CADE-16

The 15th International Conference on Automated Deduction

July 5-10, 1998, Lindau, Germany

The CADE-16 ATP System Competition

July 7

Geoff Sutcliffe and Christian Suttner

CASC-16 Web Site — `http://www.cs.jcu.edu.au/ tptp/CASC-16/`

CADE-16 Web Site — `http://www.th-darmstadt.de/cade-16`

Abstract

In order to stimulate ATP system development, and to expose ATP systems to interested researchers, the CADE-16 ATP System Competition (CASC-16) will be held at CADE-16, Lindau, Germany, on Tuesday, 7th July, 1998. CASC-16 will evaluate the performance of sound, fully automatic, 1st order ATP systems. The evaluation will be in terms of:

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

in the context of:

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

1 Introduction

The CADE-16 ATP System Competition (CASC-16) will be held at CADE-16, Lindau, Germany, on Tuesday 7th July 1998. CASC-16 is the third CADE ATP system competition, following the successful competitions at CADE-13 in 1996 [SS97a] and CADE-14 in 1997 [SS98a]. CASC-16 will evaluate the performance of sound, fully automatic, 1st order ATP systems. The evaluation will be in terms of the number of problems solved and the average runtime for successful solutions, in the context of a bounded number of eligible problems chosen from the TPTP Problem Library [SS98b] and a specified time limit for each solution attempt. Eighteen ATP systems, listed in Table 1, have been entered into the various competition and demonstration divisions.

Table 1: The ATP Systems and Entrants

ATP System	Entrants
System	Entrant and associates

The competition will be run on SUN Ultras. The competition organizers are Geoff Sutcliffe and Christian Suttner. The competition will be overseen by a panel of knowledgeable researchers who are not participating in the event; the panel members are Alan Bundy, Claude Kirchner, and Jeff Pelletier.

This paper documents the design, procedures, and competing systems of the competition. The design and procedures of CASC-16 evolved from those of CASC-13 [SS97c, SS97b] and CASC-14 [SS98a]. The rules, deadlines, and specifications given here, are absolute. Only the competition panel has the right to make exceptions.

The CASC-16 WWW site provides access to resources used before, during, and after the event:
`http://www.cs.jcu.edu.au/~tptp/CASC-16/`

2 Divisions

CASC-16 is divided into divisions according to problem and system characteristics. There are four competition divisions in which the systems are explicitly ranked, and one demonstration division in which systems can demonstrate their abilities without being formally ranked.

2.1 Competition Divisions

- The **MIX Division**: Mixed CNF 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 is divided into five categories:
  - The **HNE Category**: Horn with No Equality
  - The **HEQ Category**: Horn with some (but not pure) Equality
  - The **NNE Category**: Non-Horn with No Equality
  - The **NEQ Category**: Non-Horn with some (but not pure) Equality
  - The **PEQ Category**: Pure Equality
- The **UEQ Division**: Unit Equality CNF Non-Propositional Theorems
- The **FOF Division**: Mixed FOF Non-Propositional Theorems
  The FOF Division is divided into two categories:
  - The **FNE Category**: No Equality
  - The **FEQ Category**: With Equality
- The **SAT Division**: Mixed CNF Non-Propositional Non-theorems

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

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

2.2 Demonstration Divisions

- The **SHW Division**: Special Hardware
  ATP systems that cannot run on the general hardware can enter into the Special Hardware Division. The hardware can be supplied by the entrant or accessed via the Internet. The entry specifies which general hardware division's problems are to be used. The results will be presented along with the general hardware division's results, but they cannot be compared with those from the systems that ran on the general hardware.

3 Problems

The problems will be selected from the TPTP Problem Library, v2.1.0 [SS97d].

3.1 Problem Selection

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

- The TPTP uses system performance data [SS00] 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 will be eligible.
- The TPTP distinguishes versions of problems as one of standard, incomplete, augmented, especial, or biased. Standard and especial problems will be eligible.

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

3.2 Number of Problems

The minimal numbers of problems that have to be used in each division and category, to ensure sufficient confidence in the competition results, will be determined from the numbers of eligible problems in each division and category (the competition organizers have to ensure that there is sufficient CPU time available to run the ATP systems on this minimal number of problems). These minimal numbers of problems will then be used in determining the time limit imposed on each solution attempt (see Section 4).

