Model Elimination with Simplification and its Application to Software Verification

Peter Baumgartner · Dorothea Schäfer*

1 Introduction

Software verification is known to be a notoriously difficult application area for automated theorem provers. Consequently, this is the domain of interactive systems, such as KIV [Reif *et al.*, 1997], HOL [Gordon and Melham, 1993], Isabelle [Nipkow and Paulson, 1992] and PVS [Owre *et al.*, 1992]. The work described here aims to demonstrate that automated theorem provers (ATPs) can be successfully incorporated into such systems in order to relieve the user from some interactions. More specifically, we describe our approach of coupling the interactive program verification system KIV [Reif *et al.*, 1997] with our automated theorem prover PROTEIN [Baumgartner and Furbach, 1994].

The KIV system [Reif *et al.*, 1997] is a professionally engineered software verification system based on dynamic logic. Verification usually is done interactively by constructing a proof tree in a respective sequent calculus. However, the user can decide to attempt automated proofs for proof obligations which are "simple" enough. As a preliminary step then, a relevancy analysis tries to minimize the formulae necessary to prove the obligation submitted to the automated prover. Unlike typical benchmark problems used in ATP, these problems quite often contain redundant axioms, and hence having a goal-oriented prover like PROTEIN better supports focusing on the relevant ones than bottom-up methods.

Currently there are two ways of proof automatization in KIV. The first way is to call an external prover (currently only 3TAP is fully coupled). Proof obligations are sorted first-order formulae with equality then. The second, built-in way is by *simplifier rules*: these are Gentzen sequents which, by a special syntax, contain information *how* to use them, namely as conditional rewrite rules. It is assumed and pragmatically justified that simplifier rules are a terminating, but not necessarily confluent rewrite system. Simplifier rules are conditional equations, conditional implications or equivalences. They are used from left to right, based on matching.

One useful application of simplifier rules is to express a *definition* like in $X_S \leq Y_S \leftrightarrow (X_S < Y_S \vee X_S = Y_S)$. By this rule, all occurrences of "$\leq$"-literals can be eliminated. Besides lemmas, quite often axioms are treated as simplifier rules.

Simplifier rules are used to *reduce* a goal sequent to a normal form, either at the predicate or term level, depending on the type of the rule. At best, reduction arrives at an axiom in order to have a proof. Simplifier rules usually dominate the input clause set by far, they are user given, carefully selected and a highlight in KIV. They turned out to be very useful and efficient in practice, but still "too incomplete"; hence, there is the need to substitute user interactions by calls to an ATP.

* Both authors are funded by the DFG within the research programme "Deduction" under grant Fu 263/2-2

It is obvious that an ATP should deal with simplifier rules properly, i.e. as conditional rewrite rules, but not as ordinary clauses. The main technical contribution of this paper is thus the extension of the calculus underlying PROTEIN – model elimination – to handle simplifier rules properly (Section 3).

Related work. First, there is related work concerning automated theorem proving calculi. Of course, rewriting is around for many years now, *but mainly in resolution based systems* ([Bachmair and Ganzinger, 1998] is a good starting point to enter this huge area). Unfortunately, goal-oriented calculi like model elimination are inherently incompatible to rewriting. There is only few work in this direction: in [Astrachan, 1992] unit equations (demodulators) are used to simplify lemma clauses only, and in [Brüning, 1995] equivalences are exploited for an extended regularity check (a kind of loop check), but not for rewriting purposes. In sum, we conclude that "simplification" as defined below extends previous work in this direction.

Second, there are many interactive systems around that typically contain some form of automated deduction. Related work comes from two directions. The one line of research could be described by the term *homogeneous architectures*, where proof automatization was developed as constituent of the system and is tightly coupled to the logic of the system. This includes systems like *HOL* [Gordon and Melham, 1993], *Isabelle* [Nipkow and Paulson, 1992], [Kromodimoeljo *et al.*, 1992], *PVS* [Owre *et al.*, 1992], *ACL2* [Kaufmann and Moore, 1996], *Nqthm* [Boyer and Moore, 1988].

