Electronic Notes in Theoretical Computer Science 86 No. 1 (2003)
URL: http://www.elsevier.nl/locate/entcs/volume86.html 13 pages

Citius altius fortius: Lessons learned from the Theorem Prover Waldmeister

Thomas Hillenbrand [1]

Max-Planck-Institut für Informatik
Saarbrücken, Germany

Abstract

In the last years, the development of automated theorem provers has been advancing in a so to speak Olympic spirit, following the motto "faster, higher, stronger"; and the Waldmeister system has been a part of that endeavour. We will survey the concepts underlying this prover, which implements Knuth-Bendix completion in its unfailing variant. The system architecture is based on a strict separation of active and passive facts, and is realized via specifically tailored representations for each of the central data structures: indexing for the active facts, set-based compression for the passive facts, successor sets for the conjectures. In order to cope with large search spaces, specialized redundancy criteria are employed, and the empirically gained control knowledge is integrated to ease the use of the system. We conclude with a discussion of strengths and weaknesses, and a view of future prospects.

1 Introduction

Waldmeister is a prover for unit equational deduction. Its theoretical basis is unfailing completion in the sense of [5] with refinements towards ordered completion. The prover saturates the input axiomatization in a repeated cycle that works on a set of active and a set of passive facts, thereby implementing a variant of the given-clause algorithm. It is one of the strongest systems in its class, as demonstrated repeatedly in the last years' competitions at the Conference on Automated Deduction [26].

In this paper, we survey the concepts underlying our prover. Waldmeister is built on solidly engineered algorithms and data structures. As an example, the indexing technique is presented in Sect. 2 (citius). The question of how to organize the proof search into a proof procedure is discussed in

[1] Email: hillen@mpi-sb.mpg.de

Sect. 3 (FORTIUS) where we show that a carefully designed system architecture significantly increases the prover's strength for challenging proof tasks. In Sect. 4 (ALTIUS) we describe extensions of the basic techniques that amount to integrating more knowledge into the system, such that the reasoning process is lifted a bit higher. We conclude in Sect. 5 with a discussion of strengths and weaknesses, and a view of future prospects.

The theorem prover WALDMEISTER was initially developed at the university of Kaiserslautern by A. Buch, A. Jaeger, B. Löchner and the author. Ongoing research is now jointly carried out by B. Löchner in Kaiserslautern and the author in Saarbrücken. The material presented here results from our fruitful collaboration over the last years.

2 CITIUS: Tailor your algorithms and data structures!

The design and analysis of algorithms and data structures is a core subject of computer science; and its importance in the actual construction of a theorem-proving system can hardly be overestimated (cf. [28]). We will use the problem of term indexing (see [9] and [24] for an overview) as an example. Topics such as the computation of normal forms or ordering relationships have been worked out following the same principles.

The aim of indexing is to perform the central term-level operations – matching, unification, search for instances – in a set-based fashion, processing a single query term against a term database at a time instead of sequentially traversing the database entries. Since these central operations constitute a major part of the prover's work, and since the database grows to hundreds of thousands of entries, indexing has become an inevitable component in today's systems.

For WALDMEISTER we have employed perfect discrimination trees [17], which are an instance of a data structure generally known as trie [12]. Tries are trees where every edge is labeled with a single symbol of some alphabet. In essence, they allow sharing of common prefixes: The construction principle is that, for every subtree, its leaf entries all have the same prefix, namely the sequence of labels on the path from the root to that very subtree. In our case the alphabet consists of the function symbols and variables that are occuring. Every term to be inserted is interpreted as string of its symbols, e. g. $f(g(b), a)$ as $f\,g\,b\,a$. To achieve more sharing, the variables of the terms are normalized, i. e. renamed to $x_1, x_2, \ldots$ in order of their first appearance in the symbol string. The data structure depicted in Fig. 1 is an index for the term set $\{f(x_1, x_1), f(x_1, b), f(a, g(x_1)), f(g(x_1), g(x_2)), f(g(b), a)\}$.

