphism. The LNH was introduced in Falcon [62] and is also used in SEM. Effective use of the LNH requires careful cell selection. Propagation is by ground rewriting and inference rules to derive negated equalities.

Expected Competition Performance

Mace4 is not expected to win any prizes, because it uses one fixed strategy, and no tuning has been done with respect to the TPTP problem library. Overall performace, however, should be respectable. An early version of Mace4 competed in CASC-2002 under then name ICGNS.

7.10 Octopus 2004

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

Architecture

Octopus is a parallel ATP system. It is an improved version of the single-processor ATP system THEO [27]. Inference rules used by Octopus include binary resolution, binary factoring, instantiation, demodulation, and hash table resolutions. Octopus performs 5000-20000 inferences/second on each processor.

Implementation

Octopus is implemented in C and currently runs under Linux and FREEBSD. In the competition, it will run on a network of 150-200 PCs housed in various laboratories at McGill University's School of Computer Science. The processors communicate using PVM [8].

Strategies

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

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

When a proof is found to a weakened version, certain clauses in the proof are added to the base clauses of the given theorem. In addition, base clauses that participated in the proof of the weakened version are placed in the set-of-support. Octopus then tries to prove the given theorem with the augmented set of base clauses.

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

Expected Competition Performance

Octopus is only marginally better than it was last year, although it will run on about 4 times as many computers.

7.11 Otter 3.3

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

Architecture

Otter 3.3 [23] is an ATP system for statements in first-order (unsorted) logic with equality. Otter is based on resolution and paramodulation applied to clauses. An Otter search uses the "given clause algorithm", and typically involves a large database of clauses; subsumption and demodulation play an important role.

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. Otter is available from:
`http://www-unix.mcs.anl.gov/AR/otter`

Strategies

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

Expected Competition Performance

Otter has been entered into CASC-J2 as a stable benchmark against which progress can be judged (there have been only minor changes to Otter since 1996 [24], nothing that really affects its performace in CASC). This is not an ordinary entry, and we do not hope for Otter to do well in the competition.

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

7.12 Paradox 1.0

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

Architecture Paradox 1.0 [7] is a finite-domain model generator. It is based on a MACE-style [20] flattening and instantiating of the FO clauses into propositional clauses, and then the use of a SAT solver to solve the resulting problem.

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

Implementation

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

Strategies

There is only one strategy in Paradox:

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

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

Expected Competition Performance

Paradox 1.0 is the CASC-19 SAT division, model class, winner.

7.13 Paradox 1.1-casc

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

Architecture

Paradox 1.1-casc [7] is a finite-domain model generator. It is based on a MACE-style [20] flattening and instantiating of the first-order clauses into propositional clauses, and then the use of a SAT solver to solve the resulting problem.

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

The main differences with Paradox 1.0 are: a different SAT-solver, better memory behaviour, and a faster clause instantiation algorithm.

Implementation

The main part of Paradox is implemented in Haskell using the GHC compiler. Paradox also has a built-in incremental SAT solver which is written in C++. The two parts are linked together on the object level using Haskell's Foreign Function Interface.

Strategies

There is only one strategy in Paradox:

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

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

Expected Competition Performance

Paradox 1.1-casc should perform slightly better than Paradox 1.0.

7.14 SOS 1.0

John Slaney
Australian National University
`John.Slaney@anu.edu.au`

Architecture

SOS (Son Of SCOTT) Version 1.0 is yet another attempt to enhance Otter [23] by adding semantic guidance taken from models generated by a finite domain constraint solver. Its range of inference rules is the same as that of Otter: for CASC, that means it uses hyperresolution for most problems with non-unit clauses together with paramodulation and dynamic demodulation for equational reasoning. SOS does not impose any restriction on the inference rules of Otter. Like its predecessors [12] it maintains a model of some subset of the clauses deduced and updates this model from time to time during the search. Unlike its predecessors, it treats most of the clauses as *soft* constraints, thus maintaining a single approximate model of all of the clauses rather than many exact models of different subsets. Thus it represents a new solution to the problem of trading off the computational cost of computing models against the guidance resulting from them.

