The Next WALDMEISTER Loop (Extended Abstract)

Thomas Hillenbrand[1] and Bernd Löchner[2]

[1]Max-Planck-Institut für Informatik, Saarbrücken, Germany,
hillen@mpi-sb.mpg.de

[2]FB Informatik, Universität Kaiserslautern, Kaiserslautern, Germany,
loechner@informatik.uni-kl.de

1 Introduction

Very recently the question of how to organize the proof search within saturation-based theorem provers has attracted renewed attention [Vor01]: Two variants of the given-clause algorithm are under discussion, now being termed OTTER loop resp. DISCOUNT loop. For each of them, competitive systems have been built, demonstrating that both approaches meet the state of the art. But it is unclear which path to follow in the future in order to "increase performance of the modern provers by several orders of magnitude" [loc. cit.].

With this paper, we want to further the discussion of this question within the frame of the implementation workshop. We present our understanding of the DISCOUNT loop as evolved during the development of the WALDMEISTER prover for unit equational deduction [HJL99]. In our view, the benefits of this loop variant are threefold: (i) only active facts have to be indexed, (ii) passive facts may be stored in a compressed, space-saving representation, and therefore (iii) completeness can be sustained much longer. This contrasts with the need for weight limits or eligibility estimations in the case of the OTTER loop (cf. [RV99]).

In this contribution we focus on the second issue. We refine our previous conceptions towards a clarified system design that allows even more compression and thereby ends the need to sacrifice completeness after several hours of running time. Moreover, any overhead for the proof construction is removed from the proof search. It should even become possible to depart from the given-clause algorithm, to simulate a given-pair algorithm [McC97], and especially to develop hybrid versions, which to our knowledge has not been studied so far. The implementation which is currently under construction will give rise to further insights.

Algorithm 1 The proof procedure of WALDMEISTER

FUNCTION WALDMEISTER$(\mathcal{E}, \mathcal{H}, >, \varphi)$: BOOLEAN
1: $(\mathcal{A}, \mathcal{P}) := (\emptyset, \mathcal{E})$
2: **WHILE** $\neg\text{trivial}(\mathcal{H}) \wedge \mathcal{P} \neq \emptyset$ **DO**
3: $\quad e := \arg\min \varphi(\mathcal{P});\ \mathcal{P} := \mathcal{P} \backslash \{e\}$
4: $\quad$ **IF** $\neg\text{orphan}(e)$ **THEN**
5: $\quad\quad e := \text{Normalize}^{>}_{\mathcal{A}}(e)$
6: $\quad\quad$ **IF** $\neg\text{redundant}(e)$ **THEN**
7: $\quad\quad\quad (\mathcal{A}, P_1) := \text{Interred}^{>}(\mathcal{A}, e)$
8: $\quad\quad\quad \mathcal{A} := \mathcal{A} \cup \{\text{Orient}^{>}(e)\}$
9: $\quad\quad\quad P_2 := \text{CP}^{>}(e, \mathcal{A})$
10: $\quad\quad\quad \mathcal{P} := \mathcal{P} \cup \text{Normalize}^{>}_{\mathcal{A}}(P_1 \cup P_2)$
11: $\quad\quad\quad \mathcal{H} := \text{Normalize}^{>}_{\mathcal{A}}(\mathcal{H})$
12: $\quad\quad$ **END**
13: $\quad$ **END**
14: **END**
15: **RETURN** $\text{trivial}(\mathcal{H})$

2 An inspection of Waldmeister's proof procedure

2.1 The proof procedure itself

The input to the procedure (cf. Alg. 1) consists of an axiom set $\mathcal{E}$, a set of hypotheses $\mathcal{H}$, a reduction ordering $>$, and a weighting function φ for heuristical evaluation. The saturation is performed in a cycle working on a set $\mathcal{A}$ of *active facts* which participate in inferences and a set $\mathcal{P}$ of *passive facts* waiting to become members of $\mathcal{A}$. We denote, slightly extending the imperative notation, by $\mathcal{A}_i$ the value of $\mathcal{A}$ after the i-th iteration of the loop core in lines 7–11; this iteration count gives a notion of abstract time. The sets $\mathcal{A}_i$ induce the rewrite relations $\rightarrow_i := \rightarrow_{\mathcal{A}_i} \subset >$. The weighting function φ, e.g. $\varphi(s{=}t) = |s| + |t|$, has to ensure that every passive fact becomes minimal at some point in time. This guarantees the fairness of the proof procedure.

