Cooperation between Top-Down and Bottom-Up Theorem Provers by Subgoal Clause Transfer

Dirk Fuchs*
Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
Germany
E-mail: dfuchs@informatik.uni-kl.de

Abstract

Top-down and bottom-up theorem proving approaches have each specific advantages and disadvantages. Bottom-up provers profit from strong redundancy control and suffer from the lack of goal-orientation, whereas top-down provers are goal-oriented but have weak calculi when their proof lengths are considered. In order to integrate both approaches our method is to achieve cooperation between a top-down and a bottom-up prover: The top-down prover generates subgoal clauses, then they are processed by a bottom-up prover. We discuss theoretic aspects of this methodology and we introduce techniques for a relevancy-based filtering of generated subgoal clauses. Experiments with a model elimination and a superposition-based prover reveal the high potential of our cooperation approach.

*The author was supported by the *Deutsche Forschungsgemeinschaft (DFG)*.

1 Introduction

Automated deduction is—at its lowest level—a search problem that spans huge search spaces. In the past, many different calculi have hence been developed in order to cope with problems from the area of automated theorem proving. Essentially, for first-order theorem proving two main different paradigms for calculi are in use: *Top-down calculi* like *model elimination* (*ME*, see, e.g., [Lov78], [Fit96]) attempt to recursively break down and transform a goal into subgoals that can finally be proven immediately with the axioms or with assumptions made during the proof. *Bottom-up calculi* like *superposition* (see [BG94]) go the other way by producing logic consequences from the initial clause set until the empty clause is derived.

When comparing results of various provers it is obvious that provers based on different paradigms often have quite a different behavior. There are problems where bottom-up theorem provers perform considerably well, but top-down provers poorly, and vice versa. The main reason for this is that bottom-up provers often suffer from the lack of goal-orientation of their search, but profit from their strong redundancy control mechanisms. In contrast, top-down provers profit from their goal-orientation and suffer from insufficient redundancy control thus entailing long proof lengths for many problems. Therefore, a topic that has come into focus of research is the integration of both approaches. In particular, cooperation between theorem provers based on top-down and bottom-up principles appears to be a promising way because by exchanging information each approach can profit from the other ([BS97]). Note that it is also possible to modify calculi or provers which work according to one paradigm so as to introduce aspects of the other paradigm into it. This, however, requires a lot of implementational effort to modify the provers, whereas our approach does not require changes of the provers but only changes of their input (see section 3). Thus, we can employ arbitrary state-of-the-art provers and are very flexible.

Information that is well-suited for top-down provers are lemmas deduced by bottom-up provers. These lemmas are added to the input of a top-down prover and can help to shorten the proof length by immediately solving subgoals. Since the arising problem to filter relevant lemmas has already been discussed (see, e.g., [AS92], [AL97], [Fuc97]) we are not going to deal with this aspect here.

Instead, we want to consider top-down/bottom-up integration by transferring information from a top-down prover to a bottom-up prover. By transferring top-down generated *subgoal clauses* we are able to introduce a goal-oriented component into a bottom-up prover which enables it to solve proof problems considerably faster (see section 5). However, since similar to the use of lemmas an unbounded transfer of subgoal clauses is not sensible, techniques for *filtering relevant subgoal clauses* must be developed.

In order to examine this kind of top-down/bottom-up integration we start by giving a brief overview about superposition and model elimination theorem proving in section 2. After that, in section 3 we introduce subgoal clauses and discuss effects of the integration of *ME* subgoal clauses into the search state of a superposition-based prover. In section 4, we point out two variants of a relevancy-based filtering of subgoal clauses. An experimental study conducted with the theorem provers SETHEO

([LSBB92], [MIL+97]) and SPASS ([WGR96]) reveals the potential of our techniques. Finally, a discussion and an outlook at possible future work conclude the report.

2 Automated Theorem Proving with Superposition and Model Elimination

The general problem in first-order theorem proving is to show the inconsistency of a set $\mathcal{C}$ of clauses. Clauses are sets of literals. As already discussed, theorem provers utilize either top-down or bottom-up calculi for accomplishing this task.

Typically, a bottom-up calculus contains several inference rules which can be applied to a set of clauses that constitute the search state. The superposition calculus (e.g., [BG94]) contains the inference rules superposition, equality resolution, and equality factoring. It is to be emphasized that we employ the version of the superposition calculus introduced in [BG94]. Specifically this entails that factoring is only applied to positive literals.

Usually, a bottom-up theorem prover maintains a set $\mathcal{F}^P$ of so-called *potential* or *passive clauses* from which it selects and removes one clause C at a time. This clause is put into the set $\mathcal{F}^A$ of *activated clauses*. Activated clauses are, unlike potential clauses, allowed to produce new clauses via the application of some inference rules. The inferred new clauses are put into $\mathcal{F}^P$. Initially, $\mathcal{F}^A = \emptyset$ and $\mathcal{F}^P = \mathcal{C}$. The indeterministic selection or *activation step* is realized by heuristic means. To this end, a heuristic $\mathcal{H}$ associates a natural number $\mathcal{H}(C) \in \mathbb{N}$ with each $C \in \mathcal{F}^P$, and the $C \in \mathcal{F}^P$ with the smallest weight $\mathcal{H}(C)$ is selected.

A typic top-down calculus is model elimination, essentially a restricted version of the *connection tableau calculus* (CTC) ([LMG94]). This calculus works on *connected tableaux* or *connection tableaux* for $\mathcal{C}$. A tableau T for $\mathcal{C}$ is a tree whose non-root nodes are labeled with literals and that fulfills the condition: If the immediate successor nodes $v_1, \ldots, v_n$ of a node v of T are labeled with literals $l_1, \ldots, l_n$, then the clause $l_1 \vee \ldots \vee l_n$ (*tableau clause*) is an instance of a clause in $\mathcal{C}$. A tableau is called *connected* if each inner node v (non-leaf node) which is labeled with a literal l has a leaf node v' among its immediate successor nodes that is labeled with a literal l' complementary to l. The inference rules are *start*, *extension*, and *reduction*. The start rule allows for a standard tableau expansion that can only be applied to a trivial tableau, i.e. one consisting of only one node. An expansion step means selecting a clause from $\mathcal{C}$ and attaching its literals to a leaf node of an *open* branch, i.e. a branch that does not contain two complementary literals. Note that often also a restricted (but nevertheless complete) version of the calculus (that is called CTC_{neg}) is in use that only allows to perform the start expansion with negative clauses. Tableau reduction closes a branch by unifying a *subgoal* s (i.e. the literal at the leaf of an open tableau branch) with the complement of a literal r (denoted by $\sim r$) on the same branch, and applying the substitution to the whole tableau. Extension is a combination of expansion and reduction. It is performed by selecting a subgoal s in the tableau T, applying an expansion step to s, and immediately performing a reduction step with s and one of its newly created successors.

