TECHNISCHE
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

Graduiertenkolleg:
Kooperation und Ressourcenmanagement
in verteilten Systemen

Similarity-Based Lemma Generation with Lemma-Delaying Tableaux Enumeration

Marc Fuchs

TUM-I9743
SFB-Bericht Nr. 342/26/97
August 97

TUM–INFO–08-I9743-150/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

Similarity-Based Lemma Generation with Lemma-Delaying Tableaux Enumeration

Marc Fuchs
Fakultät für Informatik, TU München
80290 München
Germany
fuchsm@informatik.tu-muenchen.de
Tel.: +49 89 289 27916 Fax: +49 89 289 27902

July 1997

Abstract

The use of bottom-up generated lemmas is an appropriate method for achieving an effective redundancy control mechanism for connection tableau search procedures as well as for reducing proof lengths. An unbounded use of lemmas, however, usually does not lead to improvements of the search process and can even complicate the search. In order to identify lemmas which are actually relevant for the proof task we develop methods for a similarity-based filtering of lemmas. Similarity between a proof goal and a lemma is measured by simulating a costly but complete tableaux enumeration method (the lemma delaying method) which uses all lemmas to prove the goal and estimating based on this simulation which lemmas may be useful. We investigate this method in detail and introduce several similarity measures. Experiments with the prover SETHEO and the lemma generator DELTA underline the effectiveness of the newly developed methods.

1 Introduction

Automated Theorem Proving (ATP) systems based on model elimination (see [Lov78]) or the connection tableau calculus (see [LMG94]) have become more and more successful in the past. One main reason for this is that these provers traverse a search space that contains *deductions* instead of clauses as in the case of resolution procedures. The special structure of the search space allows on the one hand for highly efficient implementations using prolog-style abstract machine techniques. On the other hand, efficient search pruning techniques are possible (cf. [LMG94]).

A problem regarding the use of model elimination proof procedures is the fact that these procedures are among the weakest procedures when the length of existing proofs is considered. Furthermore, the same proof goals have to be proven over and over again during a proof run. Thus, mechanisms to reduce the proof length and to avoid redundant sub-computations are needed.

One method which is appropriate for fulfilling both of these tasks is to employ (generalized) lemmas. Successful sub-deductions are stored in form of lemmas and are added to the clauses to be refuted. The lemmas can provide a proof length reduction as well as can help to avoid some repeated or subsumed sub-deductions. Often, the employed tableaux enumeration procedures can significantly profit from the proof length reduction obtained. The use of lemmas, however, also imports additional redundancy into the calculus. Hence, an unbounded generation of lemmas without using techniques for filtering *relevant* lemmas does not seem to be sensible.

In order to filter relevant lemmas we want to use *similarity criteria* (as introduced in [Fuc97a, Fuc97b]). The idea is, when judging whether a fact should become a lemma, not only to consider the fact itself but also the concrete proof task. With the help of similarity criteria between the proof goal and a possible lemma (*lemma candidate*) we shall define new methods for judging relevancy used to introduce lemmas.

In this report, we want to investigate similarity tests based on a *simulation* of a certain tableaux enumeration procedure, namely the *lemma delaying* tableaux enumeration procedure. We start by giving a brief overview about the connection tableau calculus and our techniques for generating lemma candidates in section 2 and 3, respectively. After that we introduce the principles of lemma generation based on lemma delaying tableaux enumeration procedures in section 4. First methods for measuring similarity are described in section 5 which correspond to the methods introduced in [Fuc97a]. The subsequent sections 6 and 7 deal with the development of some new distance measures. An experimental study in section 8 underlines the benefits of our approach. Finally, a discussion and comparison with other existing approaches which employ a notion of similarity or difference is provided in section 9.

2 Connection Tableau Proof Procedures

2.1 The Connection Tableau Calculus

In this section we want to give a brief overview about refutational theorem proving with connection tableau calculi. For a more detailed description we refer the reader to [LMG94]. We assume the reader to be familiar with notions like literals, terms, substitutions, and positions in terms (trees).

We are interested in refuting a set of (input) clauses $\mathcal{C}$. In order to introduce the *connection tableau calculus*, we want to start with the basic (free variable) tableau calculus [Fit96] for clauses. Basically, a calculus consists of a set of objects $\mathcal{O}$ it works on. Furthermore, a set of inference rules $\mathcal{I}$ is needed in order to manipulate the objects. In our context, the set of objects $\mathcal{O}$ is given by the set of clausal tableaux for $\mathcal{C}$. A tableau T for $\mathcal{C}$ is a tree whose non-root nodes are labeled with literals and that fulfills the condition: If the immediate successor nodes $v_1, \ldots, v_n$ of a node v of T are labeled with literals l_i, then the clause $l_1 \vee \ldots \vee l_n$ (*tableau clause*) is an instance of a clause in $\mathcal{C}$. The set of inference rules $\mathcal{I}$ consists of two inference rules, namely the *expansion* and the *reduction* rule (see [Fit96]). An application of the expansion rule means selecting a clause from $\mathcal{C}$ and attaching its literals to a *subgoal* s which is a literal at the leaf of an *open* branch (a branch that does not contain two complementary literals). Tableau reduction closes a branch by unifying a subgoal s with the complement of a literal r (denoted by $\sim r$) on the same branch, and applying the substitution to the whole tableau.

Connection tableau calculi work on connected tableaux. A tableau is called *connected* or a *connection tableau* if each inner node v (non-leaf node) which is labeled with literal l has a leaf node v' among its immediate successor nodes that is labeled with a literal l' complementary to l. The inference rules consist of *start*, *extension*, and *reduction*. The start rule allows for a standard tableau expansion that can only be applied to a trivial tableau, i.e. one consisting of only one node. Extension is a combination of expansion and reduction. It is performed by selecting a subgoal s in the tableau T, applying an expansion step to s, and immediately performing a reduction step with s and one of its newly created successors.

Important is the notion of a tableau derivation and a (tableau) search tree: We say $T \vdash T'$ if (and only if) tableau T' can be derived from T by applying a start rule if T is the trivial tableau, or by applying an extension/reduction rule to a subgoal in T. The reflexive and transitive closure of $\vdash$ is denoted by $\overset{*}{\vdash}$. The connection tableau calculus is not (proof) confluent: In order to show the unsatisfiability of a clausal set $\mathcal{C}$, a search tree given as follows has to be examined in a *fair* way (each node of the tree must be visited after a finite amount of time) until a closed tableau occurs (a tableau with no open branch). A *search tree* $\mathcal{T}$ for a set of clauses $\mathcal{C}$ is given by a tree, whose root is labeled with the trivial tableau. Each inner node in $\mathcal{T}$ labeled with tableau T has as immediate successors the maximal (finite) set of nodes $\{v_1, \ldots, v_n\}$, where v_i is labeled with T_i and $T \vdash T_i$, $1 \leq i \leq n$. The set of nodes of $\mathcal{T}$ is denoted by $N(\mathcal{T})$.

Note that in the case of horn formulae it is possible to restrict the applicability of extension steps in such a manner that extensions are only allowed to a negative subgoal. Furthermore, no reduction steps are needed in order to get a complete calculus. Thus, in the following we want to consider such a restricted calculus when dealing with horn clauses.

2.2 Tableaux Enumeration

Since, in comparison with resolution style calculi, not only the number of proof objects but also their size increases during the proof process, explicit tableaux enumeration procedures that construct all tableaux in $\mathcal{T}$ in a breadth-first manner are not sensible. Hence, implicit enumeration procedures are normally in use that apply *consecutively bounded iterative deepening search with backtracking*. In this approach iteratively larger finite segments of the search tree $\mathcal{T}$ which contains all possible inference chains of tableaux are explored (in depth-first search). Normally, the finite segments are defined by so-called *completeness bounds* which pose structural restrictions on the tableaux which are allowed in the current segment. A completeness bound $\mathcal{B}$ defines w.r.t. a fixed natural number, a so-called *resource*, a finite initial segment of the search tree. Iterative deepening is performed by starting with a basic resource value $n \in I\!N$ and iteratively increasing n until a proof is found within the finite initial segment of $\mathcal{T}$ defined by a bound $\mathcal{B}$ and n. Prominent examples for search bounds are the *depth bound*, *inference bound, weighted-depth bound*, and *copy bound*.

The depth bound limits the maximal depth of nodes in a tableau (ignoring leaf nodes) where the root node has depth 0. In practice, the depth bound is quite successful (cf. [LMG94, Har96]) but it suffers from the large increase of the finite initial segment (defined by a resource n) caused by an increase of n. The inference bound allows for a level by level exploration of the search tree (cf. [Sti88]). In comparison with the depth bound, the inference bound allows for a smooth increase of the search space, but the bound is inferior to the depth bound in practice. In order to combine the advantages of depth and inference bound the weighted-depth bound was introduced in [MIL$^+$97]. Basically, this bound gives a fixed resource value to a subgoal depending on the current depth of the subgoal in its associated tableau. But furthermore, depending on the inferences performed so far, an additional amount of resources is given to the subgoal. Thus, the number of inferences allowed in a segment of the search tree can be reduced by decreasing this additional part if a lot of inferences have been performed so far. In practice the bound has proven to be very successful. The copy bound limits the occurrences of instantiations of input clauses in a tableau. A fixed number limits the number of "copies" of each clause in a tableau. In practice the copy bound proved to be inferior to the bounds mentioned above.

An interesting property of connection tableau search procedures is their independence of the chosen subgoal selection strategy. In order to obtain a complete enumeration strategy it is possible to use an arbitrary selection strategy which tries to solve the open subgoals of a tableau in a fixed order (cp. [LMG94]).