Another line of research could be called *heterogeneous architectures*, where the ATPs were developed as general-purpose systems outside of the combined system, or can be identified as clearly separated subsystems. Examples here are *Isabelle* coupled with *LeanTAP* [Beckert and Posegga, 1995], *ILF*, ProofPad [Dahn *et al.*, 1997], *STEP* [Bjoerner *et al.*, 1996a; Bjoerner *et al.*, 1996b], and *KIV* which is already fully coupled with the *3TAP* prover [Beckert *et al.*, 1996][1].

A concrete point is this: "Simplification" by means of conditional rewrite rules is a widespread idea. To our impression, in many systems *completeness* of the automated prover is not considered as a primary goal, and indeed they are not (this applies to e.g. Nqthm, at least as far as we could figure out from the system descriptions).

When looking at this related work we observe that now quite a few systems are around which are similar to ours. We think, however, that our approach is not subsumed by any one of these, as any system has its strength/weaknesses and most suitable application domain(s) with respective knowledge. To our impression, exploiting this knowledge is a challenge for every individual combination of the interactive system and the ATP. In particular, the rewrite rules are highly domain-dependent, and making best use of them might differ in every case.

The rest of this paper is structured as follows: in the next section we briefly review the model elimination (ME) calculus. Section 3 then describes our extensions of ME with rewriting. We present both a "static" version, which is a transformation on the input clause set, and a "dynamic" version which operates during proof time. Completeness is

[1] One of the reasons that 3TAP did not show optimal performance was its lack of goal-orientedness (PROTEIN is based on Model Elimination and hence is goal-oriented).

discussed as well. Section 4 reports on experiments carried out with examples from the KIV environment.

2 Model elimination

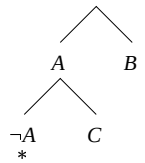
We assume the reader to be familiar with the basic concepts of first order logic (e.g. [Chang and Lee, 1973]).

A *clause C* is an implicitly universally quantified disjunction $L_1 \vee \cdots \vee L_k$ of literals, also written as the multiset $\{L_1, \cdots, L_k\}$. A *connection* (L_1, L_2) is a pair of literals which can be made complementary by application of a substitution σ. Usually we are interested in connections where σ is a most general unifier (MGU). A most general unifier for two multisets of literals is also referred to by the term MGU.

We use "$\overline{\ }$" as the complement-operator for literals. It extends to conjunctions of literals as $\overline{L_1 \wedge \ldots \wedge L_n} = \overline{L_1} \vee \ldots \vee \overline{L_n}$, and, similarly, to disjunctions of literals as expected.

The following presentation of model elimination follows [Baumgartner *et al.*, 1997] and differs from the original chain notation of [Loveland, 1969] by using a path-multiset notation. Formally, a *path* is a sequence of literals written as $p = \langle L_1, \cdots, L_n \rangle$. L_n is called the *leaf of p* which is also indicated by $leaf(p)$. The symbol $\circ$ stands for the append function of sequences, the symbol $\dot{\in}$ for membership in a sequence.

The path sets we construct below can best be visualized as trees. This also explains the connection to semantic tableaux (see [Fitting, 1990]). For example, consider the semantic tableaux on the right. It contains the clauses $A \vee B$ and $\neg A \vee C$. Since the leftmost branch contains a pair of complementary literals it is *closed*, which is indicated by the "$*$". The other two branches are *open*. We represent semantic tableaux by path sets, *but keep only the open branches.* Thus, this tree would be the path set $\{\langle A, C\rangle, \langle B\rangle\}$.



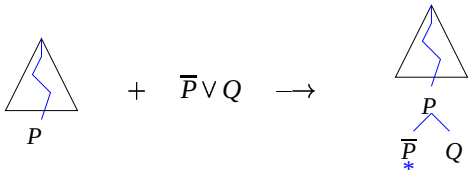
For the following definition we assume as given a set of clauses and a *computation rule* which selects from a given path set one element; we write $\mathcal{P} \cup \{p\}$ to indicate that p is the selected path in this path set; the letter $\mathcal{P}$ always denotes a path set, and p a path.