We detected two points in the implementation of the data structure which are responsible for a considerable speed-up. First, as can also be seen in Fig. 1, the nodes within the tree can have different sizes. But there is no need to represent them by lists, because for each node the size is fixed over its lifetime. Hence arrays are preferable; and in effect one can during matching index

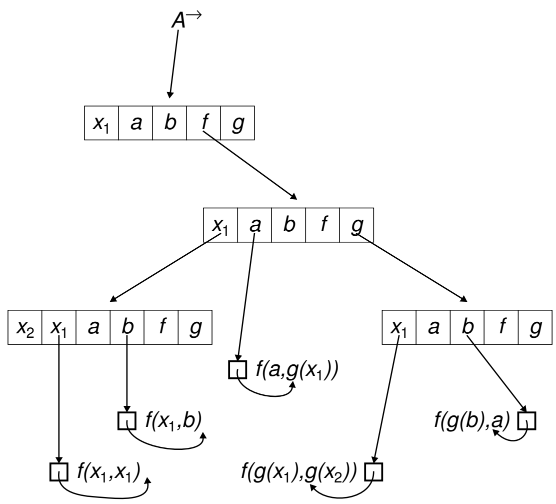

Fig. 1. Implementation view of a discrimination tree

such an array by the currently treated query function symbol and directly jump to the child, if it exists. So the low-level operations on the tree become quite simple.

Second, for a more compact representation, every subtree containing a single leaf node only is collapsed into a single node. This decreases the size of the data structure in memory and therefore also the number of memory locations that have to be inspected during retrieval. Since thereby the number of cache faults is reduced, one is rewarded with faster execution.

Abstracting over these observations, we have learned that in the design of algorithms one should take today's computer architecture into account and strive for compact representations of data. The gain is better cache utilization, i. e. first-class execution. Furthermore we found it useful, if some functionality can be realized in alternative ways, to assess these alternatives by experimental evaluation, namely detailed comparison of the time and space consumption of respective implementations, whenever complexity analysis is too coarse-grained.

On a larger scale, this principle has been applied in the COMPIT joint initiative [21] for the comparison of indexing techniques. Regarding performance in the retrieval of generalizations, our discrimination tree implementation showed to be still quite competitive with the implementations of the newer concepts code trees and context trees, although the code had not been revised for several years. We have learned since that simply by giving up jump lists and compactifying the node format the retrieval speed can nearly be doubled. Anyway, faster indexing techniques should exist out there...

3 FORTIUS: Design your proof procedure carefully!

Deduction methods like completion are usually stated in the form of an inference system, i. e. as a non-deterministic algorithm. One can usually distinguish *generating* inferences, which extend the knowledge base, and *contracting* inferences, which simplify the base in some sense. When constructing a theorem prover, the non-determinisms of the calculus have to be resolved; and there are different strategies for doing so – concerning generating as well as contracting inferences. Renewed interest in this question has been raised not long ago by [28]. In the following we present the view that WALDMEISTER is based on, and the implications it leads to.

3.1 A detailed inspection: being lazy vs. being eager

The procedure shown in Alg. 1 is an instance of the *given-clause algorithm* (cf. [15] and [19, Chap. 5.5.1]). The input consists of an equation set $\mathcal{E}$, the *axioms,* a ground equation set $\mathcal{C}$, the *conjectures,* a reduction ordering $>$ total on ground terms, and a *weighting function* φ mapping equations to *weights.*