3 Bottom-Up Lemma Generation

A simple definition of lemma use in model elimination is to record seemingly useful solutions of subgoals as lemmas and to augment the input clauses with these bottom-up generated formulae. Assume that a literal s is associated with the root node of a closed sub-tableau T^s. Let $l_1, \ldots, l_n$ be the literals that are used in reduction steps and that are outside of T^s. Then, the clause $\neg s \vee \neg l_1 \vee \ldots \vee \neg l_n$ may be derived as a new lemma (since it is a logical consequence of the tableau clauses in T^s) and added to the input clauses. From now on, we want to consider only *unit lemmas* because of their ability to close open branches and not to simultaneously introduce new open subgoals.

There are a lot of possibilities of how to generate lemmas. On the one hand lemmas can be generated dynamically during the proof run (cf. [AS92, Iwa97, AL97]). On the other hand lemmas can be generated statically in a preprocessing phase (cf. [Sch94]). We want to concentrate on static lemma use in the following. In this approach in a preprocessing phase the "bottom-part" of a search space $\mathcal{T}^{BU}$, which is defined by a clause set $\mathcal{C}$, a completeness bound $\mathcal{B}^{BU}$, and a fixed resource value n^{BU}, is discovered and *unit lemmas* are generated as follows:

Most general queries $\neg P(x_1, \ldots, x_n)$ (and $P(x_1, \ldots, x_n)$ in the case of non-horn clauses) are added to $\mathcal{C}$ for each predicate symbol P. For those queries all solution substitutions $\sigma_1, \ldots, \sigma_m$ that can be obtained within $\mathcal{T}^{BU}$ are enumerated. The unit clauses $l_1 = P(\sigma_1(x_1), \ldots, \sigma_1(x_n)), \ldots, l_m = P(\sigma_m(x_1), \ldots, \sigma_m(x_n))$ are produced as lemmas after subsumption deletion and deleting facts from $\mathcal{C}$, resulting in a set $\mathcal{L}_0$. Further iterations of this process are imaginable by starting a new iteration with the clause set $\mathcal{C} \cup \mathcal{L}_i$ and collecting the newly generated clauses in $\mathcal{L}_{i+1}$. The unit literals generated in this preprocessing phase, which finishes after a fixed number of cycles, are possible lemma candidates that can be added to the input clauses.

This kind of lemma generation mainly aims to reach a reduction of the proof length. When using a suitable bound $\mathcal{B}^{BU}$ for lemma generation (e.g., the depth bound) always a reduction of the proof length is possible in the case of horn clauses. When using non-horn clauses it may be that no proof length reduction is possible (cf. [Fuc97a]). By obtaining a proof length reduction it is possible to save the search effort for proving a useful lemma. Furthermore, when employing the "right" bound for the final top-down proof search (e.g., the inference or the depth bound) a proof can always be found with lower resources in the case of horn formulae (see also [Fuc97a]). This might lead to a dramatic improvement of the performance of the prover.

But although lemmas provide the possibility to reach a proof with a smaller number of inferences, they simultaneously increase the branching rate of the search tree. Thus, it is quite probable that an unfiltered use of all lemmas generated in the preprocessing phase will not allow for a speed-up of the search for the proof. Hence, methods for filtering relevant lemmas are needed. It is clear that relevant means in this context being able to contribute to a proof.

So far only simple filter criteria ignoring the proof goal have been used. Criteria like the length (number of symbols) or nesting depth of a fact have been applied in order to decide whether a fact should become a lemma (added to the input clauses) or not. Facts with small values are chosen because it is more probable that these facts can close open subgoals. It is clear, however, that a consideration of the proof task at hand may allow for an improved measuring of the quality of lemmas. The next chapter describes general principles for a similarity-based filtering of lemmas that considers the given clause set to be refuted when judging the quality of facts.

4 Similarity-Based Lemma Selection

In this section we want to introduce the principles of similarity-based lemma selection with lemma delaying tableaux enumeration procedures as a general framework for filtering important lemmas. Firstly, we introduce our notion of relevancy of a lemma for a proof goal. Secondly, we deal with the question how to a priori estimate this relevancy with similarity measures. Finally, we show how a lemma selection can take place based on such estimations.

4.1 Notion of Similarity

As mentioned before we want to filter relevant lemmas, i.e. lemmas that are able to contribute to a proof. We want to make the notion of relevancy 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 $\mathcal{T}$ where the start clause is an instance of S, and that contains as tableau clauses only instances of clauses from $\mathcal{C} \cup \mathcal{L}$. Furthermore, at least one $l \in L$ is used for closing a branch. Not every set $\mathcal{L}$, similar to S w.r.t. the search tree $\mathcal{T}$ defined by $\mathcal{C} \cup \mathcal{L}_0$, is an "interesting" set for using it as a lemma set (augmenting $\mathcal{C}$ with $\mathcal{L}$) 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 when using $\mathcal{B}$. There should be no $\tilde{\mathcal{L}} \subseteq \mathcal{L}_0$, $\tilde{\mathcal{L}} \neq \mathcal{L}$ that can help to refute $\mathcal{C}$ with smaller resources. Furthermore, it would be worthwhile that $\mathcal{L}$ has minimal size, i.e. no lemma can be deleted in order to obtain a proof with resource n. We call such lemma sets *most similar* to S w.r.t. $\mathcal{B}$.

According to this notion of similarity a sensible selection mechanism would choose, assuming that a set of lemma candidates $\mathcal{L}_0$ is given, a subset $\mathcal{L} \subseteq \mathcal{L}_0$ that is most similar to S. This test, however, necessitates the consideration of *all* subsets of $\mathcal{L}_0$.

Another possibility is to employ a *local* notion of relevancy of a lemma for refuting a clause set. We can employ a weaker notion of similarity between a subgoal S and a lemma $l \in \mathcal{L}_0$ by calling a lemma l *weakly similar* to S w.r.t. a search tree $\mathcal{T}$ if there is a closed tableau in $\mathcal{T}$ that can be reached with start clause S and that contains as tableau clauses instances of l and of some clauses from $\mathcal{C} \cup \mathcal{L}_0$. Analogously we can say a lemma is *most weakly similar* to S if there is a lemma set $\mathcal{L}$ that contains l and that is most similar to S. The union of these lemmas which are most weakly similar to S is a lemma set $\mathcal{L}_{loc} \subseteq \mathcal{L}_0$ which can refute $\mathcal{C}$ (employing start clause S) with minimal resources. Furthermore, normally $\mathcal{L}_{loc}$ has a rather small size compared with $\mathcal{L}_0$. Thus, a possible lemma selection mechanism can also try to find a rather well suited lemma set based on local tests.

4.2 Estimating the Similarity

After clarifying our notion of a posteriori similarity or relevancy we want to introduce our principles for a priori estimating this similarity. With the help of similarity measures (cp. [Ric92]) we want to obtain some estimations whether or not a set $\mathcal{L}$ is most similar to S. Then the subset of $\mathcal{L}_0$ with the best estimations can be used as lemma set. Another possibility is to try to estimate which lemmas are most weakly similar to S and choose some of the lemmas with the best estimations.

Our approach for computing such estimations is based on a fast simulation of the deduction process. It is clear that conventional complete tableaux enumeration procedures allow for finding a subset of $\mathcal{L}_0$ being most similar to S. The following method allows for finding a subset of $\mathcal{L}_0$ which is most similar to S by providing a sound and complete test for $Sim_{\mathcal{T}_m}(S, \mathcal{L})$. The search tree $\mathcal{T}_m$ is defined by $\mathcal{C}$ and $\mathcal{L}$, a completeness bound $\mathcal{B}$, and the resource n_m which is at least needed in order to refute $\mathcal{C} \cup \mathcal{L}_0$. Such a similarity test can be obtained by enumerating the set $\mathcal{O}$ of all open tableaux in $\mathcal{T}_m^{\mathcal{C}}$ which is defined by $\mathcal{C}$, S, $\mathcal{B}$, and n_m and then checking whether or not the open subgoals (*front clause*) $SG(T)$ of a tableau $T \in \mathcal{O}$ can be solved by extension with some facts of $\mathcal{L}$ such that the resulting tableau is part of $\mathcal{T}_m$. We want to call this method *lemma delaying* tableaux enumeration. Naturally, in practice iterative deepening is needed in order to determine the value n_m. Further information on this enumeration method and on other possible methods can be found in [Fuc97a].

Analogously one can determine whether a lemma l is most weakly similar to S. Instead of generating front clauses we have to enumerate front literals and perform an unification test with l (local test). If we want to employ local tests, however, it is not sufficient to generate front clauses that are units (*front literals*) in $\mathcal{T}_m^{\mathcal{C}}$. 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 sub-proofs of applicable lemmas from $\mathcal{L}_0$.

A fast simulation of the complete similarity tests as introduced above consists of an estimation of the value n_m in order to define a finite search tree $\mathcal{T}_m^{\mathcal{C}}$ which has to be enumerated. Then, in order to simulate a similarity test between S and a lemma set $\mathcal{L}$ we enumerate a subset of the set of front clauses which belong to the tableaux in $\mathcal{T}_m^{\mathcal{C}}$ and compute some *unification distances* between the front clauses and the lemma set $\mathcal{L}$. This values can be used for estimating the similarity of S and $\mathcal{L}$. Analogously, the usefulness of a fact $l \in \mathcal{L}_0$ for refuting S in $\mathcal{T}_m$ can be estimated by generating a subset of the set of front literals F occurring in tableaux in $\mathcal{T}_{m,e}^{\mathcal{C}}$ and then measuring a (unification) distance between the front literals and the complement of l.

