Similarity-Based Lemma Generation for Model Elimination

Marc Fuchs[*]

1 Introduction

ATP systems based on model elimination (ME) [Lov78] or the connection tableau calculus [LMG94] have become more and more successful in the past. These provers traverse a search space that contains *deductions* instead of clauses as in the case of resolution procedures. The structure of the search space allows on the one hand for efficient implementations using prolog-style abstract machine techniques. On the other hand, efficient search pruning techniques are possible. A problem regarding the use of ME is the fact that ME proof procedures are among the weakest procedures when the length of existing proofs is considered. Furthermore, ME involves a huge amount of redundancy because the same subgoals have to be proven over and over again during the proof process.

The use of lemmas offers a solution to both of these problems. The aim of *lemmaizing* is to store successful sub-deductions in form of lemmas and to support the calculus with these *bottom-up* generated formulae. But although lemmas can help to decrease the proof length they also increase the branching rate of the search space. Hence, an unbounded generation of lemmas without using techniques for filtering *relevant* lemmas does not seem to be sensible. Actually, filter mechanisms are in use that ignore the proof task at hand (cp. [AS92, Sch94]). A method that seems to be more appropriate is the use of *similarity criteria* between the proof goal and a possible lemma (*lemma candidate*) in order to filter lemmas. In the following we will present a model for similarity-based lemmaizing.

2 Connection Tableau Calculus

The connection tableau calculus (CTC) works on clausal tableaux and consists of three inference rules. Let $\mathcal{C}$ be a given set of clauses to be refuted. The *start* rule allows for a conventional tableau expansion applied to the trivial tableau which consists of only one unlabeled node. Tableau expansion attaches a clause from $\mathcal{C}$ at the leaf node of a tableau (see [Fit96]). *Reduction* allows for closing an open branch by unifying the literal at the leaf with the complement of another literal on the same branch. *Extension* is a combination of tableau expansion and immediately performing a reduction step.

In order to refute a clause set with CTC a closed tableau has to be enumerated starting with the trivial tableau. Normally implicit enumeration procedures are employed performing consecutively iterative deepening search with backtracking (cp. [Sti88]). Iteratively, larger finite segments of the search tree $\mathcal{T}$ which is the tree of all possible derivations of tableaux are explored. The segments are defined by so-called *completeness bounds* which pose structural restrictions on the tableaux which are allowed in the current segment. Prominent examples for such bounds are the *depth* and *inference* bound (see [LSBB92]). The depth bound limits the maximal depth of nodes (the root node has depth 0) in a tableau (ignoring leaf nodes) according to a fixed *resource* $n \in \mathbb{N}$, i.e. only tableau with depth smaller or equal to n are allowed. Analogously, the inference bound limits the number of inferences allowed for the derivation of a tableau according to a resource value. Iterative deepening, using a bound $\mathcal{B}$, is performed by starting with a basic resource n

[*]Fakultät für Informatik, 80290 München, `fuchsm@informatik.tu-muenchen.de`

and iteratively increasing n until a proof is found within the finite initial segment of $\mathcal{T}$ defined by $\mathcal{B}$ and n.

3 Bottom-Up Lemma Generation

One approach to enrich CTC is to employ lemmas. If a literal s is head of a closed subtableau T and the literals $l_1, \ldots, l_n$ are used to perform reduction steps from outside T then the clause $\neg s \vee \neg l_1 \vee \ldots \vee \neg l_n$ may be derived as a new lemma. One can generate lemmas *dynamically* during the proof run as in [AS92] or *statically* in a preprocessing phase as in [Sch94]. In the rest of this paper we want to consider the last method where *unit lemmas* are added to the clause set $\mathcal{C}$ to be refuted. Lemmas are generated by adding queries $\neg p(X_1, \ldots, X_n)$ (and $p(X_1, \ldots, X_n)$ in the case of non-horn clauses) to $\mathcal{C}$ and enumerating all solution substitutions $\sigma_1, \ldots, \sigma_m$ which can be obtained in the segment of the search tree defined by the depth bound and a fixed (small) resource n_D. Subsumed facts from $\{\sigma_i(p(X_1, \ldots, X_n)) : 1 \leq i \leq m\}$ are deleted, resulting in a set $\mathcal{L}_0$. Then a top-down proof run takes place using $\mathcal{C} \cup \mathcal{L}_0$ and a fixed bound (the depth bound in [Sch94]). With this method always a reduction of the proof length can be obtained when using horn clauses (see [Fuc97a]). More interesting is whether the process of *finding* a proof can profit from using lemmas ("macro operators") since the branching rate of the search tree increases.