(L. 1) Initially, $\mathcal{A}$ is empty and $\mathcal{P}$ contains the input axiomatization. (L. 2) The saturation proceeds as long as the hypotheses have not become trivial and there are still passive facts to be selected. Inside the completion loop, the following steps are performed: (L. 3) Select an equation e with minimal weight from the set $\mathcal{P}$. (L. 4, 5) Unless a parent equation has been reduced, simplify e to a normal form. (L. 6, 7) Interreduce the set $\mathcal{A}$ with e, i.e. take active facts that are reducible by e out of $\mathcal{A}$. (L. 8) Orient e if possible, and add it to $\mathcal{A}$. (L. 9) Generate all new critical pairs. (L. 10) Normalize them and the reduced active facts, and insert everything into $\mathcal{P}$. (L. 11) Normalize the hypotheses.

As can be seen in the proof procedure, the passive facts are subject to normalization only right after their generation and in case they get selected. In contrast, within an OTTER loop the whole set of passive facts is normalized in every iteration of the cycle. Line 10 then would read "$\mathcal{P} := \text{Normalize}^{>}_{\mathcal{A}}(\mathcal{P} \cup P_1 \cup P_2)$". Some implementations moreover use the rewrite relation generated by the union of active and passive facts, i.e. they always employ $\text{Normalize}^{>}_{\mathcal{A}\cup\mathcal{P}}$ instead of $\text{Normalize}^{>}_{\mathcal{A}}$. In that case the orphan predicate (L. 4) must always be evaluated to FALSE, cf. Sect. 2.2.

2.2 Aspects of space consumption

Typically $\mathcal{P}$ contains far more equations than $\mathcal{A}$, as a rule of thumb $|\mathcal{P}| = O(|\mathcal{A}|^2)$ on the average. Therefore the space requirements of WALDMEISTER are dominated by the representation of the passive facts. Hence, methods to save space mainly have to tackle $\mathcal{P}$. One approach is to minimize the number of elements, another, independent one, is to minimize the space needed for one element.

As to the first, redundancy criteria are under demand. But it is clear that such redundancy checks must not be too expensive, since otherwise the DISCOUNT loop would fall back into an OTTER loop again.

A cheaper and more efficient check is a special feature of the DISCOUNT loop, named *"orphan" criterion*: A critical pair can be deleted if one of its "parent" active facts has been interreduced, for the critical pair has not participated in any inference. This would not be possible if passive facts were used for simplification purposes.

The second approach, namely reducing the space of a single passive fact, means to employ simple *compression schemes*. Within WALDMEISTER, this can be done in three ways: (i) Not at all, i.e. flatterms are employed at the cost of 12 bytes per symbol. (ii) On the level of terms: The passiveness of facts in the DISCOUNT loop entails that they, after generation, are not touched again until they get selected, which in theorem proving is the case for a small fraction only. It is therefore possible to store terms simply as strings of symbols, at the cost of one byte per symbol. (iii) Overlap representation: After heuristical evaluation, the terms themselves are thrown away, and just minimal reconstruction information is kept, at constant cost.

The influence of these representations on the process size over abstract time can be studied in Fig. 1. For each variant, the WALDMEISTER system was run on two large proof tasks from the TPTP problem library on a UltraSPARC-IIi workstation. Stringterms allow compression rates of about 10, which is comparable to what can be achieved with a shared term representation [LS01]. The overlap representation improves on this by another factor of nearly 10. This is quite typical for long-lasting runs.

Nevertheless, the space consumption becomes prohibitive after the activation of several thousand facts (cf. Fig. 1). After some hours we are currently faced with the decision to give up or to throw away critical pairs and to sacrifice completeness. In the sequel we will show how to escape this dilemma.

2.3 A drawback and its remedy

The above-mentioned overlap representation is apparently the best one in terms of space. We have employed it when memory was tight, but it has up to now not become our method of choice. The reason for this is that the proof search is influenced: When a critical pair e is generated at time i, it is normalized to some $e{\downarrow_i}$ with respect to the then current rewrite relation $\rightarrow_i$. Let e be selected at time $j \gg i$. The reconstruction process then yields $e{\downarrow_j}$, whereas with the two other representations the critical pair would enter the completion loop in the shape $(e{\downarrow_i}){\downarrow_j}$. Since the rewrite relation changes during completion, these shapes may differ. According to our observations, the effect on the proof search is frequently negative.