4.3 Lemma Selection

Now, we want to describe how based on these principles a selection of possibly well suited lemmas from a set $\mathcal{L}_0$ can be conducted. We want to concentrate on the local method which provides similarity tests with rather small costs.

We assume that for a given top-down search bound $\mathcal{B}$, a clause set $\mathcal{C}$, and $S \in \mathcal{C}$ a method for creating front literals is given. Furthermore, we assume the existence of a distance measure d mapping two literals to $I\!R$.

In [Fuc97a] two methods for choosing an appropriate subset $\mathcal{L} \subseteq \mathcal{L}_0$ of useful lemmas are introduced. Because of its practical success we want to use the following *front literal based method*. We generate for each clause which should serve as start clause for refuting $\mathcal{C}$ front literals. The generated front literals are interpreted as samples for proofs (in $\mathcal{T}_{m,e}^{\mathcal{C}}$) which can be concluded with a lemma. Based on d we are able to choose for each of these front literals a lemma that seems to be most suitable for concluding a proof. Thus, a possible approach for filtering lemmas is to use the union of the lemmas which are most similar to a front literal w.r.t. d.

By employing such a strategy we have good chances to find a subset of $\mathcal{L}_0$ that leads to a proof with the resource value which is at least needed in order to obtain a proof with $\mathcal{L}_0$. Furthermore, normally we can restrict ourselves to a rather small subset, though not of minimal size (cf. section 8). In the following we want to substantiate this framework for a similarity-based lemma-selection with some concrete methods.

5 Basic Techniques

In this section we want to recall some techniques for generating front literals as well as for measuring distances between front literals and unit lemmas as introduced in [Fuc97a]. Furthermore, we discuss some weaknesses of these methods.

In the following we are interested in finding a well suited subset $\mathcal{L}$ of a set of lemma candidates $\mathcal{L}_0$ for refuting a clause set $\mathcal{C}$ employing a start clause S and a completeness bound $\mathcal{B}$ for iterative deepening.

5.1 Generating Front Literals

The aim of the generation of front literals is to get a good "coverage" of the search tree $\mathcal{T}_{m,e}^{\mathcal{C}}$. We want to enumerate front literals in such a way that for each useful lemma at least a structural similar front literal is given even though no front literal unifiable with the lemma can be produced. In general this is no easy task since we have no a priori knowledge how to choose n_m and thus the form of $\mathcal{T}_m^{\mathcal{C}}$ and of the "expansion" $\mathcal{T}_{m,e}^{\mathcal{C}}$ of $\mathcal{T}_m^{\mathcal{C}}$ is unknown.

We handle this problem by simply assuming n_m is "fairly large" and consider possibilities for approximating a similarity test in such a large search space. An appropriate method is to employ breadth-first search. We simply enumerate all front literals occurring in tableaux that can be reached with at most k inference steps for a fixed value $k \in IN$. Thus, we can follow each inference chain needed to obtain a front literal f^l that is unifiable with a needed lemma l until a certain depth. If we are able to generate a front literal f where inferences are performed similar to those that are needed to find f^l then it is quite probable that f and l are structurally similar (cp. the following section). This may allow us to identify l as useful for the proof. In the following let $\phi(\mathcal{C}, S, k)$ be the set of front literals which can be obtained in the search tree which is defined by $\mathcal{C}$ and start clause S with at most k inferences.

This method is not necessarily sound, i.e. it may be that not all generated front literals belong to tableaux from $\mathcal{T}_{m,e}^{\mathcal{C}}$. In our experiments, however, we have normally obtained sound tests. Furthermore, a sporadic violation of the soundness is no real problem because it is more problematic to ignore useful facts from $\mathcal{L}_0$ than to select some few unnecessary ones.

An interesting question is how much front literals should be generated. At first sight, it seems to be reasonable to use a high inference resource in order to increase the probability for producing front literals which are structural similar to a needed lemma l, i.e. to produce a literal which is almost unifiable with l. Since also a lot of unnecessary paths are explored, however, it may be that also a lot of unnecessary facts are considered to be useful. Therefore, we employ the following method: We generate the set $\phi(\mathcal{C}, S, n)$ where n is chosen as the maximal value such that $|\phi(\mathcal{C}, S, n)|$ is smaller than the size of the generated lemmas. Thus, we have the guarantee that our filtering as described previously will actually choose a pure subset of $\mathcal{L}_0$.

Note that besides this method of generating front literals in a finite segment defined by the inference bound and a fixed resource, also other bounds may be employed. But from a heuristical point of view the inference bound seems to be more appropriate. On the one hand we can follow each proof path in $\mathcal{T}_{m,e}^{\mathcal{C}}$ and have no restrictions, e.g. concerning the structure of the generated tableaux. On the other hand, we can control the number of generated front literals in a rather fine-grain manner. In section 8 we demonstrate the usefulness of the inference bound with some experimental results.

5.2 Measuring Unification Distances

In this section we want to develop methods for comparing a front literal and a lemma l. 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 $f^l \in F \backslash \phi(\mathcal{C}, S, n)$ can be produced that can be closed with l. F denotes again the set of front literals occurring in tableaux from $\mathcal{T}^{\mathcal{C}}_{m,e}$.

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: Firstly, 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 which prevent $\sim f$ and l from being unifiable. Secondly, 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. Altogether, one can say that the more inferences are "correct" when producing f the more term structure f and l probably share.

In order to develop (unification) distance measures we want to distinguish the following two cases: On the one hand it may be that in order to close f^l with l no instantiations of f^l and l are needed or only instantiations with terms of a small size. On the other hand it may be necessary that instantiations with terms of larger size have to take place.

5.2.1 Non-Instantiating Distance Measures

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 F and l. As described above we have to weaken this structural equality test to a *structural similarity* test. A common method in order to allow for such a similarity test of structures is to employ so-called features.

Features allow for an abstraction of literals by mapping them to numbers. We introduce a feature as a function φ mapping a literal to a natural number. $\varphi(l)$ is called the *feature value* of l. Usually, many different features $\varphi_1, \ldots, \varphi_n$ are used and a literal l is represented by its *feature value vector* $(\varphi_1(l), \ldots, \varphi_n(l))$. Similarity at the level of literals can be shifted to the level of features by defining a distance measure on features: We consider literals to be similar if they share common features (properties). Thus, we claim that a lemma is useful if there is a front literal that has a similar feature value vector.

The features used in [Fuc97a] concentrate on simple syntactical properties. Some fixed feature functions measuring properties like the number of distinct variables or the number of leaf nodes of a term have been used. Furthermore, two feature schemata have been applied that compute feature values for each function symbol of the given signature. We measure the occurrence as well as the nesting depth of each function symbol in a literal.

For measuring distances on metrically represented data a large variety of possibilities exist. We decided to use the simple Euclidean distance and let it additionally work on *standardized* data (cp. [Fuc97a]). Thus, we employ the following distance measure d_F^* defined on literals where $d_E(u,v)$ measures the Euclidean distance between the standardized feature value vectors of u and v.

Definition 5.1 **(d_F^*)** Let $\mathcal{C}$ be a set of clauses given in signature sig. Let $\mathcal{V}$ be a set of variables and let $Lit(sig,\mathcal{V})$ be the set of literals over sig and $\mathcal{V}$. Let d_E be as introduced before. We define $d_F^* : Lit(sig,\mathcal{V})^2 \to I\!R$ by

$$d_F^*(u,v) = \begin{cases} \infty & ; (u \equiv P(t_1,\dots,t_n),\ v \not\equiv \neg P(s_1,\dots,s_n)) \text{ or} \\ & \quad (u \equiv \neg P(t_1,\dots,t_n),\ v \not\equiv P(s_1,\dots,s_n)) \\ 0 & ; \sim u \text{ and } v \text{ are unifiable} \\ d_E(\sim u, v) & ; \text{otherwise} \end{cases}$$

In order to improve this first version of a distance measure we want to take more care of the situation that not all inferences which need to be applied to subgoals on the path from the root node of T_{f^l} to f^l are also performed when producing f. This may occur because of the lack of resources (inferences that are allowed) and because of wrong instantiations coming from the solutions of other subgoals. But this can even result in different predicate symbols of f and l although nearly the inferences are performed to f that are needed to infer f^l. At least we can reconstruct (path) inferences that do not require some specific term structure. Transformations that require only a specific predicate symbol (e.g. the symmetry of an equality predicate) can be rebuilt.