Algorithm 1 The system's proof procedure

FUNCTION $\textsc{Waldmeister}(\mathcal{E}, \mathcal{C}, >, \varphi) : \mathrm{BOOL}$
1: $(\mathcal{A}, \mathcal{P}) := (\varnothing, \mathcal{E})$
2: **WHILE** $\neg\mathrm{trivial}(\mathcal{C}) \wedge \mathcal{P} \neq \varnothing$ **DO**
3: $\quad e := \min_{\varphi}(\mathcal{P});\ \mathcal{P} := \mathcal{P} \setminus \{e\}$
4: $\quad e := \mathrm{Normalize}^{>}_{\mathcal{A}}(e)$
5: $\quad$ **IF** $\neg\mathrm{redundant}(e)$ **THEN**
6: $\qquad (\mathcal{A}, P_1) := \mathrm{Interred}^{>}(\mathcal{A}, e)$
7: $\qquad \mathcal{A} := \mathcal{A} \cup \{\mathrm{Orient}^{>}(e)\}$
8: $\qquad P_2 := \mathrm{CP}^{>}(e, \mathcal{A})$
9: $\qquad \mathcal{P} := \mathcal{P} \cup \mathrm{Normalize}^{>}_{\mathcal{A}}(P_1 \cup P_2)$
10: $\qquad \mathcal{C} := \mathrm{Normalize}^{>}_{\mathcal{A}}(\mathcal{C})$
11: $\quad$ **END**
12: **END**
13: **RETURN** $\mathrm{trivial}(\mathcal{C})$

The saturation is performed in a cycle working on a set $\mathcal{A}$ of *active facts* that participate in inferences and a set $\mathcal{P}$ of *passive facts* waiting to become members of $\mathcal{A}$. The function φ heuristically assesses passive facts. It has to ensure that every passive fact becomes minimal at some point in time and hence selected. This guarantees the fairness of the proof procedure.

The active facts $\mathcal{A}$ induce, via their orientable instances $\mathcal{A}^{>}$, an ordered rewrite relation $\rightarrow_{\mathcal{A}}$ and thereby a normal form relation $\rightarrow^{!}_{\mathcal{A}}$. We stipulate a total *normal form function* $\mathrm{Normalize}^{>}_{\mathcal{A}} \subseteq \rightarrow^{!}_{\mathcal{A}}$. Furthermore, an *interreduction function* $\mathrm{Interred}^{>}$ is needed that, given $\mathcal{A}$ and a new equation e, returns both those active facts that are e-reducible and the remainder of $\mathcal{A}$. Finally, let $\mathrm{CP}^{>}(e, \mathcal{A})$ denote the set of all critical pairs between the equation e and all equations in $\mathcal{A}$.

We comment now on some details of the proof procedure (cf. Alg. 1). (L. 1) Initially, the set of active facts is empty, whereas the set of passive facts contains the axiomatization. (L. 2) The saturation proceeds as long as (i) the conjectures have not become trivial, i. e. equations of the form $t{=}t$, and (ii) there are still passive facts. (L. 3) Select an equation e from $\mathcal{P}$ which has minimal weight with respect to φ. (L. 5) Skip if e has become trivial now, or redundant in some other way, e. g. via subsumption. (L. 6) Remove those active facts from $\mathcal{A}$ that are subject to backward simplification wrt. e. (L. 7) Orient the equation, if possible, and add it to the active facts. (L. 8) Generate the critical pairs between e and $\mathcal{A}$. (L. 9) Normalize the generated critical pairs and the reduced active facts, and add the non-redundant ones to $\mathcal{P}$. (L. 10) Normalize the conjectures.

For this proof procedure, the following two invariants hold: (i) Every non-redundant inference conclusion of $\mathcal{A}$ is contained in $\mathcal{A}\cup\mathcal{P}$. (ii) The active facts $\mathcal{A}$ are completely interreduced. Hence, in case of termination with $\mathcal{P} = \varnothing$, $\mathcal{A}$ is an interreduced rewrite system, convergent on ground terms and equivalent to $\mathcal{E}$, i. e. a decision procedure for the uniform word problem of $\mathcal{E}$. Furthermore, if φ ensures fairness, then the procedure demonstrates any valid identity on ground terms in $\mathcal{C}$ within finite time.

