The CADE-14 ATP System Competition

Dep't of Computer Science, James Cook University
Technical Report 98-01

Geoff Sutcliffe
Department of Computer Science
James Cook University, Australia
Email: geoff@cs.jcu.edu.au

Christian B. Suttner
Institut für Informatik
TU München, Germany
Email: suttner@informatik.tu-muenchen.de

Abstract

This paper documents the design, procedures, competing systems, results, and conclusions of the CADE-14 ATP System Competition (CASC-14).

1 Introduction

The CADE-14 ATP System Competition (CASC-14) was held on Wednesday 16th July 1997, as part of CADE-14 at the Sheraton Breakwater Hotel, Townsville, Australia. CASC-14 was the second CADE ATP system competition, following the successful competition at CADE-13 in 1996 at Rutgers University, New Brunswick, USA [SS97b]. CASC-14 evaluated the performance of sound, fully automatic, 1st order ATP systems. The evaluation was in terms of the number of problems solved and the average runtime for successful solutions. The evaluation was done in the context of a bounded number of eligible problems chosen from the TPTP Problem Library [SSar, SS97f] and a specified time limit for each solution attempt. Eighteen ATP systems, listed in Table 1, were entered into the various competition and demonstration divisions.

ATP System	Entrants
Allpaths	Joe Horton, Scott Buffet
DISCOUNT-TSM	Stephan Schulz
Fiesta	R. Nieuwenhuis, P. Nivela, J.M. Rivero, M.A. Vallejo
Gandalf	Tanel Tammet
I-THOP	Koji Iwanuma
MACE-1.2	Bill McCune
OCTOPUS	Monty Newborn
OSCAR	John Pollock
Otter-3.0.5	Bill McCune, Rusty Lusk, Ross Overbeek, Larry Wos
PROTEIN	Peter Baumgartner
P-SETHEO	Ortrun Ibens
RRTP	M. Paramasivam, David Plaisted
SCOTT	John Slaney, Kahlil Hodgson
SETHEO	Ortrun Ibens
SPASS V0.77	Christoph Weidenbach
TGTP	Monty Newborn
THINKER	Jeff Pelletier, Julian Fogel
Waldmeister	Arnim Buch, Thomas Hillenbrand

Table 1: The ATP Systems and Entrants

The competition was run on nine SUN Ultra 140s, supplied by SUN Education Australia. The competition was organized by Geoff Sutcliffe and Christian Suttner. The competition was overseen by a panel consisting of Alan Bundy, Claude Kirchner, and Jeff Pelletier.

This paper documents the design, procedures, competing systems, results, and conclusions of the competition. The design and procedures of CASC-14 evolved from those of CASC-13 [SS97e, SS97c]. Many minutiae that were also relevant to CASC-13 are not covered here, as they are fully discussed in the CASC-13 papers. The CASC-14 WWW site provides access to resources used before, during, and after the event:

`http://www.cs.jcu.edu.au/~tptp/CASC-14/`

2 Divisions

CASC-14 was divided into divisions according to problem and system characteristics. There were three competition divisions in which the systems were explicitly ranked, and two demonstration divisions in which systems demonstrated their abilities without being formally ranked.

2.1 Competition Divisions

The **MIX Division**: Mixed CNF Non-Propositional Theorems
Mixed means Horn and non-Horn problems, with or without equality, but not unit equality problems (see the UEQ division below). The MIX division was divided into four categories:

- Horn with No Equality (HNE)
- Horn with Equality (HEQ)
- Non-Horn with No Equality (NNE)
- Non-Horn with Equality (NEQ)

Eleven systems were entered into the MIX Division, but due to installation difficulties (see Section 4.2) only eight competed. They were Allpaths, Gandalf, I-THOP, Otter, SCOTT, SETHEO, SPASS, and TGTP.

The **UEQ Division**: Unit Equality CNF Non-Propositional Theorems
Six systems competed in the UEQ Division. They were DISCOUNT-TSM, Fiesta, Gandalf, Otter, SPASS, and Waldmeister.

The **SAT Division**: Mixed CNF Non-Propositional Non-theorems
Three systems were entered into the SAT Division, but one withdrew. The two competitors were MACE and SPASS.

Entry into the competition divisions was subject to the following rules:

- ATP systems could be entered only at the division level.
- ATP systems could be entered into more than one division. A system that was not entered into a particular division is assumed to be useless for that type of problem.
- The ATP systems had to run on a single locally provided standard UNIX workstation (the *general hardware* - see Section 3.1). ATP systems that could not run on the general hardware had to be entered into the Special Hardware Division (see Section 2.2).

2.2 Demonstration Divisions

The **FOF Division**: First Order Form Non-Propositional Theorems
The infrastructure for a FOF competition division was not ready for CASC-14. However, systems that can deal with FOF syntax could demonstrate their abilities in the FOF Division[1]. Ideally the

[1] The tptp2X utility contains a classifier that could have been prepended to any CNF system to form a FOF system. The tptp2X runtime would have been included in the total runtime.

systems ran on the general hardware, but use of any hardware supplied by the entrant or accessed via the Internet was acceptable. The results are presented along with the competition divisions' results, but no formal ranking was done. Three systems were entered into the FOF Division. They were Otter, SPASS, and THINKER.

The **Special Hardware Division**
ATP systems that could not run on the general hardware had to be entered into the Special Hardware Division. The hardware could be supplied by the entrant or accessed via the Internet. The entry specified which general hardware division's problems were to be used. The results are presented along with the competition division's results, but they cannot be compared with those from the systems that ran on the general hardware. Two systems were entered into the Special Hardware Division. They were OCTOPUS (MIX Division problems) and OSCAR (FOF Division problems).

3 Infrastructure

3.1 Hardware and Software

Nine SUN Ultra 140s were provided as the *general hardware* for CASC-14. Each machine had a 140 MHz CPU and 64MB memory. The operating system was Solaris 2.5.1. The machines were connected in a stand-alone network with no other users. The machines were physically located in Brisbane, about 1200km from the conference site. Access to the machines was possible only via a single modem line, using `kermit` for data transfer.

The organizers installed perl 5.003, PVM 3, Eclipse Prolog 3.5.2, and SICStus Prolog v3#5 on the general hardware, as required by some systems and for the competition scripts. System binaries were provided by the systems' entrants.

In the Special Hardware Division the hardware was supplied by the entrants. OCTOPUS ran on a network of 19 pentium computers running LINUX, with clock speeds ranging from about 100 MHz to over 200 MHz. These machines were access via Internet from the conference site. OSCAR ran on a Power Tower Pro 225 (a Mac clone running at 225 Mhz), with 128MB of RAM. The operating system was the Mac OS 7.6, and the prover ran in Macintosh Common LISP, v4.1. The entrant used the machine directly at his home site.

3.2 Problems

3.2.1 Problem Selection

The problems were selected from the TPTP Problem Library, v2.0.0. The problems had to meet certain criteria to be eligible for selection:

- The TPTP uses system performance data [SS97a] to classify problems as one of:
  - Easy: Solvable by all state-of-the-art ATP systems
  - Difficult: Solvable by some state-of-the-art ATP systems
  - Unsolved: Solvable by no state-of-the-art ATP systems
  - Open: Theorem-hood unknown

  Difficult problems were eligible for all the competition divisions, and in the SAT Division easy problems were also eligible due to the inadequate number of difficult problems available. Problems of any classification were eligible for the FOF Division.

- The TPTP distinguishes versions of problems as one of standard, incomplete, augmented, or special (see [SS97f] for details). Standard problems were eligible for all the divisions, and in the SAT and FOF Divisions special problems were also eligible due to the inadequate number of standard problems available.

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

3.2.2 Number of Problems

The method used to decide on the number of problems to be used in each division and category was similar to that used for CASC-13. The minimal number of problems that had to be used, to ensure sufficient confidence in the competition results, was determined from the number of eligible problems in each division and category (the competition organizers had to ensure that there was sufficient CPU time available to run the ATP systems on this minimal number of problems). The largest number of eligible problems was 182, in the UEQ Division, which required 21 problems to be used. This minimal number of problems was then used in determining the time limit of 300 seconds imposed on each solution attempt (see Section 3.3).

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

$$number\ of\ problems = \frac{number\ of\ workstations \times time\ allocated}{number\ of\ ATP\ systems \times time\ limit (300\ \text{seconds})}$$

It was a lower bound on the number of problems because it assumes that every system would use all 300 seconds for each problem. Since some solution attempts would obviously succeed before the 300 second limit was reached, more problems could actually be used.

At the time of calculation it was believed that there would be eight workstations available (the ninth became available only shortly before the competition). The competition was scheduled to run from 10:30am until 4:30pm, giving six hours for the competition. Thus there were 172800 seconds CPU time available for the competition. At the time of calculation 23 ATP systems were to be run on the general hardware over all the divisions. These parameters made it possible to select at least 25 problems in each division. The actual numbers used were determined according to the judgement of the competition organizers. In particular, more problems were used in the MIX Division in order to meet the confidence requirements in each of the MIX Division categories. Table 2 gives the number of eligible and number of selected problems for the divisions and categories.

	MIX				UEQ	SAT	FOF
	HNE	HEQ	NNE	NEQ			
Eligible	128	106	12	172	182	31	171
Selected	20	20	12	20	50	30	30

Table 2: Numbers of Eligible and Selected Problems

3.2.3 Problem Preparation

It was necessary to ensure that no system received 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, was 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 (this was introduced in CASC-14 due to feedback from entrants in the CASC-13 UEQ Division),
- remove equality axioms that are not needed by some of the ATP systems,
- output the problems in the formats required by the ATP systems. If the tptp2X utility did not support the format required, then it was the entrant's responsibility to provide a formating program.

Further, to prevent systems from recognizing problems from their file names, symbolic links were 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 were specified to the ATP systems using the symbolic link names.

In the Special Hardware Division the same problems were used as for the general hardware, with the same tptp2X transformations applied. However, the original file names were retained.

3.3 Time Limits and Timing

A time limit was imposed on each solution attempt. On the general hardware a CPU time limit was imposed, while in the Special Hardware Division the entrant could choose to use either a CPU or a wall clock time limit. Only one system, OCTOPUS, used a wall clock time limit.

The method used to decide on the time limit was similar to that used for CASC-13. A minimal time limit of 180 seconds was used, based on system performance data [SS97a] and experience gained in CASC-13. This minimal time limit was advertised in the CASC-14 Call for Participation, and no entrant expressed dissatisfaction. The maximal time limit was determined using the relationship given in Section 3.2.2, with 21 (the minimal number of problems) as the *number of problems*. This gave a maximal time limit of 357 seconds. A 300 second time limit was chosen as a reasonable value within the range allowed.

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

4 Entry Requirements and Procedures

Registration for CASC-14 closed on 15th June 1997, but three registrations were accepted after this deadline. For each system entered, a person had to be nominated to handle all issues (including execution difficulties) arising before and during the competition. It was not necessary for entrants to physically attend the competition.

Entering many similar versions of the same system was deprecated. Entrants could have been required to limit the number of system versions that they entered. Winners were expected to provide public access to their system's source code.

It was assumed that each entrant had read all the documents related to the competition (they were made available to entrants) and complied with the competition rules. Non-compliance with the rules could have lead to disqualification. A "catch-all" rule was used to deal with any unforseen circumstances: *No cheating is allowed.* The panel was allowed to disqualify entrants due to unfairness and to adjust the competition rules in case of misuse.