Definition 1 (Model elimination (ME)). *The model elimination calculus consists of two inference rules:*

– *The inference rule* extension *transforms a path multiset set and a clause into a path multiset and is defined as follows:*

$$\frac{\mathcal{P} \cup \{p\} \quad L \vee C}{(\mathcal{P} \cup \{p \circ \langle L_i \rangle \mid L_i \in C\})\sigma} \quad \textit{if } (L, leaf(p)) \textit{ is a connection with MGU } \sigma$$

Illustration, at the ground level:

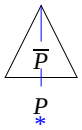


The selected path p is extended by the literals of an input clause. The resulting path containing the connection is deleted, which is marked by a $*$.

– *The inference rule* reduction *is defined as follows:*

$$\frac{\mathcal{P} \cup \{p\}}{\mathcal{P}\sigma} \quad \textit{if } (L, \textit{leaf}(p)) \textit{ is a connection with MGU } \sigma, \textit{ for some } L \dot{\in} p$$

Illustration, at the ground level:



A path containing a connection with its leaf may be closed.

A sequence $\langle \mathcal{P}_1, \cdots, \mathcal{P}_n \rangle$ *is called a* (model elimination) derivation *from a clause set M iff*

1. $\mathcal{P}_1 = \{\langle L_1 \rangle, \ldots, \langle L_k \rangle\}$ *for some clause* $C = \{L_1, \cdots, L_k\}$ *from M. C is called the* goal clause *of the derivation.*
2. $\mathcal{P}_{i+1}$ *is obtained from* $\mathcal{P}_i$ *by an extension step applied to* $\mathcal{P}_i$ *and some new variant C of a clause from M, or* $\mathcal{P}_{i+1}$ *is obtained from* $\mathcal{P}_i$ *by a reduction step.*

A model elimination derivation consists of successive application of the inference rules.

A (model elimination) refutation *is a derivation where* $\mathcal{P}_n = \emptyset$.

It is well-known that ME is complete for any computation rule, provided that the goal clause stems from a minimal unsatisfiable subset of *M*. Various refinements and variants for model elimination are known in the meantime. e.g. [Baumgartner *et al.*, 1997; Letz *et al.*, 1994]. For the purpose of the present paper, however, it suffices to stick to the very basic form just defined.

Model elimination is implemented e.g. in the Setheo prover [Letz *et al.*, 1994] and in our PROTEIN (*PRO*ver with a *T*heory *E*xtension *IN*terface) [Baumgartner and Furbach, 1994], which we used for the experiments below.

3 Simplification in model elimination

The idea of "simplification" is to replace formulae by equivalent ones which are smaller wrt. some well-founded ordering. Doing so would substitute nondeterministic search by deterministic computation. Simplification at the term level and at the predicate level is well known and discussed in the literature (e.g. [Lee and Plaisted, 1989; Bjoerner *et al.*, 1996b; Bronsard and Reddy, 1992]) in the context of saturating, bottom-up calculi like resolution.

Unfortunately, there is a principal difficulty in combining model elimination (and related calculi, such as linear resolution) with "rewriting". We briefly indicate why: a

central idea in model elimination is its goal-orientedness, i.e. that every inference is connected to the goal to be proved. At the same time, inferences among the axioms are forbidden. As a trivial example consider the task to prove that $\{A, A \to B, B \leftrightarrow C\} \models B$ (for some propositional formulae as stated, where $B \leftrightarrow C$ is considered as a rewrite rule, ordered from left to right). Now, rewriting the goal B with $B \leftrightarrow C$ yields the new goal C which would be unprovable. This is only a trivial example, and there are many more traps.