Thus, we use a finite set $\mathcal{R}$ of correct rewrite rules like $P(x_1,\dots,x_n) \to R(t_1[x_1,\dots,x_n],\dots,t_m[x_1,\dots,x_n])$ in order to rewrite a front literal f to other literals $f_1,\dots,f_k$. Since $\mathcal{R}$ usually is not terminating and since we do not want to risk the production of literals whose respective tableaux are not in $\mathcal{T}_e$, we only allow a small number of rewrite steps. We use the set $\mathcal{R}^k(f)$, which is the set of literals that can be obtained from f by applying at most k rewrite steps with rules from $\mathcal{R}$, instead of f in order to measure unification distances to l. A set of rewrite rules $\mathcal{R}$ can be obtained, e.g. in analogy to the statical generation of lemmas introduced before. We add most general queries $\neg P(x_1,\dots,x_n)$ (and $P(x_1,\dots,x_n)$ in the case of non-horn clauses) to the clause set $\mathcal{C}$ to be refuted. Then the front literals $f_1,\dots,f_m$ that can be obtained in a finite segment of the search space without instantiating the original query are collected and m valid rules $\neg P(x_1,\dots,x_n) \to f_i$, $1 \le i \le m$, are generated. Actually, we use the inference bound with a resource value that limits the number m of generated rules to a value smaller than 10. Furthermore, we use $k = 5$. In summation, we get the distance measure d_F:

Definition 5.2 (d_F) Let $\mathcal{C}$ be a set of clauses given in *sig*, and let $\mathcal{R}$ be a set of rewrite rules for $\mathcal{C}$ as described above. Let $\mathcal{V}$ be a set of variables. Let d_F^* be as in the previous definition. Let k be a natural number. We define $d_F : Lit(sig, \mathcal{V})^2 \to I\!R$ by

$$d_F(u, v) = \min_{s \in \mathcal{R}^k(u)} \{d_F^*(s, v)\}.$$

5.2.2 Instantiating Distance Measures

In [Fuc97a] an instantiating method based on an inference system for conventional unification has been developed. An appropriate instantiation (substitution) has been obtained by *pseudo-unifying* two literals (terms). Basically, the inference system $\mathcal{UD}$ for pseudo-unifying two terms is analogous to a conventional inference system for unification (cp. [Ave95]) but when normally an unification fails because an "occur-check" fails or because different functions symbols occur, then this mismatch is ignored and the unification process preceeds. Consider the following example:

Example 5.1 Let $t_1 \equiv f(x, g(x))$ and $t_2 \equiv f(a, g(b))$ be two terms given in a signature. Then t_1 and t_2 pseudo-unify with $\sigma_1 = \{x \leftarrow a\}$ or $\sigma_2 = \{x \leftarrow b\}$.

For a definition of $\mathcal{UD}$ and some remarks for controlling the inference system (making it deterministic) compare [Fuc97a]. After the instantiation process a comparison of the term structure with d_F is possible. As a possible instantiating method we get:

Definition 5.3 (d_S) Let *sig* be a signature and $\mathcal{V}$ be a set of variables. Let d_E be as in definition 5.1. We define the distance measure $d_S^* : Lit(sig, \mathcal{V})^2 \to I\!R$ by

$$d_S^*(u, v) = \begin{cases} \infty & ; (u \equiv P(t_1, \ldots, t_n),\ v \not\equiv \neg P(s_1, \ldots, s_n)) \text{ or} \\ & (u \equiv \neg P(t_1, \ldots, t_n),\ v \not\equiv P(s_1, \ldots, s_n)) \\ d_E(\sim \sigma(u), \sigma(v)) & ; \text{otherwise} \end{cases}$$

where $(\{(\sim u, v)\}, id) \overset{*}{\vdash}_{\mathcal{UD}} (\emptyset, \sigma)$. Let $\mathcal{C}$ be a set of clauses given in *sig*, and let $\mathcal{R}$ be a set of rewrite rules for $\mathcal{C}$ as described above. Let k be a natural number. We define $d_S : Lit(sig, \mathcal{V})^2 \to I\!R$ by

$$d_S(u, v) = \min_{s \in \mathcal{R}^k(u)} \{d_S^*(s, v)\}.$$

The basic idea of this method is that the term structure incorporated to a front literal f (created in the simulation) by certain inferences is judged to be correct (analogous to f^l which can be closed with a useful lemma l) as long as the term structure of f does not prevent f and l from being unifiable. If this assumption is true one can obtain a correct instantiation of the lemma l and the front literal f resulting in a higher structural similarity of f and l that can increase the probability that l will be recognized to be useful. The following example shows that, if this assumption does not hold, incorrect substitutions may be computed that may even increase the structural differences between f and l.

Example 5.2 Let $l \equiv equal(g(g(h(a,b),h^3(y)),c),h^3(y))$ be a useful lemma and let $f^l \equiv equal(g(g(x,h^3(b)),c),h^3(b))$ be the front literal that can be closed against l. Let g be an associative operator. Now consider the case that because of the lack of resources when generating front clauses instead of f^l only this front literal f may be obtained: $f \equiv equal(g(x',g(h^3(b),c)),h^3(b))$. Although with instantiation $\sigma_1 = \{x' \leftarrow h(a,b), y \leftarrow b\}$ (an instantiation equivalent modulo renaming variables to that needed to close f^l) $\sigma_1(f)$ and $\sigma_1(l)$ would be structurally very similar (cp. the features used in [Fuc97a]) the inference system $\mathcal{UD}$ would obtain $\sigma_2 = \{x' \leftarrow g(h(a,b),h^3(b)), y \leftarrow b\}$ resulting in more dissimilar terms $\sigma_2(l) \equiv equal(g(g(h(a,b),h^3(b)),g(h^3(b),c)),h^3(b))$ and $\sigma_2(l) \equiv equal(g(g(h(a,b),h^3(b)),c),h^3(b))$.

Thus, in order to allow for a better handling of such transformations a more flexible instantiation scheme (as introduced in section 7.2) may be more appropriate.

5.2.3 Summary

When considering the methods previously introduced we can mention the following weaknesses. Although the first method allows for an abstraction from structural equality that may be helpful in order to allow for differences between the term structure of a front literal and a lemma it simultaneously risks to fall in an "over-abstraction". This possibly prevents distinguishing useful from useless facts by only considering their feature value vectors. Thus, it may be sensible to combine the abstraction provided by the features with some maintenance on the given term skeleton.

Our second method may suffer from the inflexible instantiation scheme based on a unification attempt of a front literal and a lemma (see example 5.2). In addition to this it may be sensible to allow for a more flexible instantiation scheme that allows for (small) deviations of the term structure by associating term positions and solving the resulting unification problems with $\mathcal{UD}$.

6 Front Literal Generation based on Adaptive Refinements

So far we have introduced a similarity test as a simulation process of deduction. A complete similarity test can be performed by enumerating a large discrete space of tableaux (given by $\mathcal{T}^{\mathcal{C}}_{m,e}$) and using the front literals of the tableaux for similarity assessments to lemmas. In order to decrease this high number of tableaux to be considered we employ breadth-first search until a given (usually) small depth with the hope to produce front literals similar to useful lemmas w.r.t. the principles previously introduced.

Similar to conventional simulation processes, e.g. in the area of numerical simulation, it is worthwhile to investigate whether an *adaptive refinement* of the simulation process at "interesting positions" (nodes) of $\mathcal{T}^{\mathcal{C}}_{m,e}$ may allow for an improved simulation of the deduction process. Adaptive refinement means in our context to allow for more (inference resources) at certain points in the search space (nodes in the search tree) when generating front literals. By employing more inference resources when generating front literals one can obtain a more exact simulation since the probability to identify a useful lemma increases when more inferences are used to generate front literals.

Hence, the question comes up which positions are interesting, i.e. where a refined generation of front literals may be sensible.

6.1 Principles of Adaptive Refinement

We present two different possibilities for identifying nodes in the search tree which may be interesting for choosing them as starting points for a refined generation of front literals. Let $\mathcal{T}$ be the search tree given by start clause S and the clause set $\mathcal{C}$.

The first approach tries to identify *correct inferences*, i.e. inferences needed in a proof and chooses nodes in $\mathcal{T}$ which are generated by performing these inferences as starting points for adaptive refinements. A possible method for front literal generation could start by generating front literals in breadth-first search employing a rather small resource value. Then, based on similarity assessments to lemmas, correct inferences can possibly be identified (to be described shortly) and nodes of the search tree reached by performing such inferences can be used as starting points for a refined generation of front literals. Thus, one implicitly obtains higher resource capacities at interesting points in the search space in order to create front literals. This can be done although the whole number of generated front literals may not exceeds the number of front literals generated in the conventional way with breadth-first search.

Our second method does not try to allow for more inference resources in each area of the search space where an application of a lemma is expected but tries to refine *positions of high uncertainty*. If a front literal gives significant evidence that a certain lemma is needed (the distance to this lemma is much smaller than the distances to other lemmas) it may not be necessary to use more inference resources, e.g. in order to solve some of the subgoals occurring when creating the front literal. Instead, we want to use more inference resources in such areas of the search space where a higher uncertainty about the selection of lemmas exists. Only there the precision of the simulation should be increased.

6.2 An Algorithm for Improved Front Literal Generation

As described before we want to employ a (possibly) iterated front literal generation as follows: At first an initial set F_0 of front literals is created starting from the tableau obtained by expanding the trivial tableau with S. Now a set of front literals F_{i+1} is created from F_i by identifying interesting points of the search space which are on a path from a starting point for the front literal generation of F_i and some tableaux belonging to front literals from F_i. These nodes can be identified by a quality predicate $\mathcal{G}$ which measures the quality of nodes (for adaptive refinements) according to the abstract principles sketched before. Finally, a lemma selection takes place after the generation of F_m if the set $\phi = \bigcup_{i=0}^{m} F_i$ is sufficiently large. For each front literal the lemma which is most similar to it is chosen.

A critical point regarding this method is the selection of certain tableaux (nodes of the search tree) that serve as starting points for further generation loops of front literals. In the following we describe our notion of quality of tableaux and explain how starting points for an adaptive refinement can be chosen. We conclude the section with an algorithm for front literal generation.