The passive facts are, as can be seen in the proof procedure, subject to normalization only right after their generation and in case they get selected. This variant of given-clause algorithm is called *DISCOUNT loop*. In contrast, within an *OTTER loop* the whole set of passive facts is normalized in every iteration of the cycle. Line 9 then would read "$\mathcal{P} :=$ Normalize$^{>}_{\mathcal{A}}(\mathcal{P} \cup P_1 \cup P_2)$" instead. Some OTTER loop implementations moreover use the rewrite relation generated by the union of active and passive facts, i. e. they always employ Normalize$^{>}_{\mathcal{A}\cup\mathcal{P}}$ instead of Normalize$^{>}_{\mathcal{A}}$. The loop variants are named after well-known respective implementations, cf. [1] and [18]. An early experimental comparison of both loops was done in [13, p. 131], where, based on observations when completing the group axioms using the ALDES Data-Type Completion System, preference is given to the OTTER loop. The loop names have been coined in [25, Chap. 2 and 6], where also extensive experiments of both variants are reported. A discussion within the frame of resolution and superposition is contained in [29, Sect. 3 and 5.3].

It follows from the above that the DISCOUNT loop spends *much less* work per active fact than the OTTER loop, so that the development of $\mathcal{A}$ (and $\mathcal{P}$) progresses *much faster*. The price one has to pay is that there might be a useful, initially large passive fact which by full normalization has intermediately become smaller and so would be selected later. But in our experience the gain is *much higher* than the price.

3.2 Another architectural spin-off

Theorem proving procedures are refutationally complete under the idealized assumption of unlimited resources. But implementations thereof are run on finite machines and so can only analyze a finite part of the infinite search space. For provers based on saturation procedures, this resource restriction shows up as sky-high growing memory consumption if, in order to solve a difficult problem, they need to be run not for a few minutes only, but for say hours to days; and a bundle of preventive measures thereagainst has been suggested (see e. g. [25]).

In the WALDMEISTER system, we initially used a representation for the set $\mathcal{P}$ of passive facts where every single equation was compressed into the string of its symbols. Nevertheless more than 95 % of the memory was occupied by $\mathcal{P}$, which was the limiting factor for overall memory consumption. We discovered not before two years ago that the inherent structure of $\mathcal{P}$ itself can be used as starting point to dramatically cut down memory requirements [10].

The key is to condense *whole sets* of critical pairs, namely those generated at the same time i, descending from the same newly activated equation e_i. They are stored as one entry of constant size, essentially the number i and the minimum of the weights of the yet unselected pairs. To avoid excessive recomputation we employ a buffer to cache a fixed number of "promising" critical pairs individually. The weighting function φ is used to separate the subset of individually stored or already selected equations from the rest collectively represented in the set entry. This changes the asymptotic space behavior of $\mathcal{P}$ over abstract time from quadratic to linear [10, p. 493f]. Our technique requires for completeness reasons that the successive normal form functions developed during completion remain accessible. This can easily be achieved with an additional discrimination tree.

With this new architecture it is now possible to solve problems that were previously out of reach. So WALDMEISTER is now able to prove with simple standard heuristics that a ring with $x^5 = x$ is commutative (RNG036-7), or that the Robbins Axiom together with the second Winker Lemma entails Boolean (ROB007-1). For these problems the size of $\mathcal{A}$ exceeds 70 000, and more than 500 million equations are inserted into $\mathcal{P}$. For such a large search state 200 MByte suffice, most of which are occupied by $\mathcal{A}$. To our knowledge the only other system that can solve ROB007-1 is McCune's prover EQP [19]. The EQP proof has been found with built-in AC-axioms and an incomplete strategy. Clauses exceeding a weight limit are discarded. It has been noted that too simplistic discarding heuristics are harmful to the success of the proof search. So some tuning knowledge concerning weight or clause limits is required, and it may be specific for problem domains or even individual problems. For the WALDMEISTER loop, however, such knowledge is not required.

