CADE-15

The 15th International Conference on Automated Deduction

July 5-10, 1998, Lindau, Germany

The CADE-15 ATP System Competition

July 7

Geoff Sutcliffe and Christian Suttner

CASC-15 Web Site — `http://www.cs.jcu.edu.au/ tptp/CASC-15/`

CADE-15 Web Site — `http://www.th-darmstadt.de/cade-15`

Abstract

In order to stimulate ATP system development, and to expose ATP systems to interested researchers, the CADE-15 ATP System Competition (CASC-15) will be held at CADE-15, Lindau, Germany, on Tuesday, 7th July, 1998. CASC-15 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-15 ATP System Competition (CASC-15) will be held at CADE-15, Lindau, Germany, on Tuesday 7th July 1998. CASC-15 is the third CADE ATP system competition, following the successful competitions at CADE-13 in 1996 [SS97b] and CADE-14 in 1997 [SS98]. CASC-15 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 [SSar] 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
AI-SETHEO	Joachim Draeger
ANDP	Dafa Li
Bliksem 1.00	Hans de Nivelle
DISCOUNT/TSM 2.1	Stephan Schulz, Felix Brandt
E 0.1	Stephan Schulz
Gandalf c-1.1	Tanel Tammet
HERBY 1.1	Mohammed Almulla, Monty Newborn
MACE 1.3.2	William McCune, Larry Wos, Ross Overbeek, Rusty Lusk
Octopus	Monty Newborn
Otter 3.0.5	William McCune, Larry Wos, Ross Overbeek, Rusty Lusk
PHERBY 1.0	Mohammed Almulla, Monty Newborn
p-SETHEO	Andreas Wolf, Reinhold Letz, Gernot Stenz
Functional Satchmo	Tim Geisler, Sven Panne
SCOTT v3.1.0	Kal Hodgson, John Slaney
SPASS 1.0.0.a	Christoph Weidenbach, Harald Ganzinger, Christoph Meyer
TGTP	Monty Newborn
Vampire	Andrei Voronkov, Alexander Ryazanov
Waldmeister 798	Thomas Hillenbrand, Andreas Jaeger, Bernd Loechner, Arnim Buch

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-15 evolved from those of CASC-13 [SS97d, SS97c] and CASC-14 [SS98]. The rules, deadlines, and specifications given here, are absolute. Only the competition panel has the right to make exceptions.

The CASC-15 WWW site provides access to resources used before, during, and after the event:
`http://www.cs.jcu.edu.au/~tptp/CASC-15/`

2 Divisions

CASC-15 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 [SS97e].

3.1 Problem Selection

The problems have 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 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-15 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 AI-SETHEO

J. Draeger
Institut für Informatik, Technische Universität München
draeger@informatik.tu-muenchen.de

7.1.1 Architecture

AI-SETHEO is a hybrid system combining a connection tableaux theorem prover (SETHEO [GL+94, MI+97]) and a lemma generator (DELTA [Sch94]). This combines aspects of bottom-up reasoning with the top-down reasoning in connection tableaux theorem provers. For avoiding a performance breakdown due to the overwhelming number of brute-force generated lemmata a lemma selection procedure rejects both trivial and irrelevant lemmata.

7.1.2 Implementation

AI-SETHEO is implemented as a shell script which invokes the prover SETHEO, the lemma generator DELTA, and the lemma selection procedure. The selection procedure is realized in GAWK.

7.1.3 Expected Competition Performance

AI-SETHEO is still under construction; thus the competition performance can only be guessed. Early experiments show that it should be equal or better to the performance of SETHEO.

References

[GL+94] Goller C., Letz R., Mayr K., Schumann J. (1994), SETHEO V3.2: Recent Developments, Bundy A., *Proceedings of the 12th International Conference on Automated Deduction* (Nancy, France), pp.778-782, Lecture Notes in Artificial Intelligence 814, Springer-Verlag.

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

[Sch94] Schumann J. (1994), DELTA - A Bottom-up Preprocessor for Top-Down Theorem Provers, Bundy A., *Proceedings of the 12th International Conference on Automated Deduction* (Nancy, France), pp.774-777, Lecture Notes in Artificial Intelligence 814, Springer-Verlag.

