Lemma Generation for Model Elimination by Combining Top-Down and Bottom-Up Inference

Marc Fuchs
Fakultat fur Informatik
TU Miinchen
80290 Miinchen, Germany
fuchsm@informatik.tu-muenchen.de

Abstract

A very promising approach for integrating top-down and bottom-up proof search is the use of bottom-up generated lemmas in top-down provers. When generating lemmas, however) the currently used lemma generation procedures suffer from the well-known problems of forward reasoning methods, e.g., the proof goal is ignored. In order to overcome these problems we propose two relevancy-based lemma generation methods for top-down provers. The first approach employs a bottom-up level saturation procedure controlled by top-down generated patterns which represent promising subgoals. The second approach uses evolutionary search and provides a self-adaptive control of lemma generation and goal decomposition.

1 Introduction

Top-down and bottom-up approaches for automated theorem proving in first-order logic each have specific advantages and disadvantages. Top-down approaches (like model elimination (ME) [9] or the connection tableau calculus (CTC) [8]) are goal oriented but suffer from long proof lengths and the lack of an effective redundancy control. Bottom-up approaches (like superposition [2]) provide more simplification power but lack in their purest form any kind of goal orientation. Thus, an integration of these two paradigms is desirable.

Two approaches have been in the focus of interest in the last years. The methods from [14; 10] are bottom-up theorem proving approaches. There, the bottom-up inferences are restricted to the use of *relevant clauses* which are detected by additional top-down computations. Thus, goal orientation is combined with redundancy control. In other approaches top-down provers are assisted by lemmas ([12; 1; 5]). Also these methods combine goal orientation with redundancy control provided by the lemmas. The use of additional clauses can also reduce the proof length which may lead to large search reductions. But this has to be paid for by an increase of the branching rate of the search space. Thus, mechanisms for selecting relevant clauses are needed.

Our integration approach is based on [12; 5] where lemmas have been used in the CTC. There, in order to refute a set of *input clauses* with the CTC, in a preprocessing phase the input clauses are augmented by bottom-up generated clauses (lemmas). Then, the prover tries to refute this augmented clause set. Lemmas are obtained by creating a pool of possible lemmas which are able to shorten the proof length (generation *step).* Then, in the *selection step* some possibly relevant lemmas are selected which are then used for refuting the given proof task. In [5] it is concentrated on the selection of lemmas and rather simple lemmas have successfully been used. In order to speed-up the proof search in a more effective manner harder lemmas are needed. The generation approaches as used in [12; 5], however, have severe difficulties in generating harder lemmas in a controlled way. They suffer from the earlier mentioned problems of saturation based proving.

Thus, we focus now on the aspect of the generation of possible lemmas and propose two new approaches for a *controlled generation of lemmas for top-down provers* based on the combination of top-down and bottom-up search. The first technique generates lemmas in a systematic way based on some kind of level saturation (as in [12; 5]). The lemma generation, however, is combined with a decomposition of generalized proof goals which represent possibly solvable subgoals in a compact way. These generalized goals are used for detecting possibly irrelevant lemmas. The other approach uses *genetic programming* [7]. In the evolution process simultaneously top-down and bottom-up inferences are performed which are controlled by a fitness function which measures the similarity between open subgoals and derived lemmas. Thus, lemma generation and goal decomposition are focused on promising clauses in a self-adaptive way. We evaluate the usefulness of the new methods at hand of experiments performed with the prover SETHEO [11].