6.2.1 Quality of Tableaux

Before we explain our notion of quality of a tableau we want to start with explaining the notion of quality of a front literal. According to the abstract principles introduced previously we want to define two quality functions g_1 and g_2. g_1 judges whether or not the tableau associated with a front literal contains a lot of "correct" inferences, i.e. inferences occurring in a proof that can be obtained by using lemmas. It is clear that the distance measures introduced in the previous sections can be used for this very issue. Thus, a possible realization of a quality function g_1 defined on a set F of front literals can be as follows (a higher value indicates a higher quality):

$$g_1(f) = \max\{\min\{d(f', l) : l \in \mathcal{L}_0\} : f' \in F\} - \min\{d(f, l) : l \in \mathcal{L}_0\}$$

g_2 judges the uncertainty of a front literal when selecting a certain lemma. If a lot of lemmas contain similar distance values to a front literal it may be that a useful lemma will be omitted although it has nearly the same distance values as a chosen though not needed fact. We define g_2 as follows where d^f_{min} denotes the smallest distance value of a lemma to $f \in F$.

$$g_2(f) = \max\{\sum_{l \in \mathcal{L}_0} (d(f', l) - d^{f'}_{min})^2 : f' \in F\} - \sum_{l \in \mathcal{L}_0} (d(f, l) - d^f_{min})^2$$

Using this notion of quality of front literals we want to introduce a possible quality function of tableaux associated with nodes from $\mathcal{T}$. We assume that the front literals are generated starting from a set of initial tableaux T_{init}. Now we want to rate tableaux that are on a path (in $\mathcal{T}$) from a tableau from T_{init} and a tableau belonging to a front literal.

Considering the first notion of quality, a tableau T should be considered to be of high quality if a lot of front literals can be derived (starting from T) that have high quality values w.r.t. g_1. Then it is quite probable that T is inferred with correct inferences. Thus, a refined generation of front literals starting from T increases the probability to produce a front literal most similar to a useful lemma which might not be selected otherwise. Furthermore, we should take care of the fact that the generation of front literals starting from T actually generates new front literals. Thus, the tableaux from T_{init} should not obtain the best ratings since the front literal generation starting from these tableaux does not allow for the generation of new front literals (assuming that we do not want to increase the number of generated front literals compared with breadth-first search). In general there is the problem of finding a compromise of being pessimistic, i.e. choose only tableaux with a small "inference distance" to the tableaux from T_{init}, and being optimistic, i.e. choose tableaux which have a larger inference distance. A pessimistic choice allows with a higher probability to choose a tableaux which is created with correct inferences. Thus, a lot of front literals which are well suited for selecting lemmas (are similar to useful facts) may be *derivable* from the tableaux. But in practice it might be problematic to *actually generate* useful front literals from these tableaux because of the small inference resources which are applicable. Otherwise, an optimistic choice increases the probability to produce useful front literals if the starting point is produced with correct inferences but the probability to select such a starting point is rather small.

Thus, we choose the approach to use a predicate P_I judging the inferences needed to produce a tableau in addition to the distance values of inferable front literals. P_I is defined by $P_I(T)$ iff $\min\{n : T' \overset{n}{\vdash} T, T' \in T_{init}\} = n_{min}$ where n_{min} is a fixed natural number. n_{min} has to define a well suited compromise between a too pessimistic and too optimistic choice of starting points for front literal generation.

A possible realization of a quality function $\mathcal{G}_1$ defined on tableau is as follows where F is a given finite set of front literals and T^f for $f \in F$ is the tableau which contains f as front literal. Let further $L(T)$ be the subset of front literals from F which are inferable from T, i.e. $L(T) = \{f \in F : T \stackrel{*}{\vdash} T^f\}$.

$$\mathcal{G}_1(T) = \begin{cases} \sum\limits_{f \in L(T)} g_1(f) & ; L(T) \neq \emptyset,\ P_I(T) \\ \infty & ; L(T) = \emptyset,\ P_I(T) \\ 0 & ; \neg P_I(T) \end{cases}$$

A high value of $\mathcal{G}_1(T)$ indicates that T is a well suited starting point for a refined front literal generation. In addition to the remarks above we rate nodes in the search tree (tableaux) best where no front literals could be obtained in the preprocessing phase in order to increase the probability to produce front literals in this area.

Analogously, we can use g_2 in order to define a quality value $\mathcal{G}_2(T)$ for a tableau T which represents the uncertainty which lemmas may be needed for a proof which contains the inferences needed to produce T. A high value $\mathcal{G}_2(T)$ indicates that it may be useful to generate more front literals starting from T. A generation of more front literals starting from T may allow for a better exploration of regions of rather high uncertainty about useful lemmas.

Now we want to consider the problem of how to choose appropriate starting points based on this notion of quality. We assume that front literals are generated from an initial set T_{init} of tableaux. Let $T_1, \ldots, T_n$ be the tableaux which can be obtained from a tableau from T_{init} with n_{min} inferences. We order these tableaux according to a quality function $\mathcal{G}$ defined on tableaux. We set $Gen_{\mathcal{G}}(T_{init}) = (T_{i_1}, \ldots, T_{i_n}),\ i_j \in \{1, \ldots, n\}$ for $j \in \{1, \ldots, n\}$, $i_j \neq i_k$ for $j \neq k$, and $\mathcal{G}(T_{i_j}) \geq \mathcal{G}(T_{i_{j+1}}),\ 1 \leq j < n$.

Now, new front literals can be built choosing the elements of $Gen_{\mathcal{G}}(T)$ as new starting points for the generation of front literals according to the occurrence in the sequence.

6.2.2 The final algorithm

In summation we get the following algorithm $\mathcal{FG}$ for front literal generation w.r.t. a set of clauses $\mathcal{C}$, a query $S \in \mathcal{C}$, and a parameter M which serves as an upper bound for the number of generated front literals. The algorithm is parameterized by a distance measure d and a function $Gen_{\mathcal{G}}$ which depends on a given quality function $\mathcal{G}$ (e.g. $\mathcal{G}_1$ or $\mathcal{G}_2$) and a fixed value n_{min}. Let $FL(\mathcal{C}, T, k)$ be the set of front literals which can be obtained in the search tree $\mathcal{T}$ defined by $\mathcal{C}$ when enumerating all tableau in $\mathcal{T}$ that can be reached with at most k inference steps starting from the tableau T. Furthermore, let T^S be the tableau obtained when applying the start rule to the trivial tableau by attaching the clause S to the unlabeled root.

Algorithm 6.1 $\mathcal{FG}$

Input: **- clause set** $\mathcal{C}$
- (start) clause $S \in \mathcal{C}$
- maximal number of front literals M
Output: **set of (front) literals** F

```
begin
  inf := 0
  Init := {T^S}
  while |FL(C, T^S, inf)| ≤ M
     inf := inf + 1
  endwhile
  inf := max({inf − 2, 0})
  F := ∅
  F_0 := FL(C, T^S, inf)
  k := 0
  while |F_k| ≤ M
    Seq = (T^1, ..., T^m) := Gen_G(Init)
    Init := {T^1, ..., T^m}
    F_{k+1} := F_k
    i := 1
    while |F_{k+1}| ≤ M ∧ i ≤ m
      F := F_{k+1}
      F_{k+1} := F_{k+1} ∪ FL(C, T^i, inf)
      i := i + 1
    endwhile
    k := k + 1
    if |F_k| ≤ M
      F := F_k
  endwhile
  return F
end
```

Thus, compared with the original method as described in section 5.1 we start with an initial set of front literals which is created with a (by an amount of 1) smaller inference resource. Then, successively at interesting points in the search space a refined generation of front literals takes place. The generation of new front literals starting from a chosen tableau is done with the same inference resource *inf* as used in order to generate F_0. Although already a value of $inf - n_{min} + 1$ can be sufficient in order to generate new front literals the value *inf* proved to be well suited in our experiments. We used $n_{min} = 3$ in our experiments which was always smaller than the value *inf* which is dynamically computed at the beginning of algorithm $\mathcal{FG}$. The upper limit M of front literals can be, analogously as in section 5, the size of a given lemma set $\mathcal{L}_0$. In section 8 we will give some further remarks on appropriately choosing M.

7 Distances Based on Structured Features

In order to improve our methods for measuring unification distances we want again develop non-instantiating as well as instantiating methods.

7.1 Non-Instantiating Methods

As described before our first distance measure d_F, which only compares differences between feature value vectors, risks to fall in an "over-abstraction". This possibly makes it impossible to distinguish useful from useless lemmas when only considering their feature value vectors. Thus, we want to try to combine properties of the two methods described before.

Our main idea is to represent a term (literal) with a tree where each node of the tree represents the structural properties of the respective subterm of this node. Looking at the problem that due to the wrong solutions of subgoals occurring when generating a front literal f the "deep" positions (subterms at these positions) may not be very important when their term structure is considered, we only represent term properties until a given depth. This structure allows us not only to compare properties of the whole terms but also of their sub-terms.

