Otter: The CADE-13 Competition Incarnations

WILLIAM McCUNE and LARRY WOS *
Mathematics and Computer Science Division, Argonne National Laboratory
Argonne, IL, U.S.A
mccune@mcs.anl.gov, wos@mcs.anl.gov

Abstract. This article discusses the two incarnations of Otter entered in the CADE-13 Automated Theorem Proving Competition. Also presented are some historical background, a summary of applications that have led to new results in mathematics and logic, and a general discussion of Otter.

Key words: Automated theorem proving, competition, Otter, automated reasoning, equational deduction, paramodulation, resolution.

1. Introduction

Otter [17, 19] is an automated deduction system for first-order logic with equality. Two versions of Otter were entered in the CADE-13 Automated Theorem Proving System Competition, and the main purpose of this article is to give a detailed presentation of our entries. The first version (called Otter-304z) is essentially Otter 3.0.4 operating in its autonomous mode, and the second (called Otter-Wos) is a minor variation of Otter 3.0.4.

Because this article also serves as a general reference and overview of Otter, we also present some background, a summary of applications of Otter, and features of Otter that are not directly related to the competition.

1.1. Historical Background

Research in automated theorem proving at Argonne started in 1962. We have always placed great importance on implementing and testing our ideas, so many computer programs have been written. The first generation consisted mainly of two programs. PG1 (Program 1), designed by Dan Carson, George Robinson, and Wos in 1963, had the unit preference strategy [37]; experimentation with PG1 led to the set of support strategy [39] and demodulation [40]. The program RG1 [38], designed by George Robinson and Wos in 1967, had binary resolution, factoring, paramodulation, demodulation, and the set of support strategy.

The second generation, started by Ross Overbeek in 1970, was based on the NIUTP (Northern Illinois University Theorem Prover) series [10, 11],

* Supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

which evolved into AURA (AUtomated Reasoning Assistant) and its variants [28], with contributions from Brian Smith, Rusty Lusk, Bob Veroff, Steve Winker, and Wos. The NIUTP/AURA generation (written mostly in IBM assembly language, with some PL/1) included the first high-performance implementations of hyperresolution, demodulation, and paramodulation, and it introduced unit-resulting resolution and weighting. The result was the first *practical* set of programs in that their use led to answers to several open questions in equivalential calculus [41], ternary Boolean algebra [30] and semigroups [32].

The third generation of Argonne theorem provers, started in 1980 by Overbeek, Lusk, and McCune, consisted mostly of the LMA [9] (Logic Machine Architecture), a toolkit (written in Pascal) for building deduction systems, and ITP (Interactive Theorem Prover), constructed with LMA. The functionality of ITP was similar to that of AURA. The motivations for LMA and ITP were sound software engineering and portability. Several experimental theorem provers based on technology for compiling logic programs were also part of the third generation.

Otter is a member of the fourth generation. We started writing code for a new theorem prover in June 1987, and by December it was useful for our research on inference rules and search strategies. The program was later named Otter (Organized theorem-proving techniques for effective research). Other members of the fourth generation are ROO [8] (Radical Otter Optimization), by John Slaney, Lusk, and McCune, which is a parallel version of Otter; FDB [1] (Formula DataBase), by Overbeek and Ralph Butler, which is a unification and indexing toolkit; MACE [16] (Models And CounterExamples), by McCune, which searches for finite models; and EQP [18] (EQuational Prover), by McCune, a program for equational logic, which incorporates associative-commutative unification.

Aside from effective inference rules and strategies for proving theorems, speed and portability were the main considerations in building Otter, so C was used. The functionality of AURA and ITP, especially the inference and search methods, was quite useful in practice, so Otter retained most of it. However, low-level algorithms such as indexing techniques [13] were improved, resulting in sharp speedups over ITP. Major releases of Otter occurred at CADE-9 in May 1988 (version 0.9), in January 1989 (version 1), in March 1990 (version 2), and in January 1994 (version 3). The current version is 3.0.4, released in August 1995.