Only recently has it turned out that the revealed structure of the search state paves the way to an easily implemented parallelization of the prover with modest communication requirements and rewarding speed-ups. Roughly, the new representation of critical pairs is used as compact interchange format, so that communication bandwidth does not become a limiting factor. Details can be found in [8].

4 Altius: Integrate more intelligence!

4.1 Tackling redundancy

When attacking large proof tasks, the performance of unfailing completion procedures is often not satisfactory in that a large number of apparently very similar facts is generated. This happens for example in the presence of associative-commutative (AC) operators, and motivates the search for methods to reduce the search space while still retaining.

A prominent way of doing this is to build in such problematic equational theories into the calculus itself (see e. g. [14,23]), thereby reasoning on equivalence classes instead on terms. But to implement this method in an existing prover is not an easy exercise, since most parts have to be changed or rewritten, so that one might look for "cheap substitutes" [28, Sect. 6] instead.

Unfailing completion is a constructive instance of ordered completion, which serves as a useful framework here (cf. [4, Chap. 4] for details). This framework is built upon the notion of an ordering on proofs, which is derived from the reduction ordering $>$. Then an equation is redundant if for each of its ground instances there is a proof in the remaining equational system smaller than the instance itself. This is for example the case if the equational system E contains a ground complete subset E_0 and some equation e is logically entailed by E_0. Then every ground instance of e has a rewrite proof in E_0, which turns out to be smaller than e itself. Hence e can be discarded.

In the case of an AC operator $+$, if a ground total lexicographic path ordering (LPO) or Knuth-Bendix ordering (KBO) is employed, such a ground complete subset is given with the AC axioms and the additional equation $x+(y+z) = y+(x+z)$. It follows that any other equation $s{=}t$ where s and t are identical modulo AC can be deleted. This criterion is easy to implement, and in experiments described in [2], the number of active facts necessary to find a proof has reduced by a factor of up to 3.7, whereas run-time has decreased by a factor of up to 20.

To go beyond given fixed theories, we have extended the approach by a case splitting on the possible ordering relations between instances of variables of the equation at hand [16]. If left- and right-hand side can be joined under each of these constraints, then the equation essentially is redundant, since again every ground instance has a smaller proof. We had to learn that if such redundant equations are deleted themselves, then the simplification relation is weakened such that many proof tasks are no longer completed within reasonable time. So these equations are kept for simplification, but no critical pairs need to be built from them, i. e. the only work they cause is representing them. Speaking in more abstract terms, an equation that has been shown redundant in an expensive way may itself be useful for cheap, rewrite-based redundancy proofs.

Our method reaches its limit when a variable has to be compared to another subterm, so that redundant equations may remain undetected. The full decision procedure given in [7] requires the construction of confluence trees on top of solvers for ordering constraints. The experiments with an implementation described in [3] show that this is quite costly if the proof search is to be sped up (cf. [20, Probl. 7, 11]). Ongoing work is to develop cheaper, constraint-based redundancy criteria.

4.2 Automating control

The proof procedure (Alg. 1) is parameterized by a weighting function φ and a reduction ordering $>$. The influence of distinct classifications and especially orderings is tremendous, often leading to differences in proof times of orders of magnitude. Due to the dynamic nature of the proof process, instantiating these parameters well is a non-trivial task. It seems very difficult, if possible at all, to tackle this task theoretically in sufficient generality (cf. [20, Probl. 3]).

However, we have experienced that proof problems sharing major parts of their axiomatizations often behave similar in the proof search. This might be explained by two reasons: First, the saturation process is quite independent of the conjecture to be proved. Second, if the axiom sets basically coincide, the saturations will produce many common conclusions.

Of course we cannot expect a potential deduction user to supply the prover with detailed parameter settings on each invocation. Therefore it is important to extend the system with a component that automatically adapts it to the problem at hand. This has enabled us to integrate our domain-specific knowledge into WALDMEISTER.