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

$$\text{Number of problems} = \frac{\text{Number of workstations} * \text{Time allocated}}{\text{Number of ATP systems} * \text{Time limit}} \tag{1}$$

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

3.3 Problem Preparation

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

- replace all predicate and function symbols by meaningless symbols,
- randomly reorder the clauses and literals in CNF problems,
- randomly reorder the formulae in FOF problems,
- randomly reverse the unit equalities in UEQ problems,
- remove equality axioms that are not needed by some of the ATP systems,
- output the problems in the formats required by the ATP systems. If the tptp2X utility does not support the format required, then it is the entrant's responsibility to provide a formating program.

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

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

4 Time Limits and Timing

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

A minimal time limit of 180 seconds will be used. The maximal time limit will be determined using the relationship used for determining the number of problems (see Section 3.2), with the minimal number of problems as the `Number of problems`. The time limit will be chosen as a reasonable value within the range allowed.

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

On the general hardware a wall clock time limit will be imposed in addition to the CPU time limit, in order to prevent very high memory usage that causes swapping. The wall clock time limit will be double the CPU time limit.

5 Entry Requirements and Procedures

System registration for CASC-16 closed on 8th June 1998, and no registrations were accepted after the 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 is 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 are expected to provide public access to their system's source code.

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

5.1 System Properties

The ATP systems have 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. No shell features, such as input or output redirection, may be used in the command line.

The ATP systems that run on the general hardware have to be interruptable by a SIGXCPU signal, so that the CPU time limit can be imposed on each solution attempt. The default action on receiving this signal is to exit (thus complying with the time limit, as required), but systems may catch the signal and exit of their own accord. Both approaches are acceptable for the competition. If any system runs past the time limit this will be noticed in the timing data and the system will be considered to have not solved that problem. In the Special Hardware Division the wall clock time limit will be imposed using SIGALRM.

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

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

5.2 System Installation

Access to the general hardware (or equivalent) was made available from the 16th June 1998, and the entrants had to have their ATP systems installed and operational by 26th June 1998. No late systems were accepted. Entrants had to ensure that their systems execute in the competition environment, according to prescribed checks.

After the installation deadline access to the general hardware was denied, and no further changes to the ATP systems were permitted. The organizers then tested the ATP systems, first to check that the systems execute correctly (according to the checks), and secondly to check for soundness.

For the soundness testing, non-theorems (selected from the TPTP) were submitted to the systems participating in the MIX, UEQ, and FOF Divisions, and theorems were submitted to the systems participating in the SAT Division. Finding a proof of a non-theorem or a model for a theorem indicated that the system is unsound. 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.

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

5.3 System Execution

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

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

The execution of the Special Hardware Division systems will be supervised by their entrants.

6 Performance Evaluation

The systems will be ranked within each competition division and category according to the number of problems solved. If several systems solve the same number of problems, then those systems will be ranked according to their average runtimes over solutions found. Winners will be announced in each division and category, and prizes will be awarded.

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

7 The ATP Systems

The ATP systems' descriptions start on the next page. These were written by the entrants.

7.1 System

I. Surname
Department, Institution
email

7.1.1 Architecture

7.1.2 Implementation

7.1.3 Expected Competition Performance

References

[KeyToUse] Surname I., Surname I. (year), Title, pp.pages, etc.

7.2 Bliksem 1.10

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

7.2.1 Architecture

Bliksem implements the ordered resolution + superposition calculus [BG90, DeN94, Rob65]. It supports many different orders, including reduction orders, A-orders, and non-liftable orders. Special attention has been given to resolution strategies that provide decision procedures for certain subsets of first order logic [FLTZ93, DeN98]. Bliksem is able to transform first order formulae into clausal normal form, using different structural or non-structural clause transformations. Bliksem 1.10 improves over Bliksem 1.00 by having a posteriori orders, (which will not be used during the competition), equality factoring + tautology elimination, optimized normal form transformations, non-unit demodulators, and equality subsumption (which takes into account commutativity of =).