To overcome this obstacle, that is, to make the reconstruction perfect, one has to remember history: All versions of the rewrite relation $\rightarrow_i$ and the time the critical pair at hand was constructed and normalized are recorded. This is by far not as costly as it sounds: It is sufficient to annotate the elements of $\mathcal{A}$ with their *activation* time, and their *deactivation* time if necessary, i. e. when they get reduced by a new equation. Along with each set $\mathcal{A}_i$, the order in which its elements are traversed when searching for rewrite patterns must be reproducable as well so that still the same matching rewrite rules are delivered. Then the normalization steps can be perfectly reproduced; the overlap representation becomes neutral w.r.t. the search

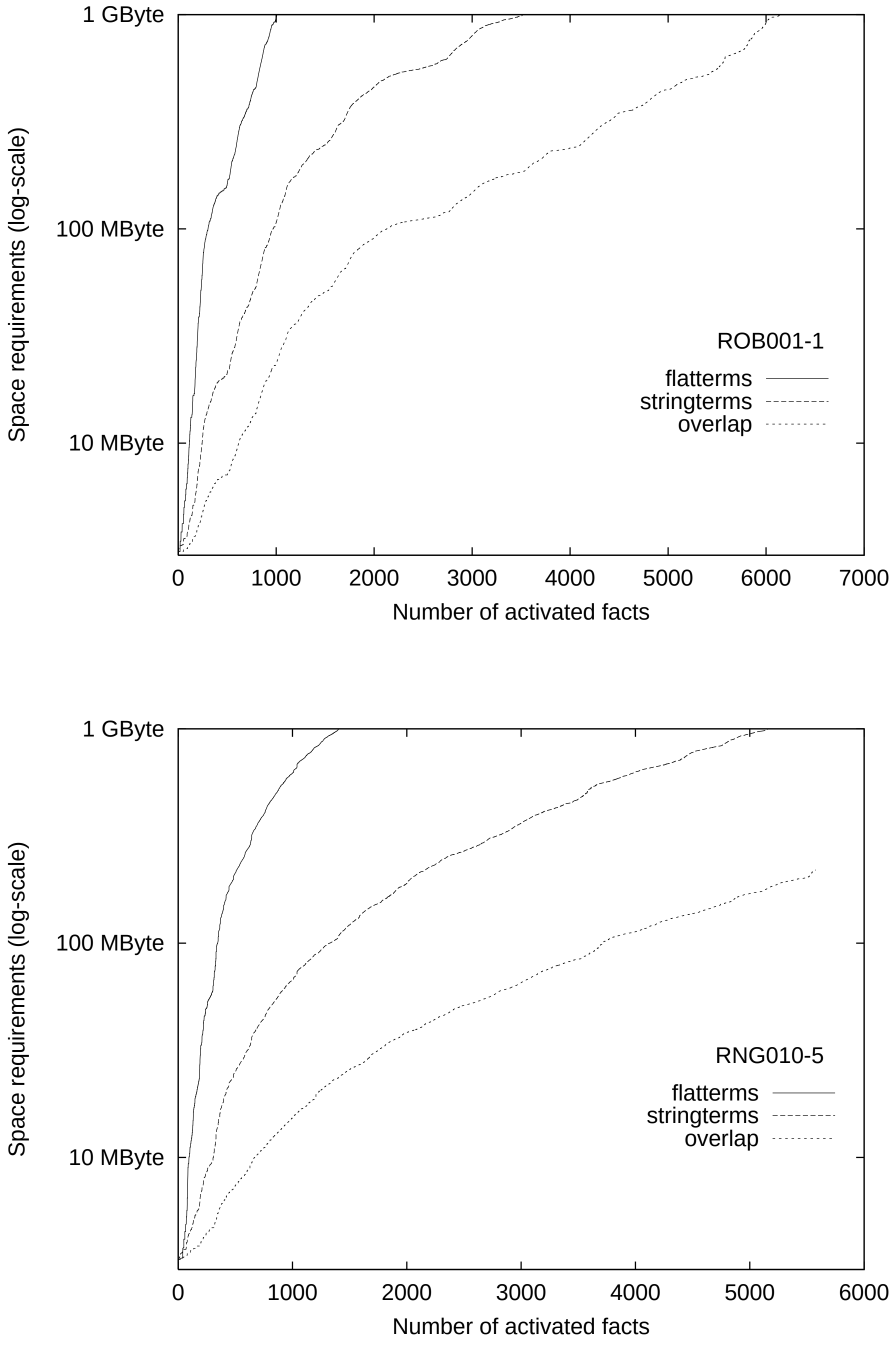

Figure 1: Size of the process over abstract time

behavior.