2 Connection Tableau Calculus

In order to refute a set C of clauses the CTC works on *connected (clause) tableaux* for C (see [8]). The inference rules are *start, extension,* and *reduction.* The start rule allows a so-called tableau expansion that can only be applied to a trivial tableau, i.e., one consist* ing of only one node. An expansion step means selecting a variant of a clause from C and attaching for each of its literals a node (labeled with the respective literal) to a leaf node of an *open* branch, i.e., a branch that does not contain two complementary literals. The start rule can be restricted to some *start clauses,* e.g., the set of negative clauses may be used (see [11]). Tableau reduction closes a branch by unifying *h subgoal $* (the literal at the leaf of the open branch) with the complement of a literal r (denoted by $\sim r$) on the same branch, and applying the substitution to the whole tableau. For defining extension we need the notion of a contrapositive. If $C = l_1 \vee \ldots \vee l_n$ is a clause then each sequence $l_i \leftarrow \sim l_1, \ldots, \sim l_{i-1}, \sim l_{i+1}, \ldots, \sim l_n$, $1 \leq i \leq n$, is a *contrapositive* of *C* with *head* l_i and *tail* $\sim l_1, \ldots, \sim l_{i-1}, \sim l_{i+1}, \ldots, \sim l_n$. Extension with a contrapositive $h \leftarrow t_1, \ldots, t_n$ of a clause from *C* is performed by selecting a subgoal 0, unifying *s* and $\sim h$ with σ, instantiating the tableau with σ, attaching $h\sigma, \sim t_1\sigma, \ldots, \sim t_n\sigma$ below $s\sigma$, and closing the branch which ends with $h\sigma$.

We say $T \vdash_C T'$ if and only if tableau *V* can be derived from *T* by applying a start rule (if *T* is the trivial tableau) or an extension/reduction rule to a subgoal in *T.* In order to refute an inconsistent clause set C, a search tree has to be examined in a *fair* way (each tree node must finally be visited) until a closed tableau occurs. A *search tree* $\mathcal{T}^C$ defined by a set of clauses *C* is a tree, whose root is labeled with the trivial tableau. Each node in $\mathcal{T}^C$ labeled with tableau *T* has as immediate successors the maximal set of nodes $\{v_1, \ldots, v_n\}$, where v_i is labeled with Ti and $T \vdash_C T_i$, $1 \leq i \leq n$.

In order to enumerate a search tree implicit enumeration procedures are normally in use that apply *iterative deepening search* with backtracking. Iteratively increasing finite initial parts of the search tree are explored in depth-first search (cp. [13]). For instance, for clause sets $\mathcal{D}$ and $\mathcal{S}$, $\mathcal{T}^{\mathcal{D},\mathcal{S},n}$ denotes the finite initial part of $\mathcal{T}^{\mathcal{D} \cup \mathcal{S}}$ where all tableaux are obtained by using only clauses from *S* for the start expansion, only the clauses from $\mathcal{D}$ for extensions, and where the tree depth of each tableau does not exceed a value of n $\in$ N. (The root node has depth 0, its successor nodes depth 1, and so on).

3 Goal Decomposition and Saturation

We provide some basic notions regarding the decomposing and saturating capabilities of the CTC which will be used in the following. We start by demonstrating how to extract *query* and *lemma* clauses from a given connection tableau. At first we introduce a method for extracting valid clauses from a connection tableau.

Definition 3.1 (subgoal clause) Let *C* be a set of clauses. Let *T* be a connection tableau for *C.* Let $s_1, \ldots, s_n$, $n \geq 1$, be the subgoals of T Then we call $s_1 \vee \ldots \vee s_n$ the *subgool clause* of T.

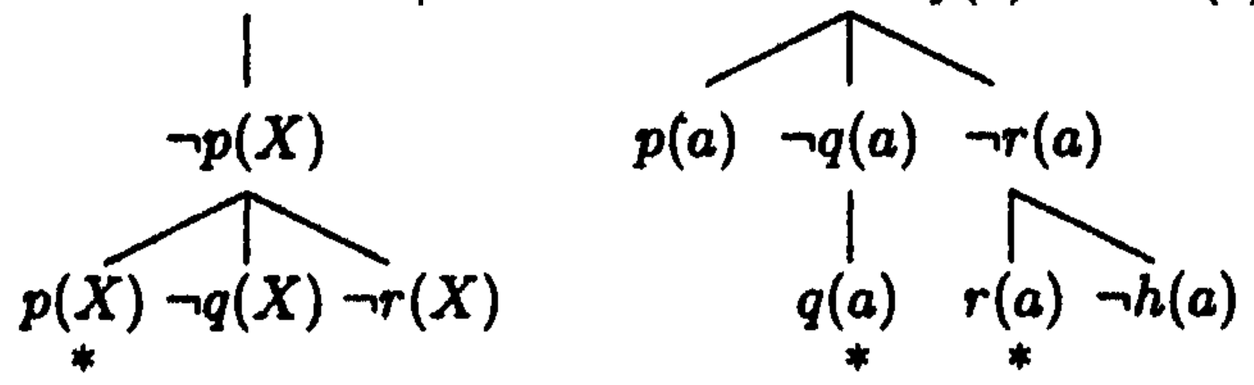
The subgoal clause of a connection tableau *T* for a clause set *C* is a logical consequence of *C* (see e.g., [8]). Subgoal clauses may be considered to be top-down generated queries or bottom-up generated lemmas depending on the form of the tableaux they are derived from. First, we consider the analytic character of subgoal clauses and define query clauses as follows (see also (3)).

