Algorithms, datastructures and other issues in efficient automated deduction

Andrei Voronkov

University of Manchester
voronkov@cs.man.ac.uk

Abstract. Algorithms and datastructures form the kernel of any efficient theorem prover. In this abstract we discuss research on algorithms and datastructures for efficient theorem proving based on our experience with the theorem prover Vampire. We also briefly overview other works related to algorithms and datastructures, and to efficient theorem proving in general.

1 Introduction

To implement an efficient automatic theorem prover, one has to put together at least three ingredients: good theory, efficient algorithms and datastructures, and clever heuristics.[1] In the recent years, a considerable progress has been made in the theory of resolution-based systems. This theory is build upon completeness theorems for resolution calculi with notions of redundant inferences and redundant derivations [6, 63]. The theory is well-understood, but a good theory alone is not enough to implement an efficient prover.

The progress in theory can be characterized by the following observation: a prover based on the theory known in 1970 would now be hopelessly inefficient (unrestricted resolution and paramodulation, use of function reflexivity axioms). A prover based on the theory known in 1980 would outperform a 1970 prover by several orders of magnitude on difficult problems (mainly because of the use of simplification orderings). Compared to a 1980 prover, a prover based on the theory of 1990 would be several orders of magnitude faster on many difficult problems and moreover more flexible due to some new theoretical results (the general theory of redundancy, selection functions).

Since the first paper on resolution theorem proving [77], many theorem provers were developed by various researchers and groups. The most consistent implementation efforts were undertaken at Argonne National Laboratory; they resulted in the development of a series of systems, including Logic Machine Architecture [44, 43] and Otter [51]. The nature of research at Argonne was formulated in [42]: *controlling redundancy in large search spaces.* Recently, several new efficient first-order theorem provers emerged, including the resolution-based provers Spass [96, 95], E [78, 79], Gandalf [89], Vampire [70, 75], Bliksem [16], and

[1] See [56] for a similar observation.

SCOTT [82]; the model-elimination based prover Setheo [40], and the equational provers Waldmeister [34] and Fiesta (see [59]).

These provers are extremely efficient. For example the Steamroller problem discussed some time ago in the literature [87] is now a trivial problem for all of these systems. Several open problems solved by Otter in 1993 [50] can now be routinely solved by some of these provers. However, there are many first-order problems coming from various applications, which are still beyond the capabilities of the state-of-the-art provers. Since first-order logic is undecidable, it is unreasonable to expect new systems to efficiently solve all of them in the near future. However, if we can increase performance of the modern provers by several orders of magnitude for a large number of such problems, many of these problems will be routinely solved, thus saving time for application developers.

In our opinion, such a drastic increase in efficiency in the near future will be mainly based on the development of new algorithms and datastructures, and understanding how the theory developed so far can be efficiently implemented on top of the existing architectures of theorem provers.

2 How do efficient datastructures influence performance

Before implementing Vampire, I implemented several more or less functioning theorem provers in REFAL [91], LISP, and Prolog. Efficiency was not an aim. I was interested in comparing their behavior when different methods were implemented (for example, [92] implemented the inverse method).

In 1993 Dominique Bolignano invited me to visit his research group at Bull, near Paris, for about two months. As part of my visit to Bull I gave a talk on theorem proving by the inverse method and seemed to convince the audience that the inverse method is worth trying. However, my old implementation of the inverse method was not maintained, while the new implementation did not exist. Then I decided to implement an *efficient* prover. The development of the new prover Vampire has changed considerably my perception of automated theorem proving. The first surprise came when I compared the behaviour of my newly implemented prover with that of Otter. When both provers used hyperresolution for solving the same problems, in the first few seconds of proof-search their performance was comparable. But after a few seconds Otter kept up making inferences at the same pace, while Vampire seemed to enter a deadlock. A simple profiling has shown that nearly all of the running time was spent on forward subsumption. So my first exercise in efficient theorem proving was to implement efficiently forward subsumption.[2] Search for literature on efficient subsumption did not help very much: although I could find a paper on efficient subsumption [29], the algorithm of this paper was not very useful when every newly generated clause had to be checked for subsumption with over 10^5 clauses in the current search space.

Search for an efficient subsumption algorithm resulted in discovery of the code tree indexing technique described in [93] and improved in [72]. After implemen-

[2] The value of subsumption was also observed by the Argonne group [97].

tation of code trees for forward subsumption, the performance of Vampire when running hyperresolution was comparable to that of Otter. From 1998 Vampire is implemented and maintained by Alexandre Riazanov. The rest of this paper describes some aspects of what Vampire has taught me about efficient theorem proving.