Furthermore, when comparing two terms t_1 and t_2 we partially abstract from the given tree structure. We try to estimate whether or not for a given subterm of a term t_1 there may be a corresponding subterm in t_2 (a subterm that shares many properties) that need not necessarily be at the same position. We want to allow for small differences in the term structures of f and l in order to provide the case that certain (needed) transformations have not been applied to f. Thus, we can combine the abstraction provided by the feature values with a flexible consideration of the concrete term structures. Now we want to present our technical realization:

Let *depth* be a natural number. Let φ be a feature. A *structured feature* φ^{Δ}_{depth} w.r.t. *depth* and φ is a function mapping literals (terms) to trees that are labeled with natural numbers. The structured feature value $\varphi^{\Delta}_{depth}(t)$ is a tree that is isomorphic to the (term) tree defined by t where all nodes with a depth greater than *depth* are deleted. Each node of $\varphi^{\Delta}_{depth}(t)$ at position p is labeled with the feature value $\varphi(t|p)$. Structured feature value vectors w.r.t. features $\varphi_1, \ldots, \varphi_n$ are vectors of labeled trees. The structured feature value vector of t is denoted by $\vec{\varphi}^{\Delta}_{depth}(t)$. For simplicity reasons we consider $\vec{\varphi}^{\Delta}_{depth}(t)$ in the following as *one* labeled tree whose nodes are labeled with feature value vectors of the respective subterms of t. Furthermore, we sometimes identify a subtree at a certain position with the label at this position. The following example illustrates the notion of a structured feature:

Example 7.1 Let $sig = (\mathcal{P} = \{eq\}, \mathcal{F} = \{mult, inv, id, glb, lub\}, \tau)$ be a signature and let $\tau(eq) = \tau(mult) = \tau(glb) = \tau(lub) = 2$, $\tau(inv) = 1$, and $\tau(id) = 0$. Let $\varphi_1, \ldots, \varphi_4$ be features. φ_1 computes a weighted length of a literal (term) where variables are counted with 1 and other symbols with 2. φ_2 counts the number of variable occurrences in a term, and φ_3 the number of leaf nodes of a term. φ_4 computes the depth of a term. Let t_1 and t_2 be terms in sig given by $t_1 \equiv eq(mult(id, x), x)$ and $t_2 \equiv eq(glb(x, lub(x, y)), x)$. Let $depth = 2$ be a constant. Then the structured feature value vectors $\vec{\varphi}^{\Delta}_{depth}(t_1)$ and $\vec{\varphi}^{\Delta}_{depth}(t_2)$ are given by (using the conventional list notation of a tree where the label of a node is denoted by $[\cdot]$):
$\vec{\varphi}^{\Delta}_{depth}(t_1) = ([8,2,3,2]([5,1,2,1]([2,0,1,0][1,1,1,0])[1,1,1,0]))$
$\vec{\varphi}^{\Delta}_{depth}(t_2) = ([10,4,4,3]([7,3,3,2]([1,1,1,0][4,2,2,1])[1,1,1,0]))$

In the following we shall clarify how a distance measure based on structured features can be defined. Essentially, we have to deal with two main tasks when construing a distance measure. On the one hand the notion of distance between subtrees of structured features has to be clarified. On the other hand we have to define a distance measure on positions. It is clear that distances between subtrees of structured features are measured according to the principles previously introduced. The Euclidean distance d_{eucl} of the associated labels can be used for this purpose. In order to measure distances between positions we employ a standard distance measure for positions of trees (cp. [Ric89]): Let p_1, $p_2 \in I\!N^*$ be two positions. Let p be a position of maximal length such that $p_1 = pq_1$ and $p_2 = pq_2$ for q_1, $q_2 \in I\!N^*$. We define $d_{pos} : I\!N^* \to I\!N$ by $d_{pos}(p_1, p_2) = |q_1| + |q_2|$ ($|\cdot|$ denotes the length of a sequence in $I\!N^*$).

With the help of d_{eucl} and d_{pos} we are able to construct our final distance measure d_{TF}. $d_{TF}(l_1, l_2)$ tries to find for each position p_1 of $\vec{\varphi}^{\Delta}_{depth}(l_1)$ a similar subtree of $\vec{\varphi}^{\Delta}_{depth}(l_2)$ at position p_2 such that the differences between the feature value vectors at positions p_1 and p_2 as well as the distance between the positions are as small as possible. Furthermore, we use a weighted sum that gives higher weights to lower positions because similarity of the whole term is considered to be more important than similarities between deep subterms.

Definition 7.1 (d_{TF}) Let $\mathcal{C}$ be a set of clauses given in signature sig. Let $\mathcal{V}$ be a set of variables. Let d_{pos} be as introduced before. Let d_{eucl} be the Euclidean distance. Let dp be a natural number and let $\varphi_1, \ldots, \varphi_n$ be features. $\mathcal{R}$ is the set of rewrite rules for $\mathcal{C}$ as introduced before. Let k be a natural number. We define $d_{TF} : Lit(sig, \mathcal{V})^2 \to I\!R$ by

$$d_{TF}(u, v) = \min_{s \in \mathcal{R}^k(u)} \{d^*_{TF}(s, v)\}$$

d^*_{TF} is defined by

$$d^*_{TF}(u,v) = \begin{cases} \infty & ;(u \equiv P(t_1,\ldots,t_n), \\ & \;v \not\equiv \neg P(s_1,\ldots,s_n)) \text{ or} \\ & \;(u \equiv \neg P(t_1,\ldots,t_n), \\ & \;v \not\equiv P(s_1,\ldots,s_n)) \\ 0 & ;\sim u \text{ and } v \text{ are unifiable} \\ d^{**}_{TF}(\vec{\varphi}^{\Delta}_{dp}(\sim u), \vec{\varphi}^{\Delta}_{dp}(v)) & ;\text{otherwise} \end{cases}$$

$$d^{**}_{TF}(\vec{\varphi}^{\Delta}_1, \vec{\varphi}^{\Delta}_2) = \frac{\sum\limits_{p \in O(\vec{\varphi}^{\Delta}_1)} (p_{max} - |p|) \cdot \min\limits_{p' \in O(\vec{\varphi}^{\Delta}_2)} \{d_{eucl}(\vec{\varphi}^{\Delta}_1|p, \vec{\varphi}^{\Delta}_2|p') \cdot d_{pos}(p,p')\}}{\sum\limits_{p \in O(\vec{\varphi}^{\Delta}_1)} (p_{max} - |p|)}$$

p_{max} denotes the maximal tree depth of $\vec{\varphi}^{\Delta}_1$.

7.2 Instantiating Methods

In order to improve our first instantiating method we want to allow for a more flexible instantiation scheme. We want to improve our estimation of similarity in situations where, because of the fact that some transformations have not been applied to certain subgoals when producing a front literal f, a simple pseudo-unification does not allow for appropriate instantiations of f and a lemma l. Consider the example 5.2. Because of "mismatches" in the term skeleton of f and l no appropriate instantiation may be found which allows for the correct estimation that l is actually needed for a proof.

We want to develop a method which allows for more flexible instantiations and partially abstracts from the given term structure. The basic idea is, before trying to solve a unification problem, at first to find a correlation $\varphi_{f,l}$ between positions of f and l and after that solving the termpairs (equations) provided by f, l, and $\varphi_{f,l}$.

Our method for finding a function $\varphi_{f,l}$ is based on principles previously introduced in order to allow for a non-instantiating method which considers the concrete term structures of f and l. We choose the similarity of subterms at the feature level in order to construct $\varphi_{f,l}$. We define $\varphi_{f,l}(p)$ for $p \in O(f)$ and $|p| \leq max$, $max \in IN$, by $\varphi_{f,l}(p) = p' \in O(l)$ where $d_{eucl}(\vec{\varphi}^{\Delta}_{depth}(\sim f)|p, \vec{\varphi}^{\Delta}_{depth}(l)|p') \cdot d_{pos}(p,p')$ $= \min\limits_q \{d_{eucl}(\vec{\varphi}^{\Delta}_{depth}(\sim f)|p, \vec{\varphi}^{\Delta}_{depth}(l)|q) \cdot d_{pos}(p,q)\}$.

Conflicts are solved by choosing the position p' with the smallest distance d_{pos} to p. Thus, we allow for (small) differences between the term structures of f and l in order to obtain more flexible instantiations. Based on $\varphi_{f,l}$ a substitution can be obtained by solving the equations $E = \{(\sim f)|p = l|\varphi_{f,l}(p) : |p| \leq max,\ p \in O(f)\}$. We control the solution of the equations from E in such a manner that the equations $(\sim f)|p = l|q$ are selected first where no $(\sim f)|p' = l|q'$ exists in E such that $p' < p$. The ordering $<$ given on positions denotes being at a "deeper" term position (cp. [Ave95]). We allow when solving the equations in E for unification failures and use $\mathcal{UD}$. Let $\sigma_E^{f,l}$ be the substitution obtained in such a way.

After computing the substitution we have to deal with the question how to measure the distance between the instantiated literals. It is possible to use the feature distance of the literals or to employ the distance measure d_{TF} previously introduced. In view of the promising experimental results of d_{TF} we have chosen this measure and finally we get the distance measure d_{VS}.

Definition 7.2 Let $\mathcal{C}$ be a set of clauses given in signature sig. Let $\mathcal{V}$ be a set of variables. Let d_{pos} be as introduced before. Let d_{eucl} be the Euclidean distance. Let $depth$ be a natural number and let $\varphi_1, \ldots, \varphi_n$ be features. Let $\sigma_E^{l_1,l_2}$ be the substitution obtainable for two literals l_1 and l_2 according to the principles introduced before. Let $\mathcal{R}$ be a set of rewrite rule for $\mathcal{C}$ as introduced before and let k be a given natural number. We define $d_{VS} : Lit(sig, \mathcal{V})^2 \to I\!R$ by

$$d_{VS}(u,v) = \min_{s \in \mathcal{R}^k(u)} \{d^*_{VS}(s,v)\}$$

d^*_{VS} is defined by

$$d^*_{VS}(u,v) = \begin{cases} \infty & ; (u \equiv P(t_1,\ldots,t_n), \\ & \quad v \not\equiv \neg P(s_1,\ldots,s_n)) \text{ or} \\ & \quad (u \equiv \neg P(t_1,\ldots,t_n), \\ & \quad v \not\equiv P(s_1,\ldots,s_n)) \\ 0 & ; \sim u \text{ and } v \text{ are unifiable} \\ d_{TF}(\sigma_E^{u,v}(\sim u), \sigma_E^{u,v}(v))\} & ; \text{otherwise} \end{cases}$$

Obviously, this method can improve on the estimations provided by d_S if transformations, e.g. of the AC-type, require more flexible instantiations. In spite of the pure heuristical character this method indeed proved to be superior in a domain (COL) where sometimes such transformations occur. Naturally, in general, this method is no improvement on d_S but merely a further contribution to a pool of possible similarity measures which improves the chances to find an appropriate similarity-based lemma method.