1.2. Applications of Otter

Otter has been applied to several areas of mathematics and logic, and many new results have been obtained with its use. Examples of such results are the existence of fixed point combinators in fragments of combinatory logic

[23, 33], logic calculi with condensed detachment [25], single axioms for group calculi [12, 15], single axioms for group theory and subvarieties [14, 24, 4, 2, 5], single axioms for ternary Boolean algebra [27], equational theorems about cubic curves [26], single axioms for lattice-like algebras [20], self-dual bases for groups and for lattices [21], implicational axioms for groups and Abelian groups [22], Robbins algebra [18, 31], Moufang loops [21, 6, 7], illiative combinatory logic [3], and proofs with particular properties [34, 35, 36].

2. Architecture

Otter reads an input file containing a set of formulas and some control information. The set of formulas represents the theory and denial of the conclusion (all proofs are by contradiction), and the control information consists of various switches and parameter settings for specifying the inference rules and search strategies. As Otter searches, it writes information (including the proof, if found) to an output file.

We now summarize how Otter works. See the manual [17] for details on the material in this section.

2.1. Logic

Otter applies to statements in classical first-order (unsorted) logic with equality. It accepts as input either clauses or quantified formulas. Quantified formulas (which may contain $\forall$, $\exists$, $\vee$, $\wedge$, $\neg$, $\rightarrow$, $\leftrightarrow$) are immediately transformed to clauses. All of Otter's inference rules and search algorithms operate on clauses.

2.2. Inference Rules

Otter's main inference rules are based on resolution or paramodulation. The resolution inference rules known to Otter are binary resolution, hyperresolution, negative hyperresolution, unit-resulting resolution, and linked unit-resulting resolution. Equational deduction is done with binary paramodulation and various forms of demodulation. (We sometimes classify demodulation as a rewriting rule rather than as an inference rule.) Factoring is not built into resolution and paramodulation; it is considered to be a separate inference rule.

Otter has one additional inference rule (called gL) for problems about cubic curves in algebraic geometry. It is a generalization rule that applies to equations and is derived from the property of cubic curves that for some kinds of statement P, if P holds at some point on a cubic curve, then P holds at every point on the curve. See [21] for details and applications.

2.3. Control

An Otter process can be viewed as a closure computation with redundancy control; that is, Otter attempts to compute the closure of a set of input statements under a set of inference rules, applying deletion rules (subsumption and demodulation) that typically preserve logical completeness of the inference system. The computation is driven by a loop.

2.3.1. *The Inference Loop*

Otter maintains three lists of clauses.

Usable. These clauses are available for application of inference rules. They actively participate in the search.

Sos. These clauses are waiting to participate in the search through application of inference rules. (Members of sos that are equations may be participating as demodulators.)

Demodulators. These are equations that are used as rewrite rules. Members of demodulators may occur in usable or sos as well.

The inference loop operates mainly on clauses in lists sos and usable:

```
While sos is not empty and no refutation has been found,
  1. Select a clause, the given-clause, in sos;
  2. Move given-clause from sos to usable;
  3. Infer and process new clauses using the inference rules
     in effect; each new clause must have the given-clause
     as one parent and members of usable as other parents;
     retained clauses are appended to sos;
End of while loop.
```

The processing of inferred clauses (Step 3 above) involves many optional retention tests and other procedures; the most important ones are listed here.

```
Given newly inferred clause C,
  1. Demodulate C (with members of list Demodulators);
  2. Orient equality literals in C;
  3. Discard C if weight(C) > the max-weight parameter;
  4. Discard C if it is subsumed by a member of usable or sos;
  5. Check if C conflicts with any clause in usable or sos;
  6. If C has passed the retention tests, then
    a. (optional) if C is an oriented equation, append it to
       Demodulators and rewrite all members of usable or sos;
    b. discard members of usable or sos subsumed by C;
    c. (optional) Factor C;
```

The inference loop can be seen as a simple implementation of the set of support strategy, because no inferences are drawn in which all of the participants are in the initial usable list. That is, the initial sos list is the initial set of support: all lines of deduction must start with a clause in the initial sos list.