7.2 ANDP: Automated Natural Deduction Prover

Dafa Li
Department of Applied Mathematics, Tsinghua University
lidafa@tsinghua.edu.cn

7.2.1 Architecture

ANDP [Li97] is a natural deduction system for first-order logic without equality, adapted from Gentzen's system. There are five connectives for ANDP, $\sim$, $\wedge$, $\vee$, $\rightarrow$, and iff, whose precedence decreases in that order. The rules of inference for the connectives are MP (Modus Ponens), MT (Modus Tollens), LDS and RDS (Left and Right Disjunctive Syllogism), CP (Conditional Proof), ADD (Addition), CNJUN (Conjunction), SIMP (Simplication), De Morgan's rule, Double negation, IMP (Implication), and DILEMMA (cases). There are two quantifiers for ANDP, the universal quantifier (Ax) and the existential quantifier (Ex). The rules of inference for the quantifiers are UG (Universal Generalization), EG (Existential Generalization), US (Universal Specialization), and ES (Existential Specialization). There are two unification algorithms for treating quantifiers. One is for introducing quantifiers, the other is for eliminating quantifiers without Skolem functions.

Search strategies: First use the rules for connectives then for quantifiers. First reason from bottom to top, then from top to bottom.

Other strategies: Subsumption with quantifiers, first eliminate existential quantifiers then universal quantifiers, minimal scopes of quantifiers.

7.2.2 Implementation

ANDP is written in Common LISP (Lucid Common Lisp, KCL) and runs on a SUN workstation. In input formulas & , *or*, and $->$ are used to stand for $\wedge$, $\vee$, and $\rightarrow$, respectively. Therefore input formulas, e.g., `(Ax)[Px & Qy -> Py]`, look like those in text books. The first-order logic formulas are stored internally in Polish style.

7.2.3 Expected Competition Performance

ANDP is entered in the FOF division. ANDP can not treat formulas with equality. To treat equality it is neccessary to introduce axioms for equality. Therefore ANDP is expected to perform well on theorems without equality, but may fail to prove theorems with equality.

References

[Li97] Li D. (1997), Unification Algorithms for Eliminating and Introducing Quantifiers in Natural Deduction Automated Theorem Proving, *Journal of Automated Reasoning* 18(1), pp.105-134.

7.3 Bliksem 1.00

H. de Nivelle
CWI Amsterdam
nivelle@cwi.nl

7.3.1 Architecture

Bliksem is a resolution based theorem prover. It supports ordered resolution, factoring, ordered paramodulation, unit simplification and demodulation. Bliksem supports many orders (KBO, RPO, LPO, simple symbol orders, *non-lifaftble* orders) and *resolution decision procedures.* At this moment it has decision procedures for the E^+-class, for the prefix class $\exists^*\forall\forall\exists^*$, and for the guarded fragment of first order logic. ([FL+93], [AvN96], [DeN94]) Additionally Bliksem also has the equality strategies of ([BG90]) There is no hyperresolution in Bliksem, but it can be simulated using an appropriate order.

Bliksem supports various transformations from first order logic to clausal normal form. In particular it supports various structural and non-structural transformations. ([BFL94]).

7.3.2 Implementation

Bliksem is implemented in C. The datastructure that is used for literals, is the *prefix with ends* representation. A literal is stored in two arrays t and e. The first contains the literal in prefix notaton. The second contains for each subterm a reference to the end of this subterm. So suppose that the term $p(X, s(s(Y)))$ is stored on position 100. This is done as follows:

t[100] = p	e[100]= 105
t[101] = X	e[101]= 102
t[102] = s	e[102]= 105
t[103] = s	e[103]= 105
t[104] = Y	e[104]= 105

Clauses are stored in another array as a sequence of indices referring to the literals.

We use discrimination trees for forward subsumption, forward demodulation, and forward unit simplification. In the discrimination trees the terms are stored in a normalized manner. In addition we use bit filters as a filter for subsumption. There are no optimized techniques for the other derivation rules.

7.3.3 Expected Competition Performance

As it is the first time for us it is difficult to give expected performances.

References

[AvN96] Andréka H., van Benthem J., Németi I. (1996), **Modal Languages and Bounded Fragments of Predicate Logic**, Research Report ML-96-03, Institute for Logic, Language and Computation, Ansterdam, The Netherlands.