Note that in the area of Horn clauses it is sufficient to employ start and expansion, i.e. the reduction inference is unnecessary. Thus, we assume that we use versions of CTC or CTC_{neg} that do not employ reduction steps in the area of Horn clauses.

CTC or CTC_{neg} do not have specific inference rules for handling equality. Instead, when dealing with equality the axiomatization must be extended by certain axioms, namely the reflexivity, symmetry, transitivity, and substitution axioms of the equality symbol.

Important is the notion of a tableau derivation and a search tree: We say $T \vdash T'$ if (and only if) tableau T' can be derived from T by applying a start rule if T is the trivial tableau, or by an extension/reduction rule to a subgoal in T. In order to show the unsatisfiability of a set $\mathcal{C}$, all derivations starting from the trivial tableau have to be enumerated until a closed tableau appears (a tableau with no open branch). Thus, the search space is given by a tableau search tree $\mathcal{T}$ defined as follows: A *search tree* $\mathcal{T}$ defined by a clause set $\mathcal{C}$ is given by a tree, whose root is labeled with the trivial tableau. Every inner node in $\mathcal{T}$ labeled with tableau T has as immediate successors the maximal set of nodes $\{v_1, \ldots, v_n\}$, where v_i is labeled with T_i and $T \vdash T_i$, $1 \leq i \leq n$.

Since, in comparison with resolution style calculi, not only the number of proof objects but also their size increases during the proof process, explicit tableaux enumeration procedures that construct all tableaux in $\mathcal{T}$ in a breadth-first manner are not sensible. Hence, implicit enumeration procedures are normally in use that apply *consecutively bounded iterative deepening search with backtracking* ([Kor85]). In this approach iteratively larger finite initial parts of the search tree $\mathcal{T}$ are explored with depth-first search. Normally, a finite segment is defined by a so-called *completeness bound* (which poses structural restrictions on the tableaux which are allowed in the current segment) and a fixed natural number, a so-called *resource*. Iterative deepening is performed by starting with a basic resource value $n \in \mathbb{N}$ and iteratively increasing n until a proof is found within the finite initial segment of $\mathcal{T}$ defined by a bound and n. A prominent example for a completeness bound is the *inference bound* (see [Sti88]) which limits the number of inferences needed to infer a certain tableau.

3 Subgoal Clauses for Top-Down/Bottom-Up Integration

3.1 Transferring top-down generated clauses to a bottom-up prover

Because of the fact that connection tableaux-based provers have a search state which contains deductions (tableaux) instead of clauses, it is at first sight not obvious how to extract valid clauses from such a search state which are well-suited for a superposition-based prover. A common method in order to extract valid clauses is to employ lemma mechanisms of ME provers: Assume that a literal s is a label of the root node of a closed subtableau T^s. Let $l_1, \ldots, l_n$ be the literals that are used in reduction steps and that are outside of T^s. Then, the clause $\sim s \vee \sim l_1 \vee \ldots \vee \sim l_n$ may be derived as a new lemma (since it is a logical consequence of the tableau clauses in T^s). Then, such a lemma could be transferred to a bottom-up prover. As appealing as this idea sounds, it has some grave restrictions: Usually, such lemmas are—due to instantiations needed to close other branches before—not as general as they could be. Hence, they often cannot be used in inferences, especially not in *contracting inferences* (subsumption, rewriting) which are very important for bottom-up provers. Moreover, they do not introduce goal-orientation and hence do not make use of the advantages of the search scheme typic for ME.

The concept of subgoal clauses, however, allows for the generation of clauses derived by inferences involving a proof goal: A *subgoal clause* S_T regarding a tableau T is the clause $S_T \equiv l_1 \vee \ldots \vee l_n$, where the literals l_i are the subgoals of the tableau T. The *subgoal clause set* $\mathcal{S}^{\mathcal{B},n,\mathcal{C}}$ w.r.t. a bound $\mathcal{B}$, a resource n, and a clause set $\mathcal{C}$, is defined by $\mathcal{S}^{\mathcal{B},n,\mathcal{C}} = (\cup S_T)\backslash\mathcal{C}$, T is a tableau which is a label of a node in the initial segment of the search tree for $\mathcal{C}$ (generated with CTC) that is defined by bound $\mathcal{B}$ and resource n. The *subgoal clause set* $\mathcal{S}^{\mathcal{B},n,\mathcal{C}}_{neg}$ is the set of subgoal clauses $\mathcal{S}^{\mathcal{B},n,\mathcal{C}}_{neg} = \{S_T : S_T \in \mathcal{S}^{\mathcal{B},n,\mathcal{C}}$, the start expansion of T is performed with a negative clause$\}$. Note that subgoal clauses are valid clauses, i.e. logic consequences of the initial clause set.

Example 3.1 Let $\mathcal{C} = \{\neg g, p_1 \wedge \ldots \wedge p_n \rightarrow g, q_1 \wedge \ldots \wedge q_m \rightarrow g\}$. Then, $\neg p_1 \vee \ldots \vee \neg p_n$ is the subgoal clause S_T belonging to the tableau obtained when extending the goal $\neg g$ with the clause $p_1 \wedge \ldots \wedge p_n \rightarrow g$. If we employ $\mathcal{B}$ = inference bound (Inf) and resource $k = 2$, then $\mathcal{S}^{\mathcal{B},k,\mathcal{C}} = \mathcal{S}^{\mathcal{B},k,\mathcal{C}}_{neg} = \{\neg p_1 \vee \ldots \vee \neg p_n, \neg q_1 \vee \ldots \vee \neg q_m\}$.

A subgoal clause S_T represents a transformation of an original goal clause (which is the start clause of the tableau T) into a new subgoal clause realized by the deduction which led to the tableau T. The set $\mathcal{S}^{Inf,k,\mathcal{C}}$ is the set of all possible goal transformations into subgoal clauses within k inferences if we use CTC, the set $\mathcal{S}^{Inf,k,\mathcal{C}}_{neg}$ is the set of all possible goal transformations into subgoal clauses within k inferences if we use CTC_{neg}. More exactly, $\mathcal{S}^{Inf,k,\mathcal{C}}$ is the closure of all (goal) clauses w.r.t. (a fixed number k of) extension and reduction steps, $\mathcal{S}^{Inf,k,\mathcal{C}}_{neg}$ is the closure of all negative (goal) clauses w.r.t. (a fixed number k of) extension and reduction steps.

Now, in order to couple a ME and a superposition prover, we generate with the inference bound and a fixed resource $k > 0$ either the set $\mathcal{S}^{Inf,k,\mathcal{C}}$ and a superposition-based prover obtains $\mathcal{C} \cup \mathcal{S}^{Inf,k,\mathcal{C}}$ as input, or we generate $\mathcal{S}^{Inf,k,\mathcal{C}}_{neg}$ and the superposition-based prover obtains $\mathcal{C} \cup \mathcal{S}^{Inf,k,\mathcal{C}}_{neg}$.