The loop can also be used to drive a Knuth-Bendix completion procedure. If the initial sos list consists of the initial set of equations, all equations (input and derived) can be oriented into terminating demodulators, a restricted form of paramodulation is used, and the inference loop terminates because sos is empty, then the resulting usable list is a complete set of reductions for the theory. Our strategies for equational theorem proving evolved separately from the Knuth-Bendix completion, but in some cases they are similar.

Two term ordering methods are available for orienting equations and guaranteeing termination of demodulation. The lexicographic recursive path ordering (LRPO) is used in the autonomous modes, with default symbol precedence $constants \prec high\text{-}arity \prec \cdots \prec binary \prec unary$, and within arity, the lexicographic ASCII ordering. The ad hoc method (which does not always guarantee termination) orders terms by user-assigned weights.

Selection of the given clause in Step 1 of the inference loop is the most important aspect of the search process; it is the next path to explore. The default selection is the smallest clause in sos, which we call best-first search. Instead, the user may specify a breadth-first search, in which the first clause in sos is selected (sos operates as a queue). We have found that a combination of best-first and breadth-first search is frequently quite valuable, and one of Otter's parameters, the *pick-given-ratio*, can be used to specify a ratio: a value of n means that through n iterations of the inference loop, the smallest clause is selected, then in the next iteration the first clause is selected, and so on.

Discarding large clauses (Step 3 in the processing of newly derived clauses) interferes with completeness, but it is quite important in practice. If $max\text{-}weight{=}\infty$, clauses are retained and appended to sos at a much higher rate than they are removed as given clauses. As a result, most retained clauses never enter the search, and memory is wasted. If many of those clauses become demodulators, much time is spent (and wasted) using them to try to demodulate previous clauses. Thus, a good value for the *max-weight* parameter is important to achieving a well-behaved search [18]. We frequently make several initial searches, varying the *max-weight* parameter until a good value is found. When multiple searches are not done (in particular in Otter's autonomous mode) the parameter *control-memory* is used to dynamically adjust the *max-weight* parameter based on the amount of memory available.

2.3.2. *The Autonomous Mode*

Although Otter is not an interactive program, we typically use it in an interactive way. We run a search, examine the output, change the switches and parameters to adjust Otter's behavior, then start another search, and so on. Because this kind of multiple search is still an art form, we have not attempted to automate it. However, Otter has a fully automatic mode, called the autonomous mode, which is useful for inexperienced users, easy problems, situations in which Otter is called from another program, and comparison with other programs.

In the autonomous mode, Otter unconditionally sets several options and partitions clauses into the usable and sos lists. Otter then checks its input for the following properties: whether all clauses are propositional, whether all clauses are Horn, whether the equality relation is present, and, if equality is present, whether equality axioms are present.

The autonomous mode algorithm is the following.

```
set max-mem to 12 megabytes; set control-memory flag;
set pick-given-ratio parameter to 4; set process-input flag;
place positive clauses in sos list;
place nonpositive clauses in usable list;

if (all clauses are propositional) then
    set propositional flag; set hyperresolution flag;
else
    if (nonunits are present) then
        set hyperresolution flag;
        if (all clauses are Horn) then
            clear ordered-hyperresolution flag;
        else
            set factoring flag; set unit-deletion flag;
    if (equality is present) then
        set knuth-bendix flag;
        if (equality axioms are present) then
            clear paramodulation flags;
```

The *max-mem* parameter limits the amount of memory available for storage of clauses and related data structures, the *process-input* flag causes input clauses to be processed as if they were derived clauses, the *propositional* flag causes several optimizations particular to propositional clauses to be in effect, the *ordered-hyperresolution* flag (set by default) prevents satellites from resolving on nonmaximal literals, the *unit-deletion* flag causes each unit clause, say P, to be used as a rewrite rule $P = \text{TRUE}$ to simplify nonunit derived clauses, and the *knuth-bendix* flag causes several additional options to be set so that the search resembles Knuth-Bendix completion (see [17]).

The two competition entries named Otter-304z were run in the (ordinary) autonomous mode.