Let us first consider the case that the *depth bound* $\mathcal{B}_D$ is used for the final top-down proof run. Let n be the resource which is at least needed to obtain a proof when using no lemmas. We assume that the resource which is at least needed when using lemmas from $\mathcal{L}_0$ can be reduced from n to $n - n_D + 1$. This is always possible in the case of horn clauses. Let $\mathcal{T}_1$ or $\mathcal{T}_2$ be the smallest segments (w.r.t. $\mathcal{B}_D$) of the search trees which are defined by $\mathcal{C} \cup \mathcal{L}_0$ or $\mathcal{C}$, respectively, that include a proof. In $\mathcal{T}_1$, despite the use of lemmas, there are *no new solutions* of subgoals compared with $\mathcal{T}_2$ because a lemma application cannot help to introduce solutions that cannot already be obtained in $\mathcal{T}_2$ by expanding the lemma proofs. Note further that when employing local failure caching (cp. [LMG94]) no duplication of segments of the search space (by duplicated solutions of subgoals) is incorporated into $\mathcal{T}_1$ which is in opposite to classical macro operator learning (cp. [Min90, Fuc97a]). Instead, inferences possible in $\mathcal{T}_2$ are *spared*, e.g. because of the subsumption test when generating lemmas (see also [Fuc97a]).

Furthermore, the number of inferences needed to explore a small search space in the preprocessing phase is normally much smaller than the number of inferences saved. In addition, the use of lemmas leads to a *restructuring* of the order in which solutions of subgoals are obtained. This may help to save the possibly large search amount for proving a useful lemma and speed up the search dramatically. When using the *inference bound* similar effects occur. But it may be that new solution substitutions are introduced to $\mathcal{T}_1$ that are not possible in $\mathcal{T}_2$ (cp. [Fuc97a]).

A dramatic improvement for the search process can be obtained by using only some *relevant* lemmas, i.e. lemmas needed in a proof. Then, one can profit from a resource reduction and does not pay this benefit for a large increase of the branching rate. Thus, mechanisms to filter relevant lemmas will be developed in the following.

4 Similarity-Based Lemma Selection

In this section we want to introduce general principles for similarity-based lemma selection. We introduce our notion of similarity and we deal with principles for a priori estimating this similarity. Then, we introduce some concrete distance measures for selecting lemmas.

4.1 General Principles for Measuring Similarity

We want to make the notion of relevance of a lemma set for a proof goal (*a posteriori similarity*) more precise. Let $\mathcal{C}$ be a set of clauses, $S \in \mathcal{C}$ be a start clause for refuting $\mathcal{C}$, and $\mathcal{L}_0$ be a set of lemma candidates. Let $\mathcal{L} \subseteq \mathcal{L}_0$ be a set of unit clauses. We say S and $\mathcal{L}$ are *similar* ($Sim_{\mathcal{T}}(S, \mathcal{L})$ holds) w.r.t. a search tree $\mathcal{T}$ if there is a closed tableau T in $\mathcal{T}$ that can be reached with start clause S and that contains as tableau clauses only instances of clauses from $\mathcal{C} \cup \mathcal{L}$. Furthermore, at least one $l \in \mathcal{L}$ is used for closing a branch. Not every set $\mathcal{L}$, similar to S w.r.t. the search tree defined by $\mathcal{C} \cup \mathcal{L}_0$, may be useful for refuting $\mathcal{C}$ when employing a bound $\mathcal{B}$. An important quality criterion is the resource value n which is at least needed in order to obtain a closed tableau T. There should be no subset of $\mathcal{L}_0$, different from $\mathcal{L}$, that can help to refute $\mathcal{C}$ with smaller resources. Furthermore, $\mathcal{L}$ should have minimal size, i.e. no lemma can be deleted in order to refute $\mathcal{C}$ with resource n. We call such lemma sets *most similar* to S w.r.t. $\mathcal{B}$.

