Improving the Performance of Automated Theorem Provers by Redundancy-free Lemmatization

Joachim Draeger and **Stephan Schulz**
Institut für Informatik, Technische Universität München, Germany
{draeger,schulz}@informatik.tu-muenchen.de

Abstract

A promising way for solving "hard" problems with automated theorem provers is the lemmatization of the original problem. The success of this method, however, depends crucially on the selection of a subset of lemmata that are likely to be useful. The paper describes an algorithm implementing the selection of lemma-knowledge by eliminating redundant lemmata from set of potentially usable clauses. A lemma a is called redundant with respect to a lemma set F, if a can be generated with very few inferences from F. Conflicts between redundancies are resolved with an importance criterion. The promising results of first experiments are discussed.

Keywords: theorem proving, lemmatization, redundancy elimination

Introduction

When dealing with difficult problems, automated theorem provers (ATPs) are still inferior to skilled mathematicians. The lack of performance of ATPs in this area of applications is caused by the overwhelming size of the search space. Since ATPs typically perform a (heuristically guided) brute-force search, they cannot handle this situation very well. Humans, on the contrary, are looking very efficiently for a solution using more selective methods that partition the problem into easier subproblems. These facts suggest the utilization of a selective search modularization in ATPs as well. A simple way to follow this idea is the lemmatization of the original problem P (Astrachan & Stickel 1992; Draeger 1998a). The key idea is to replace some parts of a traditional proof search with lemmata, which represent multiple inferences with a single intermediate result that can be repeatedly applied using a single inference.

The resulting "simplified" proofs require fewer inferences; hence they will typically offer the chance to solve more problems within a reasonable time limit. Since it is not suitable to add a full set of systematically generated lemmata to the given problem specification P (Draeger 1998; Iwanuma 1997; Schumann 1994), a *selected* subset of lemma-knowledge has to be used for supporting the brute-force search. The fundamental difficulty of this approach is the discrimination between useful and useless lemmata (Shiroma 1996). The theorem prover has to identify those lemmata which are likely to simplify the proof without over-compensating this simplification by increasing the branching rate. This problem is often called the *utility problem* (Minton 1990). An approach for its handling was demonstrated successfully by the theorem prover AI-SETHEO (Draeger 1998).

In order to solve a problem P, which is assumed to be given as clause set including one or several queries, AI-SETHEO first generates unit lemmata f by systematically enumerating consequences of the problem specification. The resulting lemmata are evaluated by a so-called information measure $I(f) = p(f) \cdot r(f)$ (Draeger 1998a), which consists of a proof complexity measure $p(f)$ and a relevancy measure $r(f)$ (Draeger & Wolf 1999; Wolf & Draeger 1999). A lemma f is considered as useful and transferred to the knowledge base F, iff $I(f)$ has a large value; thus both uninteresting lemmata with a small relevance value as well as trivial lemmata with a low proof complexity are excluded (Draeger 1998). Then the standard model-elimination prover SETHEO (Moser *et al.* 1997; Goller *et al.* 1994) is applied to the formula $P \cup F$ resulting from adding F to the original problem P. The exclusion of seemingly useless lemmata from F reduces the administrative overhead of the search process to such an extent, that the overall performance of the prover system is improved (Draeger 1998a). Many more proofs are found with AI-SETHEO than with the application of SETHEO on P without lemma support (Draeger & Wolf 1999; Wolf & Draeger 1999). Indeed, the experiments have shown that this technique can be used to solve hard problems.