We use a declarative representation consisting of three compact tables that are easy to edit. The first and the second contain descriptions of usual mathematical concepts, giving the approach a semantical stress, whereas the third factors out the control knowledge. The process starts with the problem axiomatization as input, e. g. some ring-theoretic problem. In the first table algebraic laws, such as associativity or distributivity, are described by means of pattern equations with variables for the operators:

$$\begin{aligned} F(x, F(y, z)) &= F(F(x, y), z) && \Rightarrow \mathrm{Assoc}(F) \\ F(G(x, y), z) &= G(F(x, z), F(y, z)) && \Rightarrow \mathrm{Distr_r}(F, G) \end{aligned}$$

The pattern equations are matched onto the axioms, yielding a compact representation of the detected laws, in our case:

$$\{\mathrm{Assoc}(+), \mathrm{Assoc}(\cdot), \mathrm{Distr_r}(\cdot, +), \ldots\}$$

The second table holds algebraic structures characterized as sets of algebraic laws. These sets are matched onto the result of the first stage. For the ring-theory example the following two structures match:

$$\{\mathrm{Neut_r}(F,N), \mathrm{Assoc}(F), \mathrm{Inv_r}(F,I,N)\} \Rightarrow \mathrm{Group}(F,I,N)$$

$$\{\mathrm{Assoc}(F), \mathrm{Assoc}(G), \mathrm{Distr_r}(G,F), \ldots\} \Rightarrow \mathrm{Ring}(F,N,I,G)$$

Based on axiom set inclusion, a static conflict resolution mechanism automatically prefers structures that are more specialized, here the ring structure. Finally a parameter instantiation for that structure is looked up in the third table:

$$\mathrm{Ring}(F,N,I,G) \;\Rightarrow\; > := \mathrm{LPO}(G\succ I\succ F\succ N),\; \varphi := (s{=}t) \mapsto |s| + |t|$$

The representation is lucid and easy to maintain and extend; and the staged matching is more efficient in practice than the direct one.

From our users we get very positive feedback on the autonomous mode. Despite that it was originally derived mainly from TPTP problems it seems to be a very good choice when the user's original (non-TPTP) problem is of an "algebraic" nature, e. g. stems from group theory or ring theory. In fact, the setting provided by the autonomous mode often outperforms ad-hoc parameterizations supplied by advanced users. For problems of a more "combinatoric" nature we get mixed results. Sometimes only fine-tuning by the developers helps, if at all; sometimes WALDMEISTER even beats specialized systems, as we were told by users in the area of string-rewriting systems.

4.3 Representation of conjectures $\mathcal{C}$

A nuisance of completion-based theorem proving is that, when tackling ground (i. e. skolemized universally quantified) conjectures, nearly all the time is spent completing the axiomatization, but only a tiny fraction is spent processing the conjectures. This has been critized as lack of goal orientation.

The standard approach for tackling such a conjecture is to compute normal forms of the terms and check if they are syntactically equal. Since the rewrite relation is not confluent during completion, the normal forms are not necessarily unique. Therefore instead of considering an arbitrary normal form, WALDMEISTER incrementally constructs for each term in the conjecture the set $\mathcal{S}$ of all successors [11], thereby refining the functionality specified in Line 10 of Alg. 1. The conjecture $u{=}v$ is proved if $\mathcal{S}(u) \cap \mathcal{S}(v) \neq \emptyset$.

Our method is best explained with a simple example. Figure 2 shows how sets of successors can be used in solving the proof problem GRP141-1 [27]. The terms $u \equiv f(f(a,b),c)$ and $v \equiv c$ have to be joined. The four graphs illustrate the enlarging of the conjecture. Initially, the two sets consist of only one term each (a). With the 16th rewrite rule u can be reduced to two different terms which join again in the term denoted by u'. $\mathcal{S}(u)$ now consists of four terms (b). After generating rule no. 19, u can be reduced to v in two different ways, hence $v \in \mathcal{S}(u)$ and the conjecture is proved (c)! Compare this with the standard approach which would still be stuck in u'. The last graph

(d) shows the way out of this dead end which is not possible before 31 rules have been generated. The number of generated rules is cut down by one third here. In more difficult proof tasks, even higher reduction factors may occur.

