Controlled Use of Clausal Lemmas in Connection Tableau Calculi

MARC FUCHS†

Fakultät für Informatik, TU München, Germany

Proof procedures based on model elimination or the connection tableau calculus have become more and more successful. But these procedures still suffer from long proof lengths as well as from a rather high degree of redundant search effort in comparison with resolution-style search procedures. In order to overcome these problems we investigate the use of clausal lemmas. We develop a method to augment a given set of clauses by a lemma set in a preprocessing phase and discuss the ability of this technique to reduce proof lengths and depths and to provide an appropriate reordering of the search space. We deal with the basic connection tableau calculus as well as with several calculus refinements and extensions. In order to control the use of lemmas we develop techniques for selecting relevant lemmas based on partial evaluation techniques. Experiments with the prover Setheo performed in several domains demonstrate the high potential of our approach.

1. Introduction

Top-down (backward-chaining) and bottom-up (forward-chaining) approaches for automated theorem proving in first-order logic each have specific advantages and disadvantages. Top-down approaches (like model elimination in Loveland, 1968a, 1978, or the connection tableau calculus in Letz, 1993; Letz et al., 1994) are goal oriented but suffer from long proof lengths and the lack of effective redundancy control mechanisms. Bottom-up approaches (like superposition, e.g. Plaisted, 1993; Bachmair and Ganzinger, 1994) provide more simplification power but suffer from their search method which normally neglects the goal to be proved. Thus, an integration of these two paradigms is desirable. In the following we want to consider how to integrate bottom-up elements into the top-down oriented connection tableau calculus.

Our approach is based on the work done in Schumann (1994) and Fuchs (1998a,b). There, in order to refute a set of input clauses with the connection tableau calculus, in a preprocessing phase a set of unit lemmas is created and the clauses are augmented by these bottom-up generated formulas. The lemmas have the ability to shorten the proof length when dealing with Horn clauses and can provide a redundancy control mechanism. A criticism regarding this approach is the fact that when dealing with non-Horn clauses it cannot be guaranteed whether useful lemmas are generated. Thus, in the following we will develop methods for generating an appropriate set of clausal lemmas which can provide proof length reductions. In addition, we will discuss the use of lemmas in connection with

†E-mail: fuchsm@informatik.tu-muenchen.de
refinements (structural restrictions of the allowed tableaux) and extensions (additional inference rules) of the basic connection tableau calculus.

A second main topic, besides the development of appropriate lemma generation techniques, is the intelligent control of their application. An uncontrolled use of all generated lemmas (although useful lemmas are generated which can lead to a proof length reduction) normally increases the branching rate of the search tree in such a way that the advantages are outweighed (e.g., Markovitch and Scott, 1989; Minton, 1990; Markovitch and Scott, 1993; Fuchs, 1998a,b). Criteria are needed in order to select some relevant lemmas. Thus, in this article we will develop methods for selecting lemmas which appear to be relevant for a given proof task. In the past, criteria based on syntactic properties or the derivation tree of a possible lemma have been used for this purpose. For instance, in Astrachan and Stickel (1992) and Schumann (1994) short clauses are favored in order to increase the probability that the lemmas can be applied during the proof process. Additionally, lemmas with large derivations are favored (e.g., in Markovitch and Scott, 1993; Fuchs and Wolf, 1998) because they can possibly provide large search reductions. These criteria, however, work rather uninformedly since a lemma is judged without considering the given proof task and the search scheme which is used. Thus, we introduce an approach for selecting lemmas which considers the given clauses to be refuted as well as the search method to be employed.

The article is organized as follows: after a short introduction to the basic connection tableau calculus CTC in Section 2 we outline general principles and problems regarding the use of lemmas in connection tableau calculi in Section 3. In addition, we introduce in this section the working scheme of our specific lemma technique (generation of a pool of lemma candidates and selection of lemmas from this pool). After that we discuss specific aspects in more detail. We start with techniques for generating a pool of lemma candidates. In Section 4 we present such techniques when dealing with the basic calculus CTC. We will show how we can derive clauses (lemma candidates) which can provide guarantees of proof length and depth reduction and discuss which advantages certain iterative deepening search methods can take from the application of the lemma candidates. After that we deal with refinements and extensions of CTC and discuss their influence on possible proof length and depth reductions as well as on an appropriate reordering of the search space. Then, we deal with techniques for selecting important lemmas. In Section 5 we present a notion of relevance of a lemma. Then we deal with methods for choosing important lemmas based on this notion of relevance in Section 6. We conclude the article with an experimental study conducted with the model elimination prover SETHEO (Letz et al., 1992) and a discussion in Sections 7 and 8, respectively.

2. Connection Tableau Calculus

In the following we are interested in refuting a set $C$ of input clauses with the connection tableau calculus. We use standard notations for terms, literals, clauses, and substitutions.

2.1. THE CALCULUS

The connection tableau calculus works on connected tableaux or connection tableaux for $C$. A (clausal) tableau $T$ for $C$ is a tree whose non-root nodes are labeled with literals and that fulfills the condition: if the immediate successor nodes $N_1, \ldots, N_n$ of a node $N$ of $T$ are labeled with literals $l_1, \ldots, l_n$, then the clause $l_1 \lor \cdots \lor l_n$ (tableau clause) is an
instance of a clause in $\mathcal{C}$ (see also Fitting, 1996). If a tableau clause in $T$ is an instance of an input clause $C$ we also say $C$ appears in $T$. A tableau is called connected if each inner node $N$ (non-leaf node) which is labeled with literal $l$ has a leaf node $N'$ among its immediate successor nodes that is labeled with a literal $l'$ complementary to $l$. The tableau clause which is successor of the unlabeled root node is called the head clause of the tableau. A subtableau of a tableau $T$ with head node $N$ is a labeled subtree of $T$ that contains a non-root node $N$ and all successor nodes of $N$ in $T$. Furthermore, all labels are equal to those in $T$. The literal associated with $N$ is called the head literal of the subtableau. We should note that the name subtableau is somewhat misleading since a subtableau is not a clause tableau for $C$.

The inference rules of the basic connection tableau calculus $\text{CTC}$ consist of start, extension, and reduction. The start rule allows us to perform a standard tableau expansion that can only be applied to a trivial tableau, i.e. one consisting of only one node. We furthermore restrict the start rule in such a way that it can only expand negative clauses. An expansion step means selecting a clause $C$ from $\mathcal{C}$, obtaining a new variant $C'$ of $C$, and attaching the literals of $C'$ to a leaf node of an open branch, i.e. a branch that does not contain two complementary literals. The clause attached by the start rule is also called start clause. Tableau reduction closes a branch by unifying the literal at the leaf of the open tableau branch with the complement of a literal $r$ (denoted by $\neg r$) on the same branch, and applying the substitution to the whole tableau. Extension is performed by selecting a literal $s$ at the leaf node of an open branch in the tableau $T$, performing an expansion step to $s$, and immediately performing a reduction step with $s$ and one of its newly created successors. In the following, an inference is given as a tuple $I = (r,a)$ where $r$ specifies the inference rule which has been used, and $a$ specifies the input clause applied (for extension and start) or the node of the reduction partner which is used.

A literal $s$ at the leaf node of an open branch $N$ is called a subgoal. Furthermore, we will also call $N$ a subgoal. The subgoal tableau of a connection tableau $T$ for $\mathcal{C}$ is a labeled subtree of $T$ which contains all subgoal nodes of $T$ (inclusive the labels), all ancestor nodes of the subgoal nodes of $T$ (with labels), and for each of these nodes all brother nodes in $T$ (inclusive labels). The last condition ensures that the subgoal tableau of $T$ is a connection tableau for $C$. A subgoal $s$ is closed or solved if it becomes the head literal of a closed subtableau after performing some inferences. A tableau is closed if all its branches are closed. A subtableau of a tableau $T$ is closed if all its branches are contained in branches which are closed in $T$. If a subgoal $s$ can be closed by performing the inferences $I_1, \ldots, I_n$ (involving substitutions $\sigma_1, \ldots, \sigma_n$), i.e. the instance $\sigma_1 \ldots \sigma_n$ becomes head literal of a closed subtableau, we call $\sigma_1 \ldots \sigma_n$ a solution of $s$. A closed tableau is called a proof. A closed subtableau of a tableau $T$ is a subproof (in $T$).

The depth of a tableau is the maximal depth of a node decremented by 1. The depth of the root node is 0, the depth of its immediate successor node in the tableau tree is 1, and so on. The proof depth of a given proof is the depth of the respective closed tableau. (Thus, we do not actually consider the attachment of the start clause to the tableau, i.e. the start expansion, as an inference step.) The depth of a subtableau of a given tableau $T$ is the difference of the maximal depth of a node in the subtableau (w.r.t. $T$) and the depth of the head literal of the subtableau. The subproof depth of a subproof is the depth of its associated subtableau. The proof length of a closed tableau (proof) $T$ which can be derived by inference rule applications is the number of inferences needed to infer $T$ starting from a trivial tableau decremented by 1. The subproof length of a given subproof $T'$ of a proof $T$, denoted by $I(T')$, is given as follows. Let $S'$ be a clausal tableau obtained
from $T$ by deleting the nodes below the node which is associated with the subproof head. Then $I(T')$ is the maximal number of inference steps needed to infer a tableau $S$ which is equal (modulo renaming variables) to $T$ starting from $S'$.

Finally, we want to introduce a (rather restricted) notion of subsumption on tableaux which will be useful later. We employ the following form of subsumption. We say a tableau $T_1$ subsumes another tableau $T_2$ (denoted by $T_1 \sqsubseteq T_2$) if the subgoal tableau $T'_1$ of $T_1$ subsumes the subgoal tableau $T'_2$ of $T_2$. This is the case if the trees underlying $T'_1$ and $T'_2$ are isomorphic and there is a substitution $\sigma$ such that for any pair of associated nodes $N_1$ and $N_2$ in $T'_1$ and $T'_2$ with literals $l_1$ and $l_2$, respectively, holds: $l_1 \sigma = l_2$.

2.2. PROOF SEARCH

The notion of a tableau derivation and a (tableau) search tree is important: we say $T \vdash T'$ if (and only if) tableau $T'$ can be derived from $T$ by applying the start rule if $T$ is the trivial tableau, or by an extension/reduction step to a subgoal in $T$. The search space is given by a tableau search tree $T$ defined as follows (see also Figure 1): a search tree $T$ for a set $C$ of clauses is a labeled tree, whose root is labeled with the trivial tableau. Every inner node in $T$ labeled with tableau $T$ has as immediate successors a set of nodes $\{N_1, \ldots, N_n\}$ if $n$ connection tableaux $T_1, \ldots, T_n$ can be derived from $T$ by performing different inferences. Moreover, $N_i$ is labeled with $T_i$, $1 \leq i \leq n$.

The connection tableau calculus is sound and complete in the following sense (see Letz, 1993; Letz et al., 1994). The calculus is sound, i.e. if a closed connection tableau for a clause set $C$ can be derived, then $C$ is unsatisfiable. Furthermore, the calculus is complete. For each unsatisfiable set $C$ of clauses there exists a node in the search tree which is marked with a closed connection tableau for $C$.

Interestingly, the order in which one tries to solve open subgoals has no influence on the fact of whether a closed tableau can be found. A subgoal selection function $\phi$ is a function which assigns to each open tableau a subgoal. The tableau search tree can be decreased by the use of a fixed subgoal selection function $\phi$. We can restrict the derivation relation by allowing $T \vdash T'$ only if an inference rule is applied to $\phi(T)$. A subgoal selection
function is a depth-first selection function if it selects from each open tableau a subgoal with a maximal depth. The choice of a fixed subgoal selection function does not affect the completeness of the calculus. For each tableau that can be found with subgoal selection function \( \phi_1 \), a tableau can be found with subgoal selection function \( \phi_2 \) which is equivalent modulo renaming variables.

Often the tableaux enumeration procedures which are employed do not construct all tableaux in \( T \) in an explicit manner (but see Baumgartner and Brüning, 1997, for a generative approach). Hence, we assume that implicit enumeration procedures are employed that apply consecutively bounded iterative deepening search with backtracking (e.g. Korf, 1985). In this approach, iteratively larger finite initial parts of the search tree \( T \) are explored with 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 defines w.r.t. a fixed natural number, a so-called resource, a finite initial segment of the search tree. A completeness bound is a function which assigns to each connection tableau of the search tree a natural number. Furthermore, for each \( n \in \mathbb{N} \) there is only a finite number of tableaux which are mapped by \( B \) to a value which is smaller than or equal to \( n \), and tableaux which are successors of a tableau \( T \) (w.r.t. the derivation relation) obtain larger or equal bound values as \( T \). Then, the finite segment of the search tree defined by the completeness bound and the resource value \( n \) contains all tableaux which are mapped by \( B \) to a value which is smaller than or equal to \( n \). Iterative deepening using a bound \( B \) is performed by starting with a basic resource value \( n \in \mathbb{N} \) and iteratively increasing \( n \) until a proof is found within the (increasing) finite initial segment of \( T \) defined by \( B \) and \( n \). Prominent completeness bounds are the depth bound, inference bound, and weighted-depth bound.

The depth bound maps a tableau to its depth. Thus, it restricts the depth of tableaux which is allowed in a segment according to a resource \( n \). In practice, the depth bound is quite successful (cf. Letz et al., 1994; Harrison, 1996) but it suffers from the large increase of the finite initial segment (defined by resource \( n \)) caused by an increase of \( n \). The inference bound maps a tableau to its length. Thus, it permits a level-by-level exploration of the search tree (cf. Stöckel, 1988). The length of derivations for tableaux in a segment is bounded by the resource \( n \). In comparison with the depth bound, the inference bound provides a smooth increase of the search space, but the bound is inferior to the depth bound in practice (see Letz et al., 1994; Harrison, 1996). In order to combine the advantages of depth and inference bound, the weighted-depth bound was introduced by Moser et al. (1997). This bound describes a class of possible bounds that restrict the tableau depth as well as the number of inferences allowed to infer a specific tableau. The configuration used within the prover SETHEO (see Moser et al., 1997) has proved to be quite successful.

### 3. Integrating Bottom-up Lemmas in Connection Tableau Calculi

We are interested in assisting the refutation process of a given set \( C \) of input clauses by the generation of clausal lemmas. In the following let \( T \) be a search tree for \( C \). We consider lemmas which are created as follows from a connection tableau \( T \) for \( C \). Assume that \( s \) is the head literal of the closed subtableau \( T^s \) of \( T \). Let \( l_1, \ldots, l_n \) be all literals that are used to close some literals in \( T^s \) by reduction and that are outside of \( T^s \). Then, the clause \( \neg s \lor \neg l_1 \lor \cdots \lor \neg l_n \) may be derived as a new lemma and added to the input clauses (e.g. Loveland, 1968b; Letz et al., 1992).
Now, we briefly want to discuss which advantages and disadvantages the use of lemmas can have in general. Then, we outline principles of our specific lemma technique and introduce the basic components of a lemma-based theorem prover. Later on, we will resume the discussion from Section 3.1 in order to examine in which way the form of the search space is changed by our specific lemma approach.

3.1. ADVANTAGES AND DISADVANTAGES REGARDING THE USE OF LEMMAS

At first sight, the use of (generalized) lemmas could be interpreted as introducing new edges into the search tree \( T \) because certain subdeductions can be reduced to one inference by applying a lemma. We should note, however, that the use of lemmas also inserts new nodes into the search tree. This is because the structure of a tableau \( T_1 \) where a lemma is applied differs from the structure of an, in other parts, equal tableau \( T_2 \) where the lemma proof is “unfolded”. Considering the bounds introduced in Section 2, \( T_1 \) can be enumerated with a resource value which is smaller than or equal to that needed to enumerate \( T_2 \). Under consideration of these remarks, we now summarize the advantages and disadvantages of using lemmas in connection with iterative deepening procedures. This is similar to Minton (1990) where the utility of macro operators is discussed.

A minor advantage of introducing a lemma is the advantage of decreasing path costs, i.e. the costs of reproducing the inferences needed for its proof. The major advantage of using lemmas is that they allow the restructuring of the search space. On the one hand, one can save the possibly high search effort needed for proving a useful lemma (assuming the lemma proof can be expanded within the finite segment of \( T \) to be considered). On the other hand, it is possible that a closed tableau can be reached within a smaller resource value (“resource reduction”). Then, the reordering effects may allow the solving of problems that were previously out of reach because the search procedure gets lost in the (usually exponentially) larger segment of the search tree defined by a larger resource.