3.2 Reduction of proof lengths and searches through subgoal clauses

This method gives rise to the question whether a *proof length reduction* is possible, i.e. whether there are shorter superposition proofs of the inconsistency of $\mathcal{C} \cup \mathcal{S}^{Inf,k,\mathcal{C}}$ or $\mathcal{C} \cup \mathcal{S}^{Inf,k,\mathcal{C}}_{neg}$ than of the inconsistency of $\mathcal{C}$. Note that we measure the length of a proof by counting the number of inference steps *needed in it*. It is more important whether a bottom-up prover can really profit from a possible proof length reduction in form of a *proof search reduction*, i.e. a reduction of the number of inferences the prover *needs in order to find* a proof. Specifically, it is interesting to find out in which cases the proof search reduction is high.

3.2.1 Subgoal clause generation via CTC

Firstly, we examine the case that we employ calculus CTC, i.e. generate $\mathcal{S}^{Inf,k,\mathcal{C}}$. We assume further that no equality is involved in the problem, i.e. superposition corresponds to (ordered) resolution. Then, following theorem holds true:

Theorem 3.1

1. *Let $\mathcal{C}$ be a set of ground clauses not containing equality, let $\Box \notin \mathcal{C}$, and let $k > 1$ be a natural number. Let P_1 and P_2 be minimal (w.r.t. the number of inference steps $|P_1|$ and $|P_2|$) resolution refutation proofs for $\mathcal{C}$ and $\mathcal{C} \cup \mathcal{S}^{Inf,k,\mathcal{C}}$, respectively. Then, it holds: $|P_1| > |P_2|$.*

2. *For each $k > 1$ there is a set of (non-ground) clauses $\mathcal{C}_k$ not containing equality ($\Box \notin \mathcal{C}_k$), such that no minimal resolution refutation proof for $\mathcal{C}_k \cup \mathcal{S}^{Inf,k,\mathcal{C}_k}$ is shorter than a minimal resolution refutation proof for $\mathcal{C}_k$.*

Proof:

1. Note that no factorization steps are needed in the case of ground clauses (recall that clauses are sets of literals). Then, the claim is trivial since the result of the first resolution step of each minimal proof is an element of $\mathcal{S}^{Inf,k,\mathcal{C}}$.

2. Let $k > 1$. Let $\mathcal{C}_k$ be defined by $\mathcal{C}_k = \{\{\neg p(x_1), \ldots, \neg p(x_k)\}, \{p(y_1), \ldots, p(y_k)\}\}$. Let $>= \emptyset$ be the ordering used for superposition. Then, a minimal resolution refutation proof for $\mathcal{C}_k$ requires $k-1$ factorization steps (resulting in the clause $\{p(y_1)\}$) and k resolution steps. Furthermore, in $\mathcal{S}^{Inf,k,\mathcal{C}_k}$ are only clauses which contain at least one positive and one negative literal. Thus, none of these clauses can lead to a refutation proof for $\mathcal{C}_k \cup \mathcal{S}^{Inf,k,\mathcal{C}_k}$ in less than $2k-1$ inferences.

$\Box$

Hence, a reduction of the proof length is at least in the ground case possible. However, the (heuristic) proof search of a superposition-based prover need not always profit from the proof length reduction obtained. E.g., it is possible that all clauses of a minimal refutation proof of $\mathcal{C}$ have smaller heuristic weights than the clauses from $\mathcal{S}^{Inf,k,\mathcal{C}}$ and will hence be activated before them. Consider, e.g., following example:

Example 3.2 Let $>= \emptyset$ be the ordering used for superposition. Let the clause set $\mathcal{C}$ be given by $\mathcal{C} = \{a \wedge b \rightarrow c, g \rightarrow b, a, g, \neg c\}$. The heuristic $\mathcal{H}$ corresponds to the FIFO heuristic. Further, for the first 10 times, resolvents of the two most recently activated clauses are preferred by $\mathcal{H}$. Then, following clauses are activated by the prover (in this order): $a \wedge b \rightarrow c, g \rightarrow b, a \wedge g \rightarrow c, a, g \rightarrow c, g, c, \neg c, \Box$. Furthermore, if the subgoal clauses of $\mathcal{S}^{Inf,k,\mathcal{C}}$ are inserted behind the original axioms the prover will find the same resolution refutation proof for $\mathcal{C} \cup \mathcal{S}^{Inf,k,\mathcal{C}}$ ($k \geq 0$) and the proof search can hence not profit from a possible proof length reduction.

However, since the example (especially the chosen heuristic) is somewhat contrived it can be expected that for many problems clauses from $\mathcal{S}^{Inf,k,\mathcal{C}}$ will be activated and can contribute to a reduction of the search effort.

In the case that equality is involved in the problem, a proof length reduction is not guaranteed even for ground clauses.

Theorem 3.2 *For each resource $k > 1$ there is a set of ground unit equations $\mathcal{C}$ ($\Box \notin \mathcal{C}$) where the minimal superposition refutation proofs for $\mathcal{C} \cup \mathcal{S}^{Inf,k,\mathcal{C}}$ are not shorter than minimal proofs for $\mathcal{C}$.*

Proof: Let $>= \emptyset$ be the ordering used for superposition. Consider the set of unit equations $\mathcal{C} = \{a = b, f^{k-1}(a) \neq f^{k-1}(b)\}$. We assume that $>= \emptyset$ is used as an ordering for superposition. Then, a minimal superposition refutation proof for $\mathcal{C}$ requires two inferences, a superposition step into $f^{k-1}(a) \neq f^{k-1}(b)$ resulting in the inequation $f^{k-1}(a) \neq f^{k-1}(a)$, and then an equality resolution step. In the set $\mathcal{S}^{Inf,k,\mathcal{C}}$ are either non-unit clauses whose refutation requires at least 2 inferences or the units $\mathcal{U} = \{f^j(a) \neq f^j(b), f^j(a) = f^j(b) : 0 \leq j \leq k-1\}$. Since also the refutation of $\mathcal{C} \cup \mathcal{U}$ requires 2 inferences a reduction of the proof length is impossible. $\Box$

Despite this negative result, it is sometimes possible to shorten the proof search if the search space is restructured in a favorable way.

All in all, we obtain that in general the reduction of the heuristic search for a proof cannot be guaranteed although proof lengths—at least for ground clauses and if no equality is involved in the problems—are shortened. Nevertheless, in practice it might often be the case that a restructuring of the search caused by using subgoal clauses allows for finding proofs faster.

3.2.2 Subgoal clause generation via CTC_{neg}

Secondly, we examine the case that we employ calculus CTC_{neg}, i.e. generate $\mathcal{S}_{neg}^{Inf,k,\mathcal{C}}$. First, we examine the case that no equality is involved in the proof problems. Then, even for ground clauses it is possible that minimal proofs cannot be shortened when employing subgoal clauses.