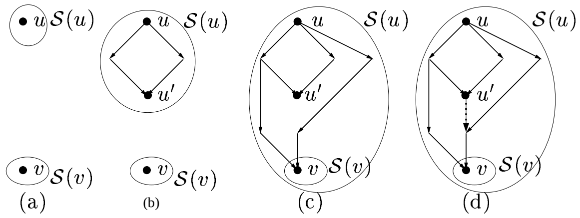

Fig. 2. Sets of successors in proving GRP141-1

This technique has improved our system impressively. When introduced, it increased the number of solvable TPTP problems from 278 to 300, and even decreased the average proof time.

5 Conclusion

The prover presented so far is quite powerful on the problem class that it has been developed for: It can solve many routine problems instantly. But for challenging problems, the amount of expertise to solve them still may be considerable, and a hindrance to spreading its application. Any suggestions here are warmly welcome.

To increase the reasoning power of the core system, current research is focused on the integration of deeper theoretical insights. In doing so, two difficulties appear quite frequently. First, the notions worked out in achieving progress in theory may, if they are to be applied constructively, turn out to be too costly in general and need to be approximated. Second, a refinement of a proof method, although retaining completeness, may change the search behavior on the proof problems of interest in an unforeseen way, and hence call for further inspection. At the moment, stronger, constraint-based redundancy criteria for specific cases of interest are under study. It will be interesting to compare this approach to the direct integration of theories. Furthermore, the question how far the principles developed here carry over at least to the Horn case deserves attention.

From a more abstract point of view, the expressiveness of the logics that WALDMEISTER deals with naturally limits the applicability of this deductive tools. On the other hand, the lessons learned in implementing a basic method of automated theorem proving should be a help in the construction of systems for other logics. It is our conviction that continuous specialization and refinement of deductive techniques is a prerequisite to future progress.

Finally, the WALDMEISTER system is available via its Web page at

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

References

[1] 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*, volume 914 of *LNCS*, pages 397–402. Springer-Verlag, 1995.

[2] J. Avenhaus, Th. Hillenbrand, and B. Löchner. On using ground joinable equations in equational theorem proving. *Journal of Symbolic Computation*, 2003. To appear.

[3] J. Avenhaus and B. Löchner. CCE: Testing ground joinability. In *Proceedings of the 1st International Joint Conference on Automated Reasoning*, volume 2083 of *LNAI*, pages 658–662. Springer-Verlag, 2001.

[4] L. Bachmair. *Canonical Equational Proofs*. Birkhäuser, 1991.

[5] L. Bachmair, N. Dershowitz, and D. A. Plaisted. Completion without failure. In *Resolution of Equations in Algebraic Structures*, volume 2, pages 1–30. Academic Press, 1989.

[6] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation and superposition. In *Proceedings of the 11th International Conference on Automated Deduction*, volume 607 of *LNAI*, pages 462–476. Springer-Verlag, 1992.

[7] H. Comon, P. Narendran, R. Nieuwenhuis, and M. Rusinowitch. Decision problems in ordered rewriting. In *Proceedings of the 13th IEEE Symposium on Logics in Computer Science*, pages 276–286. IEEE Computer Society, 1998.

[8] J.-M. Gaillourdet, Th. Hillenbrand, B. Löchner, and H. Spies. The new WALDMEISTER loop at work. In F. Baader, editor, *Proceedings of the 19th International Conference on Automated Deduction*, LNAI. Springer-Verlag, 2003.

[9] P. Graf. *Term Indexing*, volume 1053 of *LNCS*. Springer-Verlag, 1996.