Definition 3.2 (query tableau, query clause) Let *C* be a set of clauses. Let *T* be a connection tableau for C. Let $S \subsetneq \mathcal{C}$ be a set of start clauses. Let 5 be the clause below the unlabeled root of *T.* If 5 is an instance of a clause from *S* we call *T* a *query tableau* (w.r.t. *S)* and the subgoal clause of T a *query clause* (w.r.t. 5).

Essentially CTC based proof procedures implicitly enumerate query clauses w.r.t. the chosen start clauses $\mathcal{S} \subsetneq \mathcal{C}$ until the empty query clause is derived. Lemmas introduce a bottom-up element into the top-down oriented CTC. We employ the following definition of a lemma which extends and generalizes the notions of lemmas used in [12; 1; 5].

Definition 3.3 (lemma tableau, lemma clause) Let *C* be a clause set. Let *T* be a connection tableau for *C.* Let $C = s_1 \vee \ldots \vee s_n$ be the subgoal clause of *T.* Let $\mathcal{H}$ be the set of subgoals which are immediate successors of the root. If $\mathcal{H} \neq \emptyset$ we call *T* a *lemma tableau.* Then, let si, 1 $\leq i \leq n$, be the element of $\mathcal{H}$ which is left-most in T. We call the contrapositive $s_i \leftarrow \sim s_1, \ldots, \sim s_{i-1}, \sim s_{i+1}, \ldots, \sim s_n$ of *C* the *lemma clause* of *T.*

Example 3.1 Let $\mathcal{C} = \{\neg p(X), p(X) \vee \neg q(X) \vee \neg r(X), r(X) \vee \neg h(X), q(a)\}$. Let S $\{\neg p(X)\}$ be the set of start clauses. The left tableau is a query tableau representing the query $\neg q(X) \vee \neg r(X)$ (w.r.t. 5), the right tableau is no query tableau but a lemma tableau which represents the lemma $p(a) \leftarrow h(a)$.



A lemma application transforms a subgoal into a (possibly empty) set of new subgoals. An application of a lemma L to a subgoal * can be viewed as attaching the instantiated lemma tableau of *L* below *s.* Thus, it works as a macro operator in the search space. The use of a bottom-up created lemma can close a subgoal by an extension with the lemma and performing reduction steps into the introduced subgoals (tail literals) and thus reduces the proof length. In the following we will always employ such a purely bottom-up oriented view on lemmas, i.e., they replace deductions at the "tableau front". Extension steps to instantiated tail literals of lemmas are forbidden. This provides a controlled use of lemmas and prevents a nesting of lemma applications when dealing with non-unit lemmas.

4 Pattern Controlled Level Saturation

Our first method for generating lemmas is based on a combined *systematic* goal decomposition and lemma saturation. The basic idea is to employ iterative top-down and bottom-up generation procedures which produce in iteration step (level) $i > 1$ all query and lemma tableaux of depth t by the decomposition or saturation of the queries and lemmas of the previous level, respectively. Initial queries and lemmas (created in step 1) are the start clauses and the contrapoeitives of the input clauses, respectively. Then, after each iteration step subsumed tableaux are deleted (a notion of tableau subsumption can be found, e.g., in [8]). Thus, a proof of depth *d* can be obtained in $\lceil \frac{d}{2} \rceil$ top-down and bottom-up iteration steps. It is obtained by closing a query tableau generated in step $\lceil \frac{d}{2} \rceil$ using bottom-up lemmas (by extension of the query literals with lemmas and closing the introduced subgoals with reductions) which have been created in the steps $1, \ldots, \lceil \frac{d}{2} \rceil$.