The main disadvantage regarding the use of lemmas is the increase of the branching rate of the search tree. Firstly, duplications of segments of the search space can occur caused by a duplicated solution of a subgoal via a lemma and by expanding its proof. This disadvantage, however, can be overcome with local failure caching techniques when restricting the search tree by a depth-first subgoal selection function (see Moser et al., 1997). Secondly, new superfluous solutions of subgoals may be obtained which do not lead to a proof and which could not be found before within the given finite segment of the search tree. This can considerably increase the search space. Furthermore, the newly introduced lemmas cause the problem that in each inference possibly a high number of lemmas has to be tested in order to determine whether inferences are possible (applicability test). Thus, new unification attempts have to be performed.

3.2. LEMMA GENERATION TECHNIQUES

Basically, lemmas can be generated dynamically during a proof run or statically in a preprocessing phase.

A dynamic generation of lemmas, as performed with unit clauses by Astrachan and Stickel (1992); Iwama (1997); Astrachan and Loveland (1997), permits the generation of lemmas during the proof run. After each successful solution of a subgoal a lemma can possibly be generated and added to the input clauses. The aim of this type of lemma
generation is to produce lemmas that are able to reduce the search amount by eliminating repeated subdeductions. Thus, it can be viewed as a kind of redundancy control mechanism. One criticism regarding this type of lemma generation is the fact that it is unclear whether or not useful lemmas can be generated. There is no guarantee that lemmas can be produced during the proof run which can contribute to a proof, i.e. which can be “re-used”. Furthermore, the lemmas are usually not as general as possible due to instantiations coming from the solutions of subgoals previously solved. This can reduce the applicability of a lemma although the “generalized” proof could be re-used for refuting the input clauses. An extraction of the generalized proof is not sensible during the proof run since this would be too expensive compared with the conventional inferences.

There are approaches to restrict the application of the produced lemmas. In Iwanuma (1997), the use of a lemma is only allowed if it matches an open subgoal and thus the remaining alternatives in the choice point can be discarded. This remains quite uncontrolled, however, since no estimation of the usefulness of the lemma is performed and an uncontrolled reordering of the search space can take place (lemma use can help to solve subgoals which have been unsolvable before in the considered finite segment of the search tree). In Astrachan and Stickel (1992); Astrachan and Loveland (1997), lemmas are selected without considering the actual proof task and the relevance of a lemma is judged based on its syntactical structure.

In contrast to these methods, the principle of our method is to generate a set of clausal lemmas \( L \) in a preprocessing phase (similar to Schumann, 1994) and to augment the input clauses with these bottom-up generated formulas. Essentially, our prover consists of a bottom-up component which works only in a preprocessing phase and a top-down prover which tries to refute a given set of input clauses (augmented by lemmas) afterwards.

The bottom-up component consists of a lemma generator and a lemma selection component. The lemma generator is responsible for creating a pool of “interesting” lemma candidates by deriving logical consequences from the input clauses. As we will outline in the next section it can provide the generation of a pool of clauses which guarantee a reduction of the proof length compared to the refutation of \( C \) without lemmas. The lemma selection component is responsible for choosing an appropriate subset of the set of generated lemma candidates. Finally, the generated lemmas are employed in addition to the input clauses by a top-down connection tableau-based prover.

4. Bottom-up Generation of Clausal Lemmas

Now, we will deal with techniques for generating lemmas for a set \( C \) of clauses. Furthermore, we discuss the effects of the lemma techniques for reducing the proof length and the proof depth. These are the items which are used by the currently most successful search bounds to define finite segments of the search space. A reduction of the proof length and the proof depth can provide significant advantages for finding a proof. These reordering effects of the search space resulting from using lemmas are discussed in addition to the potential for proof length/depth reduction. We start with the use of lemmas in the conventional connection tableau calculus \( CTC \) as introduced before. We introduce generation procedures for unit and non-unit lemmas and discuss them in detail in Sections 4.1 and 4.2. Then, we deal with refinements and extensions of \( CTC \) in Sections 4.3 and 4.4. We investigate whether the techniques from Section 4.1 and Section 4.2 are still sufficient in order to produce lemmas which are able to reduce proof lengths and depths and to restructure the search space in an appropriate manner.
4.1. UNIT LEMMA GENERATION

The aim of our method is to generate units in the preprocessing phase which are able to close all branches of a closed tableau in a smaller depth. Then a proof length and depth reduction is obtained. As we will see this is possible when dealing with Horn clauses.

Lemma generation. We want to consider a technique for the generation of lemmas in detail which was first introduced by Schumann (1994). There, lemmas are generated according to the following procedure which obtains a clause set \( \mathcal{C} \) as input and produces a lemma set \( \mathcal{U} \) for \( \mathcal{C} \) as output.

**Procedure 4.1. (Unit lemma generation)**

1. Add most general queries \( \neg p(X_1, \ldots, X_n) \) (and \( p(X_1, \ldots, X_n) \) when dealing with non-Horn clauses) to \( \mathcal{C} \) for each predicate symbol \( p \).
2. Enumerate all closed tableaux \( T_1, \ldots, T_k \) that can be obtained within the finite segment of the search tree for \( \mathcal{C} \) which is defined by the depth bound and a depth resource value \( n_D \geq 2 \).
3. Let \( p_i \) be the head literal of tableau \( T_i \). Then, \( \neg p_i \) is obtained as a valid fact. Let \( \mathcal{U}' \) be the clause set \( \{ \neg p_1, \ldots, \neg p_k \} \).
4. Delete all facts from \( \mathcal{U}' \) which are subsumed by a clause from \( \mathcal{C} \) and identify all variants of clauses in the remaining set. This yields a set \( \mathcal{U}'' \). Now, we delete all facts from \( \mathcal{U}'' \) which are subsumed by another fact in \( \mathcal{U}'' \). Finally, we obtain a set of lemma candidates \( \mathcal{U} \).

Lemmas for a set \( \mathcal{C} \) of input clauses can be generated by employing Procedure 4.1 with input \( \mathcal{C} \). Then, the output \( \mathcal{U}_0 = \mathcal{U} \) can be used as lemma set. Additionally, we may use further repetitions of the cycle. Starting with \( \mathcal{C} \) we generate a set \( \mathcal{U}_0 \) of unit lemma candidates as described above. Then, the set \( \mathcal{U}_{i+1} \) is created using the set \( \mathcal{U}_i \cup \mathcal{C} \) as input of the procedure. If we employ \( N_I \geq 0 \) repetitions of the procedure, the unit clauses from the set \( \mathcal{U}_{N_I} \) are possible lemma candidates that can be added to the input clauses. We should note that the lemmas which are generated with an iteration number \( N_I = 0 \) are already non-trivial lemmas. We can generate lemmas whose respective proof has a maximal depth of \( N_I \cdot (n_D - 1) + n_D \). The iterated generation technique allows the use of subsumption and (possibly) selection of lemmas after each cycle.

Proof length/depth reduction. Now, we want to discuss whether we can shorten the minimal proof length or the minimal proof depth for \( \mathcal{C} \). In order to obtain a controlled use of lemmas we first make the following restriction. Lemmas are not allowed to be used as start clauses. Henceforth, in all connection tableau calculi a start expansion with a lemma is forbidden.

We want to introduce some basic notions in order to show in which way we can replace subdeductions by others.

**Definition 4.1.** Let \( \mathcal{C} \) be a set of clauses.

1. Let \( T_0 \vdash T_1 \vdash \cdots \vdash T_n, n \geq 1 \), where the inferences \( I_1, \ldots, I_n \) are applied using a fixed depth-first subgoal selection function \( \phi \). We say \( (I_{i+1}, \ldots, I_j), 0 \leq i < j \leq n \),
is the sequence of proof inferences for a node \( N \) (in \( T_n \)) if the following conditions are satisfied:

- \( N \) is a node in \( T_n \) which is the head of a closed subtableau in \( T_n \). Furthermore, \( N \) is a subgoal in a tableau \( T_k \) for \( 0 \leq k < n \).
- \( T_i \) is the tableau where \( N \) is a subgoal and \( T_i \vdash T_{i+1} \) by performing an inference step with the subgoal \( N \).
- \( T_j \) is the tableau where \( N \) is the head of a closed subtableau and \( N \) is not closed in \( T_i, T_{i+1}, \ldots, T_{j-1} \).

Furthermore, \((T_{i+1}, \ldots, T_j)\) is the proof sequence for \( N \) (induced by the inference chain \((I_{i+1}, \ldots, I_j)\)).

(2) Let \( T_0 \vdash T_1 \vdash \cdots \vdash T_n, n \geq 1 \), where the inferences \( I_1, \ldots, I_n \) are applied using a fixed depth-first subgoal selection function \( \phi \). Let \( I = (I_1, \ldots, I_n) \) be a sequence of proof inferences for \( N \) in \( T_n \). Let \( T_0 = T_0' \vdash T_1' \vdash \cdots \vdash T_m' \) where the inferences \((I_1', \ldots, I_m')\) are used w.r.t. \( \phi \). Let \( I' = (I_1', \ldots, I_m') \) be a sequence of proof inferences for \( N \) in \( T_m' \). We say \( I \) is more general than \( I' \) if \( T_n' \leq T_m' \).

The following lemma illustrates the fact that sequences of proof inferences of certain subgoals can be replaced by other more general proof inference chains without any harm (without limiting the solvability of a tableau). Furthermore, the more general inference chains can be executed starting from more general connection tableaux.

**Lemma 4.1.** Let \( \mathcal{C} \) be a set of clauses. We consider a connection tableau calculus where a fixed depth-first subgoal selection function is used. Let \( T_0 \vdash T_1 \vdash \cdots \vdash T_n \) be a tableau derivation where the inferences \( I_1, \ldots, I_n \) are applied. Let \( I = (I_1, \ldots, I_n) \), \( 0 \leq i < j \leq n \), be the sequence of proof inferences for a node \( N \). Let \( I' \) be another sequence of proof inferences for \( N \) in a tableau \( T_m \) inferred from \( T_0 \) with the inferences \( I_1, \ldots, I_i \) and the sequence \( I' \). Let \( I' \) be more general than \( I \).

Let \( T_0 \vdash T_1' \) and \( T_i' \leq T_i \). Let \( T_i' \vdash T_j' \) by executing the inferences in \( I' \). Then, there exist \( T_{j+1}', T_{j+2}', \ldots, T_n' \) where \( T_j' \vdash T_{j+1}' \vdash \cdots \vdash T_n' \) and \( T_i' \leq T_l \) for all \( l \in \{j, \ldots, n\} \).

With the help of this lemma we investigate now which proof length and depth reductions are possible when employing unit lemmas in connection with Horn clauses.

**Theorem 4.1.** Let \( \mathcal{C} \) be an unsatisfiable set of Horn clauses. Let \( \mathcal{U}_N \), be a set of unit lemma candidates for \( \mathcal{C} \) generated as described above using a resource \( n_d \geq 2 \), and an iteration number \( N_I \geq 0 \). Let the minimal proof depth for refuting \( \mathcal{C} \) be greater than 1.

(1) Let \( n \) be the minimal proof depth of a proof for \( \mathcal{C} \). Let \( n' \) be the minimal proof depth of a proof for \( \mathcal{C} \cup \mathcal{U}_N \). Then it holds that \( n' = \max\{1, n - (N_I + 1) \cdot (n_d - 1)\} \).

(2) Let \( n \) be the minimal proof length of a proof for \( \mathcal{C} \). Let \( n' \) be the minimal proof length of a proof for \( \mathcal{C} \cup \mathcal{U}_N \). Then it holds that \( n' < n \).

**Proof.** (1) At first we show the following. Let \( T \) be a tableau which is enumerated with a depth-first subgoal selection function. Let \( T_0 \vdash \cdots \vdash T_k = T \) be the derivation of \( T \) performed with the inferences \( I_1, \ldots, I_k \). Let \( N \) be the head node of a closed subtableau in \( T \) which has a depth smaller than or equal to \( N_I \cdot (n_d - 1) + n_d \). Let \( S = (I_{i+1}, \ldots, I_j) \), \( 0 \leq i < j \leq k \), be the sequence of proof inferences for \( N \). Evidently, because of our lemma
generation procedure, the following holds: there is a unit lemma \( l \in \mathcal{U}_N \) such that the application of \( l \) for an extension step to the subgoal \( N \) in \( T_1 \) is a more general sequence of proof inferences than \( S \).

Now, let \( P \) be a proof for \( \mathcal{C} \) with minimal depth. W.l.o.g. we assume that \( P \) is derived using CTC and a depth-first subgoal selection function. Via induction over the number of subgoals in depth \( \max((n-(N_I+1)(n_D-1),1)) \) of \( P \) we can show the following by means of the above considerations and Lemma 4.1: we can successively replace each sequence of proof inferences of a node in depth \( \max((n-(N_I+1)(n_D-1),1)) \) in \( P \) by a (more general) sequence of proof inferences which consists of a single extension step with a unit lemma. Then, a proof can be enumerated with lemmas which has a proof depth which is smaller than or equal to \( \max((n-(N_I+1)(n_D-1),1)) \). Since, in general, no proof depth reduction by an amount which is larger than \( (N_I+1)(n_D-1) \) can be obtained we have a proof depth reduction from \( n \) to \( \max((n-(N_I+1)(n_D-1),1)) \) when using lemmas.

(2) Analogous with the first part of the proof we can reduce the depth \( n \) of a proof \( P \) for \( \mathcal{C} \) which is minimal w.r.t. the proof length to \( \max((n-(N_I+1)(n_D-1),1)) \). Thus, the new proof depth of \( P \) when employing lemmas is smaller than the old proof depth \( n \) (since \( n_D \geq 2 \) and \( n \geq 1 \)). If the new proof depth is 1 let \( N \) be a node occurring in depth 1 (labeled with \( s \)) whose subproof \( P_s \) has a depth which is greater than 1. Otherwise, let \( s \) be a literal at a node \( N \) in \( P \) which has the depth \( n-(N_I+1)(n_D-1) \) and whose subproof \( P_s \) has depth \( N_I \cdot (n_D-1) + n_D. \) (Such nodes must exist in \( P_s \).) The proof inferences for \( N \) can be replaced by an extension with a unit lemma resulting in a new subproof \( P'_s \). It holds \( I(P'_s) < I(P_s) \). Thus, the proof length of \( P \) can be shortened with lemmas. \( \square \)

Structure of the search space. Now we want to analyze in which way the search space is reordered when all lemma candidates are used in order to refute a set of Horn clauses (see also Fuchs, 1998b). The reordering effects depend heavily on the completeness bound which is applied to refute \( \mathcal{C} \cup \mathcal{U}_N \). In the following we consider only sets \( \mathcal{C} \) whose refutation requires a proof depth greater than \( (N_I+1)(n_D-1) \). We will analyze whether because of the lemmas more or less inferences are possible in the minimal segment of the search tree for \( \mathcal{C} \cup \mathcal{U}_N \) which contains a proof compared with the minimal segment of the search tree for \( \mathcal{C} \) which contains a proof.

When using the depth bound we can decrease the resource which is at least needed in order to obtain a closed tableau from \( n \) to \( n' = n - (N_I+1) \cdot (n_D-1) \) (see above). Thus, despite the use of lemmas we have no new solutions of subgoals in the smallest segment of the search tree for \( \mathcal{C} \cup \mathcal{U}_N \) which contains a proof when compared with the segment of the search tree for \( \mathcal{C} \) defined by resource \( n \). This is because all solutions of subgoals with lemmas and resource \( n' \) can also be obtained without lemmas and resource \( n \) by “expanding” the lemma proofs. Furthermore, techniques such as local failure caching (Moser et al., 1997) can avoid the duplicated exploration of segments of the search tree caused by duplicated solutions of subgoals obtained with lemmas and by expanding the lemma proof (assuming this is possible when using resource \( n' \)). An increase of the number of inferences which are possible in the new proof segment when employing lemmas results from the application test (unification attempt) in order to close a subgoal. Thus, for each subgoal of the old proof segment which also occurs in the new segment, a lemma causes one new unification.

In contrast, inferences can be saved when using lemmas because of the occurring resource reduction (see also Schumann, 1994; Fuchs, 1998b). Then, some inference chains
of the old proof segment can be spared. These are the inferences which are impossible with clauses from \( C \) and the new smaller resource value \( n' \), and which would not lead to solutions of subgoals with resource \( n \) (thus they cannot be simulated with lemmas). This normally is only a minor advantage. More important is that certain subsumed solutions of subgoals cannot be found in the new proof segment. Consider the following example.