The most important element for the success of a controlled lemmatization is an evaluation function which selects only *nontrivial* lemmata, i.e. lemmata requiring many inferences for their proof. In the case of the information measure $I(f)$, the proof complexity measure $p(f)$ takes care of this part. However, the guaranteed non-triviality of $f \in F$ with respect to P does not exclude the case that f is trivial with respect to $P \cup F \setminus \{f\}$. In such a case, f is called *redundant* with respect to $F \setminus \{f\}$. Despite being useful on its own, f is not useful as element of F, because F and $F \setminus \{f\}$ represents the same state of knowledge. Experiments have given overwhelming evidence that every useless and hence superfluous lemma contained in a knowledge base leads to a considerable diminution of the prover performance. Thus, it is desirable to eliminate these *redundant* lemmata in the knowledge base F. This paper will discuss an algorithm performing such a redundancy elimination.

This paper is organized as follows. The next section describes the basic ideas of the redundancy elimination algorithm. The third section contains some remarks concerning the implementation, while the fourth section discusses the experimental results. The paper closes with some additional remarks in the final section.

Redundancy elimination

As defined in the introduction, a lemma f is called *redundant* with respect to $F \setminus \{f\}$, if it can easily be deduced[1] from $P \cup F \setminus \{f\}$. Such lemmata are useless, but they increase the administrative overhead of the prover considerably (Draeger 1998; 1998a). Hence, it is desirable to eliminate these redundant lemmata (Mauldin 1984). The resulting compression of the knowledge base rises the probability that the selected lemmata allow a solution of the given problem P within the existing practical resource constraints.

If we aim at the removal of as many redundant lemmata as possible, a brute-force algorithm will give the best results. However, this theoretically optimal compression cannot be achieved in practice, due to the super-exponential time complexity of the necessary brute force approach. Such an algorithm would have to test if a knowledge base F can be reconstructed from F' with a small effort for all possible $F' \subset F$.

Clearly, such a brute-force approach is not tractable in practice. In order to avoid a high computational complexity, we pursue an heuristic approach based on an iterative procedure instead. Whereas in the case of the brute-force approach, the effect of a lemma deletion is evident from the size of the compressed knowledge base F and the effort required for the reconstruction of the original knowledge base from F, now we must estimate this effect with indirect measures. Other complications are possible as well.

If the redundancy elimination is performed naively, then the removal of lemma after lemma eventually does not necessarily terminate with a plausible collection of lemmata, but instead can result in a trivial (or even empty) set. Consider the case of a sequence of lemmata $f_1, \ldots, f_n$, where each of the lemmata is a simple deduction from the preceding ones. The last lemma f_n can be recognized as redundant with respect to the remaining set of lemmata and consequently can be removed from F. During the next elimination step, however, one of the f_i, say f_{n-1}, can be recognized as redundant itself. In this way the removal of lemmata proceeds. No single step has a significant effect on F, but in the end the whole lemma set F can vanish. Such an uncontrolled iterative deletion has to be avoided. This is possible by the introduction of a flag set $u(f)$ that forbids the removal of a lemma that is required for reconstructing the original lemma set F_0 from the current knowledge base F. The flag set $u(f)$ has to assure that the lemmata contained in at least one of the sets $S_f^1, \ldots, S_f^n \subset F \setminus \{f\}$, from which the easy deduction of f is possible, remains in F. Of course, it would be contra-productive to mark the elements of all $S_f^1, \ldots, S_f^n$ as undeletable; this would not realize the full potential of the redundancy elimination procedure, because much more lemmata than necessary are retained. Instead, we will flag only the lemmata contained in one of the S_f^i against a future elimination. Formulated as pseudo-code, the resulting algorithm has the following preliminary form.

BEGIN
Construct proofs for $f \in F$ from $P \cup F$
$\forall f \in F \colon u(f) := 0$;; mark all lemmata as deletable
$F_0 := F$;; secure original knowledge base

while
  $\exists f \in F$, f redundant, $u(f) = 0$
do
  $F := F \setminus \{f\}$
  Let f be redundant w.r.t. $\{f_1, \ldots, f_n\} \subseteq F \setminus \{f\}$
  $u(f_1) := 1, \ldots, u(f_n) := 1$;; mark the f_i
  ;; as undeletable
done
END

First, all proofs for all $f \in F$ from $P \cup F$ that require only very few inferences are systematically produced and stored in a list L. All lemmata $f \in F$ are marked as undeletable, i.e. $u(f)$ is set to 0. The original knowledge base F is secured in $F_0 := F$. Now the elimination loop begins. A redundant lemma $f \in F$ with $u(f) = 0$ is eliminated from F; the redundancy is simply determined by a look-up in the proof list L.