In the automode Bliksem 1.10 is able to recognize decidable classes. However these are unlikely to show up in the competition. Apart from that, the automode is still quite primitive. We don't know how to select the right strategy + order.

Bliksem 1.00 has had soundness problems. We believe that there is no way in which the correctness of a program with the complexity of Bliksem can be guaranteed. We do not give any such guarantees for Bliksem 1.10. Instead Bliksem 1.10 is able to output its proofs with such a level of specification that they can be easily checked by another computer program. In the future we plan to have the proofs checked by a type checker, namely COQ [BBNU96].

7.2.2 Implementation

Bliksem is written in *C*. High priority has been given to portability. Bliksem has been compiled succcessfully using cc, gcc, and djpcc. The data structures have been carefully chosen after benchmark tests on the basic operations. We are still in the process of completing benchmark tests on the basic algorithms. Bliksem is available from:

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

7.2.3 Expected Competition Performance

Bliksem 1.10 improves over Bliksem 1.00. However all systems have been improved, so we cannot make any assumptions about the ranking of Bliksem 1.10. However we expect Bliksem to rank high in terms of number of generated clauses per second.

7.3 E 0.5

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

7.3.1 Architecture

E [Sch99] is a purely equational theorem prover. It was not originally designed as an independent prover, but as an preprocessor implementing the bottom-up part of the METOP calculus, with SETHEO [MIL+97] to implement the top-down part in a future combined system.

The calculus used by E combines superposition [BG94] (with selection of negative literals) and rewriting. For the special case that only unit clauses are present, this calculus degenerates into *unfailing completion* [BDP89]. For the Horn case, E can simulate variants of the positive unit strategy described in [Der91]. No special rules for non-equational literals have been implemented, i.e., resolution is simulated via paramodulation and equality resolution.

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

7.3.2 Implementation

E is implemented in ANSI C, using the GNU C compiler. The most unique feature of the implementation is the maximally shared term representation. This includes parallel rewriting for all instances of a particular subterm. A second important feature is the use of *perfect discrimination trees* with age and size constraints for rewriting and unit-subsumption.

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

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

7.3.3 Expected Competition Performance

We expect E to perform pretty good in all categories it participates in, with particularly strong showing for Horn problems. It is unlikely to make the first place in any category, though.

7.4 E-SETHEO 99csp and E-SETHEO-FLOTTER 99csp

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

7.4.1 Architecture

E-SETHEO is a compositional theorem prover for formulae in first order clause logic, combining the systems E [Sch99] and SETHEO [MIL+97]. It incorporates different calculi and proof procedures like superposition, model elimination and semantic trees (DPLL procedure). Furthermore, the system contains transformation techniques which may split the formula into independent subparts or which may perform a ground instantiation. Finally, advanced methods for controlling and optimizing the combination of the subsystems are applied.

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

On the top level, the system performs a parallelization of different strategies stemming from the system P-SETHEO [Wol98a]. In contrast to P-SETHEO, in the version 99csp of E-SETHEO, the different strategies are run sequentially, one after the other, depending on the allocation of resources to the different strategies, so-called *schedules*. A schedule is determined from rough syntactic characteristics of the input formula (e.g., if it is a Horn formula without equality). The schedules have been computed from a large number of experimental data using a combination of a genetic algorithm [SW99] and a gradient procedure [Wol98b].

7.4.2 Implementation

According to the diversity of the contained systems, the modules of E-SETHEO are implemented in different programming languages like C, Prolog, Scheme, and shell tools.

The program runs under Solaris 2.6. Sources are available freely.

7.4.3 Expected Competition Performance

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

7.5 FDP 0.9

P. Baumgartner
Institute for Computer Science, University of Koblenz
peter@uni-koblenz.de

7.5.1 Architecture

FDP is a complete theorem prover for first order clausal logic. The underlying calculus is a generalisation of the well-known Davis-Putnam procedure. The generalisation is the splitting wrt. complementary literals, which may contain variables.

The model construction technique is similar to the one of the predecessor calculus "Hyper Tableaux - The Next Generation" described in [Bau98]. Since there is no paper on FDP yet, this is the only relevant publication so far.