3 Saturation-based resolution theorem proving

All of the modern resolution-based first-order theorem provers implement some variant of the *given clause algorithm* [51, 42]. Several variants of this algorithm are overviewed in [71, 95]. One of the versions, roughly corresponding to those used in Otter and Fiesta, is shown in Figure 1 (taken from [71]). It is parametrized by several procedures explained below:

- *select* is the clause selection function. It decides which clause should be selected for activation.
- *infer* is the function that performs inferences between the current clause *current* and the set of active clauses *active*. This function returns the set of clauses obtained by all such possible inferences. This function varies from system to system. Usually, *infer* applies inferences in some complete inference system of resolution with paramodulation.
- *simplify*(*set*, *by*) is a procedure that performs simplification. It deletes redundant clauses from *set* and simplifies some clauses in *set* using the clauses in *by*. To preserve completeness, the simplified clauses are always moved to *passive*. Typically, deleted clauses include tautologies and those clauses subsumed by clauses in *by*. A typical example of simplification is rewriting by unit equalities in *by*.
- Likewise, *inner_simplify* simplifies clauses in *new* using other clauses in *new*.

When we simplify *new* using the clauses in $active \cup passive$, we speak of *forward simplification*; when we simply *active* and *passive* using the clauses in *new*, we speak of *backward simplification*. The name *given clause algorithm* is due to the fact that the clause *current* is called the given clause in Otter's terminology.

It is instructive to explain several problems facing implementors of the given clause algorithm, to show the gap between the theory and practice of resolution-based theorem proving. In theory, it is not hard to prove that this algorithm is complete, provided that the underlying logical calculus is complete. In practice, at least two problems arise. Both problems are due to the fast growth of the search space. The first problem is how to select the right current clause in a huge set of passive clauses. This problem can be solved only by a large number of experiments over a large collection of problems. The most common approach is to maintain several priority queues on passive clauses, and pick up clauses from this priority queues. A very popular technique is to maintain two queues in which the clauses are prioritized by their weight and age, respectively. The clauses are

```
input: init : set of clauses ;
var active, passive, new : sets of clauses ;
var current : clause ;
active := ∅ ;
passive := init ;
while passive ≠ ∅ do
  current := select(passive) ;
  passive := passive − {current} ;
  active := active ∪ {current} ;
  new := infer(current, active) ;
  if goal_found(new) then return provable ;
  inner_simplify(new) ;
  simplify(new, active ∪ passive) ;
  if goal_found(new) then return provable ;
  simplify(active, new) ;
  simplify(passive, new) ;
  if goal_found(active ∪ passive) then return provable ;
  passive := passive ∪ new
od ;
return unprovable
```

Fig. 1. A Given Clause Algorithm

picked from the queues using the so-called *age-weight ratio* (also called the pick-given ratio); for example if the ratio is 1:5, than out of each 6 picked clauses, 1 will be selected as the oldest clause, and 5 as the lightest clauses. The prover E maintains more than two queues. Even this simple clause selection scheme creates many problems, when clauses must be deleted from the search space (for example, when the available memory is exhausted), since every clause may be kept in several priority queues. The best clause selection strategies are not yet well-understood.

Another serious problem is caused by the proliferation of passive clauses. A large number of passive clauses results in deterioration of the proof-search speed and huge memory consumption. To illustrate this, we provide statistics on an unsuccessful run of Vampire with the time limit of 1 minute on the TPTP problem ANA003-1. During this run, 261,573 clauses were generated. The overall number of active clauses was 1,967, the overall number of passive clauses 236,389. To cope with the problems of passive clauses, several solutions have been proposed. For example, to impose a weight limit on clauses and increase the weight limit, if the prover unsuccessfully terminates with the current weight limit.

The most radical solution to the passive clauses problem was originally implemented in DISCOUNT [2] and is now used in Waldmeister and E (and can be used as an option in Spass and Vampire). These provers use a different main loop, in which passive clauses do not participate in backward simplifications until they are selected as the current clause. In fact, these provers do not store the passive clauses at all, but only store information about the inference rule used to obtain a passive clause. This implementation scheme requires double forward simplification, but is believed to be space- and time-efficient. The provers implementing the DISCOUNT loop are essentially based on the principle: *process active clauses efficiently.* The price to pay is that some very useful simplifying inferences involving a passive clause can be considerably delayed until the clause has been selected as current. The main principle of Vampire with respect to the main loop is: *do simplifications eagerly and non-simplifying inferences lazily.* Therefore, Vampire's default option is the Otter main loop. However, to work with the large among of passive clauses efficiently, the so-called *limited resource strategy* was invented [71]. This strategy is applicable when the time limit on solving a problem is specified in advanced. Vampire tries to estimate which clauses cannot be processed by the end of the time limit at all, and discards such clauses as useless. Experiments reported in [71] show high efficiency of the limited resource strategy as compared to other approaches.

4 Term indexing

To be able to process hundreds of thousands clauses in less than a minute, all most important operations should ideally be implemented on the set-at-a-time basis. For example, subsumption is NP-complete, and checking every newly generated clause for subsumption against several hundred thousand of kept clauses sequentially is hopelessly slow. There is a growing number of papers on *term indexing* (see [32, 81] for an overview): an approach that allows one to implement efficiently expensive massive operations on terms and clauses, for example, subsumption of one clause by a large database of clauses.