Let $F' \subseteq F \setminus \{f\}$ denote one of the lemma sets, with respect to which which the lemma f is redundant; this means that a short proof for f exists that uses only the clauses contained in $P \cup F'$. The lemmata $\{f_1, \ldots, f_n\} := F'$ contained in F' are marked as undeletable, i.e. $u(f_1) := 1, \ldots, u(f_n) := 1$. They must remain in the knowledge base, because otherwise f would not necessarily be trivial with respect to the resulting final knowledge base. The described elimination is iterated as long as candidates for elimination exist.

In order to use the full potential of the redundancy elimination, we have to evaluate the importance of the

[1] Weather f is trivial or not, depends of course on the specific calculus and search procedure.

lemmata that should be deleted. This is helpful especially in the case of so-called *conflicts*. Let us consider the following situation. If an equational lemma $a = b$ is considered as useful and thus retained in the knowledge base F, it is very probable that F contains the lemma $b = a$ as well. Due to the symmetry axiom $X = Y \Rightarrow Y = X$ of the equality relation, these two lemmata f_1 and f_2 are trivial consequences of each other, and hence are redundant with respect to each other. In such cases, the redundancy elimination algorithm has to decide, which one of f_1 or f_2 should be deleted, and which one should remain in the knowledge base. This can be done with an *importance evaluation* of the lemmata $f \in F$. In the case of a conflict, the lemma with higher importance should remain in F. For evaluating the importance $E(f)$ of f, the algorithm determines the number $n(f)$ of different proofs found for f and the number $a(f)$ of applications of f in all proofs contained in the list L. A high value of $E(f) := (a(f) + 1)/(n(f) + 1)$ characterizes an important lemma f, because f is often applicable, but rarely produced. This discrimination allows a suitable handling of redundancy conflicts. The importance evaluation is implemented in the following way.

- We consider lemmata for elimination in order of ascending importance, i.e. the lemma f that is considered for elimination in the next iteration should have the lowest importance $E(f)$ of all possible candidates.
- The importance criterion can also be used to determine which lemmata should be marked as undeletable. As stated above, the algorithm selects a lemma set $S_f^i = \{f_1, \ldots, f_n\}$ and marks its elements as undeletable. This assures the reconstructability of the original knowledge base F_0 from the current F. Obviously, the performance of the elimination algorithm can be increased by choosing a suitable S_f^i. But what is a suitable S_f^i? Different options are possible. In the present implementation of the elimination algorithm, we select the lemma set S_f^i with the highest average importance $\sum_{f \in S_f^i} E(f)/|S_f^i|$.

We get the new formulation of the algorithm as shown below.

BEGIN
Construct proofs for $f \in F$ from $P \cup F$
$\forall f \in F$: $u(f) := 0$;; mark all lemmata as deletable
$F_0 := F$;; secure original knowledge base
$\forall f \in F$: compute $E(f)$;; Get importance ranking
while
    $\exists f \in F$, f redundant, $u(f) = 0$
do
    Let f be such a lemma with the lowest
        importance $E(f)$
    $F := F \setminus \{f\}$
    Let f be redundant w.r.t.
        $S_f^1, \ldots, S_f^n \subset F \setminus \{f\}$
    Let $\{f_1, \ldots, f_n\} := S_f^i$ be the S_f^i
        with the highest average importance $E(S_f^i)$
        where $E(S_f^i) := \sum_{s \in S_f^i} E(f)/|S_f^i|$
    $u(f_1) := 1, \ldots, u(f_n) := 1$;; mark $f_1, \ldots, f_n$
        ;; as undeletable
done
END