Conventional tableau enumeration procedures allow for finding a subset of $\mathcal{L}_0$ most similar to S. The following method (*lemma delaying* tableau enumeration) is a sound and complete test for $Sim_{\mathcal{T}_m}(S, \mathcal{L})$ where $\mathcal{T}_m$ is the finite search tree defined by $\mathcal{C} \cup \mathcal{L}_0$, start clause S, a completeness bound $\mathcal{B}$, and the resource n_m which is at least needed to refute $\mathcal{C} \cup \mathcal{L}_0$: We enumerate the set $\mathcal{O}$ of all tableaux in $\mathcal{T}_m^{\mathcal{C}}$ defined by $\mathcal{C}$, S, $\mathcal{B}$, and n_m. Then we check whether or not the open subgoals (*front clause*) of a tableau $T \in \mathcal{O}$ can be solved by extension with some facts of $\mathcal{L}$ such that the resulting tableau is in $\mathcal{T}_m$.

With the help of *similarity measures* we want to estimate *a priori* whether $\mathcal{L} \subseteq \mathcal{L}_0$ is most similar to S. We compute such measures based on a fast *simulation* of the deduction process. We want to enumerate only a subset of the set of front clauses belonging to the tableaux from $\mathcal{O}$ and compute some *unification distances* between the generated front clauses and the lemma set $\mathcal{L}$. In order to simplify this we restrict ourselves to *local* tests and try to estimate the usefulness of a lemma in order to contribute to a refutation of $\mathcal{C}$ in $\mathcal{T}_m$ (normally together with other lemmas). Instead of generating front clauses we only enumerate front literals and choose lemmas based on distances to front literals. If we want to employ local tests, however, it is not sufficient to generate front clauses that are units (*front literals*) in $\mathcal{T}_m^{\mathcal{C}}$ and then perform unification tests. Front literals have to be generated in an "expanded" search tree $\mathcal{T}_{m,e}^{\mathcal{C}}$. This search tree contains in addition to $\mathcal{T}_m^{\mathcal{C}}$ the tableaux which can be obtained from tableaux of $\mathcal{T}_m^{\mathcal{C}}$ by expanding some subproofs of applicable lemmas from $\mathcal{L}_0$. The usefulness of a fact $l \in \mathcal{L}_0$ can be determined by generating all front literals in $\mathcal{T}_{m,e}^{\mathcal{C}}$ and performing a unification test between these literals and the complement of l. By generating a subset of the front literals and then measuring a (unification) distance between the front literals and the complement of l ($\sim l$) the usefulness of l can be estimated. The aim of this local method is to select with small costs a rather small lemma set similar to S w.r.t. $\mathcal{T}_m$.

4.2 Similarity Measures

In this section we want to clarify how to generate front literals, how to measure unification distances and how to select lemmas based on the distance values to front literals.