4.1 System Properties

The ATP systems had to be executable by a single command line, with the file name, the time limit (if required by the entrant), and entrant specified system switches (the same for all problems) as command line arguments.

The ATP systems that ran on the general hardware had to be interruptable by a SIGXCPU signal, so that the CPU time limit could be imposed on each solution attempt (see Section 3.3). The default action on receiving this signal is to exit (thus complying with the time limit, as required), but some of the systems caught the signal and exited of their own accord. Both approaches were acceptable for the competition. If any system ran past the time limit this was noticed in the timing data and the system was considered to have not solved that problem. In the Special Hardware Division the wall clock time limit used by OCTOPUS was imposed using SIGALRM rather than SIGXCPU.

When terminating, the ATP systems had to output a distinguished string (specified by the entrant) to stdout indicating the result:

- A proof exists, or
- The problem is satisfiable, or
- No solution found.

The ATP systems were not required to output solutions (proofs or models). However, systems that did output solutions are highlighted in the presentation of results (Section 7).

The precomputation and storage of any information for individual TPTP problems for usage during the competition was contrary to the spirit of the competition. The panel was allowed to disqualify unfairly doped systems. For every problem solved, the system's solution process had to be reproducible by running the system again.

The CASC-13 design differentiated between monolithic and compositional systems. Compositional systems were those that, due to some mechanism, selected one or more out of a set of available separate theorem proving techniques to be used on a given problem. Compositional systems could thus be be viewed as a combination of more rudimentary monolithic systems. This separation was eliminated for CASC-14 for two reasons. First, albeit for many systems it is clear if they are monolithic or compositional, the notion involves a gradual transition, making classification difficult for some systems. Secondly, CASC-14 had finer problem categories based on the occurrence of non-Horn clauses and equality literals. This allows the identification of compositional system components that rely on these differences, and thus disambiguates the evaluation. In CASC-14 there were two systems, Gandalf and Otter, that have a clearly compositional nature. The results for their component systems are discussed in Section 7.1.4.

4.2 System Installation

Because of the limited access to the general hardware, entrants could not install their systems directly. Rather, entrants installed their systems onto SUN Ultra machines provided by the Automated Reasoning Research Group at the TU München, Germany. Access to the TUM machines was made available from the 16th June 1997, and the entrants had have their ATP systems installed and operational by 4th July 1997. Five systems were installed only after this deadline. These late systems were accepted, but it was found that this left (too) little time for testing and taking care of installation difficulties. While the technical reasons vary, this ultimately led to three of the late systems (see Section 6.14) not being ready in time for the competition. These three systems were therefore not evaluated. In the early stages of the competition it was found that the version of Gandalf that had been retrieved from the TUM machines was the incorrect version, and only some swift action by the entrant and the organizers saved Gandalf from a poor performance.

Once an ATP system was installed the entrant informed the competition organizers, after which time no changes to the ATP system were permitted. The organizers then tested the ATP system, first to check that the system executed normally, secondly to check that the system could be interrupted by a SIGXCPU signal, and finally to check for soundness. The soundness testing had 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 the soundness testing suites of satisfiable problems were submitted to the systems participating in the MIX and UEQ Divisions, and a suite of unsatisfiable problems was submitted to the systems participating in the SAT Division. Finding a refutation for a satisfiable problem or a model for an unsatisfiable problem indicated that the system was either unsound, or not really attempting to find a solution. All systems passed this test. No soundness testing was done in the FOF Division.

In the Special Hardware Division the systems were installed on the respective hardware by the entrants, and no soundness testing was performed.

4.3 System Execution

Execution of the ATP systems on the general hardware was controlled by a `perl` script, provided by the competition organizers. For an index `N` running from 1 to 50 (the largest number of problems in any division), and within that for each division and category, a job was created for each ATP system entered into the division or category, for the Nth problem in the division or category. These jobs were then queued onto the workstations so that each workstation was running one job at a time. In this way all attempts at the Nth problems in all the divisions and categories were started before any attempts at the (N+1)th problems.

During the competition a `perl` script was regularly run to parse the systems' output files. If an ATP system's success string was found then the timing information from the `time` command was extracted. The CPU time taken, or the time limit if no solution was found, was then recorded in a summary file. The summary file was transferred to the CADE-14 conference site using `kermit`. Another `perl` script was then used to generate an HTML-file, and a WWW-browser was used to display the results. The HTML-file was regularly copied to a machine with Internet connection, in order to provide worldwide access to the results.

The execution of the Special Hardware Division systems was supervised by their entrants. The execution of OCTOPUS was controlled by a `perl` script provided by the competition organizers. For OSCAR, the entrant used his own execution control script. The systems' entrants emailed results to the organizers at regular intervals during the competition, and these were transferred by hand to a results file used for generating the HTML-file.

5 Performance Evaluation

In CASC-13, two specifically developed scoring schemes were used [SS97e]. As the analysis of results for CASC-13 shows [SS97d], the sophistication of these schemes is not required. They were replaced in CASC-14 by the following simpler scheme. The systems were ranked within each competition division according to the number of problems solved. If several systems solved the same number of problems, then those systems were ranked according to their average runtimes over solutions found.

If only one ATP system would have registered for a particular competition division, no winner could have been announced for that division, but the results for that system would still have been presented.

6 Systems

Descriptions of each of the ATP systems, discussing their architecture, implementation, and performance in CASC-14, commence on the next page.

6.1 Allpaths

J.D. Horton, D. Sharpe[1]
Faculty of Computer Science, University of New Brunswick
Email: jdh@unb.ca, dsharpe@vnet.ibm.com

ARCHITECTURE. Allpaths is a top-down procedure based on clause trees [1, 2]. A clause is chosen as the root node, typically the negation of the theorem to be proved. At each step an open leaf, selected from the most recent input clause, is resolved nondeterministically with complementary literals in the set of input clauses. If a *tautology path* is created, the resolution is rejected; if a *unifiable merge path* is created, then the merge path is non-deterministically *chosen* or not. See [1] for definitions and details. This procedure recognizes a larger set of merges and tautologies than Model Elimination or connection tableaux.

The iterative deepening bound used is the sum of the number of children clause nodes on the path from the leaf to the root. Disequalities are used to prevent the creation of tautology paths or previously searched merge paths.

IMPLEMENTATION. Prolog is used for a database generation program and a proof time program. The database generation program accepts TPTP input and generates a database of Prolog facts, one per clause literal, which the proof time program searches over. This method allows flexibility in search pruning tactics at the expense of efficiency, compared to Stickel's PTTP. Sixteen switches are available, including: ordered clauses, disequalities, the different kinds of merge and tautology paths, and choice of iterative deepening bound.

COMPETITION PERFORMANCE. There is no provision for equality in the code, so that the poor showing on the equality problems is not unexpected. Nor does Allpaths recognize whether a problem is Horn or not, a fact it could use if it did.

Thanks are due to Bruce Spencer and Scott Buffett who helped in the submission of the code to the competition.

References

[1] J. D. Horton and Bruce Spencer. Clause trees: a tool for understanding and implementing resolution in automated reasoning, *Artificial Intelligence* **92**(1997) 25–89.

[2] David Sharpe. *The Allpaths Automated Reasoning Procedure with Clause Trees*, MCS thesis, University of New Brunswick, Fredericton, NB, Canada, 1996.

[1] Now at IBM, Toronto, Canada.

6.2 DISCOUNT-TSM – Learning for Equational Deduction

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

ARCHITECTURE. DISCOUNT-TSM is a sequential version of the well-known distributed equational theorem prover DISCOUNT [1], using a new learning heuristic to control the selection of certain inference steps. DISCOUNT is based on *unfailing completion*, extended to deal with arbitrarily quantified goals. The basic algorithm repeatedly selects an unprocessed equation, rewrites it to normal form w.r.t. the already processed equations, and uses it to build critical pairs (new unprocessed equations). The equation is then used for interreduction and added to the set of processed equations. Interleaved with this completion process, *critical goals* are generated (by paramodulating facts into goals) and normalized. The process stops when either a goal has been proven or all facts and goals have been processed.

The selection of the equation to be processed is controlled by a *term space map*. Term space maps are a recently developed generalization of *term evaluation trees*, as described in [2]. A term space map is a recursive structure capturing characteristic distributions of term nodes in representative patterns of useful and useless equations collected from previous proofs.

IMPLEMENTATION. Both DISCOUNT and the supplementary programs for proof analysis and learning are implemented in ANSI C. The programs have been successfully installed under SunOS 4.3.x, Solaris 2.x, and Linux.

DISCOUNT was originally designed in 1991, before the use of efficient indexing methods became widespread. Consequently the basic inference engine is primitive by today's standards. Modern systems (e.g., Waldmeister [3]) achieve a much higher throughput.

COMPETITION PERFORMANCE. Taking into account the weaknesses of DISCOUNT's basic inference engine, we are quite pleased with the result. It shows that intelligent control using learning techniques can significantly improve the performance of a deduction system. We will now try to combine these techniques with a state-of-the-art inference engine.

References

[1] J. Denzinger, M. Kronenburg, and S. Schulz. DISCOUNT: A Distributed and Learning Equational Prover. *Journal of Automated Reasoning*, 18(2):189–198, 1997.

[2] J. Denzinger and S. Schulz. Learning Domain Knowledge to Improve Theorem Proving. *Proc. CADE-13*, LNAI 1104, 62–76. Springer, 1996.

[3] T. Hillenbrand, A. Buch, and R. Fettig. On Gaining Efficiency in Completion-Based Theorem Proving. *Proc. 7th RTA*, LNCS 1103, 432–435. Springer, 1996.

6.3 Fiesta

R. Nieuwenhuis, J.M. Rivero, M.A. Vallejo
Technical University of Catalonia
Email: roberto@lsi.upc.es

Architecture. The equational theorem prover Fiesta, in its CASC-14 version, implements unfailing completion. It uses a lexicographic path ordering where symbols are ordered first by arity and second by order of appearance in the input. The control is similar to the one of our Barcelona prover [1], i.e., fully reducing all equations before storing them, but here critical pairs are treated only if they are smaller than a given size bound. Each time the process terminates without a proof, it continues with a higher bound, thus preserving completeness.

Implementation. Fiesta is built on top of our *Dedam* toolkit of data structures and algorithms for automated deduction in first-order logic with equality [2]. *Dedam* provides all issues like input/output, memory management, a Warren Abstract Machine heap structure for term representation (as DAG's, where structure sharing is possible but not forced), as well as indexing based on substitution trees and a special data structure for sharing terms during unification and backward demodulation [1]. The actual Fiesta prover on top of this is only about a hundred lines of C-code.

Competition Performance. It is well-known that, for a prover like Fiesta, it is crucial for performance on given classes of problems to find an adequate "tuning" of parameters like the strategy for selecting the next formula to compute inferences with, the symbol orderings, and the size bounds. Fiesta was not tuned for the set of 182 eligible problems for this competition, but only for the set of 50 problems of the 1996 competition. Nevertheless, it performed well, solving 46 of the 50 problems, and with the lowest average time spent per problem solved. Surprisingly, at least two of its four timeouts correspond to problems which we know Fiesta can prove in about 20 seconds, since they were part of the 1996 competition. In CASC-14 Fiesta suffered from an inadequate term ordering for these problems, due to the random reordering of the input.

References

[1] Robert Nieuwenhuis, José Miguel Rivero, and Miguel Ángel Vallejo. The Barcelona Prover. *Journal of Automated Reasoning* 18(2), pp 171–176. Special issue CADE-13 ATP System Competition. April 1997.