Example 4.1. Let \( n > 0 \). We consider the clause set

\[
C = \{ \neg p(X) \vee \neg q(X), q(b), p(X) \vee \neg r(X), p(X) \vee \neg h(X), h(X), r(a_1), \ldots, r(a_n) \}.
\]

We perform proof search with the depth bound. If we do not employ lemmas a resource value of 2 is needed. We can find \( n + 1 \) solutions for the subgoal with literal label \( \neg p(X) \) where the \( n \) solutions \( \sigma_i = \{ X \leftarrow a_i \} \) are more specific than the existing empty solution substitution \( \sigma \).

If we generate lemmas with \( n_D = 2 \) and \( N_I = 0 \), we will generate \( n + 1 \) solutions for the query \( \neg p(X) \) which represent the lemmas \( p(X) \) and \( p(a_i), 1 \leq i \leq n \). The lemmas \( p(a_i), 1 \leq i \leq n \), are deleted and only the lemma \( p(X) \) remains. A proof for the augmented clause set \( C \cup \{ p(X) \} \) can be found with a resource value of 1. Only one solution of the subgoal with label \( \neg p(X) \) exists. This can save a lot of inferences if we modify the example in such a manner that a lot of inferences become possible to the subgoals \( \neg q(a_i) \) which do not lead to a solved tableau. If the tableau enumeration procedure finds the empty solution substitution rather late, then a lot of unnecessary inferences have to take place. (Note also that no known caching mechanism can cope with this problem.) In this case, the bottom-up generation of all solutions of \( \neg p(X) \) and the encoding of the solutions in lemma form involving subsumption is much cheaper than the enumeration of the solutions during the proof run.

A significant improvement can be obtained when only a small subset of \( U_{N_I} \) is used which already is sufficient for a resource reduction. Then, problems that were out of reach before can probably be solved because of a resource reduction from \( n \) to \( n' \) (in a normally exponentially increasing search space when increasing the resource value) and the rather small increase of the size of the finite segment defined by resource \( n' \) caused by the use of a small number of lemmas. In summary, the depth bound appears to be well-suited for the use of lemmas in Horn domains if appropriate lemma selection methods are used. It performs rather well without lemmas and controlled search space reductions are possible.

When applying the inference bound a resource reduction can be guaranteed (the exact value depends on the proof length of the tableaux from which the lemmas are extracted). Thus, analogously to the depth bound certain solutions of subgoals may be saved, e.g. because of the subsumption test. But we should note that also some new solutions of subgoals may occur in the segment which contains a proof when using lemmas compared with the proof segment when not using lemmas. It is possible that misleading paths in the search tree for \( C \) can take more benefits (resource reduction) from applying lemmas than a proof can. This is especially a problem if we deal with rather asymmetric proofs which contain only few long branches. Then, only a few lemmas may be applicable to shorten a proof. But symmetric tableaux which cannot be closed and which have not been in the minimal proof segment of the search tree for \( C \) may profit very much from a lemma use.

Thus, we have a more uncontrolled behavior in comparison with the depth bound. Nevertheless, the inference bound profits from each application of a lemma in form of a
reduction of the resource needed and not only from a reduction of the longest path. This can be useful when filtering lemmas and we are not able to select all lemmas needed to shorten the longest tableau paths. Then the depth bound probably cannot profit from the use of lemmas (although search effort may be saved in order to prove the usable lemmas) but the inference bound probably can.

It is possible that the weighted-depth bound does not profit from the use of lemmas in the form of a reduction of the resource needed to enumerate a proof (cf. Fuchs, 1997b). Thus, if no resource reduction takes place it is apparent that the use of lemmas can increase the size of the proof segment. However, also in the case of a resource reduction it may be that new solutions of subgoals are obtained by using lemmas. Nevertheless, the search space may be reordered in an appropriate way if a small number of lemmas can be selected which are able to shorten the proof length but not all branches of a proof. This is because the bound takes not only the depth but also the length of a proof into consideration. Therefore, in this situation, the bound should be superior to the depth bound where no resource reduction occurs. Furthermore, it is quite probable that it also improves on the inference bound because it is superior to the inference bound without lemmas and also a resource reduction takes place.

Regardless of the bound to be used for the final top-down proof run, the inferences performed in order to create the lemmas have to be considered. If a small value for \( n_D \) is used and the generation process is not iterated, the inferences performed in the preprocessing phase are normally negligible and are outweighed by the inferences spared by subsumption. If we iterate the lemma generation process, a rather restrictive filtering of lemmas is needed. Otherwise in rather large search spaces all tableaux have to be enumerated starting from some kind of most general start clauses. An effective discarding of unnecessary lemmas, however, usually helps to spare a number of inferences in the final proof run which exceeds the number of inferences needed to infer the lemmas by magnitudes.

In Section 7 we will investigate which bound is the most successful when dealing with Horn clauses by some experimental studies. We will see that according to our expectations, especially with the depth bound and appropriate lemma selection methods (which are able to select lemmas which can shorten each branch of a proof), significant improvements of model elimination provers can be obtained. Also the weighted-depth bound proved to be well-suited for the use of lemmas in Horn domains.

4.2. Non-Unit Lemma Generation

When refuting non-Horn clauses a generation of units cannot guarantee a reduction of the proof length. It may be that in all subproofs of a proof that have a depth which is smaller than or equal to \( N_I \cdot (n_D - 1) + n_D \) reduction steps are needed from outside of the subproof. Then, the complements of the head literals of the subproofs are no logic consequences of the tableau clauses occurring in the subproofs. Thus, no unit lemmas can be used to close the head literals of these subproofs. Although experiments performed by Schumann (1994); Astrachan and Loveland (1997) show that nevertheless in some non-Horn domains satisfactory results can be obtained with units it is sensible to develop methods which guarantee that useful (clausal) lemma candidates can be generated.

Lemma generation. In order to replace subproofs where reduction steps from outside are needed non-unit clauses such as \( C = p \lor l_1 \lor \cdots \lor l_m, m \geq 1 \), can be employed. The idea is to close a subgoal \( p' \) where reductions with “higher” literals are needed for its proof by
extension with the clause $C$ (by unifying $p$ and $\sim p'$) and by performing reduction steps into all newly introduced subgoals (instances of $l_1, \ldots, l_m$).

In order to generate a pool of lemma candidates which can provide resource reductions even in the non-Horn case, we modify the preceding procedure to a method for the generation of non-unit lemma candidates for a clause set $C$. These non-unit lemmas are used in addition to the unit lemmas. The following procedure employs a set $C$ of clauses as input and produces a set $N$ of non-unit clauses as output.

**Procedure 4.2. (Non-unit lemma generation)**

1. Add most general queries $\neg p(X_1, \ldots, X_n)$ (and additionally queries $p(X_1, \ldots, X_n)$ when dealing with non-Horn clauses) to $C$ for each predicate symbol $p$ where $n \geq 0$ is the arity of $p$.

2. Enumerate all open tableaux $T_1, \ldots, T_k$ that can be obtained in the finite segment of the search tree defined by $C$, the depth bound, and a resource value $n_D \geq 2$.

3. Let $p_i, 1 \leq i \leq k$, be the head literal of tableau $T_i$. Let $l_1, \ldots, l_{m_i}, m_i > 0$, be the subgoals of $T_i$. Then, $\sim p_i \vee l_1 \vee \cdots \vee l_{m_i}$ is obtained as a new valid clause. We also write $\sim p_i \leftarrow \sim l_1, \ldots, \sim l_{m_i}$ with head $\sim p_i$ and tail $\sim l_1, \ldots, \sim l_{m_i}$. Let $N'$ be the set of all valid clauses which can be derived from $T_1, \ldots, T_k$.

4. Identify all procedural variants in $N'$. This results in a set $N''$. Now, we delete all clauses $h \leftarrow t_1, \ldots, t_n$ from $N''$ which are procedurally subsumed by another clause from $N''$. Finally, we obtain a set of lemma candidates $N$. \((h' \leftarrow t'_1, \ldots, t'_m \text{ procedurally subsumes } h \leftarrow t_1, \ldots, t_n \text{ if there exist a substitution } \sigma \text{ and } \pi : \{1, \ldots, m\} \rightarrow \{1, \ldots, n\} \text{ with } h'\sigma = h \text{ and } t'\sigma = t_{\pi(i)}, 1 \leq i \leq m. \text{ If } \sigma \text{ only renames variables and } m = n \text{ we say } h' \leftarrow t'_1, \ldots, t'_m \text{ and } h \leftarrow t_1, \ldots, t_n \text{ are procedural variants.})

A possibility to create a set of non-unit lemmas is to employ the above procedure and to generate the set $N_0 = N$ using a clause set $C$ as input. If we want to employ an iterated generation of unit and non-unit clauses the generation process of units and non-units has to be interleaved. The sets $U_i$ and $N_i$, $i > 0$, of unit and non-unit lemma candidates are generated according to the Procedures 4.1 and 4.2, respectively, and we have to use $C \cup U_{i-1} \cup N_{i-1}$ as input. In the following let the set $L_i$ of lemma candidates be given by $U_i \cup N_i$.

The following example illustrates the saturation of lemmas with $N_1 = 1$, i.e. the lemma sets $L_0$ and $L_1$ are generated. Furthermore, we show the application of non-unit lemmas in order to shorten a proof.

**Example 4.2.** Let

$$C = \{\sim p(a), \quad p(X) \vee \sim q(X), q(X) \vee \sim r_1(X) \vee \sim r_2(X), \quad r_1(X) \vee \sim u(X) \vee \sim s(X), s(X) \vee p(X), u(X), \quad r_2(X) \vee \sim t(X), t(X)\}.$$ 

Without using lemmas there is the following proof where one reduction step occurs.
We consider the subtableau with the head literal $\neg q(a)$. This subtableau has the depth 3. We show that we can produce a lemma whose application can replace the inferences applied to create the subtableau. We use the parameters $N_I = 1$ and $n_D = 2$ for the lemma generation. In the first generation step of Procedure 4.2 we can derive the following tableau.

The tableau represents the lemma $r_1(X) \leftarrow \neg p(X)$ which is part of $N_0$. Furthermore, evidently the lemma $r_2(X)$ is part of $U_0$. With the help of these lemmas we can derive the following tableau in the next iteration step.

This tableau represents the lemma $q(X) \leftarrow \neg p(X)$. With the help of this lemma the subgoal $\neg q(a)$ can be solved by an extension and a reduction step.
Proof length/depth reduction. When using non-unit lemmas in addition to unit lemmas, proof length and depth reductions can be obtained even when dealing with non-Horn clauses. We want to state the following theorem:

**Theorem 4.2.** Let $C$ be an unsatisfiable set of clauses. Let the minimal proof depth for refuting $C$ be greater than 1. Let $L_{N_I}$ be a set containing all unit and non-unit lemma candidates for $C$ generated in $N_I \geq 0$ iterations according to Procedures 4.1 and 4.2 using a resource $n_D \geq 2$.

1. Let $n$ be the minimal proof depth of a proof for $C$. Let $n'$ be the minimal proof depth of a proof for $C \cup L_{N_I}$. Then it holds that $n' \in \{1, \ldots, \max\{n - (N_I + 1)(n_D - 1), 1\}\}$.
2. Let $n$ be the minimal proof length of a proof for $C$. Let $n'$ be the minimal proof length of a proof for $C \cup L_{N_I}$. Then it holds that $n' < n$.

**Proof.** (1) Analogously to the previous theorem we only have to show the following. Let $T$ be a tableau which is enumerated with a depth-first subgoal selection function. Let $T_0 \vdash \cdots \vdash T_k = T$ be the derivation of $T$ performed with the inferences $I_1, \ldots, I_k$. Let $N$ be the head node of a closed subtableau in $T$ which has a depth smaller than or equal to $N_I \cdot (n_D - 1) + n_D$. Let $S = (I_{i+1}, \ldots, I_j)$, $0 \leq i < j \leq k$ be the sequence of proof inferences for $N$. We have to show that with the help of a (non-unit) lemma $L \in L_{N_I}$ a more general sequence of proof inferences can be achieved which solves $N$ with a subproof of depth 1. Then, analogously to before we obtain a proof depth reduction from $n$ to a value smaller than or equal to $\max\{1, n - (N_I + 1)(n_D - 1)\}$.

Now, we show how to obtain a subproof of depth 1 by using clausal lemmas. Let $s$ be the literal occurring at node $N$ in $T_j$. If no reduction steps are needed in the sequence $S$ of proof inferences for $N$ we can replace $S$ by an extension with a unit lemma analogously to the preceding theorem. Otherwise reduction steps occur in the subproof. Let $r_1, \ldots, r_z$ be the literals in $T_j$ which are closed by earlier reduction steps with nodes $R_1, \ldots, R_z$ from outside the subproof. Due to our lemma generation procedure there exists a clause $C = s' \leftarrow r'_1, \ldots, r'_l \in L_{N_I}$, a substitution $\sigma$, and a function $\pi : \{l_1, \ldots, l_z\} \rightarrow \{1, \ldots, z\}$ such that $s'\sigma = s, r'_1\sigma = \sim r_{\pi(1)}, \ldots, r'_l\sigma = \sim r_{\pi(l)}$. The use of $C$ allows a more general solution of $s$ by performing an extension with $C$ and closing the newly introduced nodes by reduction steps with the nodes $R_{\pi(1)}, \ldots, R_{\pi(l)}$.

(2) We obtain a proof length reduction in an analogous way to that shown in the foregoing theorem. $\Box$

The proof length reduction obtained with lemmas can be larger than in the Horn case since lemma applications may be nested (see below).

**Structure of the search space.** We have noted that the properties of resource reduction when using unit lemmas can be lifted from the Horn case to the non-Horn case when additionally using clausal lemmas. If we take a qualitative look at the segment of the search space for $C \cup L_{N_I}$ which contains a proof, however, we see that due to the use of clauses an increase of the number of solution substitutions can take place (compared with the proof segment of the search tree for $C$) even when using the depth bound. This is because new subgoals (the instantiated negated tail literals of a lemma) are introduced when performing an extension step with a clausal lemma (by extension into the lemma head). Now, when using other clausal lemmas in order to solve these newly introduced
subgoals new solutions of subgoals, previously not obtainable, can be found. This effect
normally leads to an uncontrolled explosion of the search space and may introduce a lot
of misleading proof paths. Consider the following example.

**Example 4.3.** Let $C = \{p_i \lor \neg p_{i+1} : 1 \leq i < 6\} \cup \{p_6, \neg p_2, \neg p_1\}$ be a set of Horn clauses. Without lemmas $C$ can be refuted with the depth bound and a depth resource of 5. In this proof the start clause $\neg p_2$ is used. We should note that no subproof with head $\neg p_1$ is possible in this resource. If we generate unit and non-unit lemmas with an iteration number $N_I = 0$ and $n_D = 2$ we obtain $L_0 = \{p_5\} \cup \{p_i \leftarrow p_{i+2} : 1 \leq i < 5\}$. With this lemma set we can solve $\neg p_2$ as well as $\neg p_1$ with the new minimal resource value 3. Thus, we obtain a larger resource reduction than before but also new solutions of subgoals occur in the new minimal proof segment. We can easily modify the example by extending the query $\neg p_1$ to $\neg p_1 \lor \neg s_1 \lor \cdots \lor \neg s_z$, $z > 0$, and some new clauses which permit many inferences with the new subgoals $\neg s_1, \ldots, \neg s_z$. Then, a large increase of the minimal segment which contains a proof can take place (although the resource reduction is larger than when only using unit lemmas).

Furthermore, we should note that the simple addition of a lemma clause $p \leftarrow l_1, \ldots, l_n$ to $C$ can lead to an uncontrolled increase of the branching rate of the search tree since an extension step with a lemma can be performed by unification of a subgoal with a tail literal (which is only intended to be used for reduction steps). Thus, it can happen that a (valid) rule such as $\neg l_i \leftarrow l_1, \ldots, l_{i-1}, l_{i+1}, \ldots, l_n, \neg p$ may be used although it is not needed for a proof length reduction (and would therefore not be generated when applying the lemma generation method as described before). In Section 4.3 we will deal with methods to overcome all of these problems.

Finally, we want to mention that naturally analogously to before, lemma selection mechanisms can provide a drastic pruning of the search space.

### 4.3. Calculus Refinements

In the following we want to analyze whether the use of lemmas is “compatible” with the use of some calculus refinements. Calculus refinements exclude certain parts from the search space (spare the consideration of the whole search space) but still provide complete search procedures.