Proof search in FDP can be semantically guided by supplying an "initial interpretation", and proof search means to modify this initial interpretation until success. By default, a "trivial" initial interpretation is used that assign true to all positive literals. This emphasizes the role of negative clauses.

There is no built-in equality-handling (yet!) and so far FDP is restricted to clausal logic without equality. Currently, equality is treated in a preprocessing transformation similar to Brand's well-known STE-transformation.

7.5.2 Implementation

FDP is written as an interpreter in Eclipse Prolog. The implementation is highly experimental, preliminary and slow. Some, but not enough effort has been spent on clause selection for the next inference. This step is costly, but poorly implemented. The next implementation will be significantly faster by using indexing techniques.

The current implementation is available through the WWW at:
`http://www.uni-koblenz.de/~peter/FDP/`
Comments etc. are highly welcome!

7.5.3 Expected Competition Performance

No chance to seriously compete with the well-known, leading systems. I am optimistic that the situation will improve one day with a better implementation and when knowing the calculus better.

In the MIX division, I expect better performance for non-equality problems than for equality problems, and better performance for non-Horn problems than for Horn problems.

FDP decides function-free clause logic. Many SAT problems are of this kind. Therefore, FDP participates in the SAT division as well.

7.6 Gandalf c-1.8c

T. Tammet
Department of Computer Science, University of Goteborg
tammet@cs.chalmers.se

7.6.1 Architecture

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

The basic idea of the automatic mode in Gandalf — also used during the CASC-14 competition — is time-slicing: Gandalf selects a set of different search strategies, allocates time to these and finally runs the strategies one after another.

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: percentage of Horn and almost-Horn clauses, the number of clauses, percentage of clauses where literals can be ordered using term and variable depths.

There exist also versions of Gandalf for intuitionistic logic and type theory, see [Tam96] and [TS96]. More information about Gandalf can be found in [Tam98].

7.6.2 Implementation

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

The source, binaries, manual and other materials are available at:
`http://www.cs.chalmers.se/~tammet/gandalf/`.

7.6.3 Expected Competition Performance

Since Gandalf won the MIX class of CASC-14, we expect Gandalf to be one of the top performers in the MIX category during CASC-15. With some luck it may even win the MIX category again.

Gandalf is likely to do well in the UEQ, SAT and FOF categories, although it is unlikely it will win any of these.

7.7 MUSCADET v2.1

D. Pastre
Universit Ren Descartes (Paris 5), UFR de Mathmatiques et Informatique
pastre@math-info.univ-paris5.fr

7.7.1 Architecture

MUSCADET is a theorem prover based upon Natural Deduction, in the sense of [Ble71] and [Pas78]. It is built as a knowledged-based system, that is, methods are expressed by rules interpreted by an inference engine. Some rules are universal mathematical knowledge and are given to the system. Other rules are specific to some concepts and are built by metarules from the definitions of these concepts. The notions of hypothesis and conclusion to be proved are privileged. The first conclusion is the first order statement of the conjecture. ¡P¿ Methods are those used by humans, and are executed according to a plan decided by metarules and heuristics. Methods include splitting theorems into two or more subtheorems (conjonctive conclusion or disjunctive hypotheses), treatment of universal, existential or disjunctive conclusion, treatment of existential or disjunctive hypotheseses (one at a time when it is decided (by heuristic rules), replacement of the conclusion by its definition. Hypotheses are never replaced by their definition. Instead, rules are built by metarules from definitions and used to deduce new hypotheses from old ones. Universal hypotheses are not added. Instead, rules are built as from the definitions. Functional symbols may be used, but an automatic creation of intermediate objects allows to flatten deep subformulae and treat them as if concepts were defined by predicate symbols. Positive properties are privileged, in hypotheses and in conclusion. So, a special treatment of negation is done. The successive steps of a proof may be forward deduction (deduce new hypotheses from old ones), backward deduction (replace the conclusion by a new one) or refutation (only if the conclusion is a negation).

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

7.7.2 Implementation

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

7.7.3 Expected Competition Performance

The best performances of MUSCADET will be for problems manipulating many concepts in which all statements (conjectures, definitions, axioms) are expressed in a manner similar to the practice of humans, especially of mathematicians. It will have poor performances for problems using few concepts but large and deep formulas leading to many splittings.