This combined bottom-up and top-down proof search has several theoretical advantages compared to a pure bottom-up or top-down search. The top-down search is improved by bottom-up processing which avoids the recomputation of solutions for multiple occurring subgoals in query clauses. Moreover, the method improves on a pure bottom-up computation because it is more goal oriented and thus the production of a large number of irrelevant clauses may be avoided.

In practice, however, such an approach does not appear to be reasonable (for "harder problems"). An explicit storage of all generated tableaux is not sensible when dealing with ME based provers because of the huge increase of the number and size of the generated tableaux. Thus, we have to focus only on some few relevant query tableaux (or query clauses when dealing with Horn problems) and lemmas which are maintained after each level for further decomposition or saturation in the next iteration, respectively. Heuristic selection criteria for query tableaux and lemma clauses are needed. When using such normally fuzzy criteria, however, it is not guaranteed any longer that a query tableau which can be closed with lemmas can be produced after $\lceil \frac{d}{2} \rceil$ iterations. It is probable that useful queries or lemmas are discarded such that more than $\lceil \frac{d}{2} \rceil$ iterations are needed. Then, the process may be more costly than conventional top-down or bottom-up deduction. Because of the deletion of query tableaux and lemmas it is even possible that no proof can be found by clc ing a maintained query tableau with derived lemma clauses.

Thus, we employ a slightly different (lemma oriented) method which we will explain only for *Horn clauses and unit lemmas* for simplicity reasons. Instead of employing a complete top-down enumeration of all query clauses we work in an abstracted top-down search space. Literals of specific query clauses are generalized to so-called *patterns.* Patterns are literals which cover the form of several subgoals. Specifically, we try to guarantee that subgoals occurring in a proof are subsumed by some patterns. Patterns are created in d -1 steps. As initial patterns, in step 1, the literals occurring in start clauses are created. Then, in each step $i >$ 1 we successively decompose patterns of the previous step t -1 into new subgoals and generalize then the subgoals to new patterns. Thus, patterns created in step i generalize subgoals of depth i which occur in query clauses. These top-down generated patterns cannot be used for finishing a proof task (with the help of lemmas). However, they provide relevancy criteria for lemmas. If a lemma is not unifiable with the complement of a specific pattern it can be discarded. Thus, we can work with a conventional iterated lemma generation procedure whose maintenance criteria for lemmas are assisted by top-down inferences. Finally, the lemmas are used in a top-down proof run for refuting the input clauses.

We make our method more concrete. We start with the top-down goal decomposition. First, we show how to generalize literals occurring in a set $\mathcal{Q}$ of subgoal clauses to patterns. We use as patterns *N* $\in$ *N* literals from the set $Lit_{\leq L}$ for *L* $\in$. $Lit_{<L}$ is the set of all literals where each literal has a length {no. of symbols) $l \leq$, and cannot be specialized to a literal with length $l' > l$ and $l' \leq L$. Patterns should generalize the subgoals of clauses from *Q* which are the most likely to occur in a proof. In order to determine the literals to be used as patterns we employ a function *QUOIQ* on literals. This function is based on a notion of quality *qual*$_{9g}$ on subgoals. *qual*$_{9g}$ is used to estimate whether a subgoal may occur in a proof. Then, qualQ expresses how many subgoals from $\mathcal{Q}$ of a high quality are generalized by a pattern.

qual$_{9g}$*(s)* of a subgoal *$* is a value which represents the "generality" of *s* since we assume that more general subgoals can more easily be solved. For instance, small subgoals with many variables may get large values by *quals*$_9$ (cp. [5]). *qualQ* is defined using *qual*$_{9g}$ as follows.

Definition 4.1 (pattern quality) For a literal *I* let *Inst(l)* be the multi-set $\{s : C_1 \vee s \vee C_2 \in \mathcal{Q}, \exists\mu : l\mu = s\}$ We define the pattern quality *qualQ(l)* of a literal / w.r.t. $\mathcal{Q}$ by $qual_{\mathcal{Q}}(l) = \sum_{s \in Inst(l)} qual_{sg}(s)$.