As a consequence, this changes our view of the rôle of $\mathcal{P}$ in the proof procedure: Abstractly said, $\mathcal{P}$ has to enumerate in a fair way all critical pairs that are not known to be redundant. This sheds a new light on the normalization of freshly generated critical pairs: Its purpose is, besides testing redundancy, only to make a well-behaved heuristical evaluation possible; it is decoupled from the "real" inferences which occur merely at the point of selection; and only "good" critical pairs ever enter the completion loop. Going further, the one-to-one relationship between the elements of $\mathcal{P}$ and the equations they represent is no longer necessary: An item in $\mathcal{P}$ could e. g. denote the set of all critical pairs between two equations e_i and e_j, or between an equation e_i and all members of $\mathcal{A}$ (cf. Sect. 3.1).

2.4 On generating proof objects

A further issue is the provision of a proof object. The first approach realized in WALDMEISTER was to protocol each inference step in an external file. This led to a considerable slowdown of the proof search because a proof task of say 5 seconds could produce a proof log of nearly 100 MBytes.

To overcome this unacceptable situation, we next changed to recording internally for every inference step the minimal information necessary to reproduce it. When a proof is found, a dependency analysis reveals which steps actually contribute to the proof. These inferences are then re-performed step by step to generate the proof object. All in all, one gets proof objects with a run-time penalty of about 5 %.

But this approach has another, still severe drawback: Even with the minimized protocol information the space requirements of the internal proof log can become a limiting factor, for all the reductions on newly generated critical pairs have to be protocolled. Remembering the history of the rewrite relation, however, can drastically change the picture: During proof search, we just keep for every active fact the overlap information it descends from. Within the DISCOUNT loop, only members of $\mathcal{A}$ contribute to a proof. Hence, after the proof is found, the subset $\mathcal{A}^p$ of $\mathcal{A}$ actually needed for the proof is determined in a backward-oriented dependency analysis. (Experience shows that $|\mathcal{A}^p| \ll |\mathcal{A}|$.) In a second, forward-oriented phase all inferences needed to construct the elements of $\mathcal{A}^p$ are performed while logging of the inference steps is enabled. As we can see, the complete inference protocol of the proof search is no longer needed.

3 Representational issues

3.1 The set of passive facts

A simple, but seldomly exploited fact is that a position within a term or a clause can be encoded in a single integer, e.g. as number of the top symbol of $s|_p$ when the symbols in s are counted from left to right. This is straightforward especially when flatterms are employed.

Given two parent equations $s[l']_p{=}t$ and $l{=}r$, the critical pair at the overlap position p is uniquely determined as $\sigma(s[r]_p{=}t)$ where $\sigma = \mathrm{mgu}(l, l')$. The parents e_i and e_j are represented by their numbers i and j, and the critical pair is encoded as $\langle i, j, xp\rangle$ where xp denotes an extended position to clarify on which side to overlap. Instead of distinguishing "into" equation and "from" equation by the argument position in this tuple, we encode this information into the extended position as well and then may require such an arrangement that $i > j$, for the following reasons: Assuming a 32-bit address space, the tuple can with some bit-twiddling be represented in one 64-bit integer. If then such tuples are compared as integers, the result coincides with that of a comparison of the critical pairs by age. Note that many systems employ explicit age information in the facts for that purpose.

Introducing wildcards, tuples can represent sets of critical pairs, e.g. $\langle i, j, *\rangle$ for all the overlaps between the equations e_i and e_j, or $\langle i, *, *\rangle$ for everything generated at the activation of e_i. Equations stemming from interreduction can be encoded in a similar way.

Since $\mathcal{P}$ is ordered according to φ these items are augmented by a weight w. Whenever the number of elements in $\mathcal{P}$ reaches a certain limit, the following compression scheme is applied: Some or all tuples $\langle w_1,\ i, j, xp_1\rangle$ $\ldots$ $\langle w_n,\ i, j, xp_n\rangle$ are combined into $\langle w',\ i, j, *\rangle$ where $w' = \min_{1\le k\le n} w_k$. If this is not sufficient, the tuples $\langle w_k,\ i, j, *\rangle$, $1 \le k \le n$, can be packed into $\langle \min_{1\le k\le n} w_k,\ i, *, *\rangle$.