8 Experiments

We want to evaluate our method in the light of experiments with the model elimination prover SETHEO and the lemma generator DELTA. We conducted experiments in the area of PL1 (with equality). In detail we have performed experiments in the BOO, CAT, COL, and GRP domain of the public problem library TPTP [SSY94].

There have been two main aims by performing these experiments. On the one hand we want to demonstrate that similarity-based lemma generation indeed improves on an unfiltered use of lemmas, a filtering which neglects the proof task at hand, and a version of SETHEO which does not employ lemmas. On the other hand we want to perform an empirical comparison of the similarity measures introduced before. Particularly, we want to evaluate which method should be used when confronted with a given proof task. Note that in this study we do not want to consider the correlation between top-down proof bounds and the use of lemmas for a given problem (as sketched in [Fuc97a]). We always used the search bound which performs best for the conventional version of SETHEO in the considered domain. Naturally, improvements of our methods may be possible by constructing bounds especially well suited for the use of lemmas (cp. section 9).

In order to employ lemmas we have chosen the following strategy: At first the DELTA lemma generator generates lemma candidates $\mathcal{L}_0$ in the same way as described in section 3. Bottom-up lemmas are generated with the depth bound and a resource of 3. In our experiments, we have restricted the size of $\mathcal{L}_0$ to an upper bound of 1000. In most cases, the number of generated lemma candidates has not reached this upper bound such that the complete "bottom part" of the search space has been generated. In the second step, a subset $\mathcal{L} \subseteq \mathcal{L}_0$ from the set of lemma candidates is filtered and added to the input clauses. As mentioned before we have used filter criteria which ignore the proof goal (cp. [AS92, Sch94]) as well as filter criteria which are based on similarity criteria. When using syntactic criteria a fixed percentage of 25% of the lemmas in $\mathcal{L}_0$ is chosen. When employing similarity criteria a selection of lemmas is done as described in section 4. In the last step a top-down proof run of SETHEO takes place. The search bound used is the bound best suited for the conventional version of SETHEO.

8.1 Comparison with Conventional Approaches

In order to demonstrate that our approach improves on the conventional approaches we depict in table 1 the results obtained by a conventional version of SETHEO, a version of SETHEO which uses all lemmas from $\mathcal{L}_0$ (SETHEO/DELTA), a version which filters lemmas according to the number of symbols of a literal (SETHEO/SYMB), and a fixed similarity-based strategy which performs best in the respective domain. We tackled all problems that cannot be solved with the conventional version of SETHEO with a run time that is smaller than 10 seconds ("hard problems") on a Sun Ultra II. We depict the number of problems that could be solved within 15 minutes, as well as the numbers when using a time limit of 1 minute, 5 minutes, and 10 minutes.

In the BOO, CAT and GRP domain we have chosen non-instantiating methods. In both cases the distance measure d_{TF} was chosen in connection with a conventional generation method for front literals. A more flexible front literal generation as described in section 6 cannot improve on these results and

BOO	Setheo	Setheo Delta	Setheo Symb	Setheo Sim	CAT	Setheo	Setheo Delta	Setheo Symb	Setheo Sim
≤ 1	5	5	6	11	≤ 1	3	3	3	3
≤ 5	6	5	7	12	≤ 5	3	3	3	5
≤ 10	10	6	8	13	≤ 10	3	3	3	5
≤ 15	11	6	9	13	≤ 15	3	3	3	5
COL	Setheo	Setheo Delta	Setheo Symb	Setheo Sim	GRP	Setheo	Setheo Delta	Setheo Symb	Setheo Sim
≤ 1	8	6	6	14	≤ 1	5	5	4	9
≤ 5	10	26	13	29	≤ 5	9	7	5	11
≤ 10	27	28	20	32	≤ 10	9	8	6	11
≤ 15	28	31	20	33	≤ 15	9	8	6	11

Table 1: Comparison with Conventional Approaches

reaches the same success rate (with sometimes shorter top-down run times). In the COL domain instantiating methods are needed. We have depicted the results obtained with d_{VS} and a front literal generation based on adaptive refinements. This method improves in this domain over the conventional (breadth-first) front literal generation (as described shortly in more detail). The bounds we have employed are the depth bound in the BOO, GRP, and COL domain. Furthermore, we have used the weighted-depth bound in the CAT domain.

We can observe a consistent gain of efficiency compared with the standard versions of SETHEO when considering the hard problems. We want to emphasize that we can solve some *new* problems as well as *significantly decrease* the run times when employing our lemma techniques. There are a lot of problems whose run time could be decreased from a region of more than 5 minutes to a value which is less than 1 minute. This is very interesting when using interactive proof systems like ILF [DGH+94] (which exactly uses the value 1 minute as a timeout). Thus, the practical applicability of SETHEO could be obviously improved.

8.2 Comparison of the Distance Measures

Now we want to analyze the behavior of our distance measures in more detail. At first we want to consider different methods for measuring a unification distance. Then, we compare our different methods for front literal generation.

8.2.1 Unification Distances

When comparing our different distance measures we want again distinguish the cases that non-instantiating or instantiating methods are applied. At first we want to consider the question if it is possible to predict whether an instantiating or a non-instantiating method should be used. During our experiments we observed that when no instantiations or only instantiations by terms of a small size are

Problem	Setheo	Setheo/Delta	Setheo (d_S)	Setheo (d_{VS})
COL042-2	—	983s	—	58s
COL042-3	—	—	—	907s
COL042-4	—	—	—	520s
COL060-2	532s	264s	45s	11s
COL060-3	509s	252s	40s	8s
COL063-5	541s	267s	42s	37s
COL063-6	539s	262s	120s	15s
COL064-2	567s	279s	110s	22s
COL064-3	566s	275s	112s	23s

Table 2: Comparison of Distance Measures

needed really the non-instantiating methods perform better than the instantiating methods. In such cases instantiation attempts usually do not improve the quality of the comparison between front literals and lemmas and can even sometimes lead to wrong estimations.

This is an a posteriori observation, however, such that the question comes up how to a priori estimate whether instantiations of terms of larger size are needed. The following simple heuristic seems to be sufficient. If the average size of the front literals and the generated facts significantly differ, e.g. exceeds a given threshold (a value of 5 was sufficient in our experiments), then it may be wise to employ instantiating measures. By using this strategy, we can indeed automate the selection of the distance measure in such a way that always the measure which performs best in our experiments is chosen.

When comparing our different non-instantiating methods the method based on structured features normally improves on the conventional feature approach. In the domains BOO and GRP, where non-instantiating methods are used, we can obtain in the BOO domain nearly the same results (although the conventional features normally allow for slightly shorter run times) but in the GRP domain the structured features are clearly superior to the conventional features. Distance measures based on structured features (employing one of the methods for front literal generation introduced before) are already after less than 5 minutes able to solve all of the 11 problems which could be solved within 15 minutes. Furthermore, already after 1 minute 9 problems can be solved (cp. table 1). The conventional feature approach only reaches 5, 10, 10, and 11 successes when employing a timeout of 1, 5, 10, and 15 minutes, respectively.

In the COL domain instantiating methods are needed. In our experiments the measure d_{VS} could improve on the (good) results obtained with d_S. Table 2 gives an excerpt from our experiments performed in the COL domain employing the conventional method for front literal generation based on breadth-first search. We depict the run time in seconds obtained with d_S and d_{VS} as well as with the standard version of Setheo and an unfiltered use of lemmas (Setheo/Delta).

We can observe an improvement of the results obtained with d_S when using

d_{VS}. Since in the COL domain (which contains problems of combinatory logic) sometimes combinators like the combinator B defined by $((Bx)y)z = x(yz)$ are used which easily results in (small) differences of the term skeleton of a lemma and a structural similar front literal a more flexible instantiation scheme allowed for improved estimations of similarity. Nevertheless, already the simple method could obtain very good results.

8.2.2 Front Literal Generation

In the following we want to compare the method based on a regular coverage of the search space ($\mathcal{M}_1$) with the method based on adaptive refinements and quality function $\mathcal{G}_2$ ($\mathcal{M}_2$).[1] Furthermore, we want to consider the possibility to employ other schemes for front literal generation based on the use of other bounds.

We want to start with considering the possible benefits of the front literal generation via $\mathcal{M}_2$. Basically, we have detected two benefits arising from the use of $\mathcal{M}_2$ instead of $\mathcal{M}_1$. On the one hand we can observe the expected effect, namely that needed lemmas are not selected when using breadth-first search because the resources were too small in order to create front literals similar to a useful lemma. But when using $\mathcal{M}_2$ a selection of the lemmas takes place. This effect is responsible for the additional solution of the problems COL060-1 and COL061-1 when using $\mathcal{M}_2$ instead of $\mathcal{M}_1$ (using the distance d_{VS}). Thus, in the COL domain our improved front literal generation method improves on the (good) results of $\mathcal{M}_1$ by the solution of two new problems (by not simultaneously loosing some problems).