Specifically, we are interested in whether resource reductions can still be obtained when employing a restricted calculus and lemmas. We have to consider whether the application of a lemma which is needed in order to enumerate a closed tableau within a smaller inference or depth resource may be forbidden due to the calculus refinements.

Additionally, we want to consider how the structure of the search space is changed by the use of lemmas. We want to examine whether the following situation can occur: solutions of certain subgoals are possible in the minimal proof segment when using lemmas but are impossible in the old minimal proof segment (of the restricted version of $CTC$) due to the inference restrictions used there (although the lemma proof could be found in the old segment when not restricting the calculus). Then, the use of lemmas would import additional redundancy into a restricted version of $CTC$.

We start with a new calculus refinement which limits the application of lemmas. Then, we consider the refinements regularity, tautology-freeness, and subsumption.

For each calculus refinement we briefly introduce the underlying principles. Then, we
describe some methods for lemma generation which may be used when applying the calculus refinement. After that we deal with the effects of using the generated lemmas in a top-down prover which employs the considered calculus refinements.

We want to make some remarks regarding the notation used in the following. All calculus refinements can be applied to a connection tableau calculus which is an extension or restriction of $CTC$. The use of a calculus refinement $ref$ together with a calculus $Calc$ is denoted by $Calc^{(ref)}$ or simply by $Calc_{ref}$. We can apply several refinements simultaneously. Since the order in which a calculus is refined does not matter for our techniques we use $Calc^{(ref_1,ref_2)}$ instead of $(Calc^{(ref_1)})^{(ref_2)}$ and $Calc_{ref_1\cup\{ref_2\}}$ instead of $(Calc_{ref_1})^{\{ref_2\}}$ for a set of refinements $Ref_1$ and two calculus refinements $ref_1, ref_2$.

**Extension restrictions on non-unit lemmas**

As we have seen before the use of non-unit lemmas causes some problems, e.g. new subgoals are introduced when extending a subgoal with a lemma which leads to an uncontrolled increase of the search space. Thus, we want to consider how these problems can be overcome. We restrict the use of non-unit lemmas when employing extension steps.

*Technique.* Firstly, we restrict the application of clausal lemmas in such a way that for all subgoals newly generated when extending a subgoal with a lemma it is forbidden to apply an extension rule to them. Thus, in the conventional calculus $CTC$ only reduction steps can be applied to them. Note that for extensions of the connection tableau calculus (factorization and folding-up, to be described shortly) the additional inference rules can be applied to the subgoals.

Secondly, we do not allow the extension of a subgoal performed by unifying it with a tail literal of a lemma $p ← l_1, \ldots, l_n$ (i.e. we do not generate contrapositives for the clause, cf. Stickel, 1988). Thus, a lemma serves as a kind of procedural clause and not as a declarative clause. This is sensible since we want to replace a certain proof structure by the use of a lemma and are not interested in exploiting the full semantic information provided by the lemma. The calculus refinement is denoted by $extr$.

*Lemma Generation.* In order to generate lemmas we use the lemma generation algorithm as described in Section 4.1 when refuting Horn clauses and use in addition non-unit lemmas created as described in Section 4.2 when dealing with non-Horn clauses. In order to enumerate the lemmas we use $CTC^{extr}$. This is sufficient in order to generate lemmas which represent “most general tableaux” of a depth $N_I \cdot (n_D - 1) + n_D$ (cf. Example 4.2).

*Proof length/depth reduction.* When considering this modified calculus and lemmas we can see that for non-Horn problems we can still reduce proof lengths and have the same potential for proof depth reduction as in the Horn case. It is sufficient to use reduction steps into the (negated) tail literals of non-unit lemmas in order to shorten subproofs where reduction steps from outside are needed (cf. the proof of Theorem 4.2).

*Structure of the search space.* Since no extension steps into the negated tail literals of non-unit lemmas are possible and we do not generate contrapositives of the lemmas, the results regarding the reordering of the search space obtained with unit lemmas and Horn clauses can be lifted to the use of unit and non-unit lemmas and non-Horn clauses. Thus, when using the depth bound for the top-down proof search it is apparent that no new solutions of subgoals are introduced to the minimal proof segment compared with the
“old” minimal proof segment (assuming that the minimal proof depth without lemmas is greater than \((N_I + 1)(n_D - 1)\)). When employing the inference and weighted-depth bound such effects are still possible analogously to the use of unit lemmas in the Horn case.

REGULARITY, TAUTOLOGY-FREENESS, AND SUBSUMPTION

**Technique.** Regularity defines a restriction on the structure of the derivable tableaux by allowing an inference step to a tableau only if the resulting tableau is regular. A tableau is called regular if all nodes on a tableau branch are labeled with different literals. Tautology-freeness means that no tableau clause contains two complementary literals. The subsumption restriction can be formulated as follows. Let \(\succ\) be a total and acyclic ordering on the clauses to be refuted. Then, subsumption forbids the derivation of tableaux that contain a tableau clause which is an instance of an input clause or a unit lemma \(C_1\) and which is subsumed by another input clause or a unit lemma \(C_2 \neq C_1\) with \(C_1 \succ C_2\). For non-unit lemmas we employ a restricted form of subsumption. We forbid the derivation of tableaux that contain a tableau clause which is an instance of an input clause or a unit lemma \(C_1\) and which is subsumed by another input clause or a unit lemma \(C_2 \neq C_1\) with \(m \geq 0\) or a smaller input clause \(h' \lor \sim t'_1 \lor \cdots \lor \sim t'_m\) where \(h\) is connected to its predecessor literal and \(h \leftarrow t_1, \ldots, t_n\) is an instance of a non-unit lemma \(C\), if there is another smaller (unit or non-unit) lemma \(C' = h' \leftarrow t'_1, \ldots, t'_m\) with \(m \geq 0\), a substitution \(\sigma\), and \(\pi: \{1, \ldots, m\} \rightarrow \{1, \ldots, n\}\) with \(h'\sigma = h\) and \(t'_i \sigma = t_{\pi(i)}\), \(1 \leq i \leq m\).

These refinements may be used in isolation or in an arbitrary combination. We denote the regularity, tautology-freeness, and subsumption technique by \(\text{reg, taut, and subs}\), respectively.

Now, we discuss the effects resulting from the use of these calculus refinements together with our lemma techniques. We will see that we may obtain an uncontrolled increase of the number of solutions for subgoals.

**Lemma generation.** In order to generate lemmas we want to consider two different lemma generation techniques. The first variant employs no calculus refinements when enumerating lemmas. The second variant uses the same combination of calculus refinements which are used for the top-down proof run. In both cases, we use the lemma generation algorithm as described in Section 4.1 when refuting Horn clauses and use in addition non-unit lemmas created as described in Section 4.2 when dealing with non-Horn clauses. Furthermore, \(\text{extr}\) is used to enumerate the lemmas.

**Proof length/depth reduction.** At first we consider the regularity condition. Obviously, a regular tableau cannot become irregular when a new tableau is generated by replacing a subproof with head literal \(s\) by the extension of \(s\) with a suitable lemma and performing some reduction steps into the newly introduced subgoals. When using this new subproof the regularity condition cannot be violated since it has not been violated when using the old subproof and the new tableau may be even less instantiated. Thus, we can obtain analogous results for the proof length and depth reduction of proofs as described in Sections 4.1 and 4.2 regardless of the variant for the generation of lemmas. (Note that this holds although Lemma 4.1 does not hold in general when regularity is utilized.) The use of the lemmas which are needed in order to obtain the resource reductions is not forbidden by the regularity technique.

Considering the subsumption refinement for each subgoal which occurs in depth \(n - (N_I + 1)(n_D - 1)\), there must exist a clause from \(\mathcal{C} \cup \mathcal{L}_{N_I}\) which can be applied to
shorten the subproof of the subgoal and whose application is not forbidden because of the subsumption condition. Furthermore, analogously to before, the application of lemmas cannot lead to subsumed tableau clauses in other parts of the tableau.

The tautology-freeness condition imposes more problems. Whereas in the Horn case the application of the needed unit lemmas is still possible, the application of non-unit lemmas appears to be more problematic. One has to consider whether the application of a lemma $C$ needed to solve a subgoal which occurs in depth $n - (N_1 + 1)(n_D - 1)$ may be forbidden because $C$ becomes tautological when performing the extension step, when performing reduction steps into the newly introduced subgoals, or when performing inferences after the use of $C$. This situation is impossible, however, as the following theorem shows.

\textbf{Theorem 4.3.} Let $\mathcal{C}$ be an unsatisfiable set of clauses. Let the minimal proof depth for refuting $\mathcal{C}$ be greater than 1. Let $\mathcal{L}_{N_1}$ be a set containing all unit and non-unit lemma candidates for $\mathcal{C}$ generated in $N_1 \geq 0$ iterations according to Procedures 4.1 and 4.2 using $\text{CTC}^{\text{taut, extr}} \cup R$, $R \subseteq \{\text{reg, subs}\}$, and a resource $n_D \geq 2$.

\begin{enumerate}[(1)]
\item Let $n$ be the minimal proof depth of a proof for $\mathcal{C}$ using $\text{CTC}^{\text{taut, extr}} \cup R$. Let $n'$ be the minimal proof depth of a proof for $\mathcal{C} \cup \mathcal{L}_{N_1}$ using $\text{CTC}^{\text{taut, extr}} \cup R$. Then it holds that $n' = \max((n - (N_1 + 1) \cdot (n_D - 1), 1))$.
\item Let $n$ be the minimal proof length of a proof for $\mathcal{C}$ using $\text{CTC}^{\text{taut, extr}} \cup R$. Let $n'$ be the minimal proof length of a proof for $\mathcal{C} \cup \mathcal{L}_{N_1}$ using $\text{CTC}^{\text{taut, extr}} \cup R$. Then it holds that $n' < n$.
\end{enumerate}

\textbf{Proof.} (1) Let $P$ be a proof with minimal depth which is enumerated with a depth-first subgoal selection function. Furthermore, w.l.o.g. let each closed branch be minimal in $P$. This means that for each subgoal $s$ which occurs when enumerating $P$ and which is solved with substitution $\sigma$ and a subproof which contains extension steps it holds: it is impossible to close $s$ by reduction and a substitution which is more general than or equal to $\sigma$ (modulo renaming variables).

We show that we can replace the chain of proof inferences for each node $N$ (with literal $s$) in depth $n - (N_1 + 1)(n_D - 1)$ in $P$ by a more general chain of proof inferences involving an extension with a lemma and possibly some reduction steps (under consideration of the tautology restriction). Then, we obtain a proof depth reduction from $n$ to a value smaller than or equal to $n - (N_1 + 1)(n_D - 1)$. Since we employ the extension restrictions on non-unit lemmas no proof depth reduction by an amount greater than $(N_1 + 1)(n_D - 1)$ is possible.

We have to consider the case where a literal $s$ is head literal of a subproof in $P$ where the nodes $N_1, \ldots, N_m$ (labeled with $s_1, \ldots, s_m$) of the respective subtableau are closed by reduction with nodes from outside of the subproof with head $s$. Moreover, $i, j \in \{1, \ldots, m\}$ must exist with $s_i \approx s_j$ or $i \in \{1, \ldots, m\}$ with $s_i = s$. Each other literal can obviously be closed with a clause $C \in \mathcal{C} \cup \mathcal{L}_{N_1}$ analogously to before (without violating the tautology condition). This situation is the only one where possibly a tautological instance of a non-unit lemma may be needed.

Let us consider the case that $s_i \approx s_j$. Then, there exist two ancestor nodes $M_1$ and $M_2$ of $N$ in $P$ which are labeled with literals $s_1^T$ and $s_2^T$ in each tableau $T$ (being an ancestor of $P$ in the search tree) that contains $M_1$ and $M_2$ such that $s_1^T$ and $s_2^T$ are unifiable. However, this is a contradiction to the assumption that each branch in $P$ is minimal. If $s_j = s$, then the sequence of proof inferences for $N$ (labeled with $s$ in $P$) can
be replaced by a reduction step involving $N$ and the reduction partner (node) of $s_i$. This
again is a contradiction to the assumption that each branch in $P$ is minimal.

(2) Analogously to the proof of Theorem 4.1. □

Structure of the search space. We consider the case where we employ a combination of
regularity, tautology-freeness, or subsumption in the top-down proof run. Then, the use
of lemmas can increase the number of different solution substitutions which exist for
subgoals in the new minimal proof segment compared with the old proof segment. This
result holds independently of the lemma generation variant that we use.

If we do not employ calculus refinements during the lemma generation, it is quite
obvious that new solutions of a subgoal can be obtained which are not possible when not
using lemmas. The extension of a literal $s$ with a lemma which is generated by a subproof
violates the structural conditions used in the final proof run, e.g. regularity, equals
the use of a subproof for $s$ which also violates the structural conditions. This subproof
cannot be found when not using lemmas.

In addition, a local consideration of regularity, tautology-freeness, and subsumption
during the lemma generation does not prevent the possibility of an increase of the number
of solution substitutions existing for some subgoals. Thus, we may significantly increase
the minimal proof segment.

We validate this claim by demonstrating for the regularity and subsumption refinement
that the number of different solution substitutions existing for a subgoal w.r.t. a set of
clauses can be increased when employing lemmas. This is true although the lemmas
are generated without violating the structural restrictions on the allowed tableaux. An
example for tautology-freeness can be constructed analogously.

Example 4.4. (1) We assume that the regularity condition is used in the final top-
down proof run. Let

$$C = \{ \neg p_1(X), p_1(X) \lor \neg p_2(X), p_2(a) \lor \neg p_1(a), p_1(X) \lor \neg p_2(b), p_2(b) \}. $$

When using the depth bound and a resource $n_D = 3$ for the lemma generation we obtain a lemma set which includes the lemma $p_2(a)$ (which can still be obtained when additionally using tautology and subsumption tests). However, now the start clause $\neg p_1(X)$ can be proved with substitution $\{ X \leftarrow a \}$ when using the lemma $p_2(a)$ although this solution is impossible when not using lemmas. (Again the proof can also be found when using tautology and subsumption tests.) All conventional proofs which lead to the solution substitution $\{ X \leftarrow a \}$ are irregular.

(2) The extension with a lemma can also implicitly result in a use of subsumed clauses.

Consider the following set of clauses

$$C = \{ \neg p(X, f(b)), p(X, Y) \lor \neg q(X, Z) \lor p(Z, Y), p(X, f(Y)), q(a, b), \neg p(b, Y) \}. $$

Let $\succ$ be given as the transitive closure of $\succ^0$ where $p(X, Y) \lor \neg q(X, Z) \lor p(Z, Y) \succ^0 q(a, b), \neg p(b, Y), \neg p(b, Y) \succ^0 p(X, f(Y))$, and $p(X, f(Y)) \succ^0 \neg p(X, f(b))$. With these clauses the lemma $p(a, X)$ can be produced ($n_D = 2$). Thus, the subgoal $\neg p(X, f(b))$ can be closed by extension with the lemma. Explicitly performing the inferences to prove the lemma, however, is forbidden since the clause $p(a, f(b)) \lor \neg q(a, b) \lor p(b, f(b))$ is subsumed by $p(X, f(Y))$. Thus, if we use $p(a, X) \succ^0 p(X, f(Y))$ we can obtain the solution $\{ X \leftarrow a \}$ for $\neg p(X, f(b))$ with a
lemma. However, this solution cannot be obtained with any other proof (without lemmas) when using the subsumption condition.

The increase of the search space by such “hidden” violations of the structural conditions of the calculus refinements may (partially) be compensated when attaching the lemmas with inequality constraints (see Letz et al., 1992). These describe in which cases the lemma proof does not fulfill the structural restrictions. The problems regarding the tautology deletion and subsumption technique can be overcome with such techniques. However, not each hidden violation of the regularity condition may be detected.

4.4. Extensions of the Connection Tableau Calculus

While the use of previously introduced calculus refinements is harmless w.r.t. the potential for proof length and depth reductions it is not apparent whether the use of extensions of CTC, namely the use of stronger inference rules, can cause some problems. We have to discuss whether our lemmas can still work as “complete” macro operators in the extended search space, i.e. whether they can still provide resource reductions (w.r.t. the number of inferences and the proof depth). In the following, we will shortly recall two important extensions of CTC, namely factorization and folding-up, and then we discuss whether resource reductions can still be obtained.

We use a notation similar to the one previously introduced for calculus refinements. We denote a further extension of an extended version Calc of CTC or of CTC itself using the extension technique et by Calc et or CTC et. Again we employ a set notation if several extensions are used.