Because it contains Otter as a sub-program, all proofs produced by SOS are Otter proofs. The soundness of the system therefore follows trivially from the soundness of Otter.

For more information on the architecture and performance of SOS, see the forthcoming paper [36] in ECAI 2004. At the time when that paper was written, the program was provisionally called "SOFTIE".

Implementation

The system is written in C. Most of the code is that of Otter and of the constraint solver FINDER [35], both of which use very standard libraries. At present it has been compiled only under Solaris and Linux. The sources are available from:

`http://csl.anu.edu.au/ jks/software/sos.tar.gz`

Note that the program is very new and still at an experimental stage, so substantial changes are expected in future versions.

Strategies

All formulae in the set of support are tested against the guiding model and marked as true or false. Most given clauses are chosen to be the oldest of the shortest of the *false* clauses, though some proportion of choices default to Otter's criteria. The guiding model is repeatedly updated to make true as many instances (on its domain) of the usable clauses as possible. For CASC there is a time limit of 2 seconds on each model search, resulting in indeterminism because on different runs, different models may be the best found within the limit. SOS behaves differently from Otter not only because of the preference for false clauses but also because the weight reduction strategy is somewhat different: weight restriction begins more aggressively than in Otter, but subsequently each weight limit is phased in more gradually.

Expected Competition Performance

SOS should beat its parent Otter overall. It is expected to perform as well as the previous versions of SCOTT on UEQ problems, and tolerably well on Horn clause problems (HEQ and HNE). In other subsections of MIX it will fail for the same reasons as Otter. It is not expected to threaten more modern high-performance provers except possibly in UEQ. Its performance depends somewhat on the time limit, because its inference speed is lower than those of the other provers.

7.15 THEO J2004

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

Architecture

THEO [27] is a resolution-refutation theorem prover for first order clause logic. It uses binary resolution, binary factoring, instantiation, demodulation, and hash table resolutions.

Implementation

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

Strategies

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

In the last year or so, THEO has been modified so that when given a theorem, it first determines a list of potential ways to "weaken" the theorem. It then randomly selects one of the weakenings, tries to prove the weakened version of the theorem, and then uses the results from this effort to help prove the given theorem. A weakened version is created in a number of different ways, including modifying one clause by replacing a constant or function by a variable. Certain clauses from the proof of the weakened version are added to the base clauses when THEO next attempts to prove the given theorem. In addition, base clauses that participated in the proof of the weakened version are placed in the set-of-support. THEO then attempts to prove the given theorem with the revised set of base clauses.

Expected Competition Performance

THEO is only marginally better than it was last year.

7.16 Vampire 5.0

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

Architecture

Vampire [29, 30] 5.0 is an automatic theorem prover for first-order classical logic. Its kernel implements the calculi of ordered binary resolution and superposition for handling equality. The splitting rule is simulated by introducing new predicate symbols. A number of standard redundancy criteria and simplification techniques are used for pruning the search space: subsumption, tautology deletion, subsumption resolution and rewriting by ordered unit equalities. The only term ordering used in Vampire at the moment is a special non-recursive version of the Knuth-Bendix ordering that allows efficient approximation algorithms for solving ordering constraints. By the system installation deadline we may implement the standard Knuth-Bendix ordering. A number of efficient indexing techniques are used to implement all major operations on sets of terms and clauses. Although the kernel of the system works only with clausal normal forms, the preprocessor component accepts a problem in the full first-order logic syntax, clausifies it and performs a number of useful transformations before passing the result to the kernel.