[2] Robert Nieuwenhuis, José Miguel Rivero, and Miguel Ángel Vallejo. Dedam: An Kernel of Data Structures and Algorithms for Automated Deduction with Equality Clauses. Proc. CADE-14. pp 49–52. LNAI 1249. July 1997.

6.4 Gandalf

T. Tammet
Department of Computing Science, University of Göteborg
Email: tammet@cs.chalmers.se

Gandalf is a family of interdependent, code-sharing, resolution-based automated theorem provers, we are currently developing: a resolution prover for the first-order intuitionistic logic [1], for a fragment of Martin-Löf's type theory [2], and for the first-order classical logic, described here (an earlier version is described in [3]).

ARCHITECTURE. Gandalf is a resolution prover implementing a number of different basic strategies: binary ordered resolution for several orderings, versions of set-of-support resolution, binary unit resolution, and hyperresolution. Several incomplete combinations of the strategies as well as versions with limits on term depth and clause length are also used. Enhancements are used for cutting off literals and combining different strategies. Equality is handled by ordered paramodulation and demodulation.

The basic idea of the automatic mode in Gandalf used during the competition is time-slicing: Gandalf selects a set of different search strategies, allocates time to these and finally runs the strategies one after another. There is some, though little, cooperation between different strategies. The selection of strategies and the percentage of time they receive is in a somewhat ad hoc way based on the characteristics of the problem. For unit equality problems a fixed set of five different strategies was used.

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

COMPETITION PERFORMANCE. Our claim, supported by the results of the competition, is that it is important to develop a prover in a balanced manner, putting effort into strategy selection as well as developing special strategies and fast implementation.

References

[1] Tanel Tammet. A Resolution Theorem Prover for Intuitionistic Logic. In *CADE-13*, pages 2-16, *Lecture Notes in Computer Science vol. 1104, Springer Verlag,* 1996.

[2] Tanel Tammet, Jan Smith. Optimised Encodings of Fragments of Type Theory in First Order Logic. In *Types for Proofs and Programs*, pages 265-287, *Lecture Notes in Computer Science vol. 1158, Springer Verlag,* 1996.

[3] Tanel Tammet. Gandalf. *Journal of Automated Reasoning*, **18**(2): 199-204, 1997.

6.5 I-THOP Ver. 1.01

K. Iwanuma
Department of Electrical Engineering and Computer Science
Yamanashi University
Email: iwanuma@esi.yamanashi.ac.jp

ARCHITECTURE. I-THOP is based on the Model Elimination (ME) calculus [4], and fully utilizes PTTP technology [5]. Various kinds of lemma utilities [1, 3] and pruning methods are integrated within I-THOP. Especially, ones called *lemma matching rules* are quite practical and effective. These matching rules never instantiate the variables occurring in a current goal, alternatives to the goal can be cut off as soon as a matching rule is successfully applied. See [2] for the details on the effectiveness and limitations of lemma matching rules.

IMPLEMENTATION. The current I-THOP 1.01 is written in SICStus Prolog V3. Unit lemmas dynamically produced during ME-derivations are stored into Prolog's ordinary database. The latest version is available from:

`ftp://oborozukiyo.esi.yamanashi.ac.jp/pub/ITHOP/`

COMPETITION PERFORMANCE. The performance of I-THOP is not so bad for the category for non-Horn problems with no equality. This category seems to be appropriate for measuring the inference power for the most general framework, so we believe that the potential power of I-THOP is rather high. The performance for the Horn with no Equality category may be improved without difficulty by an integration of caching [1]. The most severe and difficult problem in I-THOP is the treatment of equality. The current version of I-THOP does not have any special mechanisms for dealing with equality. This is a quite important future research problem.

References

[1] O.L. Astrachan and M.E. Stickel, Caching and lemmaizing in model elimination theorem provers, *CADE-11, LNAI* **607** (1992) 778-782.

[2] K. Iwanuma, Lemma matching for a PTTP-based top-down theorem prover, *CADE-14, LNAI* **1249** (1997) 146-160.

[3] R. Letz, C. Goller and K. Mayr, Controlled integration of the cut rule into connection tableau calculi, *J. Automated Reasoning* **13** (1994) 297-338.

[4] D.W. Loveland, *Automated Theorem Proving: a logical basis* (North-Holland Publishing Company, Amsterdam, 1978).

[5] M.E. Stickel, A prolog technology theorem prover: a new exposition and implementation in prolog, *Theoret. Comput. Sci.* **104** (1992) 109-128.

6.6 OSCAR

J.L. Pollock
Department of Philosophy, University of Arizona
Email: pollock@arizona.edu

Architecture. OSCAR is an automated deductive and defeasible reasoner, which proves theorems in first-order logic as a side-effect of its more general reasoning capabilities. Viewed as a first-order theorem prover, it is a Fitch-style natural deduction system. It manipulates sequents, which are ordered pairs of suppositions (sets of formulas) and formulas. Search is bidirectional, proceeding backwards from the desired conclusion and forwards from the given premises. Forward search proceeds in terms of elimination rules, and backwards search proceeds in terms of introduction rules. Quantifiers are handled by Skolemization, which occurs during the course of the reasoning rather than by pre-processing the problem. Formulas generated by the backwards search are "reverse Skolemized". The theory behind this is available in the OSCAR Project Technical Report [1].

Implementation. OSCAR is implemented in Macintosh Common Lisp. Except for certain graphics functions, it should run in any Common Lisp. The details of the implementation are described in the OSCAR Manual [2].

Competition Performance. OSCAR solved 27 of the 30 FOF problems, thus coming equal second with Otter. OSCAR and Otter each solved one problem that the other failed on. OSCAR solved all the problems that THINKER, the other 'natural deduction' system in the FOF Division, solved.

References

[1] J.L. Pollock. Skolemization and Unification in Natural Deduction, http://www.u.arizona.edy/~pollock

[2] J.L. Pollock. OSCAR Manual, http://www.u.arizona.edy/~pollock

6.7 Otter and MACE

W.W. McCune
MCS Division, Argonne National Laboratory
Email: mccune@mcs.anl.gov

ARCHITECTURE. Otter [1] searches for proofs, and MACE [2] searches for small finite models. Both programs apply to statements in first-order (unsorted) logic with equality. The two programs are independent and work by completely different methods. (In practice, the programs are complementary.) Version 3.0.5-beta of Otter was used. Otter is based on resolution and paramodulation applied to clauses; an Otter search typically involves a large database of clauses, and subsumption and demodulation play an important role. Version 1.3.0 of MACE was used. Roughly speaking, MACE works as follows. For a given domain size, an equivalent propositional problem is constructed and a Davis-Putnam-Loveland decision procedure is applied.

IMPLEMENTATION. Both programs are written in C. Otter has changed very little since the CADE-13 competition (Otter 3.0.4), and we refer the reader to [3] for details. Otter's autonomous mode has been tuned since then in the following ways. For equational problems, there is more emphasis on breadth-first search. For mixed problems, (1) the TPTP clause type is now used to decide the initial `sos` list, (2) the inference rule UR-resolution is used in some cases as well as hyperresolution, and (3) paramodulation is less complete for non-unit equality problems.

COMPETITION PERFORMANCE. Otter's and MACE's performances in the competition were not surprising. Otter placed second (48 of 50 problems) in the UEQ Division, and its performance in the MIX Division (50 of 72 problems) was somewhat below the winner Gandalf (56 of 72 problems), which used multiple searches for each problem. MACE in the SAT Division (21 of 30 problems) could not match the performance of SPASS (a perfect score), which is actually a theorem prover that can detect satisfiability by exhausting the search for a refutation.

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.

References

[1] W. McCune. Otter. http://www.mcs.anl.gov/home/mccune/ar/otter/, 1996.

[2] W. McCune. MACE: Models and Counterexamples. http://www.mcs.anl.gov/home/mccune/ar/mace/, 1994.

[3] W. McCune and L. Wos. Otter: The CADE-13 Competition incarnations. *J. Automated Reasoning*, 18(2):211–220, 1997.

6.8 SCOTT: Semantically Constrained Otter

K. Hodgson, J. Slaney
The Australian National University
Email: {Kahlil.Hodgson,John.Slaney}@arp.anu.edu.au

ARCHITECTURE. SCOTT [1, 2, 3] is a resolution style theorem prover formed by augmenting a high performance ATP system (Otter) with a powerful model generator (FINDER). It was originally developed in collaboration with E.L. Lusk and W.W. McCune of Argonne National Laboratory. This produced an ATP system that not only knows *what* must follow, but also carries some insight into *why* this must be so.

In operation the theorem prover behaves just like Otter (see Section 6.7) except that information about the known satisfiability of fragments of the search space is used to focus on more promising inferences. Since our goal is to generate the null clause, the basic strategy is to avoid inferences for which this goal is impossible; that is, inferences based on satisfiable sets of clauses. The user may also provide semantic hints by adding clauses that affect the satisfiability of the clause space, but do not participate in the search.

IMPLEMENTATION. Otter and FINDER are both written in C, but use very different and specialized data structures. To implement SCOTT only a small amount of additional C code was needed to enable communication between the two systems.

COMPETITION PERFORMANCE. This system is still very experimental. Gathering semantic information about a search space introduces a large and sometimes prohibitive computational overhead. As the performance in the competition indicates, we still have some way to go in both minimizing this cost and maximizing the impact of the semantic information. However, we do have some anecdotal evidence which shows that the search may benefit dramatically from the additional information. Since the competition we have noted that our program was not modeling equality correctly. This was the main area where Otter was outperforming SCOTT.

References

[1] John Slaney and Kahlil Hodgson, Extending SCOTT to use multiple models, *the 15th International Joint Conference on Artificial Intelligence,* 1997.

[2] John Slaney, SCOTT: A Model-Guided Theorem Prover, *the 13th International Conference on Artificial Intelligence,* 1993, pp. 109–114.

[3] John Slaney, Ewing Lusk and William McCune, SCOTT: Semantically Constrained Otter, System Description, *the 12th International Conference on Automated Deduction,* 1994, pp. 764–768..

6.9 SETHEO – The CASC-14 System

O. Ibens
Institut für Informatik, Technische Universität München
Email: ibens@informatik.tu-muenchen.de

ARCHITECTURE. SETHEO is based on the calculus of *connection tableaux* with the additional *folding up rule* [4]. A lot of calculus refinements and search pruning methods have been integrated, which are presented in [3, 4, 5, 7]. The handling of equality is based on *lazy basic paramodulation* [2]. This method is implemented by means of a transformation from input formulae with equality axioms into formulae without equality [6].

IMPLEMENTATION. SETHEO is based on Prolog abstract machine technology. It is implemented as a shell script which invokes a preprocessor, a compiler, and the prover itself. These components are realized in C and Eclipse-Prolog. Proofs can be either visualized by means of SETHEO's tree viewer or translated into natural language by using the ILF system [1]. Binaries of SETHEO V4.0 for SunOs, Solaris, Hpux, and Linux and a draft version of the user's manual are available via ftp rsp. via email.

COMPETITION PERFORMANCE. SETHEO finished in 5th place (from 8 competitors). The CASC-14 version of SETHEO is a fusion (with improved parameter selections) of the CASC-13 versions of E-SETHEO and SETHEO [7], which finished at CASC-13 in the 1st and the 3rd places (from 8 competitors in the category 'General Hardware – Monolithic Systems – Mixed Problems'). The relatively weak position of SETHEO in CASC-14 has to be subject to further analysis.