The best *N* literals from *Lit*$_{\leq L}$ w.r.t. pual*Q* form the set *pati,s(Q)* of patterns for Q. Patterns which generalize subgoals which are part of a proof provide an exact criterion for discarding irrelevant lemmas. A pattern must be unifiable with the complement of a lemma. *L* and *N* are responsible for providing a compromise between a large pruning effect of the patterns on the number of generated lemmas *(L* large, *N* small) or a high probability that no useful lemmas are discarded (L small, *N* large). We employ a query generation algorithm which gets as input a clause set C, start clauses 5, and an iteration number $I \geq 1$. As output the sets of patterns $\mathcal{P}_1, \ldots, \mathcal{P}_I$ are delivered.

Procedure 4.1 (query generation)

```
1. $\mathcal{P}_1 := \{l : C_1 \vee l \vee C_2 \in \mathcal{S}\}$.
2. for $i := 2$ to $I$ do:
    (a) let Q be the set of the most general query
        clauses of query tableaux from $\mathcal{T}^{C,\mathcal{P}_{i-1},2}$ which
        are not part of $\mathcal{T}^{C,\mathcal{P}_{i-1},1}$.
    (b) $\mathcal{P}_i := pat_{L,N}(\mathcal{Q})$.
```

The lemma generation algorithm enumerates lemma tableaux in a similar way as in [12; 5] but additionally uses the generated patterns. It can be applied after the generation of the query pattern sets $\mathcal{P}_1,\ldots,\mathcal{P}_I$. A further input of the algorithm is again an iteration number $J \leq I$ and the set of input clauses C. As output a lemma set $\mathcal{L}$ is delivered.

Procedure 4.2 (lemma generation)

```
1. $\mathcal{L} := \{l \in C : |l| = 1,\ \exists\mu : l\mu = \sim s\mu, s \in \bigcup_{j=1}^{I} \mathcal{P}_j\}$.
2. $\mathcal{M}_1 := \{l \in C : |l| = 1\}$.
3. for $i := 2$ to $J$ do:
    (a) let $\mathcal{M}_i$ be the set of the most general lemma clauses of lemma tableaux from $\mathcal{T}^{C \cup \mathcal{L}, C, 2}$ which are not in $\bigcup_{j=1}^{i-1} \mathcal{M}_j$.
    (b) $\mathcal{L} := \{l \in \mathcal{M}_i : |l| = 1,\ \exists\mu : l\mu = \sim s\mu,$
        $s \in \bigcup_{j=1}^{I-i+1} \mathcal{P}_j\} \cup \mathcal{L}$.
```

The chance for easy lemmas to be maintained is higher than for hard lemmas since more patterns are used. This is because easy lemmas may be applicable in more depth levels of a proof. The pattern-based criterion discards lemmas immediately after their generation and thus saves space. The set $\mathcal{L}$ forms the set of possible lemmas which may finally be used in the proof run in addition to the input clauses. We consider a closed tableau for a Horn clause set of a depth of d. If $I \geq d - 1$ and the patterns from $\mathcal{P}_i$ cover the form of the subgoals with depth *i* which are needed to find the closed tableau, the lemma generation method is complete. This means that it can be guaranteed that $\mathcal{L}$ contains all lemmas needed to reduce the proof depth by an amount of $J - 1$. Specifically, it is also possible to reduce the proof length. In practice after the execution of the generation procedure lemmas are selected from $\mathcal{L}$ with a selection function (see [5]). These lemmas are used in a final proof run.

5 Evolutionary Lemma Generation

The pattern-based method has the pleasant property that it provides a systematic generation of lemmas which can guarantee the generation of useful lemmas under certain conditions. A practical advantage is that highly efficient model-elimination provers can be employed for top-down as well as bottom-up inferences.

But if the choice of the patterns is not optimal the method works as a *local optimization method* because lemmas are discarded. Lemma tableaux whose derivations require the use of (small quality) lemmas which are discarded at a certain moment cannot be generated later. Thus, it is rather probable that the generation of useful and also well judged lemmas is prevented because the generation of such lemmas may require the use of other discarded clauses.