The problem of term indexing can be formulated abstractly as follows (see [81]). Given a set L of *indexed terms* (or clauses), a binary relation R over terms (called the *retrieval condition*) and a term t (called the *query term*), identify the subset M of L that consists of the terms l such that $R(l, t)$ holds. Terms in M will be called the *candidate terms.* Typical retrieval conditions used in first-order theorem proving are matching, generalization, unifiability, subsumption, syntactic equality, variance etc. Such a retrieval of candidate terms in theorem proving is interleaved with insertion of terms to L, and deletion of them from L.

In order to support rapid retrieval of candidate terms, we need to process the indexed set into a data structure called the *index.* Indexing data structures are well-known to be crucial for the efficiency of the current state-of-the-art theorem provers. Term indexing is also used in logic and functional programming languages implementation, but indexing in theorem provers has several distinctive features:

1. Indexes in theorem provers frequently store 10^5–10^6 complex terms, unlike a typically small number of shallow terms in functional and logic programs.
2. In logic or functional language implementation the index is usually constructed during compilation. On the contrary, indexes in theorem proving are highly dynamic, since terms are frequently inserted in and deleted from

indexes. *Index maintenance* operations start with an index for an initial set of terms L, and incrementally construct an index for another set L' that is obtained by insertion or deletion of terms to or from L.

3. In many applications it is desirable for several retrieval operations to work on the same index structure in order to share maintenance overhead and memory consumption.

Therefore, along the last two decades a significant number of results on new indexing techniques for theorem proving have been published and successfully applied in different provers [81, 32, 33, 69, 84, 64, 49, 13, 30, 31, 93, 76, 72, 26, 57].

For every retrieval condition used in theorem provers, it is not necessary to retrieve exactly all candidates satisfying this retrieval condition. If the retrieval condition is used for inferences (as, e.g., unification is used for inferences by resolution and paramodulation), then it is enough to retrieve a superset of candidates, and then check the retrieval condition for every member of this superset. If the retrieval condition is used for simplifications (as, e.g., retrieval of generalizations is used for simplification by unit equalities or forward subsumption by unit clauses), then it is enough to retrieve a subset of candidates only, since simplifications only reduce the search space, but do not influence completeness. If a particular term indexing technique retrieves exactly all candidates, then this technique is said to perform *perfect filtering*. It is always an issue of debate in automated deduction, whether perfect filtering should be implemented for a particular retrieval condition. For example, [90] advocates the use of imperfect filtering for subsumption.

Term indexing is one of the main research directions for Vampire. We believe that perfect filtering is desirable for all operations for which indexing is required at all. As a consequence, Vampire stores a large number of indexes for various operations: partially adaptive code trees [93, 72] for forward subsumption, subsumption resolution, and variance check; code trees with precompiled ordering constraint for forward rewriting (also called demodulation) by unit equalities; path indexing with compiled database joins implemented using skip lists for backward subsumption and backward demodulation by unit equalities [73]; tries for unification used to implement resolution and paramodulation; and another kind of tries for storing perfectly shared terms. We learned that it pays off to spend time for implementing new indexing techniques: for nearly every retrieval condition there are problems for which this retrieval condition contributes to the running time considerably. We believe that term indexing will be in the heart of theorem proving research in the future.

It is due to term indexing that the modern provers can quickly solve problems which require search in a space of several million complex clauses. But it is also due to term indexing that implementation of new features on top of existing architectures requires non-trivial efforts and invention of completely new techniques. Every new feature brings in a tradeoff between the time/space gained by the use of the feature on one hand, and the time spent on checking applicability of this feature and space needed to implement the feature efficiently. In addition, some promising features require a non-trivial implementation.

As a result, some enhancements of theorem provers well-known in theory have never been implemented. A typical example is the basic strategy [17, 7, 60]. Although in theory basic superposition saves from performing some redundant inferences, in practice implementation of basic superposition requires considerable changes in *all* algorithms and datastructures. Therefore, all indexing techniques and all algorithms for term retrieval must be adapted to the basic strategy. As a consequence, the basic strategy has never been fully implemented.

5 Building-in equational theories

We are in search of techniques that can speed up the provers by several orders of magnitude. Built-in equational theories is one of such techniques. Many problems coming from applications are theorems about structures axiomatized by a set of equations. The most common equational theory is AC: the theory containing associativity and commutativity axioms for a function symbol.

Although the idea of built-in equational theories have been around at least from 1972 [68], the first implementation of AC was undertaken more than 20 years after in the EQP equational theorem prover [48]. This implementation resulted in probably the most celebrated event in the automated deduction community: the automatic solution of the Robbins problem [47] by EQP.

Although EQP ran for 8 days to obtain the proof, the proof-search for the Robbins problem was remarkably small. The total number of equations processed during the proof was less than 50,000. Modern provers often process such a number of clauses in a matter of a few seconds. This shows that the Robbins problem may in fact be not very difficult for a prover in which AC is implemented efficiently, including term indexing modulo AC.