**Factorization**

*Technique.* The factorization rule for model elimination or the connection tableau calculus (see Loveland, 1978; Letz et al., 1994) allows the re-use of a solution of a certain subgoal $s_2$ at node $N_2$ for solving a subgoal $s_1$ at node $N_1$. Factorization can be performed if $s_1$ and $s_2$ can be unified and all extension and reduction steps that can be used to solve $s_2$ are also possible in order to solve $s_1$. This idea is realized in the factorization inference rule which can be defined analogously to Letz et al. (1994) as follows:

**Procedure 4.3. (Factorization)** Let $T$ be a tableau and $\succ$ be a partial ordering on its tableau nodes. Let $N_1$ and $N_2$ be two nodes of $T$. $N_1$ is labeled with $s_1$, $N_2$ labeled with $s_2$. Let $\sigma$ be the most general unifier of $s_1$ and $s_2$. Furthermore, let $N$ be an ancestor node of $N_1$ and $N_2$, and $N_2$ is an immediate successor node of $N$. Let $N_2 \not\succ N_3$ where $N_3$ is an immediate successor of $N$ on the branch from the root to $N_1$.

Then, $N_1$ is *factorized* with $N_2$ by marking $N_1$ as closed, modifying $\succ$ by $\succ \leadsto \succ \cup \{(N_3, N_2)\}$ and performing the transitive closure. Additionally, $\sigma$ has to be applied to all literal labels of the tableau.

The ordering $\succ$ shows by $N \succ N'$ that the solution of $N$ depends on the solution of $N'$. $\succ$ is used to avoid cyclic (and thus unsound) applications of factorization. We start the tableaux enumeration process with an empty ordering. In the following, the factorization extension of a calculus Calc is denoted by Calc fac. Factorizations are optimistic if we factorize a node with a node whose subgoal is yet unsolved. Pessimistic factorization steps
close a node with a previously solved node. We should note that \( \text{CTC}^{\text{fac}} \) is independent of the subgoal selection function. Moreover, we can recognize that Lemma 4.1 still holds even if factorization steps are performed. This is because factorization steps can only be performed to nodes of a tableau \( T \) which are also nodes of the subgoal tableau of \( T \).

**Lemma generation.** Now, since shortest proofs or proofs with minimal depth may contain factorization steps, we have to answer the question whether our clausal lemmas can again provide a reduction of the inference and depth resource needed to enumerate a proof. As we will see our lemma mechanism is strong enough to reduce the proof depth and length. However, we have to use non-unit lemmas even when refuting Horn clauses and we have to apply factorization steps when enumerating the lemmas. Thus, in the following we will consider a lemma generation based on \( \text{(CTC}^{\text{fac}}^{\text{extr}}) \) and Procedures 4.1 and 4.2 in the Horn and non-Horn case.

**Proof length/depth reduction.** Interestingly, as the following theorem shows, the lemma technique as given above has the potential to reduce the proof length and depth of arbitrary proofs which contain factorization steps. Thus, the techniques are in some sense “compatible”. Proofs may be reduced at first by using factorization (which may also be considered some dynamic lemma technique) and then additionally by statically derived lemmas.

**Theorem 4.4.** Let \( C \) be an unsatisfiable set of clauses. Let the minimal proof depth for refuting \( C \) be greater than 1. Let \( \mathcal{L}_{N_I} \) be a set of (clausal) lemmas created in \( N_I \geq 0 \) iterations of Procedures 4.1 and 4.2 according to \( \text{(CTC}^{\text{fac}}^{\text{extr}})^{\cup R}, R \subseteq \{\text{reg, subs}\} \), and depth resource \( n_D \geq 2 \).

(1) Let \( P \) be a proof for \( C \) which has minimal depth using \( \text{(CTC}^{\text{fac}})^{\cup R} \). Let \( P' \) be a proof for \( C \cup \mathcal{L}_{N_I} \) which has minimal depth using \( \text{(CTC}^{\text{fac}}^{\text{extr}})^{\cup R} \). Let \( n \) and \( n' \) be the depth of \( P \) and \( P' \), respectively. Then, it holds that \( n' = \max\{1, n - (N_I + 1)(n_D - 1)\} \).

(2) Let \( P \) be a proof with minimal length for \( C \) using \( \text{(CTC}^{\text{fac}})^{\cup R} \). Let \( P' \) be a proof with minimal length for \( C \cup \mathcal{L}_{N_I} \) using \( \text{(CTC}^{\text{fac}}^{\text{extr}})^{\cup R} \). Let \( n \) and \( n' \) be the length of \( P \) and \( P' \), respectively. Then, it holds that \( n' < n \).

**Proof.** (1) Again we only have to show the following. Let \( T \) be a tableau which is enumerated with a depth-first subgoal selection function. Let \( T_0 \vdash \cdots \vdash T_k = T \) be the derivation of \( T \) performed with the inferences \( I_1, \ldots, I_k \). Let \( N \) (labeled with \( s \)) be the head node of a closed subtableau in \( T \) which has a depth smaller than or equal to \( N_I \cdot (n_D - 1) + n_D \). Let \( S = (I_{i+1}, \ldots, I_j) \), \( 0 \leq i < j \leq k \) be the sequence of proof inferences for \( N \). We have to show that with the help of a (non-)unit lemma \( L \in \mathcal{L}_{N_I} \), a more general sequence of proof inferences can be achieved which solves \( N \) with a subproof of depth 1. Then, analogously to before, we obtain a proof depth reduction from \( n \) to a value smaller than or equal to \( n - (N_I + 1)(n_D - 1) \). Since we employ the extension restrictions on non-unit lemmas no proof depth reduction by an amount greater than \( (N_I + 1)(n_D - 1) \) is possible.

Let \( s \) be the literal label of \( N \) in \( T_i \). Let \( t \) be the literal label of \( N \) in \( T_j \). The following cases have to be considered. At first we consider the case that in the sequence \( S \), nodes are factorized only with nodes which are successors of \( N \). Then, a unit or non-unit clause
exists in $C \cup L_N$, which can be used to close $s$. This is because factorization is performed when generating the lemmas. The obtained proof sequence is more general than $S$.

The second case is that some nodes $M_1, \ldots, M_m$ (labeled with the literals $s_1, \ldots, s_m$ in $T_j$) of the subproof with head node $N$ have been solved by factorization with nodes $N_1, \ldots, N_m$ which are outside of the subproof with head $N$. We should note that these nodes must be part of the subgoal tableau of $T_i$ and that the depth of these nodes must be smaller than or equal to the depth of $N$. Let $R_1, \ldots, R_z$ (labeled with literals $r_1, \ldots, r_z$ in $T_j$) be the nodes occurring in the subproof with head $N$ which are closed by reduction steps from outside of the subproof (with nodes $S_1, \ldots, S_z$). Then, according to our lemma generation technique, there is a lemma $C = s' \leftarrow s'_1, \ldots, s'_k, r'_1, \ldots, r'_l$ in $L_N \cup C$ which satisfies the following: there are two functions $\pi_1 : \{1, \ldots, k\} \rightarrow \{1, \ldots, m\}$ and $\pi_2 : \{1, \ldots, l\} \rightarrow \{1, \ldots, z\}$ as well as a substitution $\sigma$ such that $s'\sigma = \neg t$, and $s'_i\sigma = \neg s_{\pi_1(i)}$, $r'_i\sigma = \neg r_{\pi_2(i)}$. Furthermore, an application of $C$ is allowed in spite of the fact that calculus refinements may be used. As shown in the figure, $C$ allows us to solve the subgoal $s$ with a subproof of depth 1. Furthermore, the new solution for $s$ is more general than the old solution. The extension with $C$ introduces new nodes $\tilde{R}_1, \ldots, \tilde{R}_l, \tilde{M}_1, \ldots, \tilde{M}_k$ which can be closed by reduction with the nodes $S_{\pi_2(1)}, \ldots, S_{\pi_2(l)}$ or by factorization with the nodes $N_{\pi_1(1)}, \ldots, N_{\pi_1(k)}$.

Thus, $s$ can be closed with proof inferences more general than the inferences in $S$ using a lemma and some literals at nodes with a depth which is smaller than or equal to the depth of $N$.

(2) A proof length reduction can be obtained analogously to the proof of Theorem 4.1. □

In summary, the use of non-unit lemmas can provide a proof length and proof depth reduction regardless whether or not factorization is used.

Structure of the search space. As we have seen we can obtain results regarding both proof length and proof depth reductions which are the same as in the conventional calculus
CTC. We should note that even in the Horn case non-unit lemmas have to be used. However, then it is sufficient to allow only factorization steps into the subgoals introduced by non-unit lemmas in order to guarantee the needed resource reductions. Thus, we obtain results regarding the structure of the search space in the Horn case when employing such extension restrictions which are analogous to the results of Section 4.3. In the non-Horn case the refinement that only reduction and factorization steps are allowed into subgoals introduced by non-unit lemmas makes it possible to lift the results from Section 4.3.

FOLDING-UP

Technique. Folding-up is a generalization of the \(c\)-reduction rule for model elimination (see Shostak, 1976). The inference rule can be seen as a pessimistic variant of the factorization rule and as a type of restricted lemma mechanism, a context-lemma mechanism (see Letz et al., 1994). In order to introduce folding-up we have to extend our notion of a (labeled) tableau by introducing a further label to each node. The new label is a set of literals which are also called context lemmas in the following. Note that also the previously unlabeled root node has as label a set of context lemmas. We define the folding-up rule as follows.

Procedure 4.4. (Folding-up) Let \(T\) be a tableau extended with context-lemma labels. Let \(N\) be a non-leaf node marked with literal \(l\) which is the head of a closed subtableau. Then, let \(M\) be the deepest ancestor node of \(l\) which is used for reduction into the subproof with head \(l\) or let \(M\) be the root node if such ancestor nodes do not exist.

Then, folding-up is performed by adding the literal \(\neg l\) to the set of context-lemmas of the tableau node \(M\).

In order to use the context lemmas during the proof run we have to extend the reduction rule. We allow the closing of a subgoal \(s\) at node \(N\) if it can be unified with the complement of a literal which is an element of the set of context lemmas of an ancestor node of \(N\). Naturally, the substitution has to be applied to all literal and context lemma labels of the tableau.

\(CalcU\) is an extension of a calculus \(Calc\) that additionally uses the folding-up rule, uses the refined reduction rule, and labels each node introduced by start or extension with an empty context lemma set. Note that in contrast to factorization the folding-up step is not considered an inference step and is not counted when determining the length of a proof. Folding-up is merely used to assist certain reduction steps which serve as some kind of pessimistic factorization steps. Of course these reduction steps count as an inference. Therefore, solving a subgoal by factorization or by using a folded literal takes the same costs in numbers of inferences.

Lemma generation. In the following we assume that we generate unit as well as non-unit lemmas (even when dealing with Horn clauses). Furthermore, we use folding-up and extension restrictions on lemmas in order to enumerate the lemmas.

Proof length/depth reduction. If we apply folding-up some problems occur. These are due to the fact that we can prove a subgoal \(s\) by virtue of another subgoal \(s'\) (\(s'\) is folded up after solution) which does not have to be a brother of an ancestor of \(s\). Thus, it may be, as the following theorem shows, that certain subgoals cannot be proved by reduction.
with a folded literal since the literal would “disappear” in a lemma proof if we applied lemmas and thus cannot be folded up.

**Theorem 4.5.**

(1) For each pair \((N_I, n_D)\) with \(N_I \geq 0, n_D \geq 2\), there exists an unsatisfiable set \(C\) which has a minimal proof depth of \(n \geq 2\) using \(CTC^{fu}\) and the minimal depth of a proof for \(C \cup L_{N_I}\) is \(n' = n\) (using \(CTC^{fu|extr}\)) where \(L_{N_I}\) is the set of (clausal) lemmas created according to Procedures 4.1 and 4.2 using \(CTC^{fu|extr}\), \(N_I\), and \(n_D\).

(2) Let \(C\) be an unsatisfiable set of clauses. Let the minimal proof depth for refuting \(C\) be greater than 1. Let \(L_{N_I}\) be a set of (clausal) lemmas created with Procedures 4.1 and 4.2 according to \(CTC^{fu|extr}\), \(R \subseteq \{reg, subs\}\), the iteration number \(N_I \geq 0\), and depth resource \(n_D \geq 2\). Let \(P\) be a proof for \(C\) obtained with a minimal number of inferences using \(CTC^{fu|R}\). Let \(P'\) be a proof with minimal length for \(C \cup L_{N_I}\) using \(CTC^{fu|extr}\). Let \(n\) and \(n'\) be the numbers of inferences needed to enumerate \(P\) and \(P'\), respectively. Then, it holds that \(n' < n\).

**Proof.** (1) We consider the clause set

\[
C = \{\neg p_1^1 \lor \neg p_2^1 \lor \cdots \lor \neg p_{N_I}^{N_I(n_D-1)+n_D},
\]

\[
p_1^1 \lor \neg p_2^1, 1 \leq i \leq N_I \cdot (n_D - 1) + n_D,
\]

\[
p_2^2 \lor \neg p_2^i, 2 \leq i \leq N_I \cdot (n_D - 1) + n_D,
\]

\[
p_2^3 \}
\]

and use as start clause the clause \(\neg p_1^1 \lor \neg p_2^2 \lor \cdots \lor \neg p_{N_I}^{N_I(n_D-1)+n_D}\). There is only one proof of the minimal depth of 2 (when not using lemmas). In this proof reduction, steps with the literals \(p_2^1, \ldots, p_2^{N+1(n_D-1)}\) are executed which are folded-up to the root node.

The proof looks as follows where we use \(N = N_I \cdot (n_D - 1) + n_D\).

![Diagram](attachment:diagram.jpg)

Other proofs can only vary in the use of the folded literals. Thus, in order to reduce the proof depth each node which is labeled with \(\neg p_1^i, 1 \leq i \leq N_I \cdot (n_D - 1) + n_D\), has to be solved with a lemma. However, if we replace the subproofs of the nodes with literal labels \(\neg p_1^1, \ldots, \neg p_1^{N_I(n_D+1)}\) by extensions with unit lemmas, which is possible due to the lemma generation method, it is impossible to solve the last subgoal \(\neg p_1^{N_I(n_D-1)+n_D}\) with an extension step and possibly reduction steps. The only lemmas which can be derived and which are applicable to \(\neg p_1^{N_I(n_D-1)+n_D}\) are \(\{p_1^{N_I(n_D-1)+n_D} \lor \neg p_2^i: 1 \leq i \leq N_I\}\).
\( i \leq N_I \cdot (n_D - 1) + n_D \). (Note that the “expanded” (linear) subproof with head literal \( \neg p^N_i \) has a depth of \( N_I \cdot (n_D - 1) + n_D + 1 \).) But none of these lemmas can be used to solve the subgoal by extension and reduction steps. This is because the literals \( p^N_i, 1 \leq i \leq (N_I + 1) \cdot (n_D - 1) \), are not folded-up to the root node.

(2) We consider the set of proofs which can be enumerated with minimal inference resources. Then, it can easily be recognized that there exists a proof in this set which contains a subgoal which is closed with a subproof of a proof depth of 2 and which is obtained without using folded literals from outside of the subproof. If we consider this subproof one can see that no literal which can be folded-up from the subproof needs to be re-used later to obtain a proof of minimal length. This is because these literals are also present as facts in the input clauses. Thus, the subproof can be exchanged by a shorter and more general chain of proof inferences using lemmas. □

**Structure of the search space.** It turned out that the use of lemmas can cause some problems together with folding-up. Even when employing the depth bound and restricting the applicability of extension rules with non-unit lemmas it may be that the minimal segment of the search space which includes a proof with lemmas may be considerably larger than the minimal segment which contains a proof without lemmas. Some solutions of subgoals may be obtained in the proof segment which could not be reached when not using lemmas (in the old proof segment). Also when using the inference or weighted-depth bound the use of lemmas remains problematic. It is quite probable that only small inference reductions may be obtained by using lemmas resulting in rather small resource reductions. The use of the weighted-depth bound, however, may be more appropriate than the pure use of the depth bound since we may profit from each lemma application in form of a resource reduction.

Finally, we should not neglect that an appropriate filtering of lemmas can weaken these problems. If resource reductions can be obtained with a small number of lemmas it is to be expected that we can significantly profit from the lemma use although the resource reductions are smaller than when not employing folding-up. In Section 7 we will see that indeed with the weighted-depth bound satisfactory results could be obtained.