[10] Th. Hillenbrand and B. Löchner. The next WALDMEISTER loop. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, LNAI, pages 486–500. Springer-Verlag, 2002.

[11] A. Jaeger. Anwendungen von Nachfolgermengen in Termersetzungssystemen. Projektarbeit, Universität Kaiserslautern, 1997.

[12] D. E. Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching. Addison-Wesley, 2nd edition, 1998.

[13] W. Küchlin. An implementation and investigation of the Knuth-Bendix completion procedure. Interner Bericht 17/82, Universität Karlsruhe, Fakultät für Informatik, 1982.

[14] D. S. Lankford and A. M. Ballantyne. Decision procedures for simple equational theories with commutative-associative axioms: Complete sets of commutative-associative reductions. Technical Report ATP-39, University of Texas, Austin, 1977.

[15] E. L. Lusk. Controlling redundancy in large search spaces: Argonne-style theorem proving through the years (invited lecture). In A. Voronkov, editor, *Proceedings of the 3rd International Conference on Logic Programming and Automated Reasoning*, volume 624 of *LNAI*, pages 96–106. Springer-Verlag, 1992.

[16] U. Martin and T. Nipkow. Ordered rewriting and confluence. In *Proceedings of the 10th International Conference on Automated Deduction*, volume 449 of *LNCS*, pages 366–380. Springer-Verlag, 1990.

[17] W. McCune. Experiments with discrimination-tree indexing and path indexing for term retrieval. *Journal of Automated Reasoning*, 9(3):147–167, 1992.

[18] W. McCune. OTTER 3.0 reference manual and guide. Technical Report ANL-94/6, Argonne National Laboratory, 1994.

[19] W. McCune. 33 basic test problems: a practical evaluation of some paramodulation strategies. In R. Veroff, editor, *Automated Reasoning and its Applications: Essays in Honor of Larry Wos*, chapter 5, pages 71–114. MIT Press, 1997.

[20] R. Nieuwenhuis. Rewrite-based deduction and symbolic constraints (invited talk). In *Proceedings of the 16th International Conference on Automated Deduction*, volume 1632 of *LNCS*. Springer-Verlag, 1999.

[21] R. Nieuwenhuis, Th. Hillenbrand, A. Riazanov, and A. Voronkov. On the evaluation of indexing techniques for theorem proving. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning*, volume 2083 of *LNAI*, pages 257–271. Springer-Verlag, 2001. See also `http://www.mpi-sb.mpg.de/~hillen/compit`.

[22] R. Nieuwenhuis and A. Rubio. Basic superposition is complete. In *Proceedings of the 4th European Symposium on Programming*, volume 582 of *LNCS*, pages 371–390. Springer-Verlag, 1992.

[23] G. E. Peterson and M. E. Stickel. Complete sets of reduction for some equational theories. *Journal of the Association for Computing Machinery*, 28:233–264, 1981.

[24] I. V. Ramakrishnan, R. Sekar, and A. Voronkov. Term indexing. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 26, pages 1853–1964. Elsevier, 2001.

[25] A. Riazanov and A. Voronkov. Limited resource strategy in resolution theorem proving. Technical Report CSPP-7, University of Manchester, 2000. Accepted for publication in *Journal of Symbolic Computation*.

[26] G. Sutcliffe and C. B. Suttner. The CADE ATP system competition. See `http://www.cs.miami.edu/~tptp/CASC`.

[27] C. B. Suttner and G. Sutcliffe. The TPTP problem library (TPTP v2.2.0). Technical Report 99/02, James Cook University, Townsville, 1999.

[28] A. Voronkov. Algorithms, datastructures, and other issues in efficient automated deduction (invited talk). In R. Goré, editor, *Proceedings of the First International Joint Conference on Automated Reasoning*, volume 2083 of *LNAI*, pages 13–28. Springer-Verlag, 2001.

[29] Chr. 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, 2001.