7.8 Otter-MACE 437

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

7.8.1 Architecture

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

7.8.2 Implementation

Both programs are written in C. Otter uses shared data structures for clauses and terms, and it uses indexing for resolution, paramodulation, forward and backward subsumption, forward and backward demodulation, and unit conflict. MACE is much simpler than Otter; its Davis-Putnam-Loveland procedure uses a counting algorithm to do the unit resolutions, and forward subsumption is not applied. Otter-MACE is a shell script that runs Otter for a few seconds, then (if Otter has not found a refutation) MACE for a few seconds, then (if MACE has not found a model) Otter again for the remaining time.

7.8.3 Expected Competition Performance

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

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.

7.9 Scott IV

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

7.9.1 Architecture

Scott (Semantically Constrained Otter) [HS99] is an extension of Otter [McC94] and uses the same inference rules and overall proof search structure. This extension incorporates Finder [Sla94] (a finite model generator) as a subsystem of Otter in order to determine (maximally) consistent subsets of the clause space. These maximally consistent subsets are used to guide the proof search by allowing us to increase the likelihood of certain necessary inferences being made.

Unlike previous versions [Sla93, SLM94], the current version of Scott searches for maximally consistent subsets much more aggressively, and manages the cost of this much more intelligently. Most importantly, however, our naïve notions of semantic guidance have been superseded by those with genuine theoretical support. In addition, semantic resolution has been tentatively implemented and, provided it can be made sufficiently stable, will be incorporated in the competition version.

7.9.2 Implementation

Scott is written in C. It uses the existing program Otter to perform the proof search and additional code to determine which clauses should be chosen for inference. This additional code calls the existing program Finder in order to witness the consistency of particular sets of clauses.

7.9.3 Expected Competition Performance

In principle Scott should perform at least as well as Otter across all parts of the MIX and UEQ divisions, if only because it has Otter as a subroutine. Preliminary results show that, when Scott does find a proof, it requires at least an order of magnitude fewer inferences than Otter. However, searching for models and testing clauses against them can be costly, and so Scott may perform poorly if the time-limit is too severe. Overall, we expect good performance in comparison with cognate systems that employ relatively naïve guidance.

7.10 SSCPA 1999.6

G. Sutcliffe
Department of Computer Science, James Cook University
geoff@cs.jcu.edu.au

7.10.1 Architecture

SSCPA [SS99a] is an uncooperative multiple calculus competition parallelism ATP system, that multitasks on a single CPU. SSCPA uses the syntactic characteristics of the given problem to classify it into one of 18 predefined specialist problem classes (SPCs). SSCPA then uses performance data from available sequential ATP systems to recommend those systems that perform well for that SPC. The recommended systems are run in parallel, in one of several user selectable modes.

The choice of what syntactic characteristics are used to form the SPCs is based on community input and analysis of system performance data. Performance data for the available ATP systems, for problems in the TPTP [SS98b], is used to evaluate the systems within each SPC.

SSCPA has five modes of operation, which differ in terms of systems chosen, CPU allocation to each system, CPU priority for each system, and start time for each system. In CASC-16 SSCPA will run the ¡EM¿eager¡/EM¿ mode, which uses three or four recommended systems, CPU allocation halving from the time limit, equal CPU priority, and all systems starting together.

7.10.2 Implementation

SSCPA uses `perl` scripts that extract system recommendations from the performance data anto d control the parallel execution of the systems. The `tptp2X` utility is used to convert the problem to the formats expected by the ATP systems. A WWW interface to some SSCPA modes is publicly available at:

`http://www.cs.jcu.edu.au/cgi-bin/tptp/SystemOnTPTPFormMaker`

7.10.3 Expected Competition Performance

SSCPA is expected to perform well in the more heterogeneous divisions, i.e., MIX and FOF. The benefits of SSCPA's archtecture are less pronounced in divisions where single systems could dominate, e.g., UEQ, where Waldmesiter may dominate as it did in CASC-15 [SS99b].