Theorem 3.3 *There is a set of (non-Horn) ground clauses $\mathcal{C}$ where no minimal resolution refutation proof of the inconsistency of $\mathcal{C} \cup \mathcal{S}_{neg}^{Inf,2,\mathcal{C}}$ has a shorter length than a minimal refutation proof of the inconsistency of $\mathcal{C}$.*

Proof: Consider following set $\mathcal{C}$ (again we employ $>= \emptyset$):

$$\mathcal{C} = \left\{ \begin{array}{ccc} \{l_4, l_6, l_7\}, & \{l_4, l_6, \neg l_7\}, & \{l_3, \neg l_4\}, \\ \{l_3, \neg l_6\}, & \{\neg l_2, \neg l_4\}, & \{l_4, \neg l_5, \neg l_6\}, \\ \{\neg l_2, l_5\}, & \{l_1, l_2, \neg l_3\}, & \{\neg l_1, l_2, \neg l_3\} \end{array} \right\}$$

Each minimal refutation proof for this set requires 9 resolution steps. A minimal proof is, e.g.:

$[1, ax]$	$\{l_4, l_6, l_7\}$	$[10, res(1,2)]$	$\{l_4, l_6\}$
$[2, ax]$	$\{l_4, l_6, \neg l_7\}$	$[11, res(6,10)]$	$\{l_4, \neg l_5\}$
$[3, ax]$	$\{l_3, \neg l_4\}$	$[12, res(3,10)]$	$\{l_3, l_6\}$
$[4, ax]$	$\{l_3, \neg l_6\}$	$[13, res(7,11)]$	$\{\neg l_2, l_4\}$
$[5, ax]$	$\{\neg l_2, \neg l_4\}$	$[14, res(5,13)]$	$\{\neg l_2\}$
$[6, ax]$	$\{l_4, \neg l_5, \neg l_6\}$	$[15, res(4,12)]$	$\{l_3\}$
$[7, ax]$	$\{\neg l_2, l_5\}$	$[16, res(8,9)]$	$\{l_2, \neg l_3\}$
$[8, ax]$	$\{l_1, l_2, \neg l_3\}$	$[17, res(14,16)]$	$\{\neg l_3\}$
$[9, ax]$	$\{\neg l_1, l_2, \neg l_3\}$	$[18, res(15,17)]$	$\Box$

Now, it holds:

$$\mathcal{S}_{neg}^{Inf,2,\mathcal{C}} = \left\{ \begin{array}{ccc} \{\neg l_2, l_6, l_7\}, & \{\neg l_2, l_6, \neg l_7\}, & \{l_1, \neg l_3, \neg l_4\}, \\ \{\neg l_1, \neg l_3, \neg l_4\}, & \{\neg l_2, \neg l_5, \neg l_6\} & \end{array} \right\}$$

When enumerating all proofs for $\mathcal{C} \cup \mathcal{S}_{neg}^{Inf,2,\mathcal{C}}$ one can recognize that the minimal proof length cannot be reduced. $\Box$

Note that the above example contains non-Horn clauses. Many interesting problems, however, can be specified in Horn logic. Thus, it is interesting to investigate whether or not a proof length reduction is possible in the case that only Horn problems are considered. Since often in this area bottom-up provers employ positive hyper-resolution (e.g., OTTER ([McC94])), we assume that we have a bottom-up prover also employing this resolution variant. Then, it is not only always possible to reduce the proof length but we can even make the reduction of the proof length more precise. (Note that following theorem holds true for ground as well as non-ground clauses.)

In order to do this, we employ following notions: For natural numbers a and b, $a \dot{-} b$ is equal to $a - b$ if $a > b$, otherwise it is equal to 0. $\frac{a}{b}$ denotes the maximal number z such that $z \cdot b < a$. Moreover, for a set of clauses $\mathcal{C}$, $max_{neg}(\mathcal{C})$ is the maximal number of negative literals which occur in a clause from $\mathcal{C}$.

Theorem 3.4 *Let $\mathcal{C}$ be a set of Horn clauses, $\Box \notin \mathcal{C}$. If P_1 and P_2 are minimal positive hyper-resolution proofs for $\mathcal{C}$ and $\mathcal{C} \cup \mathcal{S}_{neg}^{Inf,k+1,\mathcal{C}}$, respectively, then*

$$|P_1| \dot{-} k \leq |P_2| \leq |P_1| \dot{-} \frac{k}{max_{neg}(\mathcal{C})}$$

Proof: Let $Neg(\mathcal{C}) = \{\neg g_1^1 \vee \ldots \vee \neg g_{n_1}^1, \ldots, \neg g_1^m \vee \ldots \vee \neg g_{n_m}^m\}$, where each $\neg g_1^i \vee \ldots \vee \neg g_{n_i}^i$ $(1 \leq i \leq m)$ is part of a minimal positive hyper-resolution refutation proof for $\mathcal{C}$. Note that in each minimal hyper-resolution proof exactly one negative clause occurs since the result of a hyper-resolution step involving a negative clause is the empty clause.

Firstly, we show that there are clause sets $\mathcal{C}_k$ where the minimal proof length $|P_2|$ when employing subgoal clauses is equal to $|P_1| \dot{-} k$. Such clause sets must be construed in such a way that each resolution step applied to a subgoal clause entails that one hyper-resolution step is saved. We define $\mathcal{C}_k$ by $\mathcal{C}_k = \{\{\neg g\}, \{\neg a_1, g\}, \{\neg a_2, a_1\}, \ldots, \{\neg a_k, a_{k-1}\}, \{a_k\}\}$. Then, obviously the proof length can be reduced to the value $|P_1| \dot{-} k$.

Secondly, we show that for each k there are clause sets $\mathcal{C}_k$ where the minimal proof length $|P_2|$ when employing subgoal clauses is equal to $|P_1| \dot{-} \frac{k}{max_{neg}(\mathcal{C}_k)}$. In order to do this we must construct clause sets $\mathcal{C}_k$ where max_{neg} resolution steps are necessary in order to save one hyper-resolution step. We define the clause set $\mathcal{C}_k$ by $\mathcal{C}_k = \{\{\neg g_1, \ldots, \neg g_k\}, \{g_1\}, \ldots, \{g_k\}\}$. Then, obviously the proof length can be reduced to the value $|P_1| \dot{-} \frac{k}{max_{neg}(\mathcal{C})}$.
Finally, we have to show that for each clause set $\mathcal{C}$ the minimal proof length when employing subgoal clauses is within the claimed limits. We perform induction over k:

$k = 1$: We distinguish between two cases: On the one hand it might be that all subgoals of clauses in $Neg(\mathcal{C})$ can be closed with axioms immediately. Then, if $k \geq minL(Neg(\mathcal{C})) = \min(\{|C| : C \in Neg(\mathcal{C})\})$ then $\square \in \mathcal{S}_{neg}^{Inf,k+1,\mathcal{C}}$. Hence, $|P_2| = 0$, i.e. $0 = |P_1| \dot{-} k \leq |P_2| \leq |P_1| \dot{-} \frac{k}{max_{neg}(\mathcal{C})} \in \{0,1\}$. If $k < minL(Neg(\mathcal{C}))$, then for all minimal proofs P_2 it holds $|P_2| = |P_1| = 1$, i.e. $0 = |P_1| \dot{-} k \leq |P_2| \leq |P_1| \dot{-} \frac{k}{max_{neg}(\mathcal{C})} = 1$. On the other hand, it might be that there is a subgoal of a clause from $Neg(\mathcal{C})$ that cannot be solved with an axiom immediately. Then, in $\mathcal{S}_{neg}^{Inf,k+1,\mathcal{C}}$ there is a subgoal clause $\neg\sigma(g_1^i) \vee \ldots \vee \neg\sigma(g_{u-1}^i) \vee \neg\sigma(a_1^i) \vee \ldots \vee \neg\sigma(a_{z_i}^i) \vee \neg\sigma(g_{u+1}^i) \vee \ldots \vee \neg\sigma(g_{n_i}^i)$ for some i $(1 \leq i \leq m)$, if there is a rule $a_1^i \wedge \ldots \wedge a_{z_i}^i \rightarrow {g_u^i}'$ $(1 \leq u \leq n_i, \sigma = mgu({g_u^i}', g_u^i))$ in $\mathcal{C}$ and a minimal refutation proof uses this rule. Then, $|P_2| = |P_1| - 1$ because the hyper-resolution step involving $a_1^i \wedge \ldots \wedge a_{z_i}^i \rightarrow {g_u^i}'$ in order to generate an instance of ${g_u^i}'$ is saved. Hence, $|P_1| \dot{-} k \leq |P_2| \leq |P_1| \dot{-} \frac{k}{max_{neg}(\mathcal{C})} \in \{|P_1| - 1, |P_1|\}$.

$k \rightarrow k+1$: When applying the induction hypotheses we obtain: There is a subgoal clause $C^k \in \mathcal{S}_{neg}^{Inf,k+1,\mathcal{C}}$ such that for each minimal hyper-resolution proof (of the inconsistency of $\mathcal{C} \cup \mathcal{S}_{neg}^{Inf,k+1,\mathcal{C}}$) P_2^k which employs this clause holds: $|P_1| \dot{-} k \leq |P_2^k| \leq |P_1| \dot{-} \frac{k}{max_{neg}(\mathcal{C})}$.
If $C^k = \square$ then $|P_2^k| = 0$ and also for all minimal proofs P_2^{k+1} of the inconsistency of $\mathcal{C} \cup \mathcal{S}_{neg}^{Inf,k+2,\mathcal{C}}$ we have $|P_2^{k+1}| = 0$. Thus, $|P_2^{k+1}|$ is in the claimed range.
In the following we can hence assume that $C^k \neq \square$. At first we assume that a minimal proof P_2^{k+1} of the inconsistency of $\mathcal{C} \cup \mathcal{S}_{neg}^{Inf,k+2,\mathcal{C}}$ uses a clause which can be derived from such a clause C^k with one resolution step. We again distinguish between two cases:
On the one hand it might be that all subgoals of this clause C^k can be solved with axioms immediately. Then: If a subgoal $sg \in C^k$ can be closed that has no brother, i.e. no other subgoals introduced by the extension that lead to sg are in C^k, we obtain by closing sg with an axiom the clause C^{k+1} being an element of $\mathcal{S}_{neg}^{Inf,k+2,\mathcal{C}}$. C^{k+1} allows for a minimal superposition proof P_2^{k+1} which needs only $|P_2^k| - 1$ inferences. Since $|P_1| \dot{-} k \leq |P_2^k|$, also $|P_1| \dot{-} (k+1) \leq |P_2^{k+1}|$. Moreover, since $|P_2^k| \leq |P_1| \dot{-} \frac{k}{max_{neg}(\mathcal{C})}$, also $|P_2^{k+1}| = |P_2^k| - 1 \leq |P_1| \dot{-} \frac{k+1}{max_{neg}(\mathcal{C})}$. Otherwise, if all subgoals of C^k have brothers, for all minimal proofs P_2^{k+1} we have $|P_2^{k+1}| = |P_2^k|$. Then, obviously $|P_1| \dot{-} (k+1) \leq |P_2^{k+1}|$. When considering the upper bound we obtain: If $|P_2^k| < |P_1| \dot{-} \frac{k}{max_{neg}(\mathcal{C})}$, then $|P_2^{k+1}| = |P_2^k| \leq |P_1| \dot{-} \frac{k+1}{max_{neg}(\mathcal{C})}$. If $|P_2^k| = |P_1| \dot{-} \frac{k}{max_{neg}(\mathcal{C})}$ and $max_{neg} \nmid (k+1)$, $|P_2^{k+1}| = |P_2^k| = |P_1| \dot{-} \frac{k+1}{max_{neg}(\mathcal{C})}$. $|P_2^k| = |P_1| \dot{-} \frac{k}{max_{neg}(\mathcal{C})}$ and $max_{neg} | (k+1)$ is impossible because in this case in C^k

there must be a subgoal having no brother.

On the other hand, there might be a subgoal in C^k that can be extended. Such an extension leads to a clause $C^{k+1} \in \mathcal{S}_{neg}^{Inf,k+2,\mathcal{C}}$ that is part of a minimal proof P_2^{k+1} with $|P_2^{k+1}| = |P_2^k| - 1$. Then, since $|P_1| \dot{-} k \leq |P_2^k|$, also $|P_1| \dot{-} (k+1) \leq |P_2^{k+1}|$. Moreover, since $|P_2^k| \leq |P_1| \dot{-} \frac{k}{max_{neg}(\mathcal{C})}$, also $|P_2^{k+1}| = |P_2^k| - 1 \leq |P_1| \dot{-} \frac{k+1}{max_{neg}(\mathcal{C})}$.

Now, we assume that a minimal proof P_2^{k+1} uses a subgoal clause which is not derived from a clause C^k. Then, no proof involving a subgoal clause derived from a clause C^k has a proof length smaller than $|P_2^k|$. Hence, we know that $|P_2^{k+1}| = |P_2^k|$ (because one resolution step can only save one hyper-resolution step). Moreover, we know that all subgoals of clauses C^k can be closed with axioms immediately and that all these subgoals have brothers (otherwise minimal proofs must only contain subgoal clauses derived with one resolution step from a C^k). By using the same arguments as above we can see that $|P_2^{k+1}|$ is within the claimed limits. □

Nevertheless, in analogy to before also in this context the proof search need not profit from the proof length reduction obtained. But our experiments have shown that often also proof search reductions are possible.