The aim of the generation of front literals is to get a good "coverage" of the search space $\mathcal{T}_{m,e}^{\mathcal{C}}$. For each useful lemma a structural similar front literal should be given. This is no easy task since a priori not even the value n_m is given and thus the form of $\mathcal{T}_{m,e}^{\mathcal{C}}$ is unknown. We handle this problem by employing breadth-first search. We enumerate all front literals occurring in tableaux which can be reached with at most k ($\in \mathbb{N}$) inference steps. Thus, we can follow each inference chain needed to obtain a front literal f^l (in $\mathcal{T}_{m,e}^{\mathcal{C}}$) that is unifiable with a lemma l until a certain depth. If we are able to generate a front literal f where inferences are performed similar to those needed to find f^l then it is quite probable that f and l are structurally similar (as described shortly). We used an inference value k which defines a compromise between computation effort and precision of the simulation and produced up to 500 front literals. Note that this method was in our experiments sound in the sense that the set of the generated front literals has always been a subset of the set of front literals $\mathcal{F}$ occurring in tableaux from $\mathcal{T}^{\mathcal{C}}_{m,e}$.

Now, we want to estimate, based on the structural differences between front literals and a fact $l \in \mathcal{L}_0$, whether or not, when allowing for more inferences when generating front literals, a front literal $f^l \in \mathcal{F}$ can be produced that can be closed with l. Due to the lack of inference resources instead of f^l only a front literal f may be generated which differs from f^l as follows: Either, it is possible that all inferences that have to be performed to literals on the path from S to f^l of the tableau T_{f^l} (whose front literal is f^l) can also be performed when creating f. Thus, some of the subgoals occurring in T_{f^l} have been solved, when generating f, by a sub-proof somewhat different to the sub-proof in T_{f^l}. Since the subgoals are variable-connected "unification failures" arise in f that prevent $\sim f$ and l from being unifiable. Or, it may be that inferences which have to be performed to literals on the path from S to f^l cannot be performed when producing a front literal f. Then, f and l may even differ in the symbol at the top-level position. Now, we want to distinguish two cases: It may be that in order to close f^l with l no instantiations are needed or only instantiations with terms of a small size. Otherwise, it may be necessary that instantiations with terms of larger size take place.

If no instantiations of f^l and l are needed a complete similarity test consists of a test of *structural equality* between front literals from $\mathcal{F}$ and l. As described above this has to be weakened to a *structural similarity* test. A common method in order to allow for such a similarity test of structures is to employ *features.* A feature is a function φ mapping literals to natural numbers. Usually, a set of features $\varphi_1, \ldots, \varphi_n$ is used and a literal u is represented by its *feature value vector* $(\varphi_1(u), \ldots, \varphi_n(u))$. We consider a lemma to be useful if a front literal has similar feature values. The features we have used concentrate on simple syntactical properties (cp. [Fuc97a, Fuc97b]). We define a distance $d_F(u,v)$ of two literals as the Euclidean distance d_E of the feature value vectors of $\sim u$ and v.

If instantiations are needed in order to use lemmas normally the previous method will not allow for good estimations. Therefore, in [Fuc97a] an instantiating method based on an inference system for unification has been developed. An instantiation (substitution) is computed by *pseudo-unifying* two literals (terms) based on an inference system $\mathcal{UD}$ which ignores certain unification failures during the unification process. A definition of $\mathcal{UD}$ and remarks for controlling $\mathcal{UD}$ can be found in [Fuc97a]. After instantiating two terms d_F can be employed. In summary, we obtain the measure d_S by $d_S(u,v) = d_F(\sigma^v_u(u), \sigma^v_u(v))\}$ where σ^v_u is the substitution obtained with $\mathcal{UD}$ for $\sim u$ and v.

In order to *select* lemmas we employ the following method which is very flexible w.r.t. the number of selected lemmas: We generate for each clause which should serve as start clause front literals and choose for each front literal the lemma most similar to the literal w.r.t. a given distance measure (d_F or d_S). If for a lot of front literals the same lemma seems to be well suited (able to conclude a proof) a small subset of $\mathcal{L}_0$ will be chosen. Otherwise, if there is a high uncertainty which lemmas are well suited, a higher number of lemmas will be chosen which may prevent useful lemmas from being ignored.