References

[1] B. Dahn, J. Gehne, T. Honigmann, L. Walther, A. Wolf. Integr. of Automated and Interactive Theorem Proving in ILF. *CADE-14*, LNAI 1247, 57-60, 1997.

[2] J. Gallier, W. Snyder. Complete Sets of Transformations for General E-Unification. *Theoretical Computer Science*, 67:203–260, 1989.

[3] O. Ibens, R. Letz. Subgoal Alternation in Model Elimination. *TABLEAUX'97*, LNAI 1227, p. 201–215, 1997.

[4] R. Letz, K. Mayr, C. Goller. Controlled Integration of the Cut Rule into Connection Tableau Calculi. *JAR*, 13:297–337, 1994.

[5] R. Letz, J. Schumann, S. Bayerl, W. Bibel. SETHEO: A High-Performance Theorem Prover. *Journal of Automated Reasoning*, 8:183–212, 1992.

[6] M. Moser, J. Steinbach. Compiling Basic Paramodulation to Logic. Technical report, Institut für Informatik, TU München, to appear 1997.

[7] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, K. Mayr. SETHEO and E-SETHEO – The CADE-13 Systems. JAR, 18:237–246, 1997.

6.10 SPASS V0.77

C. Weidenbach, C. Meyer, C. Cohrs, T. Engel, E. Keen
Max-Planck-Institut für Informatik
Email: {weidenb,meyer,cohrs,engel,keen}@mpi-sb.mpg.de

ARCHITECTURE. SPASS is based on the superposition calculus extended by specific reasoning mechanisms for sorts and a splitting inference rule for case analysis [3, 2]. Furthermore, it includes strong reduction operations like contextual rewriting, replacement resolution, static soft typing, branch condensing, and the powerful clause normal form translator FLOTTER.

IMPLEMENTATION. SPASS is implemented in C and based on the C-library EARL, which provides efficient data structures and algorithms for crucial operations like matching, unification, indexing, subsumption, and the computation of orderings. The competition version V0.77 is not available from the SPASS homepage[2], but version V1.0.0, an even stronger one including dynamic soft typing [1], is available.

COMPETITION PERFORMANCE. SPASS came second in the MIX Division, last in the UEQ Division, and won both the SAT and FOF Divisions by solving all competition problems. SPASS was the only system that participated in all four problem categories and it was not tuned towards any of these categories. The latter is in particular the reason why SPASS came last in the UEQ Division. For the MIX, SAT and FOF Divisions, the strong theory implemented by SPASS pays off.

References

[1] Harald Ganzinger, Christoph Meyer, and Christoph Weidenbach. Soft typing for ordered resolution. In *Proceedings of the 14th International Conference on Automated Deduction, CADE-14*, volume 1249 of *LNAI*, pages 321–335, Townsville, Australia, 1997. Springer.

[2] Christoph Weidenbach. Spass version 0.49. *Journal of Automated Reasoning*, 18(2):247–252, 1997.

[3] Christoph Weidenbach, Bernd Gaede, and Georg Rock. Spass & flotter, version 0.42. In M.A. McRobbie and J.K. Slaney, editors, *13th International Conference on Automated Deduction, CADE-13*, volume 1104 of *LNAI*, pages 141–145. Springer, 1996.

[2] `http://www.mpi-sb.mpg.de/guide/software/spass.html`

6.11 TGTP: The Great Theorem Prover, and OCTOPUS

M. Newborn
School of Computer Science, McGill University
Email: newborn@cs.mcgill.ca

Architecture. TGTP [1] is a resolution-refutation theorem prover, using only binary resolution and binary factoring as inference rules. There is no special treatment for equality or Horn clauses. TGTP uses an iterative deepening depth first search for a linear proof of bounded length. Proofs longer than the bound can be found, as *extended search* is carried out on certain search lines. Furthermore, information on unit clauses and instances of unit clauses is stored in a hash table. This permits the program to find more complex proofs than linear proofs. TGTP sacrifices completeness by limiting the number of literals in an inference, the length of a literal, and the number of variables in a clause. These three parameters are determined for each theorem, before search begins.

TGTP's search is really a sequence of searches. The first search terminates when it finds that a proof exists, and each subsequent search finds pieces of the proof needed by the previous search. The entire proof is constructed before TGTP declares that a proof has been found, as a hashing error may cause the first search to incorrectly think it has found a proof.

OCTOPUS is a parallel version of TGTP. The master program sends a copy of the problem to the other computers, which then search independently for a proof using different limits for the number of literals in an inference and the length of a literal. Some of the processors also use the set-of-support strategy while others do not.

Implementation. TGTP and OCTOPUS are written in C, with OCTOPUS' communication through PVM. TGTP executes in about 22MB of memory. The majority of the space is used to store clauses in the search tree (4MB) and to store information on clauses in a large hash table (16MB). Approximately 400-2000 inferences per second were carried out by TGTP during the competition.

Competition Performance. TGTP performed its best on the NNE theorems (9/12 solved) and its worst on the HEQ theorems (5/20 solved). This is consistent with its design – as said previously, TGTP has no special treatment for equality or Horn clauses. OCTOPUS outperformed TGTP, solving seven seven theorems that TGTP failed to solve while failing to solve one theorem that TGTP solved.

References

[1] M. Newborn. The Great Theorem Prover, Version 3 Newborn Software, Montreal, 1997.

6.12 THINKER

F.J. Pelletier, J. Fogel
Department of Computing Science, University of Alberta
Email: {jeffp,julian}@cs.ualberta.ca

ARCHITECTURE. THINKER [3, 4, 5] is an automated reasoning system that produces proofs of arguments in first order predicate calculus with equality, but without arbitrary function symbols (the only functions treated are constants). The system uses the "natural form" (of the connectives $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\neg$, and the quantifiers $\forall$, $\exists$), and does not convert this to any normal form. The logic system admits arguments with premises as well as pure theorems without premises. The proofs produced are in the proof system described in [1, 2].

IMPLEMENTATION. The version used in the competition was written in the string-oriented language Spitbol – a dialect of SNOBOL4 (there is also a version written in C). Formulas are stored as strings (rather than as lists or trees), and this allows the use of dynamically-sized hash tables that can be indexed by formulas, subformulas, and formula-types.The basic command in Spitbol is "Does string X match pattern Y?", where patterns can be defined to an arbitrarily high degree of abstractness, including recursive patterns. THINKER has a pattern called `PROOF`. At the top level of the program a string, `THM`, is entered (representing a possible theorem) and the program asks: "Does `THM` match `PROOF`?". The proof is generated as a side effect of the attempt to evaluate this question.

COMPETITION PERFORMANCE. THINKER was written in the mid-80's, and the standard in automated theorem proving has risen dramatically since that time. Thus THINKER's mediocre performance in the competition should not be taken as a conclusive objection either to THINKER or to natural deduction theorem proving generally. Rather, it shows merely that natural deduction automated theorem proving has not received the intellectual study and support that other styles of automated proving have.

References

[1] D. Kalish and R. Montague. *Logic*, Harcourt, Brace, Janovich, 1964.

[2] D. Kalish, R. Montague, and G. Mar. *Logic* 2nd ed., Harcourt, Brace, Janovich, 1980.

[3] F.J. Pelletier. *Completely Non-Clausal, Completely Heuristic-Driven, Automatic Theorem Proving.* Technical Report TR82-7, Dept. Computing Science, Univ. Alberta., 1982.

[4] F.J. Pelletier. *Further Developments in Thinker, an Automated Theorem Prover.* Technical Report TR-ARP-16/87 Automated Reasoning Project, Australia National Univ., 1987.

[5] F.J. Pelletier. Automated Natural Deduction in Thinker. *Studia Logica*, forthcoming.

6.13 Waldmeister

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

ARCHITECTURE. Employing Knuth-Bendix completion as the proof procedure, WALDMEISTER is a theorem prover for unit equational logic. Following an engineering approach, we started its development from a three-level model of saturation-based ATP systems [1, 2]. The main completion loop is parameterized by a selection heuristic and a reduction ordering. A new way to generate this reduction ordering has been the key to WALDMEISTER's success: while we constructed it merely syntactically in the past (an LPO with a precedence based on the arity of the operators), we now derive the reduction ordering from the algebraic structure described by the axioms. Quite surprisingly, a general purpose selection heuristic that considers term weights and orientability of the critical pairs is sufficient for most problems.

IMPLEMENTATION. When designing the main completion loop, experiments showed that, for control reasons, the set of critical pairs should be represented explicitly. However, keeping its elements permanently normalized does not prune the search space. Hence we discarded structure sharing, and instead store the critical pair terms in a compressed data structure. This avoids – without sacrificing completeness – serious space problems and paid off in the competition. Fast execution of inference steps is achieved by sophisticated indexing techniques. The treatment of goals has been improved within the scope of more comprehensive work on storing multiple normal forms according to the current set of rules. A complete step-by-step proof object is constructed and stored in memory at run-time, with hardly any loss of performance. The WALDMEISTER ATP system is implemented in ANSI–C under UNIX.

COMPETITION PERFORMANCE. Of the 49 problems that WALDMEISTER solved, 47 were solved with an average proof time of 0.7 seconds. WALDMEISTER's performance on the remaining two problems shows that the inference rate stays high when the proof tasks grow: the runtime for these two problems was eight times that taken for all the other problems together, but WALDMEISTER also produced eight times as many facts for these two problems.

References

[1] A. Buch and Th. Hillenbrand: Waldmeister: Development of a High Performance Completion-based Theorem Prover. SEKI-Report 96-01, Universität Kaiserslautern, 1996. See *http://www.uni-kl.de/AG-AvenhausMadlener/waldmeister.html*

[2] Th. Hillenbrand, A. Buch, R. Vogt, and B. Löchner: Waldmeister. *Journal of Automated Reasoning* 18(2), pp. 265–270, 1997.

6.14 Unevaluated Systems

The three systems whose installation was not completed in time for the competition (see Section 4.2) are described here.

6.14.1 PROTEIN Version 2.0 P. Baumgartner, D. Schäfer

PROTEIN is a complete theorem prover for first order clause logic. In addition it also serves as a disjunctive logic programming system for computing answers and reasoning with default negation. PROTEIN features the Model Elimination and the Restart Model Elimination [1] calculi. It includes theory reasoning [3] in a very general way. Supporting programs include a clausal-normal form translator, a compiler to turn theory axioms into a theory reasoner [2], and a library of theory reasoners. PROTEIN is based on the PTTP implementation technique [4] and runs in ECLiPSe Prolog. For more information see `http://www.uni-koblenz.de/ag-ki/Systems/PROTEIN/`.

6.14.2 P-SETHEO A. Wolf, O. Ibens

P-SETHEO is a PVM based environment that controls the distributed compilation and execution of the SETHEO ATP system [5]. P-SETHEO parallelizes two stages: the preprocessing necessary for the preparation of the executable files, and the provers' runs. The first successful prover stops the system. An ASCII file contains information about the usable hosts, defines the compilers for preparing the binary files, and defines the prover strategies. Because of the generic form of the P-SETHEO controlling mechanism it is easy to integrate new strategies. The implementation uses `perl`, PVM, C, and shell tools, in approximately 1000 lines of code.