Implementation

Vampire 5.0 is implemented in C++. The main supported compiler version is gcc 2.95.3, although in the nearest future we are going to move to gcc 3.x. The system has been

successfully compiled for Linux and Solaris. It is available from:
`http://www.cs.man.ac.uk/ riazanoa/Vampire`

Strategies

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

- Choice of the main saturation procedure : (i) OTTER loop, with or without the Limited Resource Strategy, (ii) DISCOUNT loop.
- A variety of optional simplifications.
- Parameterised simplification ordering.
- A number of built-in literal selection functions and different modes of comparing literals.
- Age-weight ratio that specifies how strongly lighter clauses are preferred for inference selection.

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

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

Expected Competition Performance

Vampire 5.0 is the CASC-19 FOF division winner.

7.17 Vampire 6.0

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

Architecture

Vampire [30] 6.0 is an automatic theorem prover for first-order classical logic. Its kernel implements the calculi of ordered binary resolution, superposition for handling equality and ordered chaining for transitive predicates. The splitting rule is simulated by introducing new predicate symbols. A number of standard redundancy criteria and simplification techniques are used for pruning the search space: subsumption, tautology deletion, subsumption resolution and rewriting by ordered unit equalities. The reduction orderings used are the standard Knuth-Bendix ordering and a special non-recursive version of the Knuth-Bendix ordering that allows efficient approximation algorithms for solving ordering constraints. A number of efficient indexing techniques are used to implement all major operations on sets of terms and clauses. Although the kernel of the system works only with clausal normal forms, the preprocessor component accepts a problem in the full first-order logic syntax, clausifies it and performs a number of useful transformations before passing the result to the kernel.

Implementation

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

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

Strategies

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

- Choice of the main saturation procedure : (i) OTTER loop, with or without the Limited Resource Strategy, (ii) DISCOUNT loop.
- A variety of optional simplifications.
- Parameterised reduction orderings.
- A number of built-in literal selection functions and different modes of comparing literals.
- Age-weight ratio that specifies how strongly lighter clauses are preferred for inference selection.

The standalone executable for Vampire 6.0 uses very simple time slicing to make sure that several kernel strategies are tried on a given problem.

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

Expected Competition Performance

Vampire 6.0 is the CASC-19 MIX division winner.

7.18 Vampire 7.0

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

Architecture

Vampire [30] 7.0 is an automatic theorem prover for first-order classical logic. Its kernel implements the calculi of ordered binary resolution and superposition for handling equality. The splitting rule and negative equality splitting are simulated by the introduction of new predicate definitions and dynamic folding of such definitions. A number of standard redundancy criteria and simplification techniques are used for pruning the search space: subsumption, tautology deletion (optionally modulo commutativity), subsumption resolution, rewriting by ordered unit equalities, basicness restrictions and irreducibility of substitution terms. The reduction orderings used are the standard Knuth-Bendix ordering and a special non-recursive version of the Knuth-Bendix ordering. A number of efficient indexing techniques is used to implement all major operations on sets of terms and clauses. Run-time algorithm specialisation is used to accelerate some costly operations, e.g., checks of ordering constraints. Although the kernel of the system works only with clausal normal forms, the preprocessor component accepts a problem in the full first-order logic syntax, clausifies it and performs a number of useful transformations before passing the result to the kernel. When a theorem is proven, the system produces a verifiable proof, which validates both the clausification phase and the refutation of the CNF. The current release features a built-in proof checker for the clausifying phase, which will be extended to check complete proofs.

Implementation

Vampire 7.0 is implemented in C++. The supported compilers are gcc 3.2.x, gcc 3.3.x, and Microsoft Visual C++. This version has been successfully compiled for Linux, but has not been fully tested on Solaris and Win32. It is available (conditions apply) from:

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

Strategies

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

- Choice of the main saturation procedure : (i) OTTER loop, with or without the Limited Resource Strategy, (ii) DISCOUNT loop.
- A variety of optional simplifications.
- Parameterised reduction orderings.
- A number of built-in literal selection functions and different modes of comparing literals.
- Age-weight ratio that specifies how strongly lighter clauses are preferred for inference selection.
- Set-of-support strategy.