## 5. Relevance of a Lemma

The most important method in order to control the use of the generated lemma candidates is an appropriate discarding of superfluous lemma candidates. In the following we want to deal with **selection** techniques which choose a set of lemmas to be used in the final proof run from a set of lemma candidates.

Essential for the selection of lemmas is a notion of relevance of a lemma set for the refutation of a given set of clauses (employing a fixed start clause). We want to make the notion of relevance of a lemma set for a proof goal more precise.

**Definition 5.1.** Let \( C \) be a set of clauses, \( S \in C \) be a start clause for refuting \( C \), and \( L_{N_I} \) be a set of lemma candidates. We say a set \( L \subseteq L_{N_I} \) is **relevant** for \( C \) and \( S \) w.r.t. a search tree \( T \) if there is a closed tableau \( T \) in \( T \) that can be reached after a start expansion with \( S \) and that contains only instances of clauses from \( C \cup L \) as tableau clauses. Furthermore, at least one \( L \in L \) appears in the tableau \( T \).

Not each set \( L \) which is relevant for \( C \) and \( S \), w.r.t. a search tree \( T \) defined by \( C \cup L_{N_I} \),
and a specific connection tableau calculus, is well-suited 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 which is at least needed in order to obtain a closed tableau \( T \) when using \( \mathcal{B} \). There should be no subset of \( \mathcal{L}_{N_1} \), different from \( \mathcal{L} \), that can help to refute \( \mathcal{C} \) with a smaller resource. We employ the following notion of strong relevance.

**Definition 5.2.** Let \( \mathcal{C} \) be a set of clauses, \( S \in \mathcal{C} \) be a start clause for refuting \( \mathcal{C} \), and \( \mathcal{L}_{N_1} \) be a set of lemma candidates. We call a set \( \mathcal{L} \subseteq \mathcal{L}_{N_1} \) strongly relevant for \( \mathcal{C} \) and \( S \) w.r.t. a search tree \( T \) and a completeness bound \( \mathcal{B} \) if only clauses from \( \mathcal{C} \) and all clauses from \( \mathcal{L} \) appear in a closed tableau in \( T \) which can be found with start clause \( S \), bound \( \mathcal{B} \), and a minimal resource (no proof for \( \mathcal{C} \cup \mathcal{L}_{N_1} \) and start clause \( S \) can be found with a smaller resource).

A sensible selection mechanism should hence try to choose a subset \( \mathcal{L} \subseteq \mathcal{L}_{N_1} \) that is strongly relevant for \( \mathcal{C} \) and \( S \). This test, however, necessitates the consideration of all subsets of \( \mathcal{L}_{N_1} \). Thus, we employ a local notion of relevance of single lemmas.

**Definition 5.3.** Let \( \mathcal{C} \) be a set of clauses, \( S \in \mathcal{C} \) be a start clause for refuting \( \mathcal{C} \), and \( \mathcal{L}_{N_1} \) be a set of lemma candidates. We call a lemma candidate \( L \in \mathcal{L}_{N_1} \) relevant for \( \mathcal{C} \) and \( S \) w.r.t. a search tree \( T \) if there is a closed tableau in \( T \) that can be reached with start clause \( S \) and which contains as tableau clauses instances of \( L \) and of some clauses from \( \mathcal{C} \cup \mathcal{L}_{N_1} \). We say a lemma is strongly relevant for \( \mathcal{C} \) and \( S \) w.r.t. a search tree \( T \) if there is a lemma set \( \mathcal{L} \) that contains \( L \) and which is strongly relevant for \( \mathcal{C} \) and \( S \) w.r.t. \( T \).

The set of the lemmas which are strongly relevant for \( \mathcal{C} \) and \( S \) is a lemma set \( \mathcal{L}_{loc} \subseteq \mathcal{L}_{N_1} \) which is the union of all strongly relevant subsets of \( \mathcal{L}_{N_1} \). With the help of this lemma set we can refute \( \mathcal{C} \) (employing start clause \( S \)) with a minimal resource. Furthermore, normally \( \mathcal{L}_{loc} \) is much smaller than \( \mathcal{L}_{N_1} \). Thus, a lemma selection mechanism can also try to find a rather well-suited lemma set based on local tests.

### 6. Lemma Selection

In this section we want to deal with general principles for estimating whether a clause \( L \) from a set \( \mathcal{L}_{N_1} \) of lemma candidates is strongly relevant for a clause set \( \mathcal{C} \) and a start clause \( S \in \mathcal{C} \). First of all we assume that a refutation of \( \mathcal{C} \cup \mathcal{L}_{N_1} \) can be found with start clause \( S \) and that the resource value \( n \in \mathbb{N} \) which is at least needed in order to find such a proof with start clause \( S \), is given. Furthermore, in order to provide a comprehensive description of our methods we solely consider a fixed calculus in the following which equals \( \text{CTC} \) or is a refinement of \( \text{CTC} \). Each of our calculus refinements may be employed. We assume that the refinement regarding the restriction of extension steps to literals introduced by non-unit lemmas is in use (see Section 4.3). Later we will make some remarks regarding an application of factorization or folding-up. We only consider bounds which map tableaux which occur at different positions in the search tree and which are equal modulo renaming variables to the same value. This is guaranteed, e.g. for the depth and inference bound.

Our principles for \textit{a priori} estimating the relevance of \( L \) are based on the fast simulation of complete tableaux enumeration procedures (\textit{lazy tableaux enumeration}). The methods...
developed may also be considered specific partial evaluation techniques. A complete test of whether \( L \) is strongly relevant for \( C \) and \( S \) can be obtained by enumerating the set \( \{T_1, \ldots, T_z\} \) of all tableaux in the finite segment \( T_{C \cup L \cup N \cup I, S, B, n} \) which is the segment defined by the clause set \( C \cup L \cup N \cup I \), start clause \( S \in C \), the completeness bound \( B \), and the resource value \( n \). Then one has to check whether \( L \) appears in a closed tableau \( T_i \), \( 1 \leq i \leq z \). Since this “test” would be too expensive (clearly it equals the use of all lemmas to refute \( C \cup L \cup N \cup I \)) we want to simulate the test by enumerating a sequence \( \tau_A = (T_{i_1}, \ldots, T_{i_k}) \), \( k \leq z \), \( i_j \in \{1, \ldots, z\} \) for \( j \in \{1, \ldots, k\} \), and \( i_j \neq i_m \) for \( i \neq m \). Then, we try to estimate whether a closed tableau \( T_j \), \( 1 \leq j \leq z \), exists that contains \( L \) (assuming such a tableau has not been enumerated). Thus, we have to deal with the question of how a “representative” subset of \( \{T_i : 1 \leq i \leq z\} \) can be enumerated and how the relevance of a lemma can be estimated based on this subset.

6.1. COMPLETE TABLEAUX ENUMERATION PROCEDURES

First we want to start with the introduction of two interesting tableaux enumeration procedures which provide complete relevance tests. The connection tableau calculus is independent of the order in which one tries to solve the subgoals of a tableau if no folding-up is employed. This means that a proof obtained with one subgoal selection function can also be obtained with another subgoal selection function (modulo renaming of variables). Therefore, we can design different tableaux enumeration methods which spare the consideration of all tableaux in \( T_{C \cup L \cup N \cup I, S, B, n} \). We want to consider the following two methods which differ from each other w.r.t. the use of a lemma in order to close subgoals. We want to consider the lemma preferring and lemma delaying method.

The lemma preferring method forces the use of a lemma candidate \( L \) in order to obtain a proof. In order to test the relevance of \( L \) we generate at first the set \( \tau_0 \) of all tableaux in \( T_{C \cup L \cup N \cup I, S, B, n} \) which satisfy the following conditions. Firstly, no tableau from \( \tau_0 \) has a branch closed by reduction or extension with a fact. Secondly, when creating a tableau from \( \tau_0 \), inference steps are applied in a depth-first manner, i.e. an extension step must be applied to a subgoal with maximal depth.

**Example 6.1.** Let

\[
C \cup L \cup N \cup I = \{\neg p(a, b), p(X, Y) \lor \neg p(X, Z) \lor \neg p(Z, Y) \lor p(a, d), p(a, f(c)), p(d, b)\}.
\]

Let \( S = \neg p(a, b) \). Let \( B \) be the depth bound and let \( n = 2 \). We depict the connection tableaux from the search tree \( T_{C \cup L \cup N \cup I, S, B, n} \) which are part of \( \tau_0 \).

Roughly speaking, the set \( \tau_0 \) contains all the tableaux which represent the possible branches which may be closed with a lemma while the other branches are unexpanded.

---

\(^{1}\)In the following we will also refer to \( \tau_A \) as a set of tableaux.
In order to test the relevance of \( L \in \mathcal{L}_{N_I} \) with the lemma preferring method we generate at first the set \( \tau^L_0 \) for \( L \in \mathcal{L}_{N_I} \) which contains all tableaux which are part of \( \mathcal{T}^{C \cup \mathcal{L}_{N_I},S,B,n} \) and which can be derived from tableaux from \( \tau_0 \) by extending a subgoal of maximal depth with \( L \) (and closing all newly introduced subgoals by reductions). Thus, because of the independence of the subgoal selection strategy, for each tableau \( T' \) in \( \mathcal{T}^{C \cup \mathcal{L}_{N_I},S,B,n} \) which contains a branch closed using \( L \) there exists a tableau \( T \in \tau^L_0 \) such that \( T \vdash \cdots \vdash T'' \) and \( T'' \) equals \( T' \) modulo renaming of variables. Furthermore, it does not hold \( T \vdash \cdots \vdash T'' \) for \( T,T' \in \tau^L_0 \). As a consequence, \( \tau^L_0 \) provides a complete and minimal set of tableaux from which all closed tableaux (modulo renaming variables) which contain a subgoal which is solved by extension with \( L \) and some reduction steps can be derived. Hence, in order to judge the relevance of \( L \) one has to enumerate all tableaux which can be reached from one of the tableaux from \( \tau^L_0 \) and to check whether a closed tableau is among these tableaux. In summary, the lemma preferring method creates at first all possible branches which may be closed with \( L \), then these branches are closed with \( L \), and finally it is used to solve the remaining open subgoals.

The lemma delaying method delays the use of \( L \) to close open branches. The method starts with the set \( \tau_0 \) analogously to the lemma preferring method. But then, in order to judge the relevance of a lemma \( L \), for each tableau \( T \in \tau_0 \) and each subgoal \( s \) which has a maximal depth in \( T \), the following tableaux are enumerated. At first, inference steps are forbidden to the subgoal \( s \) in \( T \). Then, successively all tableaux in \( \mathcal{T}^{C \cup \mathcal{L}_{N_I},S,B,n} \) are enumerated which can be reached from \( T \) while applying the mentioned inference restrictions on \( s \). Afterwards, we consider only the enumerated tableaux which are “nearly closed”. These are all tableaux which have only one open subgoal which is the subgoal at the same position as \( s \) in \( T \). We call these tableaux front literal tableaux since they have only one open subgoal at the “tableau front”. The single open subgoal of each front literal tableau is called front literal. The sequence of literals which are associated with an open branch ending with a front literal is called front literal path. Then, in order to judge the strong relevance of \( L \) for \( \mathcal{C} \) and \( S \) we check whether an extension with \( L \) is possible to a front literal tableau such that all introduced subgoals can be solved by reduction with literals from the front literal path, and the resulting tableau is part
of $T^{\Sigma_{\mathcal{L}},S,B,n}$. Again the method provides a sound and complete relevance test. In summary, the method creates at first the branches which may be closed with the lemma $L$, then solves the subgoals which are not on these branches, and finally attempts to see whether $L$ can be used to close the last open branch of one of the enumerated tableaux.

6.2. Lazy Tableaux Enumeration

Now, we want to deal with the question of how to simulate the complete but costly relevance tests. We start with the simulation of the lemma preferring method. At first we consider how to obtain information on the usefulness of a clause $L$ based on a tableau set $\tau^L_A$ which does not contain all tableaux which are needed for a complete test. We assume that $\tau^L_A$ contains at least the tableaux from $\tau^L_0$ and some further tableaux which can be derived from tableaux of $\tau^L_0$. We have to estimate whether one of the tableaux from $\tau^L_A$ lies on a path (in $T^{\Sigma_{\mathcal{L}}_1,S,B,n}$) to a proof. In Section 6.4 we will develop heuristic criteria for this purpose. In the following we assume that a function $\Sigma$ is given where $\Sigma(T) \in \mathbb{R}$ judges whether the tableau $T$ can be closed in $T^{\Sigma_{\mathcal{L}}_1,S,B,n}$. The smaller the value, the higher the probability that the subgoals of $T$ are solvable. In order to obtain reliable data it is sensible to use only those enumerated tableaux which are obtained with as many inferences as possible. Furthermore, for each closed tableau $T'$ reachable from $\tau^L_A$ the solvability of at least one tableau $T \in \tau^L_A$ with $T \vdash \cdots \vdash T'$ should be tested. Therefore, it is sensible to judge the relevance of $L$ based on the set $\tau^L_H = \{T : T \in \tau^L_A, \exists T' : T \vdash T' \land T' \not\in \tau^L_A\}$ and $T' \not\in \tau^L_A$. Thus, we use the principle of a maximal generation of a proof path (of the search tree). One tries to attain estimations by following the proof path as long as possible. Note that a sound negative result for the relevance of $L$ can be given if $\tau^L_H$ equals the empty set.

Now, we discuss how to generate an appropriate set $\tau^L_A$. It is quite probable that the relevance estimation works the better the smaller the path distance is (in $T^{\Sigma_{\mathcal{L}}_1,S,B,n}$) between the tableaux in $\tau^L_A$ and a closed tableau. Thus, we start with $\tau^L_A = \tau^L_0$. Then, we can employ depth-first or breadth-first tableaux enumeration for enumerating some further tableaux. Clearly, depth-first search is not well-suited in our context. It may happen that only after the enumeration of a large set of tableaux some tableaux are generated which have a smaller path distance to a closed tableau than a tableau from $\tau^L_0$. Thus, breadth-first search methods seem to be more appropriate since each proof path is followed systematically.

When employing lemma selection based on lemma delay at first we assume again that a set $\tau^L_A$ is given. This set should contain at least the tableaux from $\tau_0$. Furthermore, this set can contain some additional tableaux derived from tableaux of $\tau_0$. The tableaux are derived in such manner that if a tableau $T' \in \tau^L_A$ is derived from a tableau $T \in \tau_0$, then there is a subgoal in $T$ with maximal depth which has been “untouched”, i.e. no extension and reduction steps had been applied to the subgoal. So to speak this is the subgoal which should finally be closed with the lemma whose relevance should be tested.

We want to extract the information whether or not there is a front literal tableau in $T^{\Sigma_{\mathcal{L}}_1,S,B,n}$ (derivable from a tableau from $\tau_0$ as described above) which can be closed with $L$ based on $\tau^L_A$. Our method is now based on the principle of maximal proof path generation with slight path deviation. As first introduced in Fuchs (1998a) we want to use only the set $\tau^L_H \subseteq \tau^L_A$ of the enumerated front literal tableaux for the estimation. Although these tableaux normally represent deviations from a proof path, i.e. they do not lie on a branch of the search tree which ends with a proof, a comparison of the
controlled use of clausal lemmas in connection tableau calculi

structural differences between the literals of the front literal paths and the literals of $L$ can give hints on the usefulness of $L$.

The following differences between a front literal path $p^L$ (with front literal $f^L$) which can actually be closed with $L$ (by unifying $\sim f^L$ and the lemma head and closing the instantiated tail by reduction with literals from $p^L$) and an enumerated front literal path $p$ (with front literal $f$) can occur. It is possible that all inferences that have to be performed on literals on the branch from $S$ to $f^L$ of the respective front literal tableau $T_{f^L}$ are also performed when creating $f$ (generating the front literal tableau $T_f$). Thus, some of the subgoals which occur during the generation of both $T_{f^L}$ and $T_f$ and do not lie on the front literal path have been solved when generating $T_f$ by a subproof somewhat different to the subproof in $T_{f^L}$. Since all subgoals may be variable-connected “unification failures” arise at some positions in the literals of $p$ which prevent the use of $L$ in order to close $T_f$. Another possibility is that inferences which have to be performed on literals on the branch from $S$ to $f^L$ are not performed when producing a front literal $f$. This results in a higher degree of structural difference between $p$ and $L$. For instance, $f$ and the head literal of $L$ may differ in the predicate symbol at the top-level position. In our experiments we could often generate front literal tableaux whose front literal paths are created with “correct” inferences. Therefore, it is sensible to use a unification distance measure $\Delta$ (see Section 6.5) for estimating the relevance of a lemma candidate.