2.3.3. *The Auto-Wos Mode*

A different version of the autonomous mode, called *auto-wos*, was created specifically for the CADE-13 competition. The first reason for this was so that Otter could compete in the “monolithic” class, which does not allow different modules of code to be called based on properties of the input clauses. The second reason was so that we could use a different paramodulation strategy.

The *auto-wos* algorithm is the following.

```
1. set flags process-input, control-memory, hyperresolution,
   knuth-bendix, para-from-units-only, unit-deletion, factor;
2. clear flag index-for-back-demod-flag;
3. set pick-given-ratio parameter to 4;
4. if (every positive clause in sos is ground) then
       move all positive clauses to sos;
```

The property that makes *auto-wos* mode "monolithic" is that (nearly) all modules used for some type of input in ordinary autonomous mode are used for all types of problem in *auto-wos* mode. For example, clauses are indexed for paramodulation, and paramodulation is called even if no equality literals are present; and hyperresolution and factoring are called even if no nonunit clauses are present. The flag *index-for-back-demod* (default set if back demodulation is in effect) causes all terms in all clauses to be indexed so that they can be found if they can be rewritten by a newly derived demodulator. We clear this flag because indexing all terms is an expensive operation and very wasteful if no equality is present.

The second, and more practical, feature of *auto-wos* mode is that paramodulation from nonunit clauses is prohibited. This restriction is incomplete in general, but it is quite useful in practice.

The competition entry named Otter-Wos was run in the *auto-wos* mode.

2.4. Tuning for the Competition

For both Otter-304z and Otter-Wos, the value of the *max-mem* parameter was increased from 12 megabytes to 20 megabytes. This change affects performance because it can change the set of retained clauses; in particular, the behavior of the *control-memory* feature, which automatically adjusts the *max-weight* parameter, depends on the value of *max-mem*.

The *auto-wos* mode (thus Otter-Wos) was tuned with the 391 "Eligible Mixed" set of TPTP problems. First, we experimented with the initial set of support. The TPTP classifies each input clause as "axiom", "hypothesis", or "conjecture". (Otter-304z does not use this information.) To decide the initial sos list, we experimented with several rules of the form

$$\mathrm{sos} \leftarrow \mathrm{hypothesis} \cup \mathrm{conjecture};\ \ if\ P(\mathrm{sos}),\ then\ \mathrm{sos} \leftarrow \mathrm{sos} \cup f(\mathrm{axiom});$$

for various properties P and functions f. In the end, we used the rule with P = "all positive clauses are ground" and f = "positive clauses". Second, we experimented with the hot list strategy [36], which causes Otter to give special emphasis to key clauses; results indicated that our current hot list strategies are best used in the iterative-search mode rather than in autonomous modes, so the hot list strategy was not used for the competition.

3. Implementation

Otter is written in the C programming language, which was chosen for execution speed and portability. It contains about 35,000 lines of code (including comments). Clauses and terms are stored in shared data structures, which speed some of the indexing and inference operations and save memory. Specially designed and tuned indexing algorithms [13] are used to access terms and clauses for subsumption operations, application in inference rules, and application of demodulators.

Otter is designed to run in a UNIX-like environment, but versions (with several limitations) are available also for DOS computers and Macintoshes.

4. Performance in the Competition

Otter-304z in the unit equality competition. Otter placed first, proving 43 of 50 theorems in 2750 seconds. In second place was Waldmeister, with 37 proofs in 4730 seconds. Otter's performance is not surprising to us because application to real problems has driven its development and because we have focused on equational applications in the past few years.

Otter-304z in the mixed/open competition. Otter placed second in this category, with 28 proofs of 50 theorems in 7314 seconds. The winner was SPASS, with 32 proofs in 6244 seconds. Otter's performance was not surprising to us because most of these theorems are non-Horn, and many contain a mixture of equality and nonequality relations; we have worked on very few applications with these properties, and no special tuning was done for this area.