So far research on built-in equational theories was mainly built around equational unification (see e.g., [4] for a recent overview). The special case of AC-unification was discussed in a number of papers, dating back to [85, 86], but only recently efficient algorithms have been described [1]. The experience with the non-AC theorem proving shows that efficient matching is more important than efficient unification,[3] but there is essentially no literature on efficient AC-matching. Moreover, there are essentially no publications on term indexing in presence of built-in theories. The only paper we know about discusses a special case of indexing for AC-matching of linear terms [5]. Vampire has commutativity built-in in some term indexes and retrieval algorithms (see e.g., [73]).

We conclude that efficient algorithms and datastructures for built-in equational theories should become a major topic for research in automated deduction.

6 Cheap substitutes

When the price to pay for implementing a particular algorithm is too high, cheap substitutes can be used. This is especially true for the cases when com-

[3] Indeed, only 0.1% of the EQP running time on the Robbins problem was spent for AC-unification.

pleteness is not an issue, for example when implementing simplification rules. There are several examples of successful implementation of "cheap substitutes" for expensive operations. For example, Waldmeister implements specialized algorithms for AC-completion [3]. In resolution theorem proving it is desirable to quickly check whether one can apply a resolvent of two clauses which subsumes one of these clauses. Such an application simplifies the search space, since the resolvent will then replace one of its parents. However, no efficient algorithm is known for finding simplified resolvents. There is an incomplete but relatively cheap operation called *subsumption resolution* (see e.g. [6]) which can be implemented essentially without extra overhead, provided that efficient subsumption is implemented. Vampire implements subsumption resolution using the indexes for subsumption, but slightly modified algorithms. Another example of a cheap substitute is the implementation of splitting in Vampire [74] in the form of *splitting without backtracking*. Though it does not have full power of the splitting rule as implemented, e.g., in Spass [95], splitting without backtracking can be implemented without radical changes in the architecture of a theorem prover.

7 Constraints

The use of symbolic constraints in automated deduction gives stronger notions of redundancy than those formulated without constraints [62]. In addition, it is known that symbolic constraints may give a compact representation of large search spaces, for example when constraints modulo AC are used [61] to encode a doubly exponential number of AC-unifiers [36].

It is conjectured in [56] that deduction with symbolic constraints will be a major research topic in efficient automated deduction and can result in a major breakthrough in efficiency. However, ten years after the first publication on deduction with symbolic constraint [37] there is still no implementation. The problem is that algorithms and datastructures for solving symbolic constraints are not yet developed, except for some particular operations. The most advanced results on symbolic constraints are in the area of solving constraints over simplification orderings. The first algorithm for solving RPO ordering constraints were described in [14, 35], followed by a number of results on solving RPO ordering constraints [55, 15, 54], but only recently an efficient algorithm was designed [58]. In the case of Knuth-Bendix ordering constraints, the decidability of constraint solving was proved in 1999 [38, 39], but no simple algorithms have yet been described.

As shown in [94], good algorithms for solving ordering constraints are not enough for implementing constraint-based deduction. Efficient algorithms and datastructures should be developed also for constraint simplification and approximation. Moreover, first-order theories of ordering constraints are to be better understood.

8 Comparison of algorithms and datastructures

One of the main problems in research on algorithms and datastructures for automated deduction is that the algorithms are performed on terms and clauses, i.e., tree-like structures. It was observed that the worst-case complexity results for such structures can be inadequate in practice. For example, most provers use Robinson's unification algorithm [77] having worst-case exponential complexity instead of efficient linear algorithms [65, 46], since overhead on maintaining datastructures for these algorithms does not pay off in practice, where the exponential behavior of Robinson's algorithm does not show up. Subsumption is NP-complete [9, 27], but the modern provers often make subsumption-checks of 10^5 clauses against a database of 10^4 clauses in a few seconds.

A practical approach to comparing implementation of algorithms used in first-order automated deduction was recently undertaken in [57]. The essence of this approach is that the implementation techniques are compared on benchmarks taken from runs of theorem provers on real problems. It is likely that the methodology of [57] will be used for comparison of other important algorithms used in automated deduction.

9 Other aspects of efficient theorem proving

In this section we briefly overview aspects of efficient theorem proving other than those directly related to algorithms and datastructures.

9.1 Non-resolution theorem proving

Resolution is not the only automated reasoning method used in first-order automated deduction. Model elimination [41] implemented in SETHEO [40] often performs very well on problems difficult for resolution-based provers. A recent adaptation of propositional splitting to the full first-order case [8] seems to be promising. However, non-resolution based procedures often have difficulties with equality and other built-in theories, as witnessed by the results overviewed in [21]. They have been several proposals on combination of paramodulation-based reasoning and tableau-based reasoning [18, 19, 52], but none of them was implemented.

9.2 Parallel and agent-based reasoning

Parallel computing is becoming cheaper. Networks of computers are now readily available. This makes parallel and agent-based theorem proving attractive. There were early projects aiming at parallelizing theorem provers, both in the context of model elimination [80] and resolution [45]. Distributed theorem proving was considered e.g., in [53]. However, parallelization of theorem proving, especially resolution-based, requires further investigation and experiments.