[BFL94] Baaz M., Fermüller C., Leitsch A. (1994), **A Non-Elementary Speed Up in Proof Length by Structural Clause Form Transformation**, *Proceedings of the 9th Annual IEEE Symposium on Logic in Computer Science* (Paris, France), pp.213-219, IEEE Computer Society Press.

[BG90] Bachmair L., Ganzinger H. (1990), **On Restrictions of Ordered Paramodulation with Simplification**, Stickel M.E., *Proceedings of the 10th International Conference on Automated Deduction* (Kaiserslautern, Germany), pp.427-441, Lecture Notes in Artificial Intelligence 449, Springer-Verlag.

[DeN94] DeNivelle H. (1994), **Resolution Games and Non-Liftable Resolution Orderings**, Pacholski L., Tiuryn J., *Computer Science Logic*, pp.279-293, Springer-Verlag.

[FL+93] Fermuller C., Leitsch A., Tammet T., Zamov N. (1993), **Resolution Methods for the Decision Problem**, Lecture Notes in Computer Science 679, Springer-Verlag.

7.4 DISCOUNT/TSM 2.1

S. Schulz, F. Brandt
Institut für Informatik, Technische Universität München
{schulz,brandtf}@informatik.tu-muenchen.de

7.4.1 Architecture

DISCOUNT/TSM is a sequential version of the well-known distributed equational theorem prover DISCOUNT [DKS97], 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 next equation to be processed is controlled by a *term space map* (TSM) [Sch98], a generalized *term evaluation tree* [DS96]. A TSM is a recursive structure capturing characteristic distributions of term nodes in representative patterns of useful and useless equations collected from previous proofs. Suitable proofs for learning are selected using tree-based similarity criteria on TSMs and some simple features of the problem specification.

7.4.2 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 originally was designed in 1991, before the use of efficient indexing methods became widespread. Consequently, the basic inference engine is primitive by today's standards, and modern systems (e.g. Waldmeister [HBF96]) achieve a much higher throughput.

7.4.3 Expected Competition Performance

We expect this new DISCOUNT version to perform similarly to the previous versions. That is, it should be a serious contender, but will probably not be among the top three systems.

References

[DKS97] Denzinger J., Kronenburg M., Schulz S. (1997), **DISCOUNT: A Distributed and Learning Equational Prover**, *Journal of Automated Reasoning* 18(2), pp.189-198.

[DS96] Denzinger J., Schulz S. (1996), **Learning Domain Knowledge to Improve Theorem Proving**, McRobbie M., Slaney J.K., *Proceedings of the 13th International Conference on Automated Deduction* (New Brunswick, USA), pp.62-76, Lecture Notes in Artificial Intelligence 1104, Springer-Verlag.

[HBF96] Hillenbrand T., Buch A., Fettig R. (1996), **On Gaining Efficiency in Completion-Based Theorem Proving**, Ganzinger H., *Proceedings of the 7th International Conference on Rewriting Techniques and Applications* (New Brunswick, USA), pp.432-435, Lecture Notes in Computer Science 1103, Springer-Verlag.

[Sch98] Schultz S. (1998), **Term Space Mapping for DISCOUNT**, *Proceedings of the CADE-15 Workshop on Using AI Methods in Deduction* (Lindau, Germany).

7.5 E 0.1

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

7.5.1 Architecture

E 0.1 is a purely equational theorem prover. It is not primarily intended as an independent prover, but as an preprocessor implementing the bottom-up part of the METOP calculus [Mos96], with SETHEO [MI+97] to implement the top-down part in a future combined system. However, as the calculus employs inference rules very similar to those of superposition calculi [BG94], it is also possible to use E as a prover on its own.

The implemented calculus combines ordered paramodulation with rewriting. For the special case that only unit clauses are present, this calculus degenerates into *unfailing completion* [BDP89] as implemented in DISCOUNT [DKS97]. As we expect SETHEO to deal with the non-equational part of the theory, no special rules for non-equational literals have been implemented, i.e. resolution is simulated via paramodulation and equality resolution.