Otter-Wos in the mixed/monolithic competition. Otter placed second, with 32 proofs of 50 theorems in 6037 seconds. The winner was E-SETHEO, with 36 proofs in 5655 seconds. We can compare these with the mixed/open competition, because the same set of theorems was used. The positive effect of the paramodulation restriction more than offset the wasted indexing operations for theorems without equality. Both E-SETHEO and Otter did better than we expected when compared with the mixed/open results.

In the design of the competition, the open/monolithic distinction was made because it was thought that the open systems would have an unfair advantage. In this competition, at least, the monolithic systems did better, so perhaps the mixed/open and mixed/monolithic categories should be judged together. In that case, E-SETHEO wins, Otter-Wos places second, SPASS is very close behind in third, and Otter-304z is sixth.

Otter's performance in the competition clearly points to its strengths in unit equality reasoning and its weaknesses in non-Horn deduction and on problems with a mixture of equality and nonequality relations. A strength not evident from the competition is deduction in Horn theories, and a weakness not evident is propositional unsatisfiability.

5. Conclusion

One of the most important features of the results can be seen in the tables of runtimes for each system on each theorem [29]. Most of the theorems on which the top finishers failed were proved easily by at least one system. Each of the unit equality theorems was proved by at least one system, and 45 of 50 were proved in less than 20 seconds by at least one system. Of the mixed theorems, 46 were proved by at least one system, and 44 were proved in less than 17 seconds by at least one system. These results support our long-held position that the best method for automated deduction is a variety of methods.

References

1. R. Butler and R. Overbeek. Formula databases for high-performance resolution/paramodulation systems. *J. Automated Reasoning*, 12(2):139–156, 1994.
2. J. Hart and K. Kunen. Single axioms for odd exponent groups. *J. Automated Reasoning*, 14(3):383–412, 1995.
3. T. Jech. Otter experiments in a system of combinatory logic. *J. Automated Reasoning*, 14(3):413–426, 1995.
4. K. Kunen. Single axioms for groups. *J. Automated Reasoning*, 9(3):291–308, 1992.
5. K. Kunen. The shortest single axioms for groups of exponent 4. *Computers and Mathematics with Applications*, 29:1–12, 1995.
6. K. Kunen. Moufang quasigroups. *J. Algebra*, 83:231–234, 1996.
7. K. Kunen. Quasigroups, loops, and associative laws. *J. Algebra*, 1996. To appear.
8. E. Lusk and W. McCune. Experiments with ROO, a parallel automated deduction system. In B. Fronhöfer and G. Wrightson, editors, *Parallelization in Inference Systems, LNAI, Vol. 590*, pages 139–162, Berlin, 1992. Springer-Verlag.
9. E. Lusk, W. McCune, and R. Overbeek. Logic Machine Architecture: Kernel functions. In D. Loveland, editor, *Proceedings of CADE-6, LNCS, Vol. 138*, pages 70–84, Berlin, 1982. Springer-Verlag.
10. J. McCharen, R. Overbeek, and L. Wos. Complexity and related enhancements for automated theorem-proving programs. *Comp. Math. with Applications*, 2:1–16, 1976.
11. J. McCharen, R. Overbeek, and L. Wos. Problems and experiments for and with automated theorem-proving programs. *IEEE Trans. on Computers*, C-25(8):773–782, August 1976.
12. W. McCune. Automated discovery of new axiomatizations of the left group and right group calculi. *J. Automated Reasoning*, 9(1):1–24, 1992.
13. W. McCune. Experiments with discrimination tree indexing and path indexing for term retrieval. *J. Automated Reasoning*, 9(2):147–167, 1992. Invited paper.
14. W. McCune. Single axioms for groups and Abelian groups with various operations. *J. Automated Reasoning*, 10(1):1–13, 1993.
15. W. McCune. Single axioms for the left group and right group calculi. *Notre Dame J. Formal Logic*, 34(1):132–139, 1993.