7.11 SPASS 0.95TPTP

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

7.11.1 Architecture

SPASS [Wei97] is an automated theorem prover for full sorted first-order logic with equality. Its calculus is an extension of superposition by sorts [Wei98] and a splitting rule for case analysis. In contrast to many approaches to order-sorted clausal reasoning, the calculus enables sort predicates and equations to occur arbitrarily within clauses. Therefore, the sort theory is not separated from the problem clauses, but automatically and dynamically extracted [GMW97]. Nevertheless, it can be used to significantly cut the search space.

The splitting rule splits clauses, if possible, into variable disjoint parts that can be independently refuted. This rule is not necessary for completeness of the calculus but it makes SPASS an efficient decision procedure for several decidable subclasses of first-order logic [HS97].

We put a lot of effort in a clause normal form translation that produces small clause normal forms. The major techniques are renaming and improved Skolemization [NRW98].

7.11.2 Implementation

SPASS is written in ANSI-C and relies on a generic library that supports data structures and efficient functions for automated theorem proving.

The SPASS distribution containing the source code for SPASS is freely available from the SPASS homepage. The code should compile and run on any machine where the usual GNU tools are available. This has been tested on SUN Sparc workstations under Solaris and SunOS, on personal computers under Linux and Solaris and on Dec Alpha workstations under Digital Unix.

7.11.3 Expected Competition Performance

We hope to improve on the performance of SPASS at CASC-13 and CASC-14.

7.12 Waldmeister 798

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

7.12.1 Architecture

Waldmeister is a theorem prover for unit equational logic. Its proof procedure is unfailing Knuth-Bendix completion [BDP89]. We started the development from a three-level model of saturation-based ATP systems [BH96, HBVL97]. Since the efforts of the last years have led to a satisfactory inference machine, we now focus on the question how to guide the proof search.

The main completion loop is parameterized by a selection heuristic and a reduction ordering. Both are derived from the algebraic structure described by the axioms. Since last year, the derivation process has been lifted such that control knowledge is expressed declaratively, which eases the integration of human experience. For full flexibility, the search parameters may be re-instantiated during the proof search.

Recently, we also realized a database-driven mechanism which allows simple extensions of the underlying inference calculus, e.g. for reasoning on Horn clauses as described in [BDP89, pp. 24ff].

7.12.2 Implementation

Reaching efficiency both in terms of time and space has been a major goal in the development of our inference machine. As to the first, fast rewriting is achieved by sophisticated indexing techniques. As to the second, the set of critical pairs is represented in a compressed data structure.

To incorporate goal orientation into the completion procedure, the hypotheses are blown up to sets of rewrite successors and sometimes predecessors. A step-by-step proof object is constructed at run-time.

Our ATP system is implemented in ANSI-C and runs under Sun-OS, Solaris and Linux. The Waldmeister distribution contains an easy-to-use version along with documentation material and problem sets. If you are interested in tackling problems with Waldmeister, just send an e-mail to the authors.

7.12.3 Expected Competition Performance

We conjecture that Waldmeister will solve the majority of the competition problems in less time than one would need to study the corresponding problem specifications.

7.13 Waldmeister 799

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

This year's version of our unit equality theorem prover is in most parts identical to WALDMEISTER 798 that won the CASC-15 UEQ competition with an enormous lead [SS99b]. Since last year's description is included in this collection (Section 7.12), we will only report on the differences here.

7.13.1 Architecture

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

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

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

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

7.13.2 Implementation

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

7.13.3 Expected Competition Performance

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

8 Conclusion

The CADE-16 ATP System Competition will be the third large scale competition for 1st order ATP systems. The design improves on the previous competitions' designs by:

- Separating out the PEQ category in the MIX division.
- Dividing the FOF division in the FEQ and FNE categories.
- Limiting the wall clock time available for any solution attempt.
- Limiting excessive output from the ATP systems.
- Requiring proof objects to be written to `stdout`, so that they can automatically be stored as part of the system's output.

Organizational improvements include:

- Applying a strict deadline policy.
- Providing specific checks for system properties, so that entrants could ensure that their systems behave as required.
- Improved results display during the competition.