Theorem proving in which provers are considered as communicating agents was considered in [22]. It was shown that one can obtain a considerable speedup by running several provers in parallel and making them communicate by sending each other heuristically selected derived clauses.

There are several projects which put together several provers in different forms, for example, proving a common interface for running them over the Web. Examples are MathWeb [24, 23], MBASE [25], SystemOnTPTP [88]. However, the emphasis in such systems was so far on providing a graphical user interface or interactive theorem proving but not so much on efficient automated deduction.

9.3 Other research directions

There are many other aspects of efficient theorem proving not considered in this paper. We will mention some of them very briefly.

In the future modern first-order theorem prover will be tightly integrated with other systems for automated reasoning, for example inductive theorem provers (see [12] for an overview), and proof assistants such as Isabelle [66] and HOL [28], and maybe model checkers. It is likely that first-order provers integrated in such systems will also partially implement proof-search specific to these systems, for example restricted forms of induction, proofs about inductively defined types, or even restricted forms of higher-order theorem proving. This will require new lines of research, for example

- saturation-based higher-order theorem proving [10, 11];
- intelligent work with definitions [67, 20];
- built-in data types;
- recognition of irrelevant axioms;
- propositional reasoning;
- built-in theories (not only equational) [83];
- reasoning with non-standard quantifiers, for example "there exists at most n";
- finite domain reasoning.

References

1. F. Ajili and E. Contejean. Avoiding slack variables in the solving of linear diophantine equations and inequations. *Theoretical Computer Science*, 173(1):183–208, 1997.
2. J. Avenhaus, J. Denzinger, and M. Fuchs. DISCOUNT: a system for distributed equational deduction. In J. Hsiang, editor, *Proceedings of the 6th International Conference on Rewriting Techniques and Applications (RTA-95)*, volume 914 of *Lecture Notes in Computer Science*, pages 397–402, Kaiserslautern, 1995.
3. J. Avenhaus and B. Löchner. Cce: Testing ground joinability. In R. Gore, A. Leitsch, and T. Nipkow, editors, *IJCAR 2001*, 2001. this volume.
4. F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 8, pages 445–532. Elsevier Science, 2001.

5. L. Bachmair, T. Chen, and I.V. Ramakrishnan. Associative-commutative discrimination nets. In M.-C. Gaudel and J.-P. Jouannaud, editors, *Proceedings of the 4th International Joint Conference on Theory and Practice of Software Development (TAPSOFT)*, volume 668 of *Lecture Notes in Computer Science*, pages 61–74, Orsay, France, April 1993. Springer Verlag.
6. L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.
7. L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation and superposition. In D. Kapur, editor, *11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 462–476, Saratoga Springs, NY, USA, June 1992. Springer Verlag.
8. P. Baumgartner. FDPLL — a first-order Davis-Putnam-Logemann-Loveland procedure. In D. McAllester, editor, *17th International Conference on Automated Deduction (CADE-17)*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 200–219, Pittsburgh, 2000. Springer Verlag.
9. L.D. Baxter. The NP-completeness of subsumption. Unpublished manuscript, 1977.
10. C. Benzmüller and M. Kohlhase. Extensional higher-order resolution. In C.Kirchner and H. Kirchner, editors, *Automated Deduction — CADE-15. 15th International Conference on Automated Deduction*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 56–71, Lindau, Germany, 1998. Springer Verlag.
11. C. Benzmüller and M. Kohlhase. LEO—a higher-order theorem prover. In C.Kirchner and H. Kirchner, editors, *Automated Deduction — CADE-15. 15th International Conference on Automated Deduction*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 139–144, Lindau, Germany, 1998. Springer Verlag.
12. A. Bundy. The automation of proof by mathematical induction. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 13, pages 845–911. Elsevier Science, 2001.
13. J. Christian. Flatterms, discrimination nets, and fast term rewriting. *Journal of Automated Reasoning*, 10(1):95–113, February 1993.
14. H. Comon. Solving symbolic ordering constraints. *International Journal of Foundations of Computer Science*, 1(4):387–411, 1990.
15. H. Comon and R. Treinen. Ordering constraints on trees. In S. Tison, editor, *Trees in Algebra and Programming: CAAP'94*, volume 787 of *Lecture Notes in Computer Science*, pages 1–14. Springer Verlag, 1994.
16. H. de Nivelle. *Bliksem 1.10 User's Manual.* MPI für Informatik, Saarbrücken, 2000.
17. A. Degtyarev. On the forms of inference in calculi with equality and paramodulation. In Yu.V. Kapitonova, editor, *Automation of Research in Mathematics*, pages 14–26. Institute of Cybernetics, Kiev, Kiev, 1982.
18. A. Degtyarev and A. Voronkov. Equality elimination for semantic tableaux. UPMAIL Technical Report 90, Uppsala University, Computing Science Department, December 1994.
19. A. Degtyarev and A. Voronkov. Equality elimination for the tableau method. In J. Calmet and C. Limongelli, editors, *Design and Implementation of Symbolic Computation Systems. International Symposium, DISCO'96*, volume 1128 of *Lecture Notes in Computer Science*, pages 46–60, Karlsruhe, Germany, September 1996.