The automatic mode of Vampire 7.0 is derived from extensive experimental data obtained on problems from TPTP v2.6.0. Input problems are classified taking into account simple syntactic properties, such as being Horn or non-Horn, presence of equality, etc. Additionally, we take into account the presence of some important kinds of axioms, such as set theory axioms, associativity and commutativity. Every class of problems is assigned a fixed schedule consisting of a number of kernel strategies called one by one with different time limits.

Expected Competition Performance

We expect the new version to perform much better than the last MIX winner Vampire 6.0. The kernel implementation has undergone a number of significant changes, and several new features has been added, such as basicness restrictions and memory defragmentation. Another source of improvement w.r.t. Vampire 6.0 is better selection of best strategies based on extensive experiments.

7.19 Waldmeister 702

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

Architecture

Waldmeister 702 is an implementation of unfailing Knuth-Bendix completion [2] with extensions towards ordered completion (see [1]) and basicness [3, 28]. The system saturates the input axiomatization, distinguishing active facts, which induce a rewrite relation, and passive facts, which are the one-step conclusions of the active ones up to redundancy. The saturation process is parameterized by a reduction ordering and a heuristic assessment of passive facts.

Only recently, we have designed a thorough refinement of the system architecture concerning the representation of passive facts [11]. The aim of that work - the next Waldmeister loop - is, besides gaining more structural clarity, to cut down memory consumption especially for long-lasting proof attempts, and hence less relevant in the CASC setting.

Implementation

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

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

Strategies

Our approach to control the proof search is to choose the search parameters according to the algebraic structure given in the problem specification [10]. This is based on the observation that proof tasks sharing major parts of their axiomatization often behave similar. Hence, for a number of domains, the influence of different reduction orderings and heuristic assessments has been analyzed experimentally; and in most cases it has been possible to distinguish a strategy uniformly superior on the whole domain. In essence, every such strategy consists of an instantiation of the first parameter to a Knuth-Bendix ordering or to a lexicographic path ordering, and an instantiation of the second parameter to one of the weighting functions *addweight*, *gtweight*, or *mixweight*, which, if called on an equation $s = t$, return $|s| + |t|$, $|max_{>}(s,t)|$, or $|max_{>}(s,t)| \cdot (|s| + |t| + 1) + |s| + |t|$, respectively, where $|s|$ denotes the number of symbols in s.

Expected Competition Performance

Waldmeister 702 is the CASC-19 UEQ division winner.

8 Conclusion

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

References

[1] J. Avenhaus, T. Hillenbrand, and B. Löchner. On Using Ground Joinable Equations in Equational Theorem Proving. In P. Baumgartner and H. Zhang, editors, *Proceedings of the 3rd International Workshop on First Order Theorem Proving*, pages 33–43, 2000.

[2] L. Bachmair, N. Dershowitz, and D.A. Plaisted. Completion Without Failure. In H. Ait-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, pages 1–30. Academic Press, 1989.

[3] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic Paramodulation and Superposition. In Kapur D., editor, *Proceedings of the 11th International Conference on Automated Deduction*, number 607 in Lecture Notes in Artificial Intelligence, pages 462–476. Springer-Verlag, 1992.

[4] P. Baumgartner, A. Fuchs, and C. Tinelli. Darwin - A Theorem Prover for the Model Evolution Calculus. In G. Sutcliffe, S. Schulz, and T. Tammet, editors, *Proceedings of the Workshop on Empirically Successful First Order Reasoning, 2nd International Joint Conference on Automated Reasoning*, Electronic Notes in Theoretical Computer Science, 2004.

[5] P. Baumgartner and C. Tinelli. The Model Evolution Calculus. In F. Baader, editor, *Proceedings of the 19th International Conference on Automated Deduction*, number 2741 in Lecture Notes in Artificial Intelligence, pages 350–364. Springer-Verlag, 2003.