6.14.3 RRTP M. Paramasivam, D. Plaisted

RRTP [6, 7] is an instance-based refutational theorem prover. RRTP creates *replacement instances* of the input clauses using *replacement rules*. Replacement rules are implications obtained from the input clauses using *replacement strategies* to define which way literals are oriented. RRTP also incorporates unit resolution and a special strategy for Horn problems. In each round RRTP creates ground replacement instances that are tested for unsatisfiability using SATO [8]. SATO is an efficient implementation of the Davis and Putnam method written in C. RRTP exhibits superior performance in problems that do not involve equality. RRTP is written in Prolog.

References

[1] Peter Baumgartner, Ulrich Furbach, and Frieder Stolzenburg. Computing Answers with Model Elimination. *Artificial Intelligence*, 90(1–2):135–176, 1997.

[2] Peter Baumgartner. Linear and Unit-Resulting Refutations for Horn Theories. *Journal of Automated Reasoning*, 16(3):241–319, June 1996.

[3] M.E. Stickel. Automated Deduction by Theory Resolution. *Journal of Automated Reasoning*, 1:333–355, 1985.

[4] M. Stickel. A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.

[5] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, K. Mayr. SETHEO and E-SETHEO – The CADE-13 Systems. JAR, 18:237–246, 1997.

[6] M. Paramasivam *Instance-Based First-Order Methods Using Propositional Provers*. PhD Thesis, Department of Computer Science, University of North Carolina, Chapel Hill, 1997.

[7] M. Paramasivam and D. Plaisted. A Replacement Rule Theorem Prover *Journal of Automated Reasoning*, 18:221–226, 1997.

[8] Hantao Zhang and Mark E. Stickel. Implementing the Davis-Putnam algorithm by tries. Technical report, Department of Computer Science, University of Iowa, 1994.

7 Results

For each solution attempt made by an ATP system, three items of data were recorded: whether or not a solution was found, the time taken, and whether or not a solution (proof object or model) was output. This section presents the recorded data, and provides some analysis of the results.

It is important to be aware that the results obtained are modulo the competition design. There are many parameters that affect the results, and therefore the results do not necessarily determine the best overall ATP systems. Rather the results should be viewed as providing interesting insights into the ATP systems.

7.1 The MIX Division

Table 3 summarizes the results in the MIX Division. The table shows the number of proofs found and whether or not proofs were output. It also gives the number of proofs found in each of the four MIX Division categories.

The top group consists of Gandalf, SPASS, and Otter, all being clearly ahead of the remaining systems. Gandalf solved the most problems, five problems ahead of SPASS, which in turn solved one problem more than Otter. Otter and SCOTT were best on Horn problems with no equality, Otter was best on Horn problems with equality, Gandalf and SPASS were best on non-Horn problems with no equality, and Gandalf was best on non-Horn problems with equality. Five of the systems, including two of the top three, output proofs.

ATP System	MIX /72	HNE /20	HEQ /20	NNE /12	NEQ /20	Proofs output?
Gandalf	56	15	13	10	18	yes
SPASS	51	15	11	10	15	no
Otter	50	16	15	3	16	yes
SCOTT	43	16	13	3	11	yes
SETHEO	37	14	4	8	11	yes
TGTP	34	9	5	9	11	yes
I-THOP	30	11	4	8	7	yes
Allpaths	24	9	1	8	6	no
Special Hardware Division						
OCTOPUS	40	12	5	9	14	yes

Table 3: MIX Division Results

7.1.1 Number of Proofs as a Function of the Time Limit

Figure 1 shows the number of proofs that were achieved for time limits up to 300 seconds, by each of the systems. If the relative performances of the systems remain stable for a reasonable range of time limits, this provides empirical evidence that the ranking may be time limit independent.

The figure shows unstable ranking for low time limits. This is expected, and is caused by the different start up times of the systems, and possibly also by systems being particularly suited to some problems. These fluctuations are of lesser interest than the stabilized performances that are evident at time limits from about 100 seconds. No changes occur in the ranking after 200 seconds.

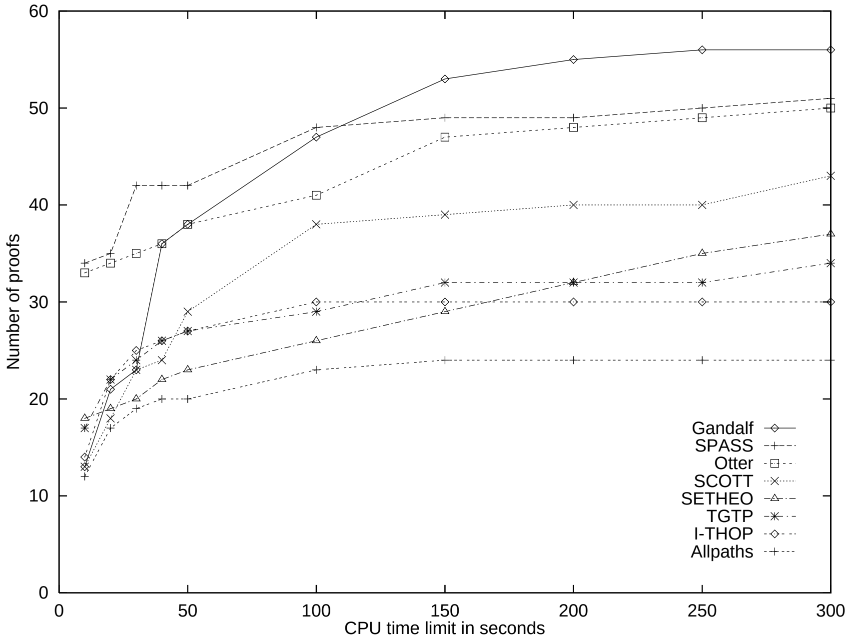

Figure 1: CPU time limit vs Number of proofs, for the MIX Division

7.1.2 Times Taken for Each Problem

Tables 4 and 5 show the CPU times taken by the systems for each problem, in seconds. The problems are listed using their TPTP names in alphabetical order (not the order that was used in the competition). For incomplete systems, ~~ddd~~ indicates termination with no solution after ddd seconds.

Three problems were not solved by any system. They are BOO015-1, COL003-8, and LCL114-1. Astute readers may recall that the eligible problems are "solvable by some state-of-the-art ATP systems". According to the performance data available at the time, each of these problems had been solved by only one system: BOO015-1 had been solved by Otter 3.0.4 in 204 seconds on a SUN SparcStation 10, COL003-8 had been solved by SETHEO 3.3 in 888 seconds on a HP 720, and LCL114-1 had been solved by Otter 3.0.4 in 539 seconds on a SUN SparcStation 10. Likely reasons why these problems could not be solved during the competition are the time limit used in the competition and the reordering of clauses and literals. Five problems were solved by only one system, namely GRP051-1, GRP057-1, GEO006-2, GEO013-1, and SET138-6. Ten problems were solved by all systems. Interestingly, problem LCL195-1 was solved by all systems except two of the top performing systems, namely Gandalf and SPASS.

It is interesting to consider naive parallelizations of some of the systems (i.e., parallelizations in which the component systems share the available time equally by multi-tasking, and communicate only when a solution is found). Running Otter in parallel with either Gandalf or SPASS would have led to the solution of 63 problems within the 300 second time limit. Even better is a combination of the top three systems: Gandalf, SPASS, and Otter. This combination solves 64 problems.

Figure 2 plots the CPU times taken by each system for each proof found, in increasing order of time taken. The highest Proof number coordinate of each plot gives the number of proofs found by that system.

Problem	General Hardware								Special Hardware
	Gandalf	SPASS	Otter	SCOTT	SETHEO	TGTP	I-THOP	Allpaths	OCTOPUS
HNE Category									
LCL016-1	-	-	122.8	119.2	-	257.6	-	-	-
LCL025-1	66.7	-	14.0	43.0	53.1	118.1	3.6	-	10.1
LCL033-1	30.5	0.0	0.0	2.0	2.0	10.8	2.2	2.6	8.6
LCL049-1	149.4	23.4	4.6	15.8	-	-	-	-	-
LCL050-1	155.7	27.5	7.8	23.1	-	-	-	-	-
LCL086-1	30.5	0.4	0.1	7.9	-	-	84.8	-	-
LCL087-1	30.6	0.1	0.1	2.9	261.1	-	47.9	90.2	20.9
LCL092-1	54.8	18.2	1.9	25.8	-	-	-	-	-
LCL102-1	31.6	1.6	46.8	24.2	2.4	-	28.3	138.6	33.1
LCL108-1	30.6	0.8	0.3	4.6	58.8	24.9	-	-	25.0
LCL114-1	-	-	-	-	-	-	-	-	-
LCL123-1	-	-	215.2	295.7	33.0	-	-	-	-
LCL182-1	80.2	283.3	3.0	13.5	2.2	140.0	4.0	8.1	16.9
LCL192-1	23.6	21.8	2.5	59.8	1.0	4.0	1.4	0.3	1.1
LCL195-1	-	-	111.3	81.2	2.4	270.1	5.9	28.6	20.2
LCL204-1	2.3	98.2	1.3	12.2	2.0	-	3.7	10.3	19.3
LCL215-1	80.0	57.6	7.3	20.1	1.9	2.5	3.5	2.6	0.9
LCL231-1	68.7	76.6	~~297.1~~	~~191.6~~	9.4	-	99.0	-	38.3
PLA010-1	-	4.3	-	-	184.8	-	-	-	-
PLA014-1	0.8	0.1	-	-	4.6	139.8	-	17.1	81.0
HEQ Category									
BOO010-1	26.5	78.5	0.9	2.0	4.9	35.4	-	-	23.8
BOO015-1	-	-	-	-	-	-	-	-	-
CAT004-1	-	-	0.8	15.7	-	-	-	-	-
CAT009-1	4.7	3.8	0.5	5.1	-	1.7	17.7	-	2.7
CAT010-1	4.6	3.6	0.5	4.5	-	1.7	20.9	-	2.8
COL003-3	-	-	2.8	-	251.5	-	-	-	-
COL003-8	-	-	-	-	-	-	-	-	-
GRP051-1	-	20.9	-	-	-	-	-	-	-
GRP057-1	38.0	-	~~26.8~~	-	-	-	-	-	-
GRP077-1	151.7	21.2	30.8	158.8	-	-	-	-	-
GRP085-1	43.8	-	2.4	15.8	-	-	-	-	-
GRP100-1	15.9	6.8	166.8	80.4	-	-	-	-	-
GRP102-1	86.3	4.9	23.4	73.6	-	-	-	-	-
GRP104-1	35.8	-	-	-	-	-	-	-	-
HEN003-3	30.3	0.1	0.1	6.5	3.3	1.2	2.8	-	0.9
HEN006-1	-	200.9	1.0	49.9	-	-	-	-	-
HEN009-3	32.0	4.7	0.8	76.5	-	-	-	-	-
HEN009-5	33.0	1.4	0.1	97.8	1.2	3.2	1.9	9.1	4.3
LCL145-1	147.2	-	9.7	-	-	-	-	-	-
RNG004-1	-	-	4.1	31.6	-	-	-	-	-