20. A. Degtyarev and A. Voronkov. Stratified resolution. In D. McAllester, editor, *17th International Conference on Automated Deduction (CADE-17)*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 365–384, Pittsburgh, 2000. Springer Verlag.
21. A. Degtyarev and A. Voronkov. Equality reasoning in sequent-based calculi. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 10, pages 609–704. Elsevier Science, 2001.
22. J. Denzinger and D. Fuchs. Cooperation of heterogeneous provers. In T. Dean, editor, *Proc. of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, volume 1, pages 10–15, Stockholm, 1999.
23. A. Franke, S.M. Hess, C.G. Jung, M. Kohlhase, and V. Sorge. Agent-oriented integration of distributed mathematical services. *Journal of Universal Computer Science*, 5(3):156–187, 1999.
24. A. Franke and M. Kohlhase. Mathweb, an agent-based communication layer for distributed automated theorem proving. In H. Ganzinger, editor, *Automated Deduction—CADE-16. 16th International Conference on Automated Deduction*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 217–221, Trento, Italy, July 1999.
25. A. Franke and M. Kohlhase. Mbase, an open mathematical knowledge base. In D. McAllester, editor, *17th International Conference on Automated Deduction (CADE-17)*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 455–459, Pittsburgh, 2000. Springer Verlag.
26. H. Ganzinger, R. Nieuwenhuis, and P. Nivela. Context trees. In R. Gore, A. Leitsch, and T. Nipkow, editors, *IJCAR 2001*, 2001. this volume.
27. M.R. Garey and D.S. Johnson. *Computers and Intractability.* Freeman, San Francisco, 1979.
28. M. Gordon and T. Melham. *Introduction to HOL.* Cambridge University Press, 1993.
29. G. Gottlob and A. Leitsch. On the efficiency of subsumption algorithms. *Journal of the Association for Computing Machinery*, 32(2):280–295, April 1987.
30. P. Graf. Extended path-indexing. In A. Bundy, editor, *Automated Deduction — CADE-12. 12th International Conference on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 514–528, Nancy, France, June/July 1994.
31. P. Graf. Substitution tree indexing. In J. Hsiang, editor, *Proceedings of the 6th International Conference on Rewriting Techniques and Applications (RTA-95)*, volume 914 of *Lecture Notes in Computer Science*, pages 117–131, Kaiserslautern, 1995.
32. P. Graf. *Term Indexing*, volume 1053 of *Lecture Notes in Computer Science.* Springer Verlag, 1996.
33. C. Hewitt. *Description and theoretical analysis of Planner: a language for proving theorems and manipulating models in a robot.* PhD thesis, Department of Mathematics, MIT, Cambridge, Mass., January 1971.
34. T. Hillenbrand, A. Buch, R. Vogt, and B. Löchner. Waldmeister: High-performance equational deduction. *Journal of Automated Reasoning*, 18(2):265–270, 1997.
35. J.-P. Jouannaud and M. Okada. Satisfiability of systems of ordinal notations with the subterm property is decidable. In J.L. Albert, B. Monien, and M. Rodríguez-Artalejo, editors, *Automata, Languages and Programming, 18th International Colloquium, ICALP'91*, volume 510 of *Lecture Notes in Computer Science*, pages 455–468, Madrid, Spain, 1991. Springer Verlag.