A particular feature of E is the term representation. Terms are maximally shared (within two classes of clauses) to minimize memory consumption, and we employ graph rewriting techniques to optimize simplification. As an additional feature, E has a very clean interface for adding and developing new search control heuristics.

7.5.2 Implementation

E is implemented in ANSI C, using the GNU C compiler. The program has been successfully installed under SunOS 4.3.x, Solaris 2.x, HP-UX B 10.20, and various versions of Linux. It uses it's own library, CLIB, for data types, I/O, and memory management.

7.5.3 Expected Competition Performance

E is in a very early phase of development, so it is hard to predict actual performance. We expect it to perform adequately in the UEQ category, while the performance in the MIX category is likely to be mediocre for the current version.

References

[BDP89] Bachmair L., Dershowitz N., Plaisted D.A. (1989), **Completion Without Failure**, Ait-Kaci H., Nivat M., *Resolution of Equations in Algebraic Structures*, pp.1-30, Academic Press.

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

[DKS97] Denzinger J., Kronenburg M., Schulz S. (1997), **DISCOUNT: A Distributed and Learning Equational Prover**, *Journal of Automated Reasoning* 18(2), pp.189-198.

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

[Mos96] Moser M. (1996), **Goal-Directed Resoning in Clausal Logic with Equality**, PhD thesis, Institut für Informatik, Technische Universität München, Munich, Germany, and Dissertationen zur Informatik 2, CS Press.

7.6 Gandalf c-1.9c

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.

References

[Tam98] Tammet T. (1998), Towards Efficient Subsumption, Kirchner C., Kirchner H., *Proceedings of the 15th International Conference on Automated Deduction* (Lindau, Germany), Lecture Notes in Artificial Intelligence, Springer-Verlag.

[Tam96] Tammet T. (1996), A Resolution Theorem Prover for Intuitionistic Logic, McRobbie M., Slaney J.K., *Proceedings of the 13th International Conference on Automated Deduction* (New Brunswick, USA), pp.2-16, Lecture Notesin Artificial Intelligence 1104, Springer-Verlag.

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

7.7 HERBY and PHERBY

M. Almulla
Department of Mathematics and Computer Science, Kuwait University M. Newborn
almulla@sun470.sci.kuniv.edu.kw
School of Computer Science, McGill University
newborn@opus.cs.mcgill.ca

7.7.1 Architecture

HERBY [Alm95, AN96] is a semantic-tree based theorem prover that implements a number of different heuristics [YAN96, YAN] for choosing ground atoms from the Herbrand base of the given problem. The version of HERBY that will participate at CADE-15 is expected to use 15 atom selection heuristics. The selection of heuristics and the percentage of time they receive is fixed and is independent of the characteristics of the problem. An earlier version of HERBY was tested on the Stickel test set.

PHERBY is a parallel version of HERBY. PHERBY uses a bigger set of heuristics than HERBY, as well as cooperative computing in determining the quality of atoms to be used. This is the first version of the program.

7.7.2 Implementation

HERBY and PHERBY are implemented in C. PHERBY runs on a PVM consisting of a network of approximately 20 120MHz and 150MHz pentium computers running the LINUX operating system.

7.7.3 Expected Competition Performance

72 theorems were used for testing ATP programs at the CADE-14 competition. HERBY solved 20 of them in less than 300 seconds.

PHERBY is expected to perform better than the serial version HERBY. However, it is expected to show much weaker performance compared to resolution-refutation theorem provers. Out of the 72 theorems that were used for testing ATP programs at the CADE-14 competition, PHERBY solved 24 in less than 300 seconds.

References

[Alm95] Almulla M. (1995), **Analysis of the Use of Semantic Trees for Proofs of Unsatisfiability**, PhD thesis, McGill University, Montreal, Canada.

[AN96] Almulla M., Newborn M. (1996), **The Practicality of Generating Semantic Trees for Proofs of Unsatisfiability**, *International Journal of Computer Mathematics* 62(1-2), pp.45-61.

[YAN96] Yu Q., Almulla M., Newborn M. (1996), **Heuristics for a Semantic Tree Theorem Prover**, Kautz H., Selman B., *Proceedings of the 4th International Symposium on Artificial Intelligence and Mathematics AI/MATH-96* (Fort Lauderdale, USA), pp.162-165.