Table 4: CPU times in seconds for the MIX Division, Part I

Problem	Gandalf	SPASS	Otter	SCOTT	SETHEO	TGTP	I-THOP	Allpaths	OCTOPUS
	General Hardware								Special Hardware
NNE Category									
ANA002-2	121.4	-	-	-	36.4	-	-	-	-
SET005-1	30.3	0.1	281.9	6.7	1.8	1.3	3.9	78.1	2.1
SET007-1	-	0.7	-	-	57.9	3.6	25.5	-	112.0
SET011-1	6.6	0.0	129.1	250.2	1.3	1.4	4.5	5.6	15.7
SET012-1	30.4	2.1	-	-	4.3	3.8	14.0	0.5	9.5
SET012-2	46.0	20.2	-	-	-	11.8	-	9.6	58.2
SET013-1	-	81.0	-	-	-	64.1	15.3	3.0	89.9
SET013-2	50.9	-	-	-	-	-	-	-	-
SET014-2	15.5	0.2	-	-	1.1	1.0	4.2	76.2	1.5
SET015-1	30.5	64.1	-	-	1.1	85.1	4.7	1.1	88.4
SET015-2	60.5	148.2	-	-	-	-	-	-	-
SET055-6	0.1	0.5	0.4	8.7	5.2	1.8	13.8	3.0	1.4
NEQ Category									
CAT011-3	0.6	0.0	-	50.8	-	26.0	-	-	20.2
GEO006-2	-	1.1	-	-	-	-	-	-	-
GEO013-1	-	-	92.0	-	-	-	-	-	-
GEO026-2	210.8	-	0.3	40.6	189.5	13.5	7.7	-	4.7
GEO041-2	125.1	4.4	87.6	-	-	-	-	-	-
GEO043-2	124.3	3.3	44.4	-	-	-	-	-	-
GEO059-2	1.5	0.1	0.6	283.8	249.2	-	-	-	46.0
NUM009-1	0.3	0.7	1.2	7.8	42.6	3.5	69.7	35.4	3.0
NUM190-1	16.3	1.2	0.6	42.3	152.1	-	-	-	197.0
SET025-4	0.8	0.8	0.4	7.9	14.3	5.1	30.5	11.0	6.8
SET061-6	15.6	0.2	0.8	58.0	142.5	49.5	-	-	31.2
SET074-6	16.4	0.5	0.9	46.1	-	-	-	-	38.4
SET101-6	0.7	0.3	0.2	28.8	23.7	2.6	14.1	4.4	2.0
SET138-6	132.1	-	-	-	-	-	-	-	-
SET168-6	18.3	-	71.2	63.3	-	12.2	-	29.2	24.0
SET192-6	15.6	0.7	149.1	-	226.0	38.2	18.3	15.2	35.9
SET199-6	9.6	-	138.6	-	236.3	3.3	17.4	-	27.2
SET479-6	13.5	1.1	-	-	149.6	3.0	16.7	10.2	23.6
SET506-6	4.1	1.5	0.5	9.5	139.5	17.9	-	-	48.3
SET562-6	70.7	22.0	126.8	-	-	-	-	-	-

Table 5: CPU times in seconds for the MIX Division, Part II

The plots show that all the systems have a point at which the time taken to find proofs starts to increase dramatically. Gandalf has the least sudden such degradation of performance in the face of increasing problem difficulty. The systems with the most continuous performance, delivering proofs steadily right up to the time limit, are SCOTT and SETHEO. These two systems seem to be the most likely to profit from s larger time limit.

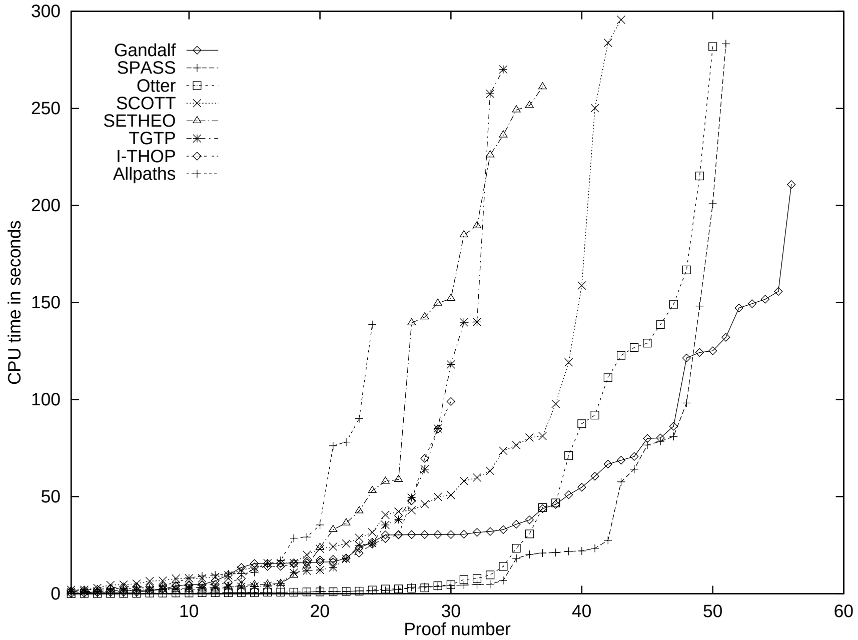

Figure 2: Proof number vs CPU time, for the MIX Division

Figure 3 plots the average CPU times taken for the first Proof number proofs found by each system, in increasing order of time taken for the proofs.

The plots show SPASS to have the lowest average times overall, which is a change since CASC-13 where Otter lead the field. Otter is fast initially as well, but slows down after the 36th proof and is eventually slower than Gandalf.

7.1.3 Subsumption Relationships between Systems

An interesting exposure of the comparative strengths and weaknesses of systems can be obtained by analyzing which systems solve supersets of the problems solved by other systems; this is referred to as *system subsumption*[3]. Figure 4 shows subsumption relationships in the MIX Division categories. An arrow from system A to system B indicates that system A subsumes system B, and ≡ indicates that the two systems solve the same problems. Systems that neither subsume other systems nor are subsumed by others are not shown.

The HNE category contains the least number of subsumption relations, indicating that for this type of problem the systems behave most distinctively. The NNE and HEQ categories involve all the systems in some subsumption relationship. Quite amazingly, in the NNE category Otter is subsumed by every other system. In the HEQ category Otter subsumes all systems except Gandalf and SPASS, while in the NEQ Gandalf subsumes all systems except Otter and SPASS. Thus Gandalf and SPASS are the only two systems that are unsubsumed in all categories.

[3]The authors prefer to call this the "ATP system food chain".

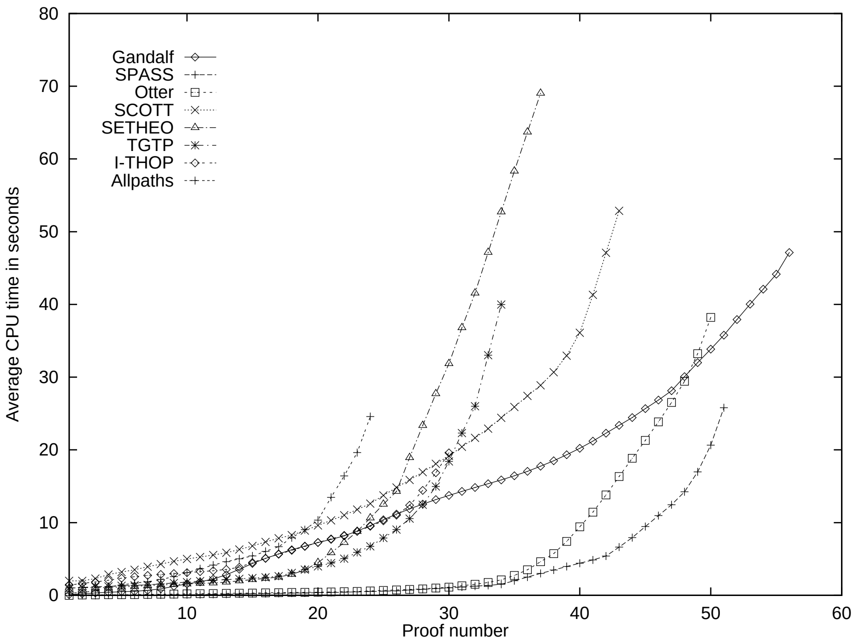

Figure 3: Proof number vs Average CPU time, for the MIX Division

7.1.4 The MIX Division Categories

Two systems in the MIX Division, Gandalf (Section 6.4) and Otter (Section 6.7), are clearly compositional, i.e., they select one or more out of a set of available separate theorem proving techniques to be used on a given problem. Gandalf selects several techniques to be used, and allocates some fraction of the available time to each selected technique. The selected techniques are then tried in succession until a proof is found or the allocated time is used up. In some cases unit clauses generated by one technique are retained for use by the next technique. Otter selects a single technique to be used for the entire proof attempt. The performances of these two systems have each been analysed, first to see if certain techniques are responsible for the successes in each category, and secondly to suggest if any of the categories should be further subdivided because of problem characteristics that suit different techniques.

The successes of Gandalf came from many different techniques. The greatest majority solved by a single technique is in the HEQ category, where `binary resolution` with the `unit strategy` and `dynamic demodulation` produced 9 of the 13 proofs. Seven of those 9 proofs were for pure equality problems, suggesting an analysis of performance on pure equality problems, which is done below. In the HNE category 9 of the 15 proofs came from using `binary resolution` with the `unit strategy`, and in the NEQ category 8 of the 18 proofs came from using `binary resolution` with the `unit strategy`, the `sos strategy`, and `dynamic demodulation`. In the NNE category the picture is less clear, but 9 of the 10 proofs were based on the use of `binary resolution` with the `sos strategy`, with other components also active. Thus Gandalf's success was largely due to the use of binary resolution with some combination of the `unit` and `sos` strategies, with `dynamic demodulation` active when equality was present.

Otter's selection of technique corresponded directly to the MIX Division categories. In the HNE category Otter used unordered hyperresolution. In the HEQ category Otter used unordered hyperresolution, the Knuth-Bendix method, and paramodulation from and into units. In the NNE category Otter used ordered hyperresolution, UR resolution, factoring, and unit deletion. In the NEQ category Otter used a composite of the techniques used in the HEQ and NNE categories.

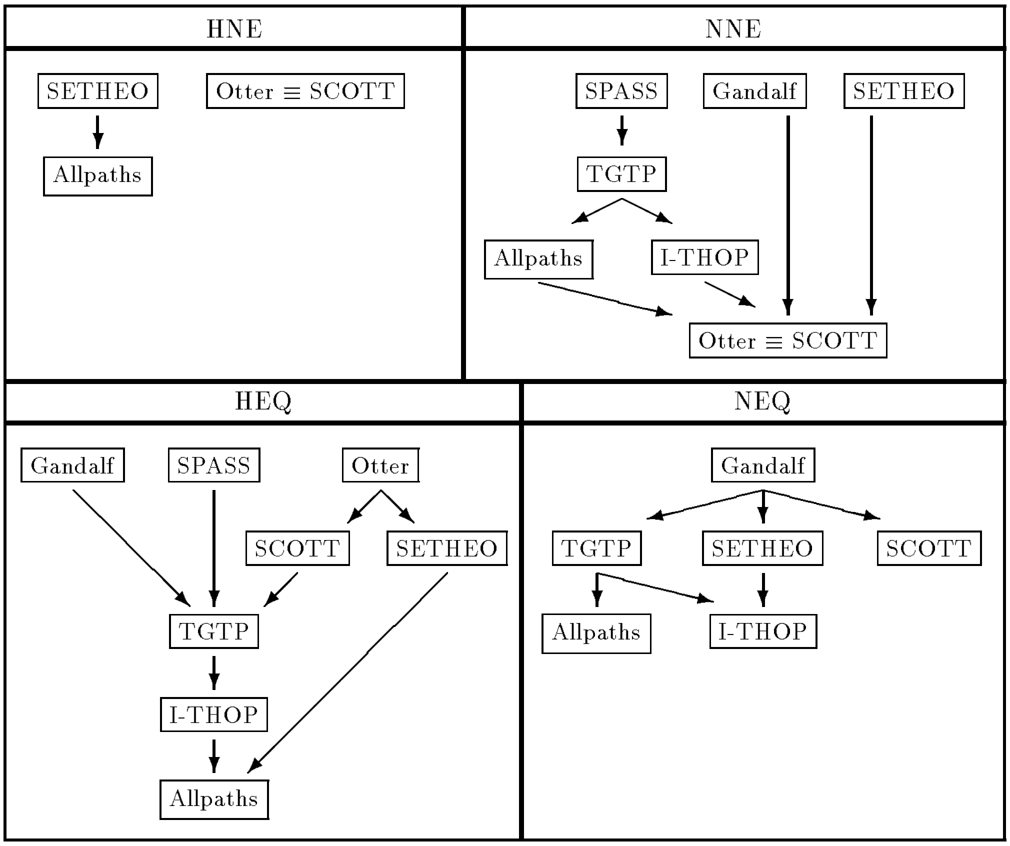

Figure 4: The ATP System Food Chains for the MIX Division

Thus Otter's success was based on the use of unordered hyperresolution in the HNE category, with other components being active as required for the more complex categories.