36. D. Kapur and P. Narendran. Double-exponential complexity of computing a complete set of AC-unifiers. In *Proc. IEEE Conference on Logic in Computer Science (LICS)*. IEEE Computer Society Press, 1992.
37. C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue Francaise d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on automated deduction.
38. K. Korovin and A. Voronkov. A decision procedure for the existential theory of term algebras with the Knuth-Bendix ordering. In *Proc. 15th Annual IEEE Symp. on Logic in Computer Science*, pages 291–302, Santa Barbara, California, June 2000.
39. K. Korovin and A. Voronkov. Knuth-Bendix constraint solving is NP-complete. Preprint CSPP-8, Department of Computer Science, University of Manchester, November 2000. to appear in ICALP 2001.
40. R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A high-performance theorem prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.
41. D.W. Loveland. Mechanical theorem proving by model elimination. *Journal of the Association for Computing Machinery*, 15:236–251, 1968.
42. E.L. Lusk. Controlling redundancy in large search spaces: Argonne-style theorem proving through the years. In A. Voronkov, editor, *Logic Programming and Automated Reasoning. International Conference LPAR'92.*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 96–106, St.Petersburg, Russia, July 1992.
43. E.L. Lusk, W. McCune, and R.A. Overbeek. Logic machine architecture: Inference mechanisms. In D.W. Loveland, editor, *Proceedings 6th Conference on Automated Deduction (CADE '82)*, number 138 in Lecture Notes in Computer Science, pages 85–108, New York, 1982. Springer Verlag.
44. E.L. Lusk, W. McCune, and R.A. Overbeek. Logic machine architecture: Kernel functions. In D.W. Loveland, editor, *Proceedings 6th Conference on Automated Deduction (CADE '82)*, number 138 in Lecture Notes in Computer Science, pages 70–84, New York, 1982. Springer Verlag.
45. E.W. Lusk, W.W. McCune, and J. Slaney. ROO: a parallel theorem prover. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 731–734, Saratoga Springs, NY, USA, June 1992. Springer Verlag.
46. A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.
47. W. McCune. Solution of the robbins problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.
48. W. McCune. Automatic proofs and counterexamples for some ortholattice identities. *Information Processing Letters*, 65(6):285–291, 1998.
49. W.W. McCune. Experiments with discrimination-tree indexing and path indexing for term retrieval. *Journal of Automated Reasoning*, 9(2):147–167, 1992.
50. W.W. McCune. Single axioms for the left group and the right group calculi. *Notre Dame J. of Formal Logic*, 34(1):132–139, 1993.
51. W.W. McCune. OTTER 3.0 reference manual and guide. Technical Report ANL-94/6, Argonne National Laboratory, January 1994.
52. M. Moser, C. Lynch, and J. Steinbach. Model elimination with basic ordered paramodulation. Technical Report AR-95-11, Fakultät für Informatik, Technische Universität München, München, 1995.
53. W.W. McCune M.P. Bonacina. Distributed theorem proving by peers. In A. Bundy, editor, *Automated Deduction — CADE-12. 12th International Conference on Au-*

tomated Deduction, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 841–845, Nancy, France, June/July 1994.
54. P. Narendran, M. Rusinowitch, and R. Verma. RPO constraint solving is in NP. In G. Gottlob, E. Grandjean, and K. Seyr, editors, *Computer Science Logic, 12th International Workshop, CSL'98*, volume 1584 of *Lecture Notes in Computer Science*, pages 385–398. Springer Verlag, 1999.
55. R. Nieuwenhuis. Simple LPO constraint solving methods. *Information Processing Letters*, 47:65–69, 1993.
56. R. Nieuwenhuis. Rewrite-based deduction and symbolic constraints. In H. Ganzinger, editor, *Automated Deduction—CADE-16. 16th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, pages 302–313, Trento, Italy, July 1999.
57. R. Nieuwenhuis, T. Hillenbrand, A. Riazanov, and A. Voronkov. On the evaluation of indexing techniques for theorem proving. In R. Gore, A. Leitsch, and T. Nipkow, editors, *IJCAR 2001*, 2001. this volume.
58. R. Nieuwenhuis and J.M. Rivero. Solved forms for path ordering constraints. In *In Proc. 10th International Conference on Rewriting Techniques and Applications (RTA)*, volume 1631 of *Lecture Notes in Computer Science*, pages 1–15, Trento, Italy, 1999.
59. R. Nieuwenhuis, J.M. Rivero, and M.Á. Vallejo. The Barcelona prover. *Journal of Automated Reasoning*, 18(2):171–176, 1997.
60. R. Nieuwenhuis and A. Rubio. Basic superposition is complete. In *ESOP'92*, volume 582 of *Lecture Notes in Computer Science*, pages 371–389. Springer Verlag, 1992.
61. R. Nieuwenhuis and A. Rubio. AC-superposition with constraints: no AC-unifiers needed. In A. Bundy, editor, *Automated Deduction—CADE-12. 12th International Conference on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 545–559, Nancy, France, June/July 1994.
62. R. Nieuwenhuis and A. Rubio. Theorem proving with ordering and equality constrained clauses. *Journal of Symbolic Computations*, 19:321–351, 1995.
63. R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier Science, 2001.
64. H.J. Ohlbach. Abstraction tree indexing for terms. In H.-J. Bürkert and W. Nutt, editors, *Extended Abstracts of the Third International Workshop on Unification*, pages 131–135. Fachbereich Informatik, Universität Kaiserslautern, 1989. SEKI-Report SR 89-17.
65. M. Paterson and M. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16:158–167, 1978.
66. L.C. Paulson. *Isabelle: A Generic Theorem Prover.* Springer-Verlag LNCS 828, 1994.
67. D. Plaisted and Y. Zhu. Replacement rules with definition detection. In R. Caferra and G. Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *Lecture Notes in Artificial Intelligence*, pages 80–94. Springer Verlag, 1999.
68. G. Plotkin. Building-in equational theories. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 7, pages 73–90. Edinburgh University Press, 1972.
69. P.W. Purdom and C.A. Brown. Fast many-to-one matching algorithms. In J.-P. Jouannaud, editor, *Rewriting Techniques and Applications, First International Conference, RTA-85*, volume 202 of *Lecture Notes in Computer Science*, pages 407–416, Dijon, France, 1985. Springer Verlag.