[YAN] Yu Q., Almulla M., Newborn M. (Submitted), **Heuristics used by HERBY for Semantic Tree Theorem Proving**, *Annals of Mathematics and Artificial Intelligence*.

7.8 Otter 3.0.5 and MACE 1.3.2

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

7.8.1 Architecture

Otter [McC94a, McC96, MW97] searches for proofs, and MACE [McC94b, McC94c] 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.

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.

7.8.3 Expected Competition Performance

These versions of Otter and MACE are nearly identical to the versions entered in last year's CASC-14. Both programs did well last year, but neither won any top prizes. We do not expect any winning performances in CASC-15.

References

[McC94a] McCune W.W. (1994), **Otter 3.0 Reference Manual and Guide**, Technical Report ANL-94/6, Argonne National Laboratory, Argonne, USA.

[McC94b] McCune W.W. (1994), **A Davis-Putnam Program and its Application to Finite First-Order Model Search: Quasigroup Existence Problems**, Technical Report ANL/MCS-TM-194, Argonne National Laboratory, Argonne, USA.

[McC94c] McCune W.W. (1994), **MACE: Models and Counterexamples**, http://www.mcs.anl.gov/home/mccune/ar/mace/.

[McC96] McCune W.W. (1996), **Otter**, http://www.mcs.anl.gov/home/mccune/ar/otter/.

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

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 p-SETHEO

A. Wolf, R. Letz, G. Stenz
Institut für Informatik, Technische Universität München
{wolfa,letz,stenzg}@in.tum.de

7.9.1 Architecture

p-SETHEO [Wol98] (single processor version) is an environment for strategy parallel competition of automated theorem provers. Actually, it is used for controlling strategies which consist of different parameterizations of the automated theorem prover SETHEO [MI+97]. p-SETHEO time-slices different strategies. The first successful sub-prover stops the system.

7.9.2 Implementation

The implementation uses Perl, C and shell tools in approximately 1000 lines of code (without SETHEO).

7.9.3 Expected Competition Performance

p-SETHEO is still under construction, so we cannot give any estimations.

References

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

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

7.10 Functional Satchmo

T. Geisler and S. Panne
Insitut für Informatik, Universität München
{Tim.Geisler,Sven.Panne}@informatik.uni-muenchen.de

7.10.1 Architecture

Functional Satchmo [GPS97] implements the *positive unit hyper-resolution (PUHR) tableau method* [BY96], a calculus for range-restricted clauses, which is actually a formalisation of Satchmo [MB88].

PUHR tableaux are trees in which every node is labeled with a ground atom (*unit*) or with a *positive ground clause*. PUHR tableaux for a given clause set are built starting from the tree consisting of a single unlabeled node, by repeatedly applying the following two rules to the leaf of a branch. The *PUHR rule* resolves the body literals of an input clause simultaneously with the units on the branch and extends the branch with a new leaf, labeled with the (hyper-)resolvent. The *splitting rule* extends the branch with n new leaf nodes $B_1, \ldots, B_n$, if the branch contains a positive ground clause $B_1 \vee \ldots \vee B_n$, but none of the units B_1 or ... or B_n. If the empty clause is selected, the branch is marked as *closed*. If none of these rules is applicable, then the set of units of a branch represents a Herbrand model of the input clause set. If all branches are closed, then the input clause set is unsatisfiable.

Complement splitting, a variant of the above splitting rule, is used to enhance efficiency and to guarantee minimality of the leftmost generated model [BY96].

Incremental evaluation avoids repeated applications of the PUHR rule with the same clauses and accelerates its applicability test: If a unit is added to the branch in a splitting step, then exactly those PUHR steps using this unit have to be applied as the next steps. This leads to significantly more efficient implementations of the PUHR tableau method [SG96].

7.10.2 Implementation

Functional Satchmo is implemented in the purely functional language Haskell, which performs lazy evaluation. A PUHR tableau is represented as a (lazy) list of its open branches. The sets of units in tableau branches are represented as standard discrimination trees.

7.10.3 Expected Competition Performance

We expect average performance in the SAT division.