In the HEQ category 8 of the 20 selected problems are pure equality problems. Table 6 shows the performances of the systems on those 8 pure equality problems (PEQ), and on the other 12 problems that mix equality with non-equality (MEQ). While Otter has the best performance over the whole category, Gandalf is the strongest on the pure equality problems. Otter proves the most of the 12 mixed predicate problems. This difference in ranking suggests that the HEQ category should be split into these two subactegories in future competitions. There are no pure-equality problems in the NEQ category.

ATP System	HEQ /20	PEQ /8	MEQ /12
Gandalf	13	7	6
SPASS	11	4	7
Otter	15	5	10
SCOTT	13	5	8
SETHEO	4	1	3
TGTP	5	1	4
I-THOP	4	1	3
Allpaths	1	1	0
Special Hardware Division			
OCTOPUS	5	1	4

Table 6: HEQ Category Results

7.2 The UEQ Division

Table 7 summarizes the results in the UEQ Division. In contrast to CASC-13, where Otter clearly dominated, CASC-14's results show much less difference between the top performing systems. Waldmeister solved all but one problem, followed by Otter, Gandalf, Fiesta, and DISCOUNT-TSM, each of which solved one problem less than the previous system. As was the case in CASC-13, the top performing systems all return a proof object.

ATP System	UEQ /50	Model output?
Waldmeister	49	yes
Otter	48	yes
Gandalf	47	yes
Fiesta	46	yes
DISCOUNT-TSM	45	no
SPASS	37	no

Table 7: UEQ Division Results

7.2.1 Number of Proofs as a Function of the Time Limit

Figure 5 shows the number of proofs that were achieved for time limits up to 300 seconds, by each of the systems. The plot shows a stable system ranking for a large range of time limits, with no changes after 200 seconds. Gandalf is the only system which starts out slowly.

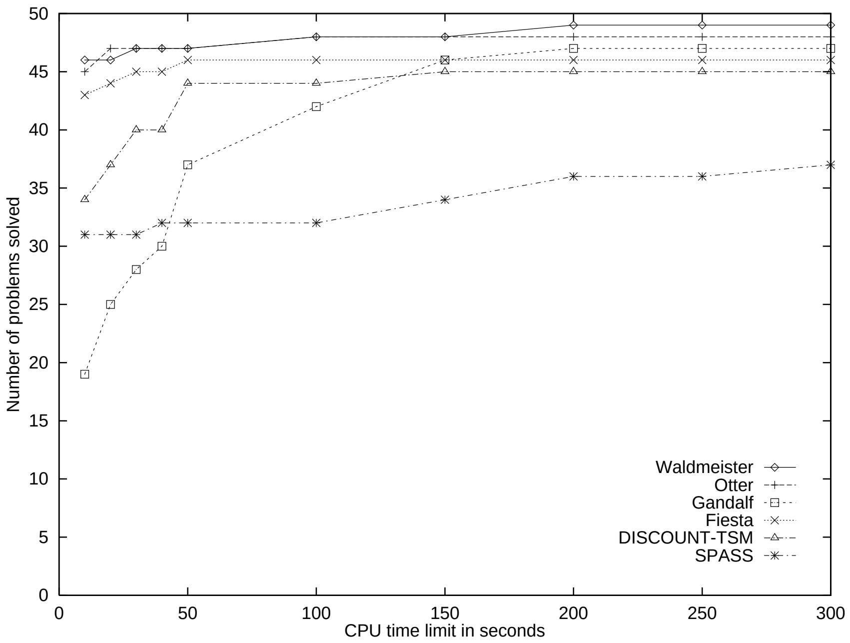

Figure 5: CPU time limit vs Number of proofs, for the UEQ Division

7.2.2 Times Taken for Each Problem

Table 8 shows the CPU times taken by the systems, for each problem, in seconds. Every problem was solved by some system, and most solutions were found very quickly. One problem was solved by only two systems, namely GRP167-3 (solved by Waldmeister and DISCOUNT-TSM). There were 34 problems solved by all the systems. These results show that the selected problems were in general too easy for the ATP systems. Comparing the results for the 16 problems which were used both in CASC-13 and CASC-14, it is found that the performance of the systems seems to have improved since CASC-13 (no definite comparison is possible due to the reordering of clauses and reversal of equalities). At the same time, 157 of the 162 problems eligible for CASC-13 remained eligible for CASC-14. Thus apparently high performance should be expected. Such undifferentiated high performance is not desirable for a competition, and ways to correct this are discussed in Section 8.

Figure 6 plots the CPU times taken by each system for each proof found, in increasing order of time taken. The difference between the top two systems, Waldmeister and Otter, is very small. Apparently only SPASS may benefit from a higher time limit. The other systems appear able to solve problems of this difficulty range within the 300 second limit.

Figure 7 plots the average CPU times taken for the first Proof number proofs found by each system, in increasing order of time taken for the proofs. Waldmeister has the lowest average time, followed by Otter and Fiesta.

7.3 The SAT Division

Table 9 summarizes the results in the SAT Division. SPASS solved all problems, nine problems ahead of MACE.

Table 10 shows the CPU times taken by the systems, for each problem, in seconds. For MACE, ~~ddd~~ indicates termination with no solution after ddd seconds, and + indicates an abnormal end of the run within one second (due to "too many first_order clauses"). All solutions are found very

Problem	Wald-meister	Otter	Gandalf	Fiesta	DISCOUNT-TSM	SPASS
BOO005-2	0.0	0.3	0.6	0.2	3.7	0.0
BOO009-2	0.0	0.3	0.5	0.2	2.7	0.0
BOO012-2	0.0	0.2	22.3	0.1	3.5	0.3
BOO013-2	0.0	0.2	2.2	0.2	2.0	0.2
BOO016-2	0.0	0.8	1.2	0.1	2.8	0.1
COL001-1	0.5	1.1	26.1	44.7	2.5	-
COL011-1	81.0	1.1	10.4	-	9.8	-
COL032-1	0.0	0.1	30.6	0.1	2.1	-
COL036-1	24.3	0.4	108.6	-	-	-
COL037-1	3.5	2.8	117.6	24.7	2.7	-
COL049-1	-	0.4	96.3	7.8	-	-
COL061-2	0.0	0.0	3.3	0.0	1.9	0.0
COL061-3	0.0	0.0	6.1	0.1	1.9	0.0
COL064-5	0.0	0.0	44.8	0.1	2.1	0.0
COL064-6	0.0	0.1	44.8	0.1	1.9	0.0
COL066-2	0.0	0.0	43.6	0.0	2.0	-
COL066-3	0.0	0.1	43.7	0.0	2.0	-
GRP115-1	0.0	0.3	7.8	0.1	1.9	0.1
GRP117-1	0.0	0.2	7.4	0.1	1.9	0.0
GRP121-1	3.6	-	-	1.4	4.8	284.2
GRP148-1	0.0	14.6	11.5	8.1	44.7	39.7
GRP159-1	0.0	0.1	0.7	0.1	9.0	0.0
GRP165-2	0.0	0.1	0.2	0.1	4.6	0.5
GRP166-2	0.1	0.2	81.7	0.6	4.8	0.3
GRP166-4	0.0	0.1	0.8	0.0	5.6	0.2
GRP167-3	180.2	-	-	-	113.9	-
GRP168-2	0.0	0.0	0.1	-	9.8	0.0
GRP169-1	0.8	0.4	127.4	0.4	15.3	6.2
GRP170-2	0.0	99.5	30.1	1.5	-	-
GRP171-2	0.0	0.9	6.4	0.2	5.3	1.1
GRP172-1	0.0	0.7	2.5	0.2	5.5	0.6
GRP174-1	0.0	0.3	30.0	0.1	4.9	0.6
GRP175-2	0.0	0.0	46.1	0.1	48.1	9.2
GRP185-1	0.2	6.0	43.6	0.1	6.9	-
GRP186-3	0.0	0.2	121.8	0.1	22.1	0.1
GRP191-1	0.2	2.6	182.2	0.6	18.3	141.7
GRP191-2	0.3	2.2	69.6	15.7	18.4	117.8
LCL116-2	0.1	1.4	11.0	1.0	2.2	1.5
LCL134-1	0.0	0.1	0.2	0.1	2.0	0.1
LCL140-1	0.0	0.6	2.3	0.2	2.1	0.4
LCL153-1	0.2	0.2	11.7	0.4	45.2	0.4
LCL157-1	0.1	0.8	8.7	0.2	2.3	0.4
RNG007-4	0.0	0.1	0.0	0.0	2.0	0.0
RNG008-7	0.3	14.5	98.8	0.3	4.1	2.8
RNG015-6	0.0	0.7	53.4	0.3	46.7	171.3
RNG017-6	0.0	0.1	14.1	1.2	28.0	-
RNG018-6	0.0	0.3	15.8	0.5	24.7	174.3
RNG020-6	0.1	0.2	-	2.3	-	-
ROB002-1	0.0	0.0	0.5	0.1	2.2	0.2
ROB023-1	0.8	0.3	44.8	6.8	-	1.0