In order to use unification distance measures we can normally profit from the fact that a front literal tableau shares many inferences with a closed tableau. Thus, again breadth-first search (after creating the set $T_0$) is appropriate in order to create $T_0^L$ since systematically all possible proof paths are explored. The following example illustrates the working scheme of the lemma selection based on the lemma delaying method.

**Example 6.2.** We consider the clause set

$$C \supseteq \{ \neg p(X, Y) \lor \neg q(X, Y), p(X, Y) \lor \neg r(X, Y), r(X, Y) \lor \neg r(Y, X), \neg q(g(X), g(Y)) \lor \neg q(X, Y), q(X, Y) \lor \neg q(X, Z) \lor \neg q(Z, Y), \neg q(a, a), q(a, c), q(c, d)\}.$$ 

Let $L_{N_f}$ contain the strongly relevant element $L = r(g(d), g(a))$. We assume that the following tableau shows a front literal tableau which has to be created when employing a complete relevance test with the lemma delaying method.

```
\begin{center}
\begin{tikzpicture}
  \node (p) {$p(g(a), g(d))$};
  \node (a) [below of=p] {$r(g(a), g(d))$};
  \node (b) [below of=a] {$\neg r(g(d), g(a))$};
  \node (c) [right of=p] {$q(g(a), g(d))$};
  \node (d) [below of=c] {$q(g(a), g(d))$};
  \node (e) [below of=d] {$q(a, d)$};
  \node (f) [below of=e] {$q(a, c)$};
  \node (g) [right of=e] {$q(c, d)$};
  \node (h) [below of=g] {$q(c, d)$};

  \draw[->] (p) -- (a);
  \draw[->] (a) -- (b);
  \draw[->] (c) -- (d);
  \draw[->] (d) -- (e);
  \draw[->] (e) -- (f);
  \draw[->] (f) -- (g);
  \draw[->] (g) -- (h);
\end{tikzpicture}
\end{center}
```
The following connection tableau shows a tableau which may be generated with our fast simulation of the lemma delaying method. The path which can be closed with the lemma $L$ is at the beginning represented by a tableau from $\tau_0$. Then, we assume that we allow two further inferences which can be applied to tableaux from $\tau_0$ in order to generate the set $\tau^L_A$. The connection following tableau from $\tau^L_A$ is also present in $\tau^L_R$ since it is a front literal tableau.

Thus, our useful lemma $L$ cannot be used to close an enumerated front literal tableau but it can “nearly” close an enumerated front literal path.

Figure 2 summarizes the working schemes of the lemma preferring and lemma delaying based selection techniques for a unit lemma $L$. We show two identical copies of a part of the search tree $T^{C\cup L\cup S\cup B\cup n}$. The left and right part of the figure illustrate a relevance estimation based on lemma delay and lemma preference, respectively. A black box represents a proof which can be found by closing a tableau with the lemma candidate $L$. The other proof (black oval) which is equal (modulo renaming variables) to the first proof is inferred from a tableau from $\tau^L_0$. The head node of each segment of the search tree represents a tableau from $\tau_0$. All heavily bordered ovals represent the tableaux which form the set $\tau^L_A$. The grey tableaux show the sets $\tau^L_R$ which are used for the relevance test. The simulation of the lemma delaying method lets the subgoal open which must be closed with $L$ for obtaining a proof. Instead, front literals are enumerated without touching the respective subgoal of the tableau in $\tau^L_0$. Then the usefulness of $L$ is estimated, by computing a unification distance between $L$ and the enumerated front literals. Since a structural similarity between the front literal $f_L$ and $f$ is quite probable, the small unification distance between $f$ and $L$ may help to identify $L$ as useful. In the simulation of the lemma preferring method a tableau $T$ is enumerated which lies on a path to a proof. A positive test for the solvability of the subgoals of $T$ can help to recognize that $L$ is strongly relevant for $C$ and $S$.

Up to now we have assumed that the resource value $n$ is given. Naturally, in practice, only an estimation of $n$ can be used. An incorrect choice of $n$ affects both the soundness and the completeness of our methods. If $n$ is chosen too small the completeness of our methods is influenced and no good estimations may be obtained since too few tableaux are used for the relevance test. A value too high affects the soundness. Lemma candidates may be judged to be useful although they cannot lead to a proof with a minimal resource value because tableaux not lying in the minimal proof segment are used for the judgment. Since the completeness of the test is much more important (a use of some unnecessary clauses is normally no serious problem but the needed lemmas should be chosen) we try to use a value for $n$ which is rather large and still guarantees a fast lemma selection.
A possible alternative is also to select for each iterative deepening level which employs a resource value $n \in \mathbb{N}$ in some lemmas. Then, the appropriate resource $n$ which is needed for the partial evaluation is always given for the lemma selection. In our experiments we did not employ this more costly variant.

Finally, we want to note that the above heuristic techniques are of course also applicable when folding-up or factorization is employed. If factorization is employed, variants of the lemma delaying and lemma preferring method can be defined which still provide sound and complete relevancy tests. The lemma delaying method is not complete if folding-up is employed (no literal of the branch to be closed with a lemma can be folded-up). Nevertheless, in practice, the heuristics can still provide satisfactory estimations for the relevance of lemmas.

6.3. Selecting Lemmas Based on Lazy Tableaux Enumeration

In order to select lemmas from the set $\mathcal{L}_{N_I}$ of lemma candidates for the refutation of $\mathcal{C}$ with start clause $S$ we employ a relevance value $\Phi$ for each $L \in \mathcal{L}_{N_I}$. $\Phi(L)$ can be computed as the minimal distance between $L$ and an enumerated front literal path belonging to a tableau from $\mathcal{T}_P$ or the minimal value $\Sigma(T)$ for a tableau $T \in \mathcal{T}_P$ when applying a lemma selection based on the lemma delaying or lemma preferring method, respectively. Then, a fixed percentage of the smallest clauses from $\mathcal{L}_{N_I}$ w.r.t. $\Phi$ can be chosen. If we apply lemma selection based on the lemma preferring method this is exactly the way we proceed. When employing lemma delay experiments performed in Fuchs (1997c) they show that a slight modification of the technique attains better results. A selection of lemmas is performed by choosing the nearest lemma for each front literal path resulting in a set $\mathcal{L}_{N_I}$. Then, from this set the best lemmas w.r.t. $\Phi$ are chosen up to a given upper bound.

6.4. Lemma Selection Based on the Lemma Preferring Method

Now, we instantiate the abstract principles described above with some concrete methods and heuristics. We start with the lemma preferring method. We want to introduce
methods for a solvability estimation of a tableau and show how we generate the sets \( \tau^L_A \) for a lemma preference based lemma selection.

Our first problem is to estimate whether a tableau \( T \) is solvable within the search tree \( T_{C∪S,B,N} \). We weaken this problem and only try to define a function \( \Sigma \) as follows. If \( T_1 \) lies on a path to a proof and \( T_2 \) does not lie on a path to a proof or lies only on paths to proofs whose derivations require larger resource values, then \( \Sigma(T_1) \) should be smaller than \( \Sigma(T_2) \). This function is sufficient to give some of the tableaux from \( \tau^R_L \) of a needed lemma \( L \) the best ratings.

In order to define an approximation of this function we have to estimate how difficult the solution of a subgoal may be in general (regarding a given bound). Furthermore, we have to consider which effects are caused by the difficulty of specific subgoals for the derivation of a closed tableau (by solving these subgoals) when they occur at certain tableau positions.

In order to estimate how difficult the solution of a subgoal may be (w.r.t. a specific bound) we use a complexity function \( |\cdot| \). We employ a rather simple function which neglects the bound to be employed. The complexity \( |s| \) of a subgoal \( s \) is given by a weighted sum of the symbols occurring in \( s \). Variables are counted as 1 and function symbols as 2. Thus, small and general subgoals are preferred against large and complex subgoals. Naturally this complexity function could be refined in the future.

Now, we want to consider how to compute a solvability value \( \Sigma(T) \) for a whole tableau regarding a given completeness bound \( B \). At first we want to consider the case that \( B \) is the depth bound.

**Definition 6.1.** Let \( d(s) \) be the depth of a subgoal in a tableau. Let \( f_1, f_2 \in \mathbb{R} \). Then, we define the complexity \( \Sigma^D \) of a tableau \( T \) with subgoals \( s_1, \ldots, s_k \) by

\[
\Sigma^D(T) = \max\{(f_1 \cdot |s_i| + f_2 \cdot d(s_i)) : 1 \leq i \leq k\}.
\]

The factors \( f_1, f_2 \) describe the correspondence between the difficulty of a subgoal estimated from its syntactical structure and the resource available for solving it. An increase of the literal complexity by an amount of \( f_2 \) can be compensated by an increase of the available resource value by an amount of \( f_1 \). Currently we apply the values \((f_1, f_2) = (1, 4)\).

If we use the inference bound the difficulty of all subgoals (and not only of the hardest ones) influences the solvability of a tableau with given resources.

**Definition 6.2.** Let \( C \) be a set of clauses. Let \( T \) be the search tree for \( C \). Let \( N \) be a node in \( T \) which is labeled with tableau \( T \). \( I(T) \) denotes the number of inferences needed to infer \( T \) in \( T \). Let \( s_1, \ldots, s_k \) be the subgoals of \( T \). We define the complexity \( \Sigma^I \) by

\[
\Sigma^I(T) = I(T) + \sum_{i=1}^{k} |s_i|.
\]

When employing the weighted-depth bound, resources are shared between subgoals. The number of inferences needed to solve certain subgoals influences the resource values of further subgoals. Additionally, the depth of a subgoal influences its resource value (see Moser et al., 1997).

**Definition 6.3.** Let \( C \) be a set of clauses. Let \( T \) be the search tree for \( C \). Let \( N \) be a
node in $T$ which is labeled with tableau $T$. We define the complexity $\Sigma^W_D(T)$ by

$$\Sigma^W_D(T) = \Sigma^D(T) \cdot \Sigma^I(T).$$

Finally, we have to discuss how an enumeration of $\tau_A = \bigcup_{L \in L_N} \tau^L_0$ can take place. Breadth-first search is applied. We choose a value $n$ as the largest value such that the number of elements from $\tau^0$ does not exceed a given threshold. We use a value of 1000 tableaux. Then, we create the set $\bigcup_{L \in L_N} \tau^L_0$ and add further tableaux to this set which are enumerated in breadth-first search up to a value of 100,000 tableaux. These tableaux form the set $\tau_A$. The sets $\tau^L_R$ used for the lemma selection are given as described previously. The overhead caused by the lemma selection is normally rather small. For most examples the selection based on the lemma preferring method takes less than 20 seconds.

6.5. Lemma Selection Based on the Lemma Delaying Method

In order to instantiate our abstract framework for choosing lemmas based on the lemma delaying method we have to introduce methods for measuring unification distances as well as for enumerating front literal tableaux. We want to start with our methods for measuring unification distances.

As described in Section 6.2 we want to rate the structural distances between a clausal lemma and a front literal path. Because of the fact that normally too few inferences are applied when generating the front literal paths in the deduction simulation, there is no front literal tableau which can be closed with a useful lemma. If subgoals of the generated front literal tableau are closed with a “wrong” subproof, unification failures arise.

At first we describe our approach for computing unification distances when using unit lemmas. We compare the structure of a front literal $\sim f$ and a lemma candidate $L$ as follows.

At first we try to “pseudo-unify” $\sim f$ and $L$. We adapt a conventional inference system for unification but ignore failures because two different function symbols occur at the same positions in $\sim f$ and $L$ or occur-check failures. As a result of this unification attempt we obtain a substitution which tries to minimize the structural differences between $\sim f$ and $L$ after applying the substitution.

**Definition 6.4.** The inference system $UD$ works on tuples $(E, \sigma)$ where $E$ is a set of term-pairs, which are annotated with positions, and $\sigma$ is a substitution. It consists of the following inference rules:

$$\begin{align*}
T_1 & : \frac{E \cup \{(s, s) | p\}, \sigma}{E, \sigma} \\
T_2 & : \frac{E \cup \{(x, t) | p\}, \sigma}{E, \sigma} \quad x \neq t, \ x \text{ is a variable occurring in } t \\
T_3 & : \frac{E \cup \{(x, t) | p\}, \sigma}{E \sigma_1, \sigma_1 \circ \sigma} \quad x \text{ is a variable which does not occur in } t, \ \sigma_1 = \{x \leftarrow t\} \\
T_4 & : \frac{E \cup \{(f(s_1, \ldots, s_m), f(t_1, \ldots, t_n)) | p\}, \sigma}{E \cup \{(s_i, t_i) | p: 1 \leq i \leq n\}, \sigma} \\
T_5 & : \frac{E \cup \{(f(s_1, \ldots, s_m), g(t_1, \ldots, t_n)) | p\}, \sigma}{E, \sigma} \quad f \neq g.
\end{align*}$$
Applying the inference system $\mathcal{UD}$ we can compute a substitution (considering literals to be terms) by computing $\left(\left(\sim f, L\right)\right|\epsilon) \vdash_{\mathcal{UD}} \left(\emptyset, \sigma\right)$ where $\vdash_{\mathcal{UD}}$ denotes a derivation step with $\mathcal{UD}$, $\epsilon$ is the empty position, and $\text{id}$ is the empty substitution. There is a non-determinism in the application of the inference rules that can lead to different substitutions $\sigma$. We employed a fixed strategy which favors the application of rules to term-pairs with smaller positions because it is more likely that these term positions are obtained by “correct” inferences (also needed in order to generate $f^L$) than larger positions. Thus, our implementation of $\mathcal{UD}$ computes a unique substitution for a front literal $f$ and a lemma $L$.

After the computation of such a substitution $\sigma$ we rate the structural differences between $\sim f \sigma$ and $L \sigma$ using so-called features. Literals are represented by a vector of feature values. A feature is a function mapping a literal to a natural number and a feature value of a literal w.r.t. a feature $\phi$ equals $\phi(u)$. Examples for features are the number of function symbols of a literal or the depth of the tree representing the literal. An exact definition of the features we have used can be found in Fuchs (1997b) (see also Fuchs, 1997a). Actually a fixed set of features is used which may be improved in future by the use of techniques for feature selection or even feature extraction (see Sherrah et al., 1997).

If we use non-unit lemmas only the front literal and the head literal of the lemma are employed for computing a substitution due to efficiency reasons. Then, we instantiate the literals occurring in the front literal path and the tail of the lemma clause. The structural distance measure which is applied then rates the distance between the front literal and the lemma head as well as the differences between the tail literals and the path literals. Among several sensible realizations of these ideas we have chosen the following distance measure.

**Definition 6.5.** Let $d_F(u, v)$ be the Euclidean distance between the feature representations of two literals $u$ and $v$. Let $C$ be a clause set. Let $\mathcal{L}_{N_I}$ be a set of lemma candidates for $C$ and let $k \in \mathbb{N}$. We define the distance measure $d$ which maps two literals to a real number by

$$d(u, v) = \begin{cases} \infty & ; (u = P(t_1, \ldots, t_n), v \neq -P(s_1, \ldots, s_n)) \text{ or } (u = -P(t_1, \ldots, t_n), v \neq P(s_1, \ldots, s_n)) \\ d_F(\sim u, v) & ; \text{otherwise.} \end{cases}$$

Now, let $P = (p_1, \ldots, p_m)$ be a front literal path (with path end literal $p_m$). Let $C = h \leftarrow t_1, \ldots, t_n$ be a lemma clause. Let $\left(\left(\sim p_m, h\right)\right|\epsilon, \text{id}) \vdash_{\mathcal{UD}} \left(\emptyset, \sigma\right)$. We define $\Delta(P, C) \in \mathbb{R}$ by

$$\Delta(P, C) = \max \left\{ d(p_m \sigma, h \sigma), \min_{\phi: \{1, \ldots, n\} \rightarrow \{1, \ldots, m\}} \left\{ \{Max_\phi(P, C)\} \right\} \right\}$$

where $Max_\phi(P, C)$ is given by

$$Max_\phi(P, C) = \max\{d(p_{\phi(i)} \sigma, \sim t_i \sigma) : 1 \leq i \leq n\}.$$

Now we want to deal with the question of how to appropriately generate a set $\tau_A$ of tableaux. Analogously to before a simple approach is to generate tableaux via breadth-first search starting from the tableaux of $\tau_0$. We generate up to 500 front literals or stop if 100 000 tableaux are enumerated. The time needed for the lemma selection based on
lemma delay is normally less than 10 seconds. Further methods for measuring unification distances and enumerating front literal tableaux can be found in Fuchs (1997c).