In the case that equality is involved in our proof problems, we obtain an analogous theorem as before:

Theorem 3.5 *For each resource $k > 1$ there is a set of unit equations $\mathcal{C}$ where the minimal superposition refutation proofs of $\mathcal{C} \cup \mathcal{S}_{neg}^{Inf,k,\mathcal{C}}$ are not shorter than minimal proofs for $\mathcal{C}$.*

Proof: In analogy to theorem 3.2. □

3.3 Relevancy criteria for subgoal clauses

Finally, we want to discuss in which cases the integration of subgoal clauses into the search state of a superposition-based prover promises a strong gain of efficiency:

Firstly, it is important that some of the subgoal clauses can be proven quite easily, especially more easily than the original goal(s). In order to estimate this, it is necessary to judge whether they can *probably be solved* with the help of clauses of the initial clause set. E.g., measuring similarity between a goal and other clauses with the techniques developed in [DF94] may be well-suited for this estimation.

Secondly, a solution of a newly introduced subgoal clause *should not always entail a solution of an original goal within few steps of the prover.* If this were the case then the integration of new subgoal clauses would not promise much gain. Criteria in order to estimate this are: On the one hand, the transformation of an original goal clause into a subgoal clause by a ME prover should have been performed by using many inferences, i.e. k should be quite high. Then, it is possible that a solution of a new subgoal clause does not entail a solution of an original goal within few steps because the probability is rather high that a bottom-up prover cannot—due to its heuristic search—quickly reconstruct the inferences needed to infer the original goal. On the other hand, if there is a subgoal clause S_T and some of the tableau clauses of the tableau T have a high heuristic weight regarding the heuristic of the superposition-based prover, a high gain of efficiency may occur if the prover can prove S_T. This is due to the fact that inferences needed to infer the original goal may not be performed by the prover.

As we will see in the following, such criteria are helpful for a relevancy-based filtering of subgoal clauses.

4 Relevancy-Based Selection of Subgoal Clauses

Already when using small resources k the sets $\mathcal{S}^{Inf,k,\mathcal{C}}$ and $\mathcal{S}_{neg}^{Inf,k,\mathcal{C}}$ can become quite large. Thus, it is not sensible to integrate all subgoal clauses from $\mathcal{S}^{Inf,k,\mathcal{C}}$ or $\mathcal{S}_{neg}^{Inf,k,\mathcal{C}}$ into the search state of a superposition-based prover: Integrating too many clauses usually does not entail a favorable rearrangement of the search because the heuristic "gets lost" in the huge number of clauses which can be derived from many subgoal clauses. Hence, it is reasonable to develop techniques for filtering subgoal clauses that appear to entail a large gain of efficiency for a superposition prover if they can be proven. I.e. we are interested in filtering *relevant* subgoal clauses. Therefore, our approach is as follows: At first, we generate a *set of subgoal clause candidates* and then we select some subgoal clauses from this set. The chosen subgoal clauses are added to the search state of the bottom-up prover. We shall first introduce two techniques for generating a set of subgoal clause candidates. After that we shall deal with the selection of relevant subgoal clauses.

In order to generate a set of interesting subgoal clauses it is important that we employ a large resource for generating subgoal clauses. As we have already discussed, subgoal clauses that are generated with a small number of inferences do not promise much gain because a bottom-up prover may easily reconstruct the inferences needed to infer them. Thus, if the prover is able to prove the subgoal clause it can also prove the original goal clause with few inferences and we do not gain much efficiency. However, it is not possible to generate all subgoal clauses $\mathcal{S}^{Inf,k,\mathcal{C}}$ or $\mathcal{S}_{neg}^{Inf,k,\mathcal{C}}$ for a sufficiently large resource k as subgoal clause candidates because their large number entails too high costs for the generation and additional selection. Hence, we are only able to choose as a set of subgoal clause candidates a *subset* of $\mathcal{S}^{Inf,k,\mathcal{C}}$ or $\mathcal{S}_{neg}^{Inf,k,\mathcal{C}}$, k sufficiently large (see section 5).

Our first variant, an *inference-based* method, starts by generating subgoal clauses from $\mathcal{S}^{Inf,k,\mathcal{C}}$ or $\mathcal{S}_{neg}^{Inf,k,\mathcal{C}}$ for a rather large resource k and stops when N_{sg} subgoal clause candidates are generated. The advantage of this method is that it is very easy and can efficiently be implemented: Tableaux are enumerated with a fixed strategy for selecting subgoals for inferences (usually left-most/depth-first) and for each tableau its subgoal clause is stored. The main disadvantage of this method is that due to the fixed strategy and the limit of the number of subgoal clauses, we only obtain subgoal clauses which are inferred from goal clauses by expanding certain of their subgoals with a high number of inferences, other subgoals only with a small number of inferences. (See also Figure 1: Ovals are tableaux in a finite segment of the search tree $\mathcal{T}$, the lines represent the $\vdash$ relation. Grey ovals represent enumerated tableaux, i.e. their subgoal clauses are stored, white ovals represent tableaux which are not enumerated within N_{sg} inferences.) Thus, the method is somewhat unintelligent because no information

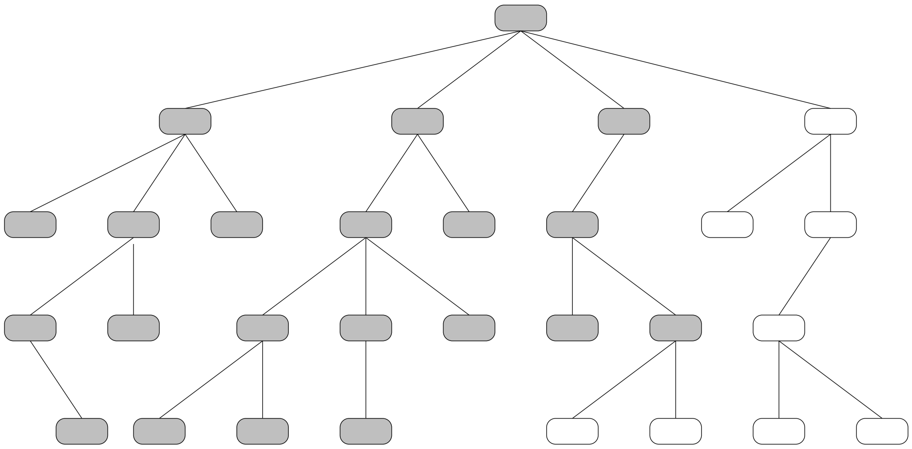

Figure 1: Inference-based generation of a set of subgoal clause candidates

about the quality of the transformation of an original goal clause into a subgoal clause is used. Certain transformations are favored against others only due to the uninformed subgoal selection strategy.