The organizers believe that the competition will fulfill its main motivations: stimulation of research, motivation for improving implementations, evaluation of relative capabilities of ATP systems, and providing an exciting event. For the entrants, their research groups, and their systems, there will be substantial publicity both within and outside the ATP community. The significant efforts that have gone into developing the ATP systems will receive public recognition; publications, which adequately present theoretical work, have not been able to expose such practical efforts appropriately. The competition will provide an overview of which researchers and research groups have decent, running, fully automatic ATP systems.

References

[Bau98] P. Baumgartner. Hyper Tableaux - The Next Generation. In H. de Swart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods: Proceedings of TABLEAUX'98*, number 1397 in Lecture Notes in Computer Science, pages 60–76. Springer-Verlag, 1998.

[BBNU96] E. Becht, A. Bloesch, R. Nickson, and M. Utting. Ergo 4.1 Reference Manual. Technical Report 96-31, Software Verification Research Centre (SVRC), University of Queensland, Brisbane, Australia, 1996.

[BCF+97] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, A. Meier, E. Melis, W. Schaarschmidt, J. Siekmann, and V. Sorge. OMEGA: Towards a Mathematical Assistant. In W.W. McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction*, number 1249 in Lecture Notes in Artificial Intelligence, pages 146–160. Springer-Verlag, 1997.

[BDP89] 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.

[BG90] L. Bachmair and H. Ganzinger. On Restrictions of Ordered Paramodulation with Simplification. In M.E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, number 449 in Lecture Notes in Artificial Intelligence, pages 427–441. Springer-Verlag, 1990.

[BG94] L. Bachmair and H. Ganzinger. Rewrite-Based Equational Theorem Proving with Selection and Simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[BH96] A. Buch and T. Hillenbrand. Waldmeister: Development of a High Performance Completion-Based Theorem Prover. Technical Report SEKI-Report SR-96-01, AG Effiziente Algorithmen, Universität Kaiserslautern, Kaiserslautern, Germany, 1996.

[Ble71] W.W. Bledsoe. Splitting and Reduction Heuristics in Automatic Theorem Proving. *Artificial Intelligence*, 2:55–77, 1971.

[DeN94] H. DeNivelle. Resolution Games and Non-Liftable Resolution Orderings. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic*, pages 279–293. Springer-Verlag, 1994.

[DeN98] H. DeNivelle. A Resolution Decision Procedure for the Guarded Fragment. 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 191–204. Springer-Verlag, 1998.

[Der91] N. Dershowitz. Ordering-Based Strategies for Horn Clauses. In Mylopolous J. and R. Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence* , pages 118–124. Morgan-Kaufmann, 1991.

[FLTZ93] C. Fermüller, A. Leitsch, T. Tammet, and N. Zamov. *Resolution Methods for the Decision Problem.* Number 679 in Lecture Notes in Computer Science. Springer-Verlag, 1993.

[GMW97] H. Ganzinger, C. Meyer, and C. Weidenbach. Soft Typing for Ordered Resolution. In W.W. McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction*, number 1249 in Lecture Notes in Artificial Intelligence, pages 321–335. Springer-Verlag, 1997.

[HBVL97] T. Hillenbrand, A. Buch, R. Vogt, and B. Löchner. Waldmeister: High Performance Equational Deduction. *Journal of Automated Reasoning*, 18(2):265–270, 1997.

[HJL99] 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.

[HS97] U. Hustadt and R. Schmidt. On Evaluating Decision Procedures for Modal Logic. Technical Report MPI-I-97-2-003, Max-Plank Institut Fur Informatik, Saarbrucken, Germany, 1997.

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

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

[McC00a] W.W. McCune. MACE: Models and Counterexamples. http://www-unix.mcs.anl.gov/AR/mace/, 2000.

[McC00b] W.W. McCune. Otter: An Automated Deduction System. http://www-unix.mcs.anl.gov/AR/otter/, 2000.

[MIL+97] 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.

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

[NRW98] A. Nonnengart, G. Rock, and C. Weidenbach. On Generating Small Clause Normal Forms. 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 397–411. Springer-Verlag, 1998.

[Pas78] D. Pastre. Automatic Theorem Proving in Set Theory. *Artificial Intelligence*, 10:1–27, 1978.

[Pas89] D. Pastre. MUSCADET : An Automatic Theorem Proving System using Knowledge and Metaknowledge in Mathematics. *Artificial Intelligence*, 38:257–318, 1989.