At the end of this section, a remark about the performance of the algorithm presented above should be made. Though it selects the elimination candidates carefully, the final result of the applied selection strategy is not always optimal. The importance evaluation $E(f)$ is based on local information only, and does not consider the effect of more than one step at a time. Thus, it is possible that the elimination of redundant lemmata $r_1, \ldots, r_n$, considered as suitable based on the available local data, turns out to be disadvantageous from a global point of view. The deleted lemmata $r_1, \ldots, r_n$ could eventually have led to significant eliminations later. The elimination of seemingly less suitable lemmata $r'_1, \ldots, r'_m$ at the beginning, on the contrary, might lead to the preservation of the lemmata $r_1, \ldots, r_n$, enabling the potentially more significant eliminations later. The heuristic approach tries to avoid such disadvantageous situations as much as possible, however, they cannot not be excluded totally. This is the trade-off one has to accept in exchange for the lower computational complexity of the iterative algorithm as compared to the full brute-force search.

Implementation

The proposed redundancy elimination algorithm was implemented in the theorem prover AI-SETHEO, which is based on the standard model elimination prover SETHEO (Moser *et al.* 1997; Goller *et al.* 1994). AI-SETHEO is a fully automated theorem proving system that adds modularity through lemmatization to the standard proof search. A description of the basic system is given in (Draeger 1998).

One of the main modules is the goal-oriented brute-force theorem prover SETHEO, which uses the model elimination calculus (Loveland 1968; 1978) for first-order clause logic as refined special tableau method. The other two main modules of AI-SETHEO are the lemma generator and the lemma selection procedure. The lemmatization mechanism includes the redundancy elimination algorithm as described in this paper and works as follows.

At first, AI-SETHEO tries to solve the original problem P in a certain amount of time, using SETHEO's standard proof procedure. If this attempt is unsuccessful, a set S of additional unit-lemmata is generated in a brute-force way by the DELTA-Iterator (Schumann 1994), which is also based on SETHEO. Then AI-SETHEO evaluates the information measure $I(f)$ for all $f \in S$. A lemma f considered as potentially useful is transferred to the knowledge base F. Now, the redundancy elimination algorithm is applied to F. For

Domain	# Hard Problems	SETHEO	AI-S. -RE	AI-S. +RE
BOO	52	5	1	21
CAT	29	2	15	25
COL	68	11	10	31
FLD	188	8	13	27
GEO	103	6	19	48
GRP	249	13	24	48
HEN	35	1	9	34
LAT	34	0	4	7
LCL	181	10	41	78
LDA	22	0	6	14
NUM	285	2	10	14
PUZ	15	3	7	6
RNG	82	3	5	17
SET	735	23	35	100
Others	421	15	13	28
Total	2499	102	212	498

Table 1: Successes on TPTP domains

efficiency reasons this algorithm is scheduled as a two stage process. In the first stage we use the saturating theorem prover E (Schulz 1999) in filter mode to eliminate clauses that are directly subsumed by others. In this mode, E just eliminates strictly redundant clauses, i.e. clauses which can be trivially deduced from just one parent clause. Such lemmata f represent redundancies of F in a trivial sense; they would be eliminated by the second stage representing the general redundancy elimination algorithm as well. However, saturating theorem provers are constructed to deal with large clause sets, and hence the simple filter operation is performed much more efficiently by E. This first stage typically reduces the size of the knowledge base considerably, making the much more expensive general redundancy elimination algorithm described in the previous section much cheaper to apply. Thus, the two stage approach reduces the total running time of the elimination process compared with a simple one stage process considerably.