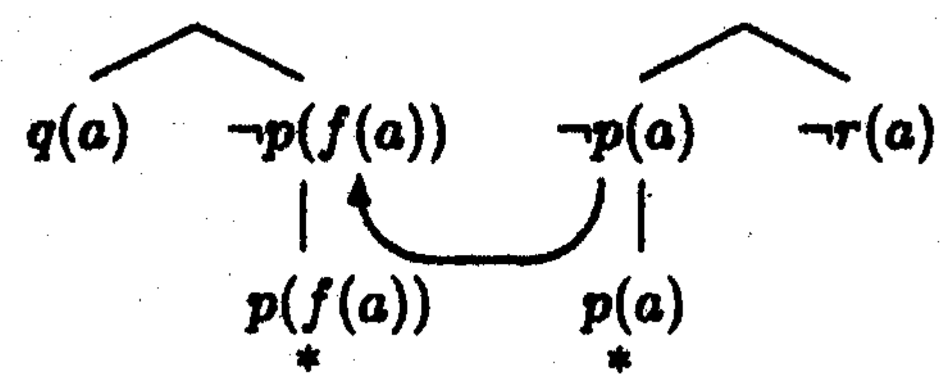
Our solution to this problem is the use of evolutionary techniques for lemma generation which are based on the genetic programming (GP) paradigm. For a detailed introduction to genetic algorithms or genetic programming we refer to [6] or [7], respectively. Our application of GP combines the evolution of query and lemma tableaux. The abstract principles of our method are as follows. An individual corresponds to a connection tableau which represents a lemma or a query clause. Thus, we work with (possibly) partial solutions (lemmas) of our initial problem and with goal decompositions which represent problems which are still open. The fitness of one lemma is given by its ability to solve or "almost" solve an open subproblem. A tableau which represents a query is the fitter the more subgoals are solvable (almost solvable) by lemmas. The genetic operators are based on the exchange of sub tableaux. Thus, good subdeductions which may be part of a proof are used in order to create new (and possibly fitter) individuals. Building blocks (subtableaux) of the fittest individuals persist with a high probability and can contribute to a generation of lemmas or query clauses which appear in a proof.

Thus, the lemma and query tableaux used for GP are used in a deductive sense by producing new lemma and query clauses in order to solve the original problem or at least to generate useful lemmas. Further, they play a role as control elements. The lemma production influences the query decomposition and vice versa. This is similar to our first pattern-based approach. But now also the top-down decomposition is influenced by some kind of distance to given valid lemmas. Hence the search is concentrated on "interesting" regions of the search space in a self-adaptive way. Furthermore, the probabilistic character of GP offers the chance to avoid a naive hill climbing based search and can produce needed lemmas although ancestors are judged by low fitness values.

The technical realization of these ideas is as follows. As already mentioned we use a fixed sized population of tableaux. Each tableau represents a query or a lemma. The population is initialized using the given input clauses to be refuted. Each query tableau obtainable by applying the start rule with a clause allowed as a start clause is added to the population. Analogously for each contrapositive of an input clause a lemma tableau is built. Additionally, it is possible to use some further selected lemma or query clauses in the initial population (see Section 6).

We employ three genetic operators, namely *reproduction* , and variants of *crossover* and *mutation.* Reproduction copies one element from the old population to the next. Our crossover operator differs from standard GP where two randomly chosen subtrees of two ancestor individuals are exchanged. Since such an operation would normally not result in a connection tableau crossover has to be constrained. One approach is to allow crossover at nodes v_1 and v_2 (labeled with l_1 and l_2) of two individuals T_1 and T_2, respectively, only if $\mu = mgu(l_1, l_2)$ exists. Then, an exchange of the subtrees could take place and the resulting tableaux are instantiated with μ. As appealing as this sounds it neglects the fact that a re-use of a subdeduction below v_1 in tableau T_1 may be possible below v_2 in T_2 although the above criterion is not applicable. This is because the subdeduction may be more general when viewed in an isolated way and is "over-instantiated" in T_1. Consider following example.

Example 5.1 Let $C \supseteq \{q(a) \vee \neg p(f(a)), p(f(X)), p(a), \neg p(X) \vee \neg r(X)\}$. The following figure shows two connection tableaux for C. The arrow (which is also called link) shows that the subdeduction below $\neg p(f(a))$ which represents a proof $p(f(X))$ can be used below the goal $\neg p(a)$ which would take the form $\neg p(Y)$ when deleting the subproof below the subgoal.