[Pas93] D. Pastre. Automated Theorem Proving in Mathematics. *Annals of Mathematics and Artificial Intelligence*, 8:425–447, 1993.

[Rob65] J.A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1):23–41, 1965.

[Sch99] S. Schulz. System Abstract: E 0.3. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 297–301. Springer-Verlag, 1999.

[Sla93] J.K. Slaney. SCOTT: A Model-Guided Theorem Prover. In R. Bajcsy, editor, *Proceedings of the 13th International Conference on Artificial Intelligence* , pages 109–114. Morgan-Kaufmann, 1993.

[Sla94] 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.

[SLM94] J.K. Slaney, E. Lusk, and W.W. McCune. SCOTT: Semantically Constrained Otter, System Description. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in Lecture Notes in Artificial Intelligence, pages 764–768. Springer-Verlag, 1994.

[SS97a] G. Sutcliffe and C.B. Suttner. Special Issue: The CADE-13 ATP System Competition. *Journal of Automated Reasoning*, 18(2), 1997.

[SS97b] G. Sutcliffe and C.B. Suttner. The Procedures of the CADE-13 ATP System Competition. *Journal of Automated Reasoning*, 18(2):163–169, 1997.

[SS97c] C.B. Suttner and G. Sutcliffe. The Design of the CADE-13 ATP System Competition. *Journal of Automated Reasoning*, 18(2):139–162, 1997.

[SS97d] C.B. Suttner and G. Sutcliffe. The TPTP Problem Library (TPTP v2.1.0), 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.

[SS98a] 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.

[SS98b] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[SS99a] G. Sutcliffe and D. Seyfang. Smart Selective Competition Parallelism ATP. In A. Kumar and I. Russell, editors, *Proceedings of the 12th Florida Artificial Intelligence Research Symposium*, pages 341–345. AAAI Press, 1999.

[SS99b] G. Sutcliffe and C.B. Suttner. The CADE-15 ATP System Competition. *Journal of Automated Reasoning*, 23(1):1–23, 1999.

[SS00] G. Sutcliffe and C.B. Suttner. ATP System Results for the TPTP Problem Library. http://www.cs.jcu.edu.au/~tptp/TPTP/Results.html, 2000.

[SW99] G. Stenz and A. Wolf. Strategy Selection by Genetic Programming. In A. Kumar and I. Russell, editors, *Proceedings of the 12th Florida Artificial Intelligence Research Symposium*, pages 346–350. AAAI Press, 1999.

[Tam96] T. Tammet. A Resolution Theorem Prover for Intuitionistic Logic. 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 2–16. Springer-Verlag, 1996.

[Tam98] T. Tammet. Towards Efficient ATPProgress. 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.

[TS96] T. Tammet and J. Smith. Optimised Encodings of Fragments of Type Theory in First Order Logic. In S. Berardi and M. Coppo, editors, *Types for Proofs and Programs: Proceedings of the International Workshop TYPES'95*, number 1158 in Lecture Notes in Artificial Intelligence, pages 265–287. Springer-Verlag, 1996.

[Wei97] C. Weidenbach. SPASS: Version 0.49. *Journal of Automated Reasoning*, 18(2):247–252, 1997.

[Wei98] C. Weidenbach. Sorted Unification and Tree Automata. In W. Bibel and P.H. Schmitt, editors, *Automated Deduction, A Basis for Applications*, volume 1 Foundations - Calculi and Methods, pages 291–320. Kluwer Academic Publishers, 1998.

[WGR96] 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.

[Wol98a] A. Wolf. p-SETHEO: Strategy Parallelism in Automated Theorem Proving. In H. de Swart, editor, *Proceedings of International Conference TABLEAUX'98: Analytic Tableaux and Related Methods*, number 1397 in Lecture Notes in Artificial Intelligence, pages 320–324. Springer-Verlag, 1998.

[Wol98b] A. Wolf. Strategy Selection for Automated Theorem Proving. In F. Giunchiglia, editor, *Proceedings of the 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA)*, number 1480 in Lecture Notes in Artificial Intelligence, pages 452–465. Springer-Verlag, 1998.