Our second variant, an *adaptive* method, tries to overcome the disadvantages of the first in the following way: Instead of allowing for more inferences when generating subgoal clauses due to an uninformed subgoal selection strategy, we want to allow for more inferences at certain *interesting positions* of the search tree $\mathcal{T}^{\mathcal{C}}$ for a given set of clauses $\mathcal{C}$.

In detail, our approach is as follows: At first, we generate all subgoal clauses $\mathcal{S}^{Inf,k_1,\mathcal{C}}$ or $\mathcal{S}^{Inf,k_1,\mathcal{C}}_{neg}$ with a resource k_1 which is small compared to the first variant. Then, a fixed number N_{ref} of subgoal clauses is chosen which promise the highest gain of efficiency regarding the criteria from section 3. More exactly, we choose subgoal clauses which are maximal w.r.t. a selection function function ψ. A possible realization of ψ is the following:

$$\begin{aligned}\psi(S_T) = \alpha_1 \cdot I(S_T) + \alpha_2 \cdot \max\{\mathcal{H}(C) : C \text{ is a tableau clause in } T\} \\ + \alpha_3 \cdot \max\{sim(S_T, C) : C \in \mathcal{C}, |C| = 1\}\end{aligned}$$

The higher the number of inferences $I(S_T)$ is which are needed to infer S_T the higher $\psi(S_T)$ should be. Hence, α_1 should be positive. Also a rating $\alpha_2 > 0$ is sensible. If there are tableau clauses in T which have a high heuristic weight regarding the heuristic $\mathcal{H}$ of the superposition-based prover we can perhaps gain a lot of efficiency. The function *sim* measures whether literals from S_T could probably be solved with unit clauses from $\mathcal{C}$. We utilized a variant of the function *occnest* which is defined in [DF94] for accomplishing the task. We set $\alpha_1 > \alpha_2 > \alpha_3 \geq 0$ due to the increasing vagueness of the criteria.

Now, let $M^{N_{ref}} \subseteq \mathcal{S}^{Inf,k_1,\mathcal{C}}$ or $M^{N_{ref}} \subseteq \mathcal{S}^{Inf,k_1,\mathcal{C}}_{neg}$ be the set of chosen subgoal clauses. Then, we generate again with a resource k_2 subgoal clauses but employ as start clauses for the tableau (subgoal clause) enumeration the clauses from $M^{N_{ref}}$. We call the set of subgoal clauses generated with this method $\mathcal{S}^{Inf,k_2,\mathcal{C},M^{N_{ref}}}$. (Consider also Figure 2: The dotted line shows which subgoal clauses are generated with resource k_1. Then

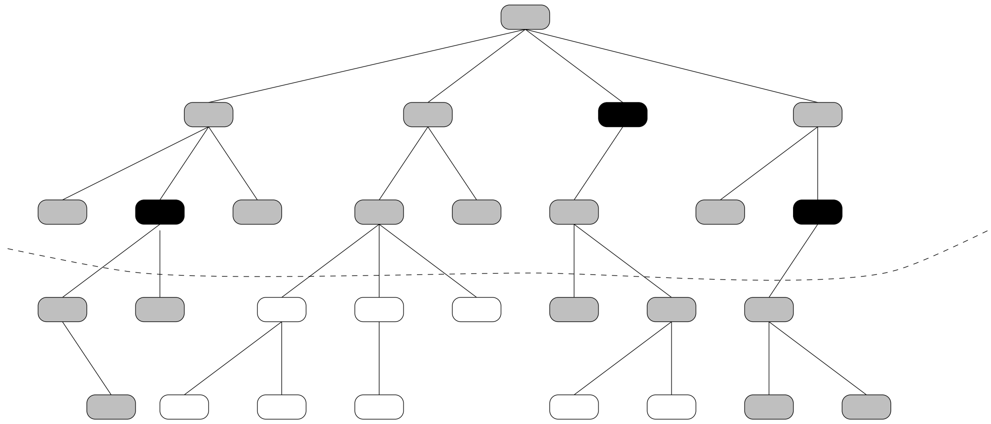

Figure 2: Adaptive generation of a set of subgoal clause candidates

some of them are selected (black ovals) and used as starting points for the generation of new subgoal clauses (grey ovals) with resource k_2.) The resource k_2 should again be not too high in order to allow for a fast enumeration of the subgoal clauses. The set of subgoal clause candidates is then given by $\mathcal{S}^{Inf,k_1,\mathcal{C}} \cup \mathcal{S}^{Inf,k_2,\mathcal{C},M^{N_{ref}}}$ (if we employ CTC) or by $\mathcal{S}_{neg}^{Inf,k_1,\mathcal{C}} \cup \mathcal{S}^{Inf,k_2,\mathcal{C},M^{N_{ref}}}$ (if we employ CTC_{neg}). Thus, subgoal clause candidates are on the one hand all subgoal clauses derived with a certain number of inferences which may—at least if no equality is involved in the problem—guarantee a reduction of the proof length. On the other hand, we have some subgoal clause candidates derived with a higher number of inferences, at most $k_1 + k_2$. These subgoal clauses promise a high gain of efficiency because they are derived from subgoal clauses selected with function ψ, i.e. they are derived from clauses which promise a high gain of efficiency for a superposition-based prover.

For selecting subgoal clauses from the set of subgoal clause candidates we employed function φ—which is mainly based on the function ψ—and selected clauses with the highest weight regarding φ. In detail, φ is defined by $\varphi(S_T) = \psi(S_T) - \theta(S_T)$. θ simply counts a weighted sum of the number of variables in S_T and two times the number of function or predicate symbols in S_T. Hence, quite "general" subgoal clauses are preferred. This is sensible because they can more often be used in inferences of a bottom-up prover.

5 Experimental Results

In order to conduct an experimental evaluation of our concept of integrating top-down/bottom-up provers by cooperation, we coupled two renowned provers: the ME prover SETHEO and the superposition prover SPASS. Each prover runs on an own processor and obtains the initial clause set $\mathcal{C}$ as input. Because of the fact that when tackling simple problems it is unnecessary to let the provers cooperate, each prover tries to solve the problem independently with a timeout of 4 seconds. If no prover could solve the problem, the top-down prover generates subgoal clauses with one of the two variants. Note that in our context this does not require changes in the top-down prover but can be performed with built-ins of the PROLOG-style input language of SETHEO. Note further that SETHEO employs CTC_{neg}. Thus, we experimented only with subgoal clauses obtained with negative start clauses. Then, these subgoal clauses are filtered, transferred to the bottom-up prover, and integrated into its search state. Finally, the provers proceed to tackle the problem in parallel.

Hence, we can efficiently solve simple problems. Moreover, cooperation can be performed for harder problems after the first timeout. Note that during this cooperation phase it is also possible to add some clauses that the bottom-up prover has generated to the axiomatization of the ME prover. Thus, we can achieve cooperation by exchanging lemmas and subgoal clauses without one concept disturbing the other.