On the other hand the effect occurred that a smaller number of front literals seemed to be sufficient in order to choose appropriate lemmas. Since we implicitly obtain higher inference resources in interesting areas of the search space we were able to restrict ourselves to a smaller number of front literals without missing the front literals needed in order to select useful lemmas. This smaller number of front literals normally also reduces the number of chosen lemmas. Thus, in a lot of cases we were able to reach the same proofs as when using breadth-first search (for front literal generation) but with a significantly smaller number of chosen lemmas. Consequently, we decided to work in connection with adaptive refined front literal generation with a number of front literals which has been about 20% smaller as when employing breadth-first search.

Consider e.g. the example BOO006-2 of the BOO domain. Method $\mathcal{M}_1$ solves the problem (employing d_F) in 24 seconds. Our adaptive refined version only needs 10 seconds. This is due to the fact that, though the same proof is found, considerably less lemmas are chosen (21 instead of 27). Employing $\mathcal{M}_1$ 160 front literals are chosen (with inference resource 7) where 169 lemmas are generated.

[1] With quality function $\mathcal{G}_1$ similar results could be obtained.

Problem	Setheo	$\mathcal{M}_1$	$\|F\|$	$\|L\|$	$\mathcal{M}_2$	$\|F\|$	$\|L\|$
BOO003-2	344s	2s	104	16	6s	73	14
BOO004-2	497s	2s	104	16	6s	73	15
BOO005-2	629s	21s	160	25	10s	126	19
BOO006-2	387s	24s	160	27	10s	126	21
BOO012-2	—	131s	98	23	131s	78	23
BOO012-4	—	408s	98	17	390s	90	16
Problem	Setheo	$\mathcal{M}_3$	$\|F\|$	$\|L\|$	$\mathcal{M}_{wd}$	$\|F\|$	$\|L\|$
BOO003-2	344s	6s	87	14	6s	73	14
BOO004-2	497s	6s	87	14	6s	73	14
BOO005-2	629s	20s	137	23	16s	130	20
BOO006-2	387s	—	137	24	20s	130	21
BOO012-2	—	—	82	21	—	80	23
BOO012-4	—	—	90	16	—	88	17

Table 3: Comparison of Different Techniques for Front Literal Generation

When using $\mathcal{M}_2$ only 126 front literals are generated by successively performing a refined front literal generation at interesting points of the search space starting with 54 front literals created with inference resource 6. Although less front literals are generated we are able to identify the useful lemmas since a chosen tableau for front literal generation is identical to an initial part of the proof. Note that with only 54 front literals no appropriate estimations are possible such that the method $\mathcal{M}_1$ cannot be improved by simply reducing the number of generated front literals. Similar effects occurred all over the BOO domain resulting in smaller run times as mentioned previously.

This observations naturally gives raise of the question whether or not it may be only needed to employ bounds which can regulate the number of front literals in a more fine-grain way than the inference bound and use a slightly smaller number of front literals as chosen by the inference bound in our experiments. Thus, we experimented with the weighted-depth bound which was employed in such a way that a similar number of front literals is generated as when using method $\mathcal{M}_2$ (cp. table 3, method $\mathcal{M}_{wd}$). But obviously this method is not well suited for generating front literals. This appears to be comprehensible since we cannot obtain a representative view of the search space and have some unneeded depth restriction when creating front literals.

In addition to these experiments we depict in table 3 results with an adaptive refined method which favors tableaux as starting points for front literal generation which are predecessor nodes of tableaux where the front literals are dissimilar to lemmas ($\mathcal{M}_3$). The results show that indeed an refinement at interesting points of the search space (as defined in section 6.2) is needed in order to obtain sensible estimations. The table shows the run time as well as the number of front literals ($|F|$) and chosen lemmas ($|L|$) for some selected examples in the BOO domain.

9 Discussion

We have introduced the principles of a similarity-based lemma use in model elimination. In order to judge similarity we have developed a general framework which is based on the (fast) simulation of deduction processes. In this paper we concentrated on measuring similarity based on the complete lemma delaying tableaux enumeration method. This method consists of the generation of front literals as well as of the measurement of unification distances between literals. In order to generate front literals we developed methods for covering a large search space. Similar to conventional simulation processes applied for numerical simulation we developed a method based on a regular coverage of the set of front literals (breadth-first search until a given maximal depth). Furthermore, methods for an adaptive refined generation of front literals at certain interesting points of the search space have been applied. The more flexible method based on adaptive refinements indeed proved to be superior to the conventional method in our experiments. In order to measure unification distances we developed a couple of possible methods well suited in different (usually a priori identifiable) situations. Experiments with the similarity-based filtering of lemmas demonstrated the superiority to conventional filter methods.

In summation our approach can be seen as a similarity-based method for restructuring the search space (expand the search space at certain interesting points). Up to now there have been only few approaches for a similarity-based restructuring of the search space.

There are approaches for controlling the search via difference reduction techniques where techniques similar to our pseudo-unifying method have been used. In [BS88, Dig85] or [BW93] techniques for *partial* or *difference unification* have been applied, respectively, with the aim to obtain control knowledge to guide the search. In these approaches the detected differences have been *explicitly* exploited in order to guide the deduction system. Paramodulation steps or rippling were intended to reduce the differences remaining after unification. Thus, the indeterminism of the applied (partial) unification algorithm causes severe problems. It remains unclear which substitutions can be gainfully applied. Furthermore, one has to cope with the question how to reduce the differences which remain after instantiation. Since we use the difference test only to estimate whether a lemma may be *useful in general* but not whether it should be used at a certain position, i.e. in order to prove *a certain subgoal* (note that a lemma may not logically imply a front literal which is most similar to it), we do not fall into these drawbacks.

Structural difference measures which are subsumed by our simple unification distance measure based on features as introduced in section 5 have been applied in [DF94]. The structural distances are explored in order to construct a heuristic for a saturating theorem prover, i.e. to control the selection of formulae to be processed. Thus, similar to our approach no explicit use of the *difference information* is made. But a main difference is that no explicit consideration of possible *deductions* is made in [DF94]. Only local assessments according to a distance w.r.t. an actual proof goal are done. Thus, this technique works only in very specialized domains where only facts similar to a proof goal in the considered sense are needed or in combination with general purpose saturating heuristics. Using our technique which tries to simulate the process of deduction and considers whole proof attempts we can obtain much better estimations of the usefulness of certain facts.

Future work will deal with the development of further distance measures. Since our methods may have weaknesses in domains where structural differences may not be very meaningful (e.g., domains which are closely related to propositional logic) the development of further methods based on the simulation of other tableaux enumeration procedures is sensible. The *lemma preferring* method as introduced in [Fuc97a] may allow for better estimations in such cases. Furthermore, it seems to be very interesting to support the application of lemmas by the use of appropriate bounds which favor proofs containing lemmas. Based on the results of the simulation process it remains to be investigated whether properties of a closed tableau (with lemmas) may be extracted from the similarity test. This may help to restructure the search space in such a manner that proofs with lemmas are favored.

References

[AL97] O.L. Astrachan and D.W. Loveland. The use of Lemmas in the Model Elimination Procedure. *Journal of Automated Reasoning*, 19(1):117–141, 1997.

[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.

[Ave95] J. Avenhaus. *Reduktionssysteme*. Springer, 1995.

[BS88] K.H. Bläsius and J.H. Siekmann. Partial Unification for Graph Based Equational Reasoning. In *Proceedings of CADE-9*, pages 397–414. Springer, 1988.

[BW93] D. Basin and T. Walsh. Difference Unififcation. In *Proceedings of IJCAI-93*, 1993.

[DF94] J. Denzinger and Matt. Fuchs. Goal oriented equational theorem proving. In *Proceedings of KI-94*. Springer, 1994.

[DGH+94] B. I. Dahn, J. Gehne, Th. Honigmann, L. Walther, and A. Wolf. *Integrating Logical Functions with ILF*. Preprint, Humboldt University Berlin, Department of Mathematics, 1994.

[Dig85] V.J. Digricoli. The management of heuristic search in boolean experiments with RUE resolution. In *Proceedings of IJCAI-85*, 1985.

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

[Fuc97a] M. Fuchs. Principles of Lemmaizing based on Similarity Criteria. AR-Report AR-97-02, Technische Universität München, Institut für Informatik, 1997.

[Fuc97b] M. Fuchs. Similarity-Based Lemma Generation for Model Elimination. In *Proc. FTP-97 (to appear)*, 1997.

[Har96] J. Harrison. Optimizing proof search in Model Elimination. In *Proceedings of CADE-13*, pages 313–327. Springer, LNAI 1104, 1996.

[Iwa97] K. Iwanuma. Lemma Matching for a PTTP-based Top-down Theorem Prover. In *Proceedings of CADE-14*, pages 146–160, Townsville, Australia, 1997. Springer LNAI 1249.

[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.

[MIL+97] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. The Model Elimination Provers SETHEO and E-SETHEO. *special issue of the Journal of Automated Reasoning*, 1997.

[Ric89] M.M. Richter. *Prinzipien der künstlichen Intelligenz*. Teubner, 1989.

[Ric92] M.M. Richter. Classification and Learning of Similarity Measures. In *Proceedings der Jahrestagung der Gesellschaft für Klassifikation*. Springer, 1992.

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

[SSY94] G. Sutcliffe, C.B. Suttner, and T. Yemenis. The TPTP Problem Library. In *Proceedings of the 12. International Conference on Automated Deduction (CADE)*, pages 252–266. Springer LNAI 814, 1994.

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