References

[BY96] Bry F., Yahya A. (1996), **Minimal Model Generation with Positive Unit Hyper-resolution Tableaux**, *Proceedings of TABLEAUX'96: the 5th Workshop on Theorem Proving with Analytic Tableaux and Related Methods* (Palermo, Italy), pp.143-159, Lecture Notes in Artificial Intelligence 1071, Springer-Verlag.

[GPS97] Geisler T., Panne S., Schütz H. (1997), **Satchmo: The Compiling and Functional Variants**, *Journal of Automated Reasoning* 18(2), pp.227-236.

[MB88] Manthey R., Bry F. (1988), **SATCHMO: A Theorem Prover Implemented in Prolog**, Lusk E. Overbeek R., *Proceedings of the 9th International Conference on Automated Deduction* (Argonne, USA), pp.415-434, Lecture Notes in Computer Science 310, Springer-Verlag.

[SG96] Schütz H., Geisler T. (1996), **Efficient Model Generation through Compilation**, McRobbie M., Slaney J.K., *Proceedings of the 13th International Conference on Automated Deduction* (New Brunswick, USA), pp.431-447, Lecture Notes in Artificial Intelligence 1104, Springer-Verlag.

7.11 SCOTT v3.1.0

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

7.11.1 Architecture

Scott (Semantically Constrained Otter) [SH98] is an extension of Otter [McC94] and uses the same inference rules and overall proof search structure as Otter. It adds to its parent prover a module which searches dynamically for small finite models of subsets of the clauses in order to extract semantic information which may help to guide the proof search. For the purposes of CASC-15, the models are used only to adjust the weights of clauses in order to make Otter prefer clauses which are false in the guiding model or models. The difference between the present version of Scott and previous ones [SLM94, Sla93] is mainly that it now generates and uses many models rather than just one, resulting in richer semantic control and in greater stability under small perturbations of problem formulations and or settings. Although many more elaborate uses of the models are easy to envisage, for the present the extra weight assigned to clauses is a function simply of the number of models in which they are true.

7.11.2 Implementation

Scott is written in C. It uses the existing program Otter as a sub-program to perform the proof search and the existing constraint satisfaction program Finder [Sla94] to generate the models. The additional code implements the interface between the two, maintains the list of known models, labels clauses with the numbers of models in which they hold and controls calls to the model-generator.

7.11.3 Expected Competition Performance

In principle Scott should perform at least as well as Otter across all parts of the Mix division, if only because it has Otter as a subroutine. However, in practice it does not invariably do better than Otter, for three reasons. Firstly it has the extra overheads of searching for models and testing clauses against them. Secondly, the settings of Otter in autonomous mode are fine-tuned for searching without semantic guidance and may not be ideal when a semantic component is added. Thirdly, there are no magic bullets: any strategy may lose sometimes, and in order to make the comparison interesting we always make Scott try a model-guided search if possible. Overall, we expect good performance in comparison with cognate systems that employ relatively naïve guidance.

References

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

[SH98] Slaney J.K., Hodgson K. (1998), **System Description: MSCOTT**, TR-ARP-06-98, Automated Reasoning Project, Australian National University, Canberra, Australia.

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

[Sla93] Slaney J.K. (1993), **SCOTT: A Model-Guided Theorem Prover**, Bajcsy R., *Proceedings of the 13th International Conference on Artificial Intelligence* (Chambery, France), pp.109-114, Morgan-Kaufman.

[Sla94] Slaney J.K. (1994), **Finite Domain Enumerator, System Description**, Bundy A., *Proceedings of the 12th International Conference on Automated Deduction* (Nancy, France), pp.764-768, Lecture Notes in Artificial Intelligence 814, Springer-Verlag.

7.12 SPASS 0.95TPTP

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

7.12.1 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].

7.12.2 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.

7.12.3 Expected Competition Performance

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

References

[GMW97] Ganzinger H., Meyer C., Weidenbach C. (1997), **"Soft Typing for Ordered Resolution**, McCune W.W., *Proceedings of the 14th International Conference on Automated Deduction* (Townsville, Australia), pp.321-335, Lecture Notes in Artificial Intelligence 1249, Springer-Verlag.

[HS97] Hustadt U., Schmidt R.A. (1997), **On Evaluating Decision Procedures for Modal Logics**, Pollack M.E., *Proceedings of the 15th International Joint Conference on Artificial Intelligence* (Nagoya, Japan), pp.202-207.