Thus, *asymmetric link relations* between tableaux can be built which show which subdeductions can replace others (see also [4]). Our crossover variant produces *one* new individual. Consistently with the link relation in a destination tableau a (possibly empty) subdeduction is replaced by a subdeduction in a *source tableau.* Then, the modified destination tableau is instantiated in an appropriate manner (more details can be found in [4]). In the above example the left and the right tableau serve as source and destination tableau, respectively. The tableau resulting from crossover represents the query $\neg r(f(X))$. The crossover operator can be viewed as a *generalized extension step* which allows us to attach subdeductions and not only clauses to (inner) nodes. We use a mutation operator which serves as a *generalized reduction step* (see [4]). It is needed in the non-Horn case to preserve the completeness of the genetic operators in order to create each useful lemma.

The genetic operators are applied to individuals chosen probabilistically proportionate to their fitness. We use a similarity measure between query and lemma tableaux for computing fitness. In the Horn case only the query and lemma clauses are considered. In the non-Horn case we may also consider open branches for judging the similarity. We use a similarity measure which considers certain syntactic properties of literals (cp. [4]).

The evolutionary search stops if a query tableau can be extended to a closed tableau using the lemma tableaux or a given maximal number of generations is reached. In the latter case a selection function (see [5]) chooses lemma clauses of the current population which are used in the final proof run. In summation the GP approach cannot guarantee that useful lemmas *are* generated during the search. But at least one can show that when fulfilling weak conditions each needed lemma can be created with a probability greater than 0 [4]. The self-adaptation car pabilities and randomized effects can allow the solution of problems which are out of reach of conventional search techniques (see Section 6).

6 Experimental Results

We want to analyze the performance of the newly developed lemma generation procedures. We have chosen the

Table 1: Experimental Results in the TPTP library

domain		SETHEO	SETHEO/PAT	SETHEO/OP
BOO	≤ 5	6	12	12
	≤ 10	10	14	14
	≤ 15	11	14	14
CAT	≤ 5	4	&	5
	≤ 10	3	5	5
	≤ 15	3	5	5
COL	≤ 5	10	29	27
	≤ 10	27	32	32
	≤ 15	28	3d	33
GRP	≤ 5	9	11	15
	≤ 10	9	11	15
	≤ 15	9	11	15
SET	≤ 5	36	48	50
	≤ 10	39	50	52
	≤ 15	41	50	54

high performance model elimination prover SETHEO for the final top-down proof run as well as for the pattern based lemma generation procedure.

As test set domains of the problem library TPTP v2.0.0 [15] are used. The domains BOO, CAT, COL, GRP, and SET have been chosen. BOO, COL, and GRP mostly contain Horn problems whereas in the other domains often non-Horn problems occur. We tackled only "hard problems". These problems cannot be solved with the conventional SETHEO system within 10 seconds. We have used a SUN Ultra 2 and a run time limit of 15 minutes for each problem. This includes the time for the lemma generation and the final refutation run.

In Table 1 one can find the performance of the newly developed systems in comparison with SETHEO. SETHEO is configured as described in [11]. Specifically, this includes the use of *folding-up* (see [8]) in the proof run. SETHEO/PAT generates lemmas based on our first method. The lemmas are then added to SETHEO. The final proof run is done with the same version of SETHEO which is used without lemmas. We restrict the lemma generation to unit lemmas which are sufficient to obtain good results in the considered domains. The lemma generation Procedure 4.2 is employed with iteration number J = 3. The iteration number for the pattern generation was set to J = 6. We used for the pattern length *L* and the pattern number TV the combinations (L, *N) m* (3,3) and *(L,N)* = (7,10). We depict for each example the best result which could be obtained with a configuration. The selection function is defined using the *lemma delaying method* as introduced in {5]. SETHEO/GP is based on genetic programming. We have initialized the evolutionary lemma generation in such a manner that the lemmas which are produced by the pattern based method with two bottom-up iteration steps are used in the initial population. Furthermore, we use some selected queries (see [4]). Thus, the procedure can start at an interesting point in the search space. Unit lemmas are selected from the final population using the lemma delaying method. We show the best results obtained in 5 runs for each problem. The exact configuration of our genetic algorithm can be found in [4].