The compressed knowledge base F produced by the redundancy elimination algorithm is then appended to the original formula P, and the whole proof process performed by AI-SETHEO is repeated with the lemmatized formula $P \cup F$ instead of P. This iteration process stops in the case of success or when a time limit is reached. As the next section shows, many more proofs are found in this way as with the brute-force application of SETHEO on P (Draeger 1998; Draeger & Wolf 1999; Wolf & Draeger 1999) alone.

Results

The performance of the redundancy elimination algorithm was tested on all hard problems contained in the problem collection TPTP (Sutcliffe*et al.* 1994), version 2.2.1. Here, a problem is called hard, if standard SETHEO can not solve it within 100 seconds. The results of the tests are shown in table 1.

The entries in the first two columns give the theory domain in the TPTP and the number of hard problems in this domain. The entries in the last three columns give the performances of SETHEO and both AI-SETHEO without (version presented at the conference AIMSA-98) and with (version presented at the conference TABLEAUX-00 and described in this paper) redundancy elimination for an overall time limit of 1000 seconds. In the case of AI-SETHEO, the timing includes all operations necessary for lemma handling. The experiments were carried out on Sun Ultra 10 workstations running at 300 MHz.

As can be seen, significantly more problems are solved with redundancy elimination than without. In the case of the 'hard' problems a performance increase by an average factor of 2.5 was reached for the version with redundancy elimination; in several domains like BOO, HEN, and RNG, the performance increase reached up to a factor of 3 or more. This improvement can be explained in the following way:

Due to the increased branching factor and resulting explosion in the search space, the number of clauses SETHEO can handle efficiently is limited. However, in order to reach the same *final* size of the knowledge base F, AI-SETHEO with redundancy elimination can afford to start with a (much) larger initial set F as a compensation for the expected size reduction. The removal of unimportant lemmata from F then leads to a higher diversity of the remaining elements in F or, formulated in another way, into an enlargement of the *effective* size of F. This effect raises the probability of finding a proof, because the higher diversity of F enables the prover to pursue more solution approaches simultaneously during the direct search for a proof. In other words, the larger effective size of F leads to a larger fault tolerance with respect to lemma selection. This increased robustness against erroneous lemma evaluations is an essential factor for the success of the redundancy elimination algorithm. It explains the larger number of successes for AI-SETHEO with redundancy elimination. Unfortunately, the degree of size reduction of the knowledge base F under the redundancy elimination algorithm can not be estimated in advance, because it is subject to strong variations. The final result strongly depends on the given proof problem. Hence we cannot give typical numbers for this parameter.

The redundancy elimination leads to other positive effects as well. Very often, many iteration steps of the proof procedure described in the implementation section are necessary until a proof for the given problem is found. In order to avoid an explosion of the size of the knowledge base, comparatively strong selection criteria for the lemmata have to be chosen. This can lead to a distinct bias in the remaining lemma knowledge. In effect, the search for useful new lemmata can converge prematurely, thus preventing the discovery of knowledge essential for the solution of a given problem. The higher diversity of the lemma knowledge resulting from the larger original size of the lemma set in the presence of redundancy elimination helps to avoid such a premature convergence. Thus, the proof search can reach solutions deeper in the search space than before.

Unfortunately, redundancy elimination is not always advantageous. It prefers the most general version of a lemma, which sometimes results in an unnecessarily large search space compared with a lemma specifically instantiated for the given problem. Hence it is possible, that in a few cases the search space is enlarged rather than reduced by the application of the redundancy elimination algorithm. Indeed, some solutions found by AI-SETHEO without redundancy elimination are lost in this way. Typical examples for this effect can be found in the puzzle domain PUZ of the TPTP library. For some problems contained in PUZ it is possible to construct lemmata which subsume more than 1000 other lemmata. In other domains, the situation is not so clear, but the evidence still pointing in the same direction (Draeger 2000).