16. W. McCune. A Davis-Putnam program and its application to finite first-order model search: Quasigroup existence problems. Tech. Report ANL/MCS-TM-194, Argonne National Laboratory, Argonne, IL, May 1994.
17. W. McCune. OTTER 3.0 Reference Manual and Guide. Tech. Report ANL-94/6, Argonne National Laboratory, Argonne, IL, 1994.
18. W. McCune. 33 basic test problems: A practical evaluation of some paramodulation strategies. Preprint ANL/MCS-P618-1096, Argonne National Laboratory, 1996.
19. W. McCune. Otter. http://www.mcs.anl.gov/home/mccune/ar/otter/, 1996.
20. W. McCune and R. Padmanabhan. Single identities for lattice theory and weakly associative lattices. Preprint MCS-P493-0395, Argonne National Laboratory, 1995.
21. W. McCune and R. Padmanabhan. *Automated Deduction in Equational Logic and Cubic Curves*, Vol. 1095 of *LNAI*. Springer-Verlag, Berlin, 1996.
22. W. McCune and A. D. Sands. Computer and human reasoning: Single implicative axioms for groups and for Abelian groups. *American Math. Monthly*, December 1996. To appear.
23. W. McCune and L. Wos. The absence and the presence of fixed point combinators. *Theoretical Computer Science*, 87:221–228, 1991.
24. W. McCune and L. Wos. Application of automated deduction to the search for single axioms for exponent groups. In A. Voronkov, editor, *Logic Programming and Automated Reasoning, LNAI, Vol. 624*, pages 131–136, Berlin, 1992. Springer-Verlag.
25. W. McCune and L. Wos. Experiments in automated deduction with condensed detachment. In D. Kapur, editor, *Proceedings of CADE-11, LNAI, Vol. 607*, pages 209–223, Berlin, 1992. Springer-Verlag.
26. R. Padmanabhan and W. McCune. Automated reasoning about cubic curves. *Computers and Mathematics with Applications*, 29(2):17–26, 1995.
27. R. Padmanabhan and W. McCune. Single identities for ternary Boolean algebras. *Computers and Mathematics with Applications*, 29(2):13–16, 1995.
28. B. Smith. Reference manual for the environmental theorem prover: An incarnation of AURA. Tech. Report ANL-88-2, Argonne National Laboratory, 1988.
29. G. Sutcliffe and C. Suttner. The Results of the CADE-13 ATP System Competition. *J. Automated Reasoning*. This issue.
30. S. Winker. Generation and verification of finite models and counterexamples using an automated theorem prover answering two open questions. *J. ACM*, 29:273–284, 1982.
31. S. Winker. Robbins algebra: Conditions that make a near-Boolean algebra Boolean. *J. Automated Reasoning*, 6(4):465–489, 1990.
32. S. Winker, L. Wos, and E. Lusk. Semigroups, antiautomorphisms, and involutions: A computer solution to an open problem, I. *Math. of Comp.*, 37:533–545, 1981.
33. L. Wos. The kernel strategy and its use for the study of combinatory logic. *J. Automated Reasoning*, 10(3):287–343, 1993.
34. L. Wos. Searching for circles of pure proofs. *J. Automated Reasoning*, 15(3):279–315, 1995.
35. L. Wos. Otter and the Moufang identity problem. *J. Automated Reasoning*, 17(2):215–257, 1996.
36. L. Wos. The power of combining resonance with heat. *J. Automated Reasoning*, 17(1):23–81, 1996.
37. L. Wos, D. Carson, and G. Robinson. The unit preference strategy in theorem proving. In *AFIPS Proceedings 26*, pages 615–621. Spartan Books, 1964.
38. L. Wos and G. Robinson. Paramodulation and set of support. In *IRIA Symposium on Automatic Demonstration*, 276–310. Springer-Verlag, 1968.
39. L. Wos, G. Robinson, and D. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *J. ACM*, 12(4):536–541, 1965.
40. L. Wos, G. Robinson, D. Carson, and L. Shalla. The concept of demodulation in theorem proving. *J. ACM*, 14(4):698–709, 1967.

41. L. Wos, S. Winker, B. Smith, R. Veroff, and L. Henschen. A new use of an automated reasoning assistant: Open questions in equivalential calculus and the study of infinite domains. *Artificial Intelligence*, 22:303–356, 1984.