[NRW98] Nonnengart A., Rock G., Weidenbach C. (1998), **On Generating Small Clause Normal Forms**, Kirchner C., Kirchner H., *Proceedings of the 15th International Conference on Automated Deduction* (Lindau, Germany), Lecture Notes in Artificial Intelligence, Springer-Verlag.

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

[Wei98] Weidenbach C. (1998), **Sorted Unification and Tree Automata**, Bibel W., Schmitt P.H., *Automated Deduction, A Basis for Applications*, Kluwer Academic Publishers.

7.13 TGTP and Octopus

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

7.13.1 Architecture

TGTP [New97] is a resolution-refutation theorem prover. It carries out a sequence of iteratively-deepening searches, looking for a linear proof. While searching for a linear proof, TGTP also stores information on generated unit clauses. This information is stored in a large hash table with space for one million entries, permitting proofs that are not linear to be derived. Completeness is sacrificed only by limits on the number of literals in a clause and on the number of constants, functions, and variables in a literal. TGTP uses only two inference rules: binary resolution and binary factoring.

Octopus is a parallel verison of TGTP, with one master and as many slaves as available - 25 are expected for the CADE-15 competition. The master computer reads the given theorem from disk, spawns slaves and then sends the theorem to the slaves. Each slave carries out the same search process, but each palces a different limit on the number of literals in a clause and the number of constants, functions, and variables in a literal. In addition, some processors use the set-of-support strategy and others do not.

7.13.2 Implementation

TGTP is implemented in C, consisting of twelve -.c files, two -.h files, 150 functions, and 12286 lines of code.

Octopus modifies only slightly the code of TGTP for the parallel implementation. It is expected to run on a network of 20-25 pentium computers running at approximately 150Mhz. The processors communicate with eachother using Parallel Virtual Machine (pvm).

7.13.3 Expected Competition Performance

At CADE-14, TGTP solved 34 of the 72 theorems. The current version is slightly stronger, capable of solving a few more of the 72 theorems, but remaining weak on theorems involving the equality axioms.

Octopus is slightly stronger than TGTP, solving 40 theorems at CADE-14 and currently able to solve another three or four.

References

[New97] Newborn M. (1997), **The Great Theorem Prover, Version 3**, Newborn Software, Montreal, Canada.

7.14 Vampire

A. Voronkov
Computer Science Department, Uppsala University
voronkov@csd.uu.se

7.14.1 Architecture

Vampire [Vor95] is a resolution-based theorem prover. It is not fully implemented yet, so it is unclear what features we shall be able to present at CASC.

In the near future we plan to implement binary/hyperresolution with clause splitting. Our main emphasis is on the efficient implementation as opposite to logical optimizations.

7.14.2 Implementation

Vampire is implemented in `C++` with no tuning to a particular computer or system. Except for a couple of string library functions, ANSI `C++` is used.

We use very efficient data structures for indexing. At the moment we have code trees for forward subsumption and a combination of discrimination trees and substitution trees for back subsumption and aggressive term structure sharing. We are working on code trees for binary resolution and hyperresolution.

7.14.3 Expected Competition Performance

We do not expect high performance since we shall not be able to implement equality before the competition. Our aim is to see how it works on nonequality problems and make a good version for the 1999 competition.

References

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

7.15 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

7.15.1 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, HB+97]. 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].

7.15.2 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.

7.15.3 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.

References

[BDP89] Bachmair L., Dershowitz N., Plaisted D.A. (1989), **Completion Without Failure**, Ait-Kaci H., Nivat M., *Resolution of Equations in Algebraic Structures*, pp.1-30, Academic Press.

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

[HB+97] Hillenbrand T., Buch A., Vogt R., Löchner B. (1997), **Waldmeister: High Performance Equational Deduction**, *Journal of Automated Reasoning* 18(2), pp.265-270.

8 Conclusion

The CADE-15 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

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

[SS97e] 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.

[SS98] G. Sutcliffe and C. Suttner. The CADE-14 ATP System Competition. Technical Report Technical Report 98/01, Department of Computer Science, James Cook University, Townsville, Australia, 1998.

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