Another mechanism also plays a role in limiting the potential performance increase. Some versions of the relevance measure component $r(f)$ of the information measure $I(f) = p(f) \cdot r(f)$ favour lemmata f which have a low syntactic complexity. One can argue in the following way (Draeger & Wolf 1999): The proof of a problem (if there is one) is finite; hence the number of useful lemmata is finite, too. On the other hand, the total number of lemmata which are valid in the underlying theory is typically not finite, or at least exceeds the number of useful lemmata by far. This means that the syntactic complexity of an arbitrary lemma on the average is much larger than the syntactic complexity of a useful lemma. Therefore limiting the syntactic complexity of a selected lemma will raise the probability that this lemma is useful for the construction of the actual proof. In respect to the redundancy elimination we have to note that many specifically instantiated lemmata are already eliminated during the lemma selection procedure, because they have a larger syntactical size than the corresponding general formulation[2]. Hence, the preference for syntactically simple lemmata in AI-SETHEO without redundancy elimination to a large degree favours those lemmata f which would be preferred by the redundancy elimination algorithm, too.

Outlook

The prospects of the redundancy elimination approach with respect to possible additional applications of theorem provers in the future are discussed in (Draeger 2000). All experiments carried out so far confirm the overall tendency that a better discrimination between useful and useless lemmata leads to a smaller knowledge base, and consequently improves the performance of the theorem prover. However, the full potential of controlled lemmatization has yet to be discovered.

[2]It is assumed here that the syntactical size of a lemma f is measured as the number of function, constant, and variable symbols contained in f.

References

Astrachan, O.; Stickel, M. 1992. Caching and Lemmaizing in Model Elimination Theorem Provers. In *CADE-92*, LNCS 607

Draeger, J. 1998. Acquisition of Useful Lemma Knowledge in Automated Reasoning. In *AIMSA-98*, LNCS 1480

Draeger, J.; 1998a. *Modularisierte Suche in Theorembeweisern.* Ph.D. Dissertation, Technical University Munich.

Draeger, J.; 2000. Redundancy Elimination in Lemma-Knowledge for Automated Theorem Provers. In *IC-AI-00*, CSREA Press 2000, p. 1305

Draeger, J.; Wolf, A.; 1999. Strategy Parallelism and Lemma Evaluation. In *FLAIRS-99*

Goller, C.; 1997. A Connectionist Approach for Learning Search-Control Heuristics for Automated Deduction Systems. Ph.D. Dissertation, Technical University Munich

Goller, C.; Letz, R.; Mayr, K.; Schumann, J.; 1994. SETHEO V3.2: Recent Developments. In *CADE-94*, LNAI 814

Iwanuma, K.; 1997. Lemma Matching for a PTTP-based Top-down Theorem Prover. In *CADE-97*, LNCS 1249

Loveland, D.; 1968. Mechanical Theorem-Proving by Model Elimination. JACM 15(2)

Loveland, D.; 1978. Automated Theorem Proving: a Logical Basis. North-Holland

Mauldin, M.; 1984. Maintaining Diversity in Genetic Search. In *NCAI-84*

Minton, S. 1990. Quantitative Results Concerning the Utility of Explanation-Based Learning. Artificial Intelligence 42(1990)363

Moser, M.; et al. 1997. SETHEO and E-SETHEO. The CADE-13 Systems. *JAR* 18(2):237

Schulz, S. 1999. System Abstract: E 0.3. In Ganzinger, H., ed., *Proc. of the 16th CADE, Trento*, number 1632 in LNAI, 297–391. Springer.

Schumann, J.; 1994. DELTA — A Bottom-up Preprocessor for Top-Down Theorem Provers. In *CADE-94*, LNAI 814

Shiroma, P.; 1996. Efficient production planning and scheduling. Gabler Verlag.

Sutcliffe, G.; Suttner, C.; Yemenis, T. 1994. The TPTP Problem Library. In *CADE-94*, LNAI 814

Wolf, A.; Draeger, J.; 1999. Strategy Parallelism and Lemma Evaluation. In *TABLEAUX-99*, LNCS 1617