7. Experimental Results

We want to evaluate our method in the light of experiments performed with the model elimination prover SETHEO (Letz et al., 1992) and different lemma generation techniques. We conducted experiments in the well-known problem library TPTP-v2.0.0 (Sutcliffe et al., 1994). In detail, we have used the domains BOO (boolean algebra), CAT (category theory), COL (combinatory logic), GEO (geometry), GRP (group theory), and SET (set theory). In the following we want to perform an empirical investigation of which lemma technique is the best regarding a given proof task. Furthermore, we want to compare our lemma-based variants of SETHEO with the conventional SETHEO system, an unbounded use of all generated lemmas in SETHEO, and a variant of SETHEO where lemmas are selected based on simple syntactic properties of a clause. In the following we will only consider “hard problems” which are unsolvable with the standard version of SETHEO with a run time which is smaller than 10 seconds on a Sun Ultra II.

7.1. Comparison of Different Lemma Techniques

Before we investigate different lemma techniques in more detail we want to clarify which variant of CTC we use since we have seen that the choice of the calculus has a strong influence on the choice of a sensible lemma technique. In the following we will consider the same calculi which are automatically chosen by the prover SETHEO regarding a given proof task (cf. Moser et al., 1997). We use a variant of CTC where additionally regularity, tautology-freeness, and subsumption conditions are used. Additionally, in the non-Horn case folding-up is employed which turns out to be successful.

We want to investigate for the Horn and non-Horn case which lemma generation and selection techniques, as well as which proof strategy for the final top-down proof run, is sensible if the calculi as described above are used. We will describe sensible basic settings for lemma generation, selection, and the top-down proof run. Then we pose some open questions regarding these settings which are examined in experiments afterwards.

Let us consider the Horn case. It is clear that unit lemmas are sufficient to provide a proof length and depth reduction. In order to prevent an uncontrolled increase of the minimal proof segment with lemmas we employ regularity, tautology-freeness, and subsumption tests during the generation of the lemmas. Regarding the selection of lemmas we will examine whether lemma delay or lemma preference can provide better results. Finally, the question should be answered of which search bound is appropriate for the final proof run. Since we can generate lemmas which guarantee a proof depth reduction, the depth bound seems to be appropriate. The use of some few lemmas which provide a reduction of the depth resource which is needed to enumerate a proof can significantly decrease the size of the segment of the search tree which contains a proof. Thus, it is interesting to investigate whether the depth bound and lemmas improve on the weighted-depth bound and lemmas even if the weighted-depth bound is superior without lemmas.

When considering the variant of CTC used in the non-Horn case (which involves folding-up) we have seen that the use of non-unit lemmas can provide guarantees for a proof length reduction whereas the use of unit lemmas cannot. Nevertheless, it remains an interesting question whether only unit or also non-unit lemmas should be used. We have
to examine whether non-unit lemmas actually provide some advantage by decreasing the
needed resource and do not simultaneously produce too much overhead. Furthermore, it
should be investigated in this context whether extension restrictions on lemmas can lead
to a better control of the lemma use. The use of folding-up when generating the lemmas
is sensible. Considering the reordering of the search space again, the calculus refinements
as employed in the final proof run should be used. As in the Horn case, the effectiveness
of the different selection techniques should be compared. When considering the choice of
a completeness bound for the final proof run, the weighted-depth bound appears to be
more appropriate since only a small or even no resource reduction may take place when
employing the depth bound (and thus the size of the search space may increase).

In summary, the following aspects should be examined. When considering the lemma
generation in the non-Horn case unit and non-unit lemma, mechanisms (with and without
extension restrictions) should be compared. Furthermore, we should compare lemma
delaying and lemma preference based lemma selection and study which completeness
bound appears to be the most appropriate.

**Lemma Generation.** At first we want to look at the question whether the use of non-unit
clauses is beneficial when dealing with non-Horn domains. We experimented in the non-
Horn domains GEO and SET. In our experiments we could renounce the use of non-units
in the GEO domain. There in over 70% of the proofs found with a minimal resource by
the proof system without lemmas and the weighted-depth bound (which performs best in
this domain), no reduction steps occur. Unit lemmas can provide a stable increase of the
performance of Setheo. The use of non-unit lemmas cannot increase the performance.
In the SET domain in 27 of 39 proofs which can be found by Setheo with a minimal
resource value and the weighted-depth bound (which again performs best in this domain),
reduction steps occur. In this domain the use of non-units is profitable.

We compare the performance of the conventional Setheo system with a system which
only employs unit lemmas (Setheo/U), a system where units and non-units are used
(Setheo/NU), and a system where units and non-units are used in connection with ex-
tension restrictions on non-unit lemmas (Setheo/NUR). Lemmas are generated using
an upper limit of 1000 and 3000 clauses for unit and non-unit lemma candidates, respec-
tively. We do not iterate the generation process ($N_l = 0$) and use the depth resource
$n_D = 2$. The length of non-unit lemmas is limited to a value of 2, i.e. a non-unit lemma
consists of a head and a tail literal. Lemma delay based selection is employed. In all
problems the number of chosen units as well as non-units ranges from 3 to 10. In Table 1
we show the number of problems which can be solved after 1, 5, and 10 minutes run time
on a Sun Ultra II workstation.

### Table 1. Experiments in non-Horn domains.

<table>
<thead>
<tr>
<th>GEO</th>
<th>Setheo</th>
<th>Setheo/U</th>
<th>Setheo/NU</th>
<th>Setheo/NUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤ 1 min</td>
<td>10</td>
<td>15</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>≤ 5 min</td>
<td>17</td>
<td>20</td>
<td>13</td>
<td>18</td>
</tr>
<tr>
<td>≤ 10 min</td>
<td>19</td>
<td>21</td>
<td>16</td>
<td>21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SET</th>
<th>Setheo</th>
<th>Setheo/U</th>
<th>Setheo/NU</th>
<th>Setheo/NUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤ 1 min</td>
<td>18</td>
<td>40</td>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td>≤ 5 min</td>
<td>36</td>
<td>48</td>
<td>47</td>
<td>48</td>
</tr>
<tr>
<td>≤ 10 min</td>
<td>39</td>
<td>50</td>
<td>49</td>
<td>52</td>
</tr>
</tbody>
</table>
We want to note that the time for lemma generation and selection is included. We can see that there are examples where the additional use of non-unit lemmas can lead to further resource reductions which are impossible with units alone. We can see that the extension restrictions are profitable for a practical applicability of non-unit lemmas.

**Lemma Selection.** In our experiments the lemma selection based on the lemma delaying method was in general more effective than the selection based on the lemma preferring method. Whereas we could reach stable successes with lemmas selected by lemma delay, the lemma preference based selection proved to be more instable. In the BOO, CAT, and GRP domain we reached results as good as when employing lemma delay based selection. In the COL and SET domain we always performed worse than lemma delay. Finally, in GEO we reached overall worse results but could also solve problems that are out of reach of any other selection strategy we have applied up to now.

In Table 2 we show some selected results obtained in the domains COL and GEO with unit lemmas (which are generated as described before). The different versions of Setheo which are depicted are a conventional version of Setheo without lemmas, a version based on lemma delay (Setheo/LD), and a version based on lemma preference (Setheo/LP). For Setheo we have used the weighted-depth bound which performs best in these domains. The other lemma-based versions use the depth bound in the COL domain (see also the discussion of the choice of a search bound). In the GEO domain the weighted-depth bound is used for all lemma-based Setheo variants. We depict the run times in seconds obtained on a Sun Ultra II. The symbol “—” shows that no proof could be found within 10 minutes. The run times include the time for lemma generation and selection. Furthermore, we show the number of solved problems after 1, 5, and 10 minutes for the respective domains.

**Proof Search.** When not using lemmas the weighted-depth bound performs much better than the depth bound in the non-Horn domains CAT, GEO, and SET. Considering the Horn domains the weighted depth bound is superior in the COL domain. In the BOO and GRP domain the depth and weighted-depth bound show a nearly equal performance.

According to our expectations in the Horn case the use of the depth bound becomes
Table 3. Experiments in the TPTP library.

<table>
<thead>
<tr>
<th>BOO</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
<th>CAT</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤ 1</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>11</td>
<td>≤ 1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>≤ 5</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>12</td>
<td>≤ 5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>≤ 10</td>
<td>10</td>
<td>6</td>
<td>8</td>
<td>13</td>
<td>≤ 10</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COL</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
<th>GEQ</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤ 1</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>14</td>
<td>≤ 1</td>
<td>10</td>
<td>5</td>
<td>6</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>≤ 5</td>
<td>10</td>
<td>26</td>
<td>13</td>
<td>29</td>
<td>≤ 5</td>
<td>17</td>
<td>7</td>
<td>8</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>≤ 10</td>
<td>27</td>
<td>28</td>
<td>20</td>
<td>32</td>
<td>≤ 10</td>
<td>19</td>
<td>8</td>
<td>9</td>
<td>21</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GRP</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
<th>SET</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
<th>Setheo</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤ 1</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>≤ 1</td>
<td>18</td>
<td>11</td>
<td>13</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>≤ 5</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>11</td>
<td>≤ 5</td>
<td>36</td>
<td>21</td>
<td>24</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>≤ 10</td>
<td>9</td>
<td>8</td>
<td>6</td>
<td>11</td>
<td>≤ 10</td>
<td>39</td>
<td>22</td>
<td>25</td>
<td>52</td>
<td></td>
</tr>
</tbody>
</table>

more attractive when employing unit lemmas. The depth bound and lemmas perform normally as good as the weighted-depth bound and lemmas and sometimes slightly better. In the BOO and GRP domain we achieve results similar to those obtained with the weighted-depth bound. In the COL domain we achieve better results with the depth bound than with the weighted-depth bound. We can solve all problems solvable with the weighted-depth bound (and lemmas) and additionally three new problems.

In the non-Horn domains the use of the depth bound and lemmas cannot improve on the use of lemmas in connection with the weighted-depth bound. The weighted-depth bound remains clearly superior to the depth bound. In many cases it was not possible to shorten each branch of a proof.

7.2. EVALUATION OF THE PERFORMANCE

In order to demonstrate the strength of our new lemma approach we want to compare a fixed lemma system which uses the newly developed lemma selection techniques (Setheo/Rel) with a conventional version of Setheo, a version which uses all lemma candidates from \( \mathcal{L}_{N_1} \) (Setheo/Delta), and a version which selects lemmas according to the number of symbols of a literal (Setheo/Symb). The conventional version of Setheo is the same as that which is described in Moser et al. (1997).

When dealing with lemmas we use a system which is configured as follows. Unit lemmas are generated up to a value of 1000 lemmas. Non-unit lemmas are limited to at most 3000 clauses of length 2. We have used the lemma generation techniques which proved to be the best in the experimental studies as described in the preceding section. Thus, we use unit lemmas in the Horn case, and non-unit lemmas together with extension restrictions in the non-Horn case. All calculus refinements which are employed in the final proof run are also used when generating lemmas. Furthermore, the folding-up rule is used in the non-Horn case. The iteration number and the depth resource has been \( N_I = 0 \) and \( n_D = 2 \), respectively. The lemma selection is performed based on lemma delay. The search bound which has been used was the depth bound for domains which
mostly contain Horn problems (BOO, COL, GRP) and the weighted-depth bound for non-Horn domains (CAT, GEO, SET).

Table 3 shows the experimental results. We depict the number of problems which can be solved after 1, 5, and 10 minutes. We can observe a consistent gain of efficiency of SETHEO/REL compared to the other versions of SETHEO (caused from resource reductions and only small increases of the branching rate of the search tree). We can solve new problems and also significantly decrease the run times when employing our lemma techniques. For a lot of problems the run time could be decreased from over 5 minutes to less than 1 minute. This is very interesting when using SETHEO as a component in interactive proof systems such as ILF (Dahn et al., 1997). We can see that a selection of lemmas is actually needed since an unbounded use of all lemma candidates (SETHEO/Delta) normally significantly decreases the performance. Furthermore, simple selection techniques which prefer short clauses are no match for our new partial evaluation techniques.

8. Discussion and Future Work

We have extended the work of Schumann (1994) and Fuchs (1998a,b) by introducing techniques for a controlled use of clausal lemmas in connection tableau calculi. We have investigated the potential of the lemma use in order to provide a proof length and depth reduction as well as for reordering the search space in an appropriate manner. We considered the conventional connection tableau calculus as well as some refinements and extensions which proved to be relevant for a practical success of the calculus. For Horn as well as for non-Horn domains, we could reach significant improvements of our basic prover.

In the past, there have been some approaches for dynamically creating unit lemmas during the proof run (see Astrachan and Stickel, 1992; Astrachan and Loveland, 1997; Iwanuma, 1997). Lemmas are created after a closed subtree is derived during the proof search where no reduction steps from outside are used. These approaches cannot guarantee that useful lemmas can be generated during the proof run even in the Horn case. Thus, although some hard problems could only be solved with such lemma techniques (cf. Astrachan and Loveland, 1997), no stable success has been reported over a large set of problems. The use of lemmas was mainly controlled with lemma selection criteria based on syntactic properties of a lemma candidate ignoring the concrete proof task. Furthermore, in Iwanuma (1997) a method for controlling the application of lemmas during the proof run (“lemma matching”) was used but this method is not complete in order to allow resource reductions which are essential for our method. To our knowledge, no successful approach for a dynamic use of full non-unit lemmas has been reported in the past. Currently, techniques like c-reduction (Shostak, 1976) or folding-up (Letz et al., 1994) provide restricted non-unit lemma mechanisms. As discussed, these mechanisms are not fully compatible with the use of bottom-up generated lemmas. As our experiments reveal, with an appropriate choice of a completeness bound and sophisticated selection mechanisms good results are possible with our (static) lemma mechanisms and folding-up.

In Fuchs (1999) a static approach for creating clausal lemmas for connection tableau provers is described. In contrast to our approach, certain decompositions of proof goals, so-called subgoal clauses, are generated which are only used for the start expansion. The approach suffers from theoretical shortcomings in the area of non-Horn problems where
no guarantees for proof search reductions can be provided. Nonetheless, this approach also yields satisfactory results in practice.

Concerning selection strategies, in the past mainly uninformed strategies which use the literal length or derivation costs of a lemma have been used (e.g. Astrachan and Stickel, 1992; Markovitch and Scott, 1993; Schumann, 1994; Fuchs and Wolf, 1998). In contrast we apply criteria which consider the actual proof task. In order to guide deduction systems some other approaches have also used difference information similar to our partial evaluation techniques. In Digricoli (1985) and Bläsius and Siekmann (1988) techniques for partial unification have been applied which are similar to our pseudo-unifying method. In these approaches the detected differences have been explicitly exploited in order to guide paramodulation steps. 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 of how to reduce the differences which remain after instantiation. Since we only want to estimate whether a lemma may be useful in general we do not fall into these drawbacks. Structural difference measures which are subsumed by our unification distance measure have been applied by Denzinger and Fuchs (1994) (see also Denzinger et al., 1997). The distance value between the actual proof goal and an equation has been used in order to control the selection of equations to be processed by an equational theorem prover. Similar to our approach no explicit use of the difference information is made in Denzinger and Fuchs (1994). But in contrast to our method no explicit consideration of possible deductions is made. Thus, this technique mainly works in combination with other heuristics.

Future work will deal with an improved control of the use of lemmas during the proof process. While we actually employ rather sophisticated techniques for the selection of lemmas (cf. also Fuchs, 1997c), we do not control the application of lemmas in order to extend certain subgoals. The general idea is to extract knowledge about possible resource reductions obtainable with lemmas (e.g. from proof experiences) and to use this knowledge in order to forbid certain lemma applications. For instance, lemma extensions to “high” subgoals may be forbidden.

Additionally, in future we will iterate the lemma generation process in order to try to work with “harder lemmas” which have a higher potential for resource reductions. It would be interesting to discover whether even harder problems can be solved with these methods and whether stable successes are still possible. An interesting environment for employing harder lemmas is the cooperative parallel theorem prover CPTHEO (Fuchs and Wolf, 1998). Within this prover the parallel generation of ever harder lemma candidates and a partial evaluation-based selection of lemmas which becomes more precise over time is possible. Asynchronously, proof tasks are started which contain the input clauses and a lemma set which currently appears to be best-suited. Thus, this prover seems to be appropriate in order to solve hard problems. Moreover, it appears to be self adaptive to the difficulty of a problem which is important to obtain stable successes.

References