This should allow to represent most sets $\mathcal{P}$ that occur in practice within a reasonable amount of memory: In the extreme, $\mathcal{P}$ just consists of an initial segment of individual critical pairs and then is linearly bound by the size of $\mathcal{A}$, i. e., $|\mathcal{P}| = O(|\mathcal{A}|)$ even in the worst case. Whenever such a wildcard tuple is selected by the main loop, it gets expanded, i. e., the represented critical pairs are recomputed. The whole approach therefore trades time for space; but note that in general only the light-weighted tuples ever need to be expanded again.

Finally, we also get some new flexibility: The wildcards in the tuples allow to depart from the pure given-clause algorithm. Instead of generating and normalizing all new critical pairs, the corresponding tuples $\langle i, j, *\rangle$ can directly be inserted into $\mathcal{P}$. With some appropriate setting of the initial weights for these tuples it is possible to simulate a given-pair algorithm [McC97] or to develop hybrid versions.

3.2 The set of active facts and its accumulated history

To record all the rewrite relations $\rightarrow_j$, $j \leq i$, it is sufficient to store the elements e_j as they are. The problem is that for a given relation $\rightarrow_j$ and a redex t several equations may match. Which of them is chosen for reduction is determined by the traversal through the indexing structure. It is of course not feasible to save for every $j \leq i$ the index for $\mathcal{A}_j$. We would like to use only a single index for the accumulated history; how can this be achieved?

At this point the perfect discrimination trees employed in WALDMEISTER reveal a beautiful property: The ordering relation of two rewrite patterns being traversed is invariant under insertion and deletion of other elements. Hence, traversing an index over $\{e_j \mid j \leq i\}$ with some filtering to determine the desired $\rightarrow_k$ leads to the same traversal order of matching equations as in the index for $\mathcal{A}_k$. The reason for this property is that the trees are ordered ones, i.e. the order of the successors of a node is fixed. It is unclear to us how the same effect can be achieved with other indexing schemes.

For performance reasons we can keep two indexes: one of $\mathcal{A}_i$ for the current rewrite relation and another of $\{A_j \mid j \leq i\}$ for the accumulated history.

4 Conclusion

This work answers a question which has been on our agenda for several years: How to retain completeness in situations when even our best compression techniques hit the limitations of memory or address space? Compared with the methods mentioned in [Wei01, p. 1980f] our new approach is – in our opinion – superior: Neither do we lose completeness, nor do we have to restart the whole search. If the size of $\mathcal{P}$ is set to modest values we can even expect some improvements in the run time of the system: Since the process size is then considerably smaller, the working set becomes smaller and the cache utilization increases.

As an unexpected, but most welcome, effect we gain more conceptional clarity in the architecture of our prover. First, the initial generation and normalization of critical pairs can be interpreted to be for heuristical evaluation only. These steps are no “real” inferences. Second, the provision of a proof object can be completely decoupled from the proof search. There is no need to record each inference performed in case that inference will contribute to the proof. Third, the given-clause algorithm is no longer hard-wired. We can simulate the given-pair algorithm or explore new hybrid versions.

Finally, let us say that our approach, being fit for the unit equality prover WALDMEISTER, can straightforwardly be adapted to any resolution-based prover that implements the DISCOUNT loop variant. Since in the full clausal case the space requirements are usually more pronounced, the effects should be even more beneficial. Thus, the conclusion is the same for both cases: Even for long-running proof tasks there is no need to sacrifice completeness.

References

[HJL99] Th. Hillenbrand, A. Jaeger, and B. Löchner. System description: WALDMEISTER – Improvements in Performance and Ease of Use. In *Proc. CADE-16*, LNAI 1632, pages 232–236, 1999.

[LS01] B. Löchner and S. Schulz. An Evaluation of Shared Rewriting. 2nd International Workshop on Implementation of Logics (these proceedings), 2001.

[McC97] 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.

[RV99] A. Riazanov and A. Voronkov. Vampire. In *Proc. CADE-16*, LNAI 1632, pages 292–296, 1999.

[Vor01] A. Voronkov. Algorithms, datastructures, and other issues in efficient automated deduction (invited talk). In *Proc. IJCAR*, LNAI 2083, pages 13–28, 2001.

[Wei01] Chr. Weidenbach. Combining Superposition, Sorts and Splitting. In A. Robinson and A. Voronkov, editors, Handbook of Automated Reasoning, Elsevier, 2001.