We experimented in the light of problems from the well-known problem library TPTP ([SSY94]). In order to obtain a reliable collection of data, we employed two domains of TPTP as our test set, the domains CAT (category theory) and LDA (LD-algebras). The CAT domain consists of 58 problems, the LDA domain of 22. From these domains we extracted 22 and 15 hard problems, respectively, i.e. problems none of the provers can solve within 4 seconds. Note that the problems in both domains contain equality as well as non-Horn clauses. The subgoal clause candidates were generated in the following way: For variant 1 we employed the resource $k = 10$ which performed best in the experiments. The use of higher resources did not entail better results. We limited the set of subgoal clauses by $N_{sg} = 500$. For variant 2 we employed as resources $k_1 = k_2 = 9$. These resources allowed for the efficient generation of all subgoal clauses within the initial segments of the search tree. Usually with this method at most 500 subgoal clauses were generated, i.e. about the same number as when employing variant 1. As start clauses for an adaptive refinement we selected $N_{ref} = 5$ subgoal clauses.

Finally, 100 clauses were selected from the set of candidates and transmitted to SPASS. Table 1 presents results of our experiments (we omitted 7 problems from CAT which could not be solved by any of the alternatives from table 1).

Column 1 of each part of the table displays the name of the problem. Columns 2 and 3 present the runtimes of SPASS and SETHEO (on a SPARCstation 20) when working alone, columns 4 and 5 the runtimes of SPASS when it obtains subgoal clauses from SETHEO which are generated regarding variants 1 and 2, respectively. Note that the runtimes include the 4 seconds before the cooperation, the selection of subgoal clauses, and the transmission to SPASS. The entry "–" means that the problem could not be solved within 1000 seconds.

The results reveal the high potential of our approach to significantly improve on single provers. However, when considering the results of variant 1, they also show that a naive and uninformed generation of subgoal clauses usually does not entail much gain. In the LDA domain, we can solve 9 of 15 hard problems by using variant 2 for subgoal generation, whereas SPASS can only solve 5, SETHEO only 2. A simple competitive prover which employs SPASS and SETHEO in parallel would also only be able to solve 6 problems. Hence, cooperation is really important in order to increase the success rate. We can also in almost all cases significantly decrease the runtimes. E.g. problems `LDA005-2` and `LDA006-2` which require a runtime of more than 4 minutes when using SPASS can be proven in a few seconds. In the CAT domain the results are similar.

Table 1: Integration of top-down/bottom-up approaches by cooperative provers

problem	SPASS	SETHEO	inf.-bas.	adapt.	problem	SPASS	SETHEO	inf.-bas.	adapt.
LDA004-1	–	–	–	–	CAT001-1	–	–	–	–
LDA005-1	–	–	–	–	CAT001-3	134s	32s	6s	6s
LDA005-2	279s	–	265s	8s	CAT001-4	33s	11s	5s	5s
LDA006-1	–	–	–	–	CAT002-2	–	–	–	–
LDA006-2	276s	–	304s	10s	CAT003-1	–	–	–	–
LDA007-1	16s	366s	19s	21s	CAT004-3	–	23s	–	9s
LDA007-2	–	50s	7s	7s	CAT008-1	91s	126s	6s	6s
LDA008-1	–	–	–	–	CAT009-1	–	–	10s	10s
LDA008-2	–	–	–	–	CAT009-3	–	–	–	29s
LDA009-1	–	–	–	–	CAT009-4	53s	–	47s	50s
LDA009-2	–	–	–	24s	CAT010-1	–	–	11s	9s
LDA010-1	–	–	–	9s	CAT011-3	17s	–	12s	12s
LDA010-2	–	–	–	26s	CAT014-3	18s	–	11s	11s
LDA011-1	54s	–	58s	9s	CAT018-3	–	–	–	74s
LDA011-2	21s	–	35s	7s	CAT019-4	–	–	–	–

By employing subgoal clauses generated regarding variant 2 we can solve 11 of 22 hard problems. SPASS and SETHEO are only able to solve 6 and 4, respectively. In the most cases the runtimes could be decreased.

6 Discussion and Future Work

Integration of top-down and bottom-up provers by employing cooperation is very promising in the field of automated deduction. Due to certain strengths and weaknesses of provers following different paradigms, techniques that try to combine the strengths by cooperation can allow for an improvement of the deductive system. Our approach of combining top-down and bottom-up provers by processing top-down generated subgoal clauses in a bottom-up prover achieves this combination by introducing goal-orientation into a bottom-up prover thus combining strong redundancy control mechanisms and goal-directed search.

So far, related approaches mainly aimed at supporting a top-down prover by a bottom-up based lemma component. Results presented, e.g., in [Sch94] or [Fuc97], reveal that also these approaches are well-suited. However, in some domains, especially if equality is involved, superposition-based provers clearly outperform ME provers. Thus, in such domains it may be more sensible to develop techniques in order to support the more powerful bottom-up prover than the weaker top-down prover. Transmitting information from a top-down to a bottom-up prover was so far—to our knowledge—only discussed in [Sut92]. However, there lemmas generated by a ME prover were transferred to resolution-based provers and the results were not satisfactory.

Future work should deal with an extension of the empiric investigation. It would be interesting to detect in which kinds of domains (Horn/non-Horn, equality/no equality) the approach is especially well-suited. Moreover, it might be interesting to develop more complex methods for generating subgoal clause candidates. The results from section 5 suggest that an even more intelligent generation and selection of subgoal clauses leads to further improvements.

References

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

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

[BG94] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[BS97] F. Baader and K.U. Schulz(Eds.). *Applied Logic Series 3: Frontiers of Combining Systems.* Kluwer Academic Publishers, 1997.

[DF94] J. Denzinger and M. Fuchs. Goal oriented equational theorem proving. In *Proc. 18th KI-94*, pages 343–354, Saarbrücken, 1994. LNAI 861.

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

[Fuc97] M. Fuchs. Similarity-Based Lemma Generation for Model Elimination. In *Proc. Int. Workshop on First-Order Theorem Proving (FTP'97)*, pages 63–67, Linz, Austria, 1997.

[Kor85] Richard E. Korf. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *AI*, 27:97 – 109, 1985. Elsevier Publishers B.V. (North-Holland).

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

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

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

[McC94] W. McCune. Otter 3.0 reference manual and guide. Technical Report ANL-94/6, Argonne National Laboratory, Argonne, 1994.

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

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

[SSY94] G. Sutcliffe, C.B. Suttner, and T. Yemenis. The TPTP Problem Library. In *CADE-12*, pages 252–266, Nancy, 1994. LNAI 814.

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

[Sut92] G. Sutcliffe. A heterogeneous parallel deduction system. In *Proc. FGCS'92 Workshop W3*, 1992.

[WGR96] C. Weidenbach, B. Gaede, and G. Rock. Spass & Flotter Version 0.42. In *Proc. CADE-13*, pages 141–145, New Brunswick, 1996. LNAI 1104.