5 Experiments

We have performed experiments with the CTC based prover Setheo [LSBB92] and the lemma generator Delta [Sch94]. We experimented in three domains of the TPTP library, namely in the BOO, COL, and GRP domain. We employed Setheo, an unfiltered combination of Setheo and Delta (Setheo/Delta), a version where filtering was done according to conventional criteria as used in [Sch94] (Setheo/Conv, see also [Fuc97a]), and our

Problem	Setheo	Setheo Delta	Setheo Conv	Setheo Sim	Problem	Setheo	Setheo Delta	Setheo Conv	Setheo Sim
BOO003-1	35	19	1	3	COL003-2	—	481	238	705
BOO003-2	344	—	377	2	COL042-2	—	983	—	58
BOO004-2	497	—	628	2	COL042-3	—	—	—	907
BOO005-2	629	—	110	21	COL042-4	—	—	—	520
BOO006-2	387	—	42	25	COL060-2	532	264	—	11
BOO006-4	141	503	5	8	COL060-3	509	252	—	8
BOO012-2	—	—	—	131	COL063-5	541	267	—	37
BOO012-4	—	—	—	408	COL063-6	539	262	—	15
BOO013-1	8	—	—	3	COL064-2	567	279	—	22
BOO013-3	485	44	—	2	COL064-3	566	275	—	23

Table 1: Experimental Results

similarity-based version SETHEO/Sim. In the BOO and GRP domain non-instantiating methods were sufficient whereas in the COL domain the measure d_S was used. We used the depth bound and iterative deepening. Lemmas were generated with resource $n_D = 2$.

We depict some problems from the BOO and COL domain in table 1. We show the run times in seconds on a Sun Ultra II. The entry '—' denotes that no proof could be found within 1000 seconds. Considering our results we can observe that SETHEO/Sim significantly improves on the other versions. We can solve new problems as well as decrease the run times of a lot of problems from a region of more than 5 minutes to a value smaller than 1 minute. Note that the run times include the time for lemma generation and selection. Similar results could be obtained *all over* the domains in a stable way. A deeper analysis and further information can be found in [Fuc97a].

6 Conclusions and Future Work

Lemmaizing techniques have the potential to significantly improve the performance of an automated theorem proving system. We discussed static lemma generation and investigated the potential of lemmaizing for proof length and resource reduction. In order to solve the problem of introducing relevant lemmas, we developed efficient techniques based on similarity criteria between a proof goal and a set of lemma candidates. The future work will deal with the improvement of our similarity measures. It remains to be investigated whether a more intelligent generation of front clauses can improve the reliability of our similarity test. Furthermore, techniques from the area of machine learning may allow for an improvement of the similarity measures d_S and d_F.

References

[AS92] O.L. Astrachan and M.E. Stickel. Caching and Lemmaizing in Model Elimination Theorem Provers. In *Proceedings of CADE-11*, pages 224–238, Saratoga Springs, USA, 1992. Springer LNAI 607.

[Fit96] M. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer, 1996.

[Fuc97a] M. Fuchs. Principles of Similarity-Based Lemmaizing. AR-Report, AR-97-02, TU München.

[Fuc97b] M. Fuchs. Flexible Proof-Replay with Heuristics. In *Proc. EPIA-97 (to appear)*, 1997.

[LMG94] R. Letz, K. Mayr, and C. Goller. Controlled Integration of the Cut Rule into Connection Tableau Calculi. *Journal of Automated Reasoning*, (13):297–337, 1994.

[Lov78] D.W. Loveland. *Automated Theorem Proving: a Logical Basis.* North-Holland, 1978.

[LSBB92] R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A High-Performance Theorem Prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.

[Min90] S. Minton. Quantitative Results Concerning the Utility of Explanation-Based Learning. *Artificial Intelligence*, (42):363–391, 1990. Elsevier Science Publishers.

[Sch94] J. Schumann. Delta - a bottom-up preprocessor for top-down theorem provers. system abstract. In *Proceedings of CADE-12*. Springer, 1994.

[Sti88] M.E. Stickel. A prolog technology theorem prover: Implementation by an extended prolog compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.