70. A. Riazanov and A. Voronkov. Vampire. In H. Ganzinger, editor, *Automated Deduction—CADE-16. 16th International Conference on Automated Deduction*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 292–296, Trento, Italy, July 1999.
71. A. Riazanov and A. Voronkov. Limited resource strategy in resolution theorem proving. Preprint CSPP-7, Department of Computer Science, University of Manchester, October 2000.
72. A. Riazanov and A. Voronkov. Partially adaptive code trees. In M. Ojeda-Aciego, I.P. de Guzmán, G. Brewka, and L.M. Pereira, editors, *Logics in Artificial Intelligence. European Workshop, JELIA 2000*, volume 1919 of *Lecture Notes in Artificial Intelligence*, pages 209–223, Málaga, Spain, 2000. Springer Verlag.
73. A. Riazanov and A. Voronkov. An efficient algorithm for backward subsumption using path indexing and database joins. Preprint, Department of Computer Science, University of Manchester, 2001. To appear.
74. A. Riazanov and A. Voronkov. Splitting without backtracking. Preprint CSPP-10, Department of Computer Science, University of Manchester, January 2001. to appear at IJCAI'2001.
75. A. Riazanov and A. Voronkov. Vampire 1.1. (system description). In R. Gore, A. Leitsch, and T. Nipkow, editors, *IJCAR 2001*, 2001. this volume.
76. J.M.A. Rivero. *Data Structures and Algorithms for Automated Deduction with Equality*. Phd thesis, Universitat Politècnica de Catalunya, Barcelona, May 2000.
77. J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, 1965.
78. S. Schulz. System abstract: E 0.3. In H. Ganzinger, editor, *Automated Deduction—CADE-16. 16th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, pages 297–301, Trento, Italy, July 1999.
79. S. Schulz. *Learning Search Control Knowledge for Equational Deduction*, volume 230 of *Dissertationen zur künstliche Intelligenz*. Akademische Verlagsgesellschaft Aka GmmH, 2000.
80. J. Schumann and R. Letz. PARTHEO: a high performance parallel theorem prover. In M.E. Stickel, editor, *Proc. 10th Int. Conf. on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 40–56, 1990.
81. R. Sekar, I.V. Ramakrishnan, and A. Voronkov. Term indexing. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 26, pages 1853–1964. Elsevier Science, 2001.
82. J.K. Slaney, E.L. Lusk, and W. McCune. SCOTT: Semantically Constrained Otter (system description). In A. Bundy, editor, *Automated Deduction—CADE-12. 12th International Conference on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 764–768, Nancy, France, 1994.
83. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1:333–355, 1985.
84. M. Stickel. The path indexing method for indexing terms. Technical Report 473, Artificial Intelligence Center, SRI International, Menlo Park, CA, October 1989.
85. M.E. Stickel. A complete unification algorithm for associative-commutative functions. In *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, pages 71–76, Tbilisi, USSR, 1975.
86. M.E. Stickel. A complete unification algorithm for associative-commutative functions. *Journal of the Association for Computing Machinery*, 28(3):423–434, 1981.
87. M.E. Stickel. Schubert's Steamroller problem: Formulation and solutions. *Journal of Automated Reasoning*, 2(1):89–101, 1986.

88. G. Sutcliffe. Systemon tptp. In D. McAllester, editor, *17th International Conference on Automated Deduction (CADE-17)*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 406–410, Pittsburgh, 2000. Springer Verlag.
89. T. Tammet. Gandalf. *Journal of Automated Reasoning*, 18(2):199–204, 1997.
90. T. Tammet. Towards efficient subsumption. In C.Kirchner and H. Kirchner, editors, *Automated Deduction — CADE-15. 15th International Conference on Automated Deduction*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 427–441, Lindau, Germany, 1998. Springer Verlag.
91. V. Turchin. *Refal-5, Programming Guide and Reference Manual*. New England Publishing Co., 1989.
92. A. Voronkov. LISS — the logic inference search system. In M. Stickel, editor, *Proc. 10th Int. Conf. on Automated Deduction*, volume 449 of *Lecture Notes in Computer Science*, pages 677–678, Kaiserslautern, Germany, 1990. Springer Verlag.
93. A. Voronkov. The anatomy of Vampire: Implementing bottom-up procedures with code trees. *Journal of Automated Reasoning*, 15(2):237–265, 1995.
94. A. Voronkov. Formulae over reduction orderings: some solved and yet unsolved problems. Invited talk at RTA'2000, unpublished, 2000.
95. C. Weidenbach. Combining superposition, sorts and splitting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 27, pages 1965–2013. Elsevier Science, 2001.
96. C. Weidenbach, B. Afshordel, U. Brahm, C. Cohrs, T. Engel, E. Keen, C. Theobalt, and D. Topic. System description: SPASS version 1.0.0. In H. Ganzinger, editor, *Automated Deduction—CADE-16. 16th International Conference on Automated Deduction*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 378–382, Trento, Italy, July 1999.
97. L. Wos, R. Overbeek, and E. Lusk. Subsumption, a sometimes undervalued procedure. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic. Essays in Honor of Alan Robinson*, pages 3–40. The MIT Press, Cambridge, MA, 1991.