Table 8: CPU times in seconds for the UEQ Division

ATP System	SAT /30	Model output?
SPASS	30	no
MACE	21	yes

Table 9: SAT Division Results

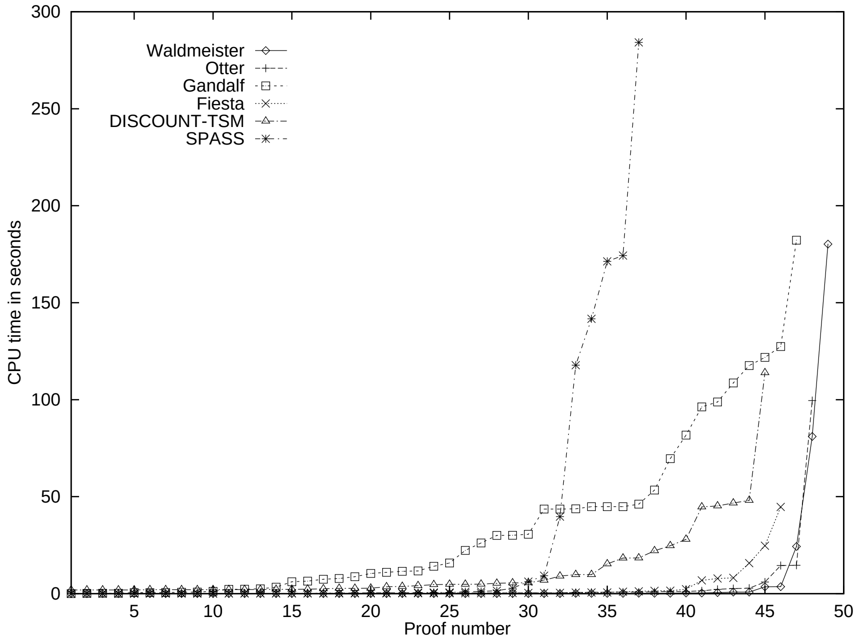

Figure 6: Proof number vs CPU time, for the UEQ Division

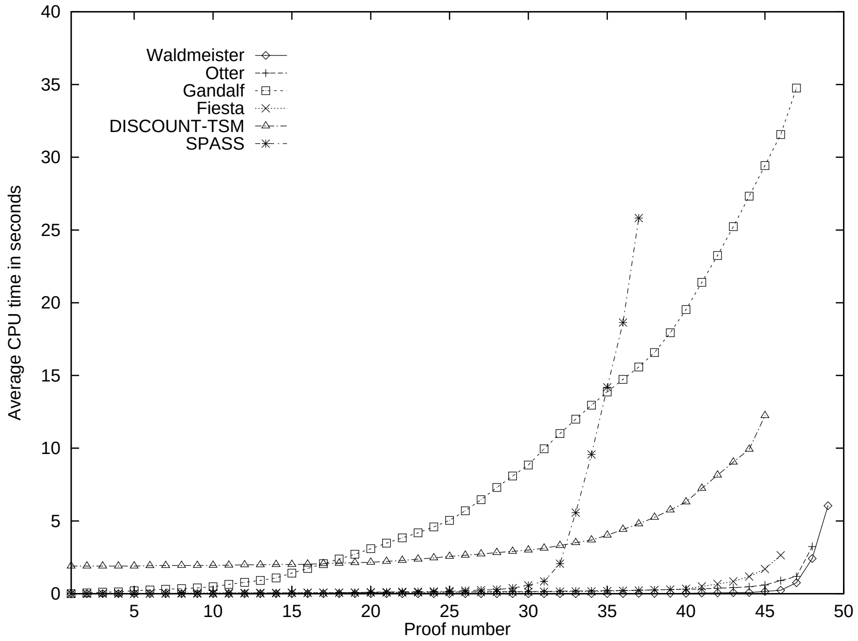

Figure 7: Proof number vs Average CPU time, for the UEQ Division

quickly, particularly by SPASS. This shows that the problem set was comparatively easy for the systems entered into this division of the competition, a fact that had been recognized during the problem preparation for the competition. However, the organizers felt that cancellation of the division or a last minute change of the problem set would have been unfair to the entrants.

Problem	MACE	SPASS
CAT015-3	~~18.1~~	0.1
CAT015-4	~~28.6~~	0.1
MSC009-1	0.3	0.1
PUZ001-3	0.3	0.1
PUZ028-1	0.5	0.1
PUZ028-2	0.6	0.3
PUZ028-3	+	0.2
PUZ028-4	+	0.1
SYN036-2	+	0.4
SYN059-1	0.3	0.1
SYN078-1	0.2	0.1
SYN084-1	0.2	0.1
SYN303-1	-	0.1
SYN304-1	0.1	0.1
SYN305-1	-	0.1
SYN306-1	0.2	0.1
SYN308-1	0.3	0.1
SYN309-1	0.2	0.1
SYN316-1	0.1	0.1
SYN317-1	0.1	0.1
SYN320-1	0.1	0.1
SYN322-1	0.2	0.1
SYN324-1	0.1	0.1
SYN329-1	0.4	0.1
SYN330-1	-	0.1
SYN335-1	-	0.1
SYN337-1	0.2	0.1
SYN342-1	0.2	0.1
SYN344-1	0.2	0.1
SYN348-1	0.2	0.1

Table 10: CPU times in seconds for the SAT Division

7.4 The FOF Division

Table 11 summarizes the results in the FOF Division. SPASS solved all the problems.

Table 12 shows the CPU times taken by the systems, for each problem, in seconds. For Otter, ddd —— indicates termination with no solution after ddd seconds. Almost all solutions were found very quickly, particularly by SPASS and Otter. Both these systems use a clause normal form transformation and work on the normal form instead of the first order form.

Four of the FOF problems contain equality, namely MGT013+1, MGT020+1, MGT023+1, and SYN080+1. SPASS and Otter solved all of these, while THINKER solved none. This suggests that the FOF Division should be split into two categories in future competitions, one category for problems with no equality and on category for problems with equality.

ATP System	FOF /30	Proof output?
SPASS	30	no
Otter	27	yes
THINKER	16	yes
Special Hardware Division		
OSCAR	27	yes

Table 11: FOF Division Results

Problem	SPASS	Otter	THINKER	OSCAR
MGT009+1	1.6	0.2	-	1.0
MGT013+1	0.4	0.1	-	1.3
MGT020+1	0.3	~~0.1~~	-	-
MGT022+2	0.0	0.1	-	0.1
MGT023+1	0.2	~~0.1~~	-	0.3
MGT036+3	0.0	0.0	-	0.1
SYN036+2	0.1	-	-	-
SYN056+1	0.0	0.0	1.3	0.1
SYN059+1	0.0	0.0	1.2	0.1
SYN060+1	0.0	0.0	0.4	0.0
SYN061+1	0.0	0.0	0.4	0.0
SYN080+1	0.0	0.0	-	0.0
SYN321+1	0.0	0.0	-	0.0
SYN328+1	0.0	0.1	-	-
SYN331+1	0.0	0.0	-	10.0
SYN338+1	0.0	0.0	0.8	0.0
SYN349+1	0.0	0.1	-	0.3
SYN352+1	0.0	0.1	-	34.9
SYN355+1	0.0	0.0	0.4	0.0
SYN364+1	0.0	0.0	-	0.4
SYN367+1	0.0	0.1	0.5	0.0
SYN369+1	0.0	0.1	0.6	0.0
SYN381+1	0.0	0.0	0.5	0.1
SYN384+1	0.0	0.0	1.6	0.0
SYN397+1	0.0	0.0	0.4	0.0
SYN398+1	0.0	0.0	0.5	0.0
SYN403+1	0.0	0.1	0.4	0.0
SYN407+1	0.0	0.0	0.5	0.0
SYN409+1	0.0	0.0	0.5	0.0
SYN410+1	0.0	0.0	0.4	0.0

Table 12: CPU times in seconds for the FOF Division

8 Conclusions

The CADE-14 ATP System Competition was the second large scale competition for 1st order ATP systems. It extended the scope of the previous competition by incorporating the SAT and FOF Divisions. With respect to the competition design, finer grained competition categories determined by problem characteristics made it possible to identify special capabilities of systems even if they did not win overall. The experiences from CASC-13 allowed the realization of more automation during the event itself.

The organizers believe that the competition fulfilled 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 has been substantial publicity both within and outside the ATP community. The significant efforts that have gone into developing the ATP systems have received public recognition; publications, which adequately present theoretical work, have not been able to expose such practical efforts appropriately. The competition has provided an overview of which researchers and research groups have decent, running, fully automatic ATP systems.

8.1 Criticisms

General criticisms of ATP system competitions, and the organizers' comments, are given in [PSS97].

At CASC-14, a specific criticism was of the seemingly easy problems in the UEQ Division. This phenomenon came about due to improvement in the ATP systems since CASC-13, while the set of eligible problems remained almost the same. In order for the eligible problems to be of approriate difficulty, it is necessary for results from ATP systems, over those problems, to be submitted to the TPTP results collection. No results for systems specialized for UEQ problems had been submitted since CASC-13, and thus many problems had unsatisfactory ratings relative to the systems entered. Further, very few new UEQ problems had been submitted and added to the TPTP since CASC-13. This seems to indicate, and this view has been supported by some members of the community, that unit equality is rarely used to encode contemporary problems. As the TPTP is maintained in the spirit of a bottom-up development, reflecting the use of ATP systems in the community, the cooperation of system developers is required to ensure a representative data collection.

A suggestion was made to include problems rated as unsolved, in order to put out a challenge to the systems. A major purpose of the competition is to differentiate between the competitors. The ideal problem therefore can be solved by at least one system, but not all systems. Problems not solved by any system do not contribute to the evaluation, and therefore decrease the quality of the comparison. The strong limitations under which systems have to operate in the competition (time limit and uniform system switches for all problems) make it very unlikely that such previously unsolved problems would be solved. Therefore the use of unsolved problems in the competition is not recommendable.

On the organizational side, the preparations for CASC-14 were hindered by limited access to the machines and late system submissions.

8.2 Future ATP System Competitions

Several experiences from this competition will be taken into account in future competitions.

At least the HEQ category and the FOF Division will be divided into pure equality and non-pure equality categories. This will further allow specialized systems to show their particular abilities.

Late submissions in CASC-14 increased the organizational effort required, and turned out to be detrimental to potential participants. A strict deadline policy will be followed in future, even if some time flexibility seems to be available. This also ensures completely equal treatment of all entrants.

In CASC-14 the organizers took on the task of ensuring that the submitted system ran correctly in the competition environment. In future the competition control scripts will be made available to the entrants, who can then ensure in advance that their systems behave as required.

A few solution attempts in CASC-14 took excessive wall clock time, even though the CPU time limit was not exceeded. This was apparently caused by very high memory usage that caused swapping. In the future this will be prevented by imposing a wall clock time limit in addition to the CPU time limit. Similarly, excessive output will be prevented by limiting the output file size.

Some of the systems output their proof objects to separate files, thus making it hard to automatically store the proofs for later examination. In future competitions the proofs objects will have to be written to stdout, so that they can automatically be stored as part of the system's output.

ACKNOWLEDGEMENTS. Thanks to Thilo Trapp for his help in preparing the competition and evaluating the results.

References

[PSS97] F.J. Pelletier, G. Sutcliffe, and C.B. Suttner. Conclusions about the CADE-13 ATP System Competition. *Journal of Automated Reasoning*, 18(2):287–296, 1997.

[SS97a] G. Sutcliffe and C.B. Suttner. ATP System Results for the TPTP Problem Library. http://www.cs.jcu.edu.au/~tptp/TPTP/Results.html, 1997.

[SS97b] G. Sutcliffe and C.B. Suttner. Special Issue: The CADE-13 ATP System Competition. *Journal of Automated Reasoning*, 18(2), 1997.

[SS97c] G. Sutcliffe and C.B. Suttner. The Procedures of the CADE-13 ATP System Competition. *Journal of Automated Reasoning*, 18(2):163–169, 1997.

[SS97d] G. Sutcliffe and C.B. Suttner. The Results of the CADE-13 ATP System Competition. *Journal of Automated Reasoning*, 18(2):271–286, 1997.

[SS97e] C.B. Suttner and G. Sutcliffe. The Design of the CADE-13 ATP System Competition. *Journal of Automated Reasoning*, 18(2):139–162, 1997.

[SS97f] C.B. Suttner and G. Sutcliffe. The TPTP Problem Library (TPTP v2.0.0), 1997. Technical Report AR-97-01, Institut für Informatik, Technische Universität München, Munich, Germany; Technical Report 97/04, Department of Computer Science, James Cook University, Townsville, Australia.

[SSar] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, To appear.