[6] J-P. Billon. The Disconnection Method: A Confluent Integration of Unification in the Analytic Framework. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Proceedings of TABLEAUX'96: the 5th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, number 1071 in Lecture Notes in Artificial Intelligence, pages 110–126. Springer-Verlag, 1996.

[7] K. Claessen and N. Sorensson. New Techniques that Improve MACE-style Finite Model Finding. In P. Baumgartner and C. Fermueller, editors, *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*, 2003.

[8] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, , and Sunderam V. *PVM: Parallel Virtual Machine: A Users Guide and Tutorial for Network Parallel Computing*. MIT Press, 1994.

[9] M. Greiner and M. Schramm. A Probablistic Stopping Criterion for the Evaluation of Benchmarks. Technical Report I9638, Institut für Informatik, Technische Universität München, München, Germany, 1996.

[10] T. Hillenbrand, A. Jaeger, and B. Löchner. Waldmeister - Improvements in Performance and Ease of Use. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 232–236. Springer-Verlag, 1999.

[11] T. Hillenbrand and B. Löchner. The Next Waldmeister Loop. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, number 2392 in Lecture Notes in Artificial Intelligence, pages 486–500. Springer-Verlag, 2002.

[12] K. Hodgson and J.K. Slaney. TPTP, CASC and the Development of a Semantically Guided Theorem Prover. *AI Communications*, 15(2-3):135–146, 2002.

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

[14] R. Letz and G. Stenz. Integration of Equality Reasoning into the Disconnection Calculus. In C. Fermüller and U. Egly, editors, *Proceedings of TABLEAUX 2002:*

Automated Reasoning with Analytic Tableaux and Related Methods, number 2381 in Lecture Notes in Artificial Intelligence, pages 176–190. Springer-Verlag, 2002.

[15] R. Letz and G. Stenz. Generalised Handling of Variables in Disconnection Tableaux. In M. Rusinowitch and D. Basin, editors, *Proceedings of the 2nd International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, 2004.

[16] B. Loechner. What to know when implementing LPO. In G. Sutcliffe, S. Schulz, and T. Tammet, editors, *Proceedings of the Workshop on Empirically Successful First Order Reasoning, 2nd International Joint Conference on Automated Reasoning*, Electronic Notes in Theoretical Computer Science, 2004.

[17] Björk M. Extending Stlmarck's Method to First Order Logic. In Pirri F Cialdea Mayer M, editor, *TABLEAUX 2003 Position Papers and Tutorials*, number Technical Report RT-DIA-80-2003, pages 23–36, 2003.

[18] Björk M. *Stlmarck's Method for Automated Theorem Proving in First Order Logic.* PhD thesis, Department of Computing Science, Chalmers University of Technology, Gothenburg, Sweden, 2003.

[19] Björk M. Adding Equivalence Classes to Stlmarck's Method in First Order Logic. In *IJCAR 2004 Doctoral Programme*, 2004.

[20] W.W. McCune. A Davis-Putnam Program and its Application to Finite First-Order Model Search: Quasigroup Existence Problems. Technical Report ANL/MCS-TM-194, Argonne National Laboratory, Argonne, USA, 1994.

[21] W.W. McCune. MACE 2.0 Reference Manual and Guide. Technical Report ANL/MCS-TM-249, Argonne National Laboratory, Argonne, USA, 2001.

[22] W.W. McCune. Mace4 Reference Manual and Guide. Technical Report ANL/MCS-TM-264, Argonne National Laboratory, Argonne, USA, 2003.

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

[24] W.W. McCune and L. Wos. Otter: The CADE-13 Competition Incarnations. *Journal of Automated Reasoning*, 18(2):211–220, 1997.

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

[26] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In Blaauw D. and L. Lavagno, editors, *Proceedings of the 39th Design Automation Conference*, pages 530–535, 2001.

[27] M. Newborn. *Automated Theorem Proving: Theory and Practice.* Springer, 2001.