In the table we have depicted the number of problems which can be solved by the considered approaches after 5,10, and 15 minutes. We can see that all lemma methods improve on the conventional SETHEO system in a stable way. Often lemma tableaux of a proof depth of 2, sometimes of a depth of 3 and 4, can be used in a proof. The use of these lemmas leads to a proof length reduction and significantly smaller search spaces which have to be traversed by the iterative deepening search procedure. In comparison with the already successful conventional lemma generation approaches (without top-down assistance) as described in [12; 5] all of our methods improve on results obtained with these methods. The level saturation procedure can be improved by the top-down generated patterns. E.g., in the SET domain where a lot of predicate and function symbols occur the pattern use is indispensable. In domains like BOO, however, the patterns cannot incorporate additional potential for deleting lemmas. The GP approach significantly improves on the level saturation method. A not fitness controlled randomized method achieves worse results (see [4]).

Considering the pattern based method and GP one can recognize that with GP generated lemmas the proof length can often be reduced in a more effective manner. The pattern based method has some difficulties in producing sufficiently specific patterns well-suited for the first iteration steps since the probability that such patterns match needed query literals decreases from iteration to iteration. GP can overcome the problem that in its initial population some useful lemmas may be missing. It can re-compute these lemmas. Furthermore, we could observe that GP sometimes generates well-suited rather hard lemmas (with depth 4).

7 Conclusion and Future Work

We presented two methods for combining top-down and bottom-up proof search aiming at generating a set of well-suited lemmas. The lemmas are then used in a final top-down proof run in addition to the given input clauses. We have seen that an evaluation of an abstracted top-down search space (with patterns) and a partial evaluation of interesting regions of the complete top-down search space (by genetic programming) is indeed able to provide sufficient information in order to control bottom-up inferences. As the experiments show the criteria are strong enough to generate interesting hard lemmas which provide large search reductions.

The study reveals that the use of our techniques in a parallel proof environment is desirable. The pattern based approach can work with different parameters for *L, N, I,* and *J* in parallel. Also the GP algorithm can profit when different incarnations run in parallel. Specifically, the evolutionary approach offers the possibility to develop high performance parallel systems which may scale up to a large number of processors.

References

[1] 0. Astrachan and D. Loveland. The Use of Lemmas in the Model Elimination Procedure. *Journal of Automated Reasoning,* 19(1):117-141,1997.

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

[3] D. Fuchs. Cooperation of Top-Down and Bottom-Up Theorem Provers by Subgoal Clause Transfer. In *Proceedings of AISC-98,* pages 157-169. Springer, LNAI1476,1998.

[4] M. Fuchs. An Evolutionary Approach for Combining Top-down and Bottom-up Proof Search. AR-Report AR-98-04,1998, Technische Universitat Munchen, Institut fur Informatik, 1998.

[5] M. Fuchs. Relevancy-Based Lemma Selection for Model Elimination using Lazy Tableaux Enumeration. In *Proceedings of ECAI-98,* pages 346-350. John Wiley *k* Sons, Ltd., 1998.

[6] J. Holland. *Adaptation in natural and artificial systems.* Ann Arbor: Univ. of Michigan Press, second edition, 1992.

[7] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, 1992.

[8] 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.

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

[10] D. Loveland, D. Reed, and D. Wilson. SATCHMORE: SATCHMO with RElevancy. *Journal of Automated Reasoning,* 14:325-351,1995.

[11] M. Moser, 0. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. The Model Elimination Provers SETHEO and E-SETHEO. *Journal of Automated Reasoning,* 18(2), 1997.

[12] J. Schumann. Delta - a bottom-up preprocessor for top-down theorem provers. system abstract. In *Proceedings of CADE-12,* pages 774-777. Springer, LNAI 814,1994.

[13] M. Stickel. A prolog technology theorem proven Implementation by an extended prolog compiler. *Journal of Automated Reasoning,* 4:353-380,1988.

[14] M. Stickel. Upside-Down Meta-Interpretation of the Model Elimination Theorem-Proving Procedure for Deduction and Abduction. *Journal of Automated Reasoning,* 13:189-210,1994.

[15] G. Sutcliffe, C. Suttner, and T. Yemenis. The TPTP Problem Library. In *Proceedings of CADE-12,* pages 252-266. LNAI 814,1994.