[28] R. Nieuwenhuis and J.M. Rivero. Basic Superposition is Complete. In Krieg-Brückner B., editor, *Proceedings of the 4th European Symposium on Programming*, number 582 in Lecture Notes in Computer Science, pages 371–390. Springer-Verlag, 1992.

[29] A. Riazanov and A. Voronkov. Vampire 1.1 (System Description). In R. Gore, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning*, number 2083 in Lecture Notes in Artificial Intelligence, pages 376–380. Springer-Verlag, 2001.

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

[31] S. Schulz. System Abstract: E 0.61. In R. Gore, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning*, number 2083 in Lecture Notes in Artificial Intelligence, pages 370–375. Springer-Verlag, 2001.

[32] S. Schulz. A Comparison of Different Techniques for Grounding Near-Propositional CNF Formulae. In S. Haller and G. Simmons, editors, *Proceedings of the 15th Florida Artificial Intelligence Research Symposium*, pages 72–76. AAAI Press, 2002.

[33] S. Schulz. Simple and Efficient Clause Subsumption with Feature Vector Indexing. In G. Sutcliffe, S. Schulz, and T. Tammet, editors, *Proceedings of the Workshop on Empirically Successful First Order Reasoning, 2nd International Joint Conference on Automated Reasoning*, Electronic Notes in Theoretical Computer Science, 2004.

[34] S. Schulz. System Abstract: E 0.81. In M. Rusinowitch and D. Basin, editors, *Proceedings of the 2nd International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, 2004.

[35] J.K. Slaney. FINDER: Finite Domain Enumerator, System Description. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in Lecture Notes in Artificial Intelligence, pages 798–801. Springer-Verlag, 1994.

[36] J.K. Slaney, A. Binas, and D. Price. Guiding a Theorem Prover with Soft Constraints. In *Proceedings of the European Conference on Artificial Intelligence*, 2004.

[37] G. Stenz. DCTP 1.2 - System Abstract. In C. Fermüller and U. Egly, editors, *Proceedings of TABLEAUX 2002: Automated Reasoning with Analytic Tableaux and Related Methods*, number 2381 in Lecture Notes in Artificial Intelligence, pages 335–340. Springer-Verlag, 2002.

[38] G. Stenz. *The Disconnection Calculus.* PhD thesis, Institut für Informatik, Technische Universität München, Munich, Germany, 2002.

[39] G. Stenz and A. Wolf. E-SETHEO: Design, Configuration and Use of a Parallel Automated Theorem Prover. In N. Foo, editor, *Proceedings of AI'99: The 12th Australian Joint Conference on Artificial Intelligence*, number 1747 in Lecture Notes in Artificial Intelligence, pages 231–243. Springer-Verlag, 1999.

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

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

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

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

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

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

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

[47] G. Sutcliffe and C. Suttner. The CADE-14 ATP System Competition. Technical Report 98/01, Department of Computer Science, James Cook University, Townsville, Australia, 1998.

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

[49] G. Sutcliffe and C. Suttner. The CADE-19 ATP System Competition. *AI Communications*, 2004.

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

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

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

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

[54] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

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

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

[57] C.B. Suttner and G. Sutcliffe. The Design of the CADE-13 ATP System Competition. *Journal of Automated Reasoning*, 18(2):139–162, 1997.

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

[59] T. Tammet. Gandalf. *Journal of Automated Reasoning*, 18(2):199–204, 1997.

[60] T. Tammet. Towards Efficient Subsumption. In C. Kirchner and H. Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction*, number 1421 in Lecture Notes in Artificial Intelligence, pages 427–440. Springer-Verlag, 1998.

[61] C. Weidenbach, B. Gaede, and G. Rock. SPASS and FLOTTER. In M. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th International Conference on Automated Deduction*, number 1104 in Lecture Notes in Artificial Intelligence, pages 141–145. Springer-Verlag, 1996.

[62] J. Zhang. Constructing Finite Algebras with FALCON. *Journal of Automated Reasoning*, 17(1):1–22, 1996.