In principle there are several solutions: first, one could also perform rewriting on the axioms. In the example, one would therefore rewrite $A \to B$ to $A \to C$. The thus simplified proof task then is $\{A, A \to C, B \leftrightarrow C\} \models C$, which is provable by model elimination. We propose such a scheme below and call it "static simplification", because it is performed as a preprocessing step before the proof search with model elimination is begun. A general treatment of this idea, however, would amount to a full-fledged superposition calculus (see e.g. [Bachmair and Ganzinger, 1998]). Since termination then is undecidable and preprocessing should not be too complex, we restrict ourselves to a simple procedure (see Sections 3.1).

A second solution is to consider *both* rewriting the goal and *not* rewriting it during the proof search. In order to make this meaningful, one would give preference of the simplified goal over the non-simplified one. In the example, one would thus first consider the simplified goal C, and then B upon failure to prove C. This idea is present in the "model elimination with dynamic simplification" (Section 3.2).

Example 1 (ME refutation). We initiate a running example to illustrate the subsequent definitions. We take an excerpt from a KIV case study "enum" [Schellhorn and Reif, 1997]. The specification deals with three sorts: natural numbers $\mathcal{N}$, the sort of the elements $\mathcal{D}$ and the sort of the finite enumerations $\mathcal{E}$. The function $s : \mathcal{N} \to \mathcal{N}$ denotes the successor function for natural numbers; $\emptyset$ is a constant denoting the empty enumeration. The operator $\oplus : \mathcal{E} \times \mathcal{D} \to \mathcal{E}$ adds an element to an enumeration. The size function $\# : \mathcal{E} \to \mathcal{N}$ gives the number of elements in a finite enumeration. The predicate $\in : \mathcal{D} \times \mathcal{E} \to \mathcal{B}$ tests for membership of an element in a finite enumeration. For one specific (very simple) problem, th-10, we need the following formulae (sorting is written using subscripts as indicated):

ax-2	$\forall N_{\mathcal{N}}, M_{\mathcal{N}} : s(N) = s(M) \leftrightarrow N = M$
ax-03	$\forall D_{\mathcal{D}} : D \notin \emptyset$
ax-06	$\#\emptyset = 0$
ax-07	$\forall D_{\mathcal{D}}, E_{\mathcal{E}} : D \notin E \to \#(E \oplus D) = s(\#E)$
th-10	$\forall D_{\mathcal{D}} : \#(\emptyset \oplus D) = s(0)$

The specification of the problem contains many more formulae which are left out for simplicity. The goal is to prove that th-10 follows from the rest.

For ease of description, we leave away sort information from now on; instead of $\overline{s = t}$ and $\overline{d \in D}$ we write $s \neq t$ and $d \notin D$, respectively. Further, we will use the clausal normal form of these formulas, which should be easy to identify. Following Prolog syntax, variables are written then in capital letters, and functions and constants begin with lowercase letters. For instance, the negated and skolemised theorem th-10, which is the goal clause of our model elimination refutations, is $\#(\emptyset \oplus d) \neq s(0)$.

KIV's simplifier rules are used in Model Elimination in a preprocessing step for *static simplification* of the input clause set, and during proof search as *dynamic simplification*. As a preliminary step, we introduce some definitions common to both.

Definition 2 (Simplification rules). *In the following, L is a literal, and ψ is a conjunction of literals, called* condition *in the following context; the case ψ being the empty conjunction (being true in every interpretation) is allowed and we write $\psi = true$ then. A* rewrite rule *is a formula of the form $\psi \to \tau_1 = \tau_2$ where τ_1 and τ_2 are terms and $Var(\tau_2) \subseteq Var(\tau_1)$. A* replacement rule *is a formula of the form $\psi \to (L \text{ op } \chi)$, where L is a literal, $\text{op} \in \{\leftrightarrow, \to\}$ and χ is a either a disjunction or a conjunction of literals, with $Var(\chi) \subseteq Var(L)$. In case $\text{op} = \leftrightarrow$ the replacement rule is called an* equivalence *rule, else an* implication rule. *Instead of $true \to \phi$ we simply write ϕ. By the term "simplification rule" we refer to both rewrite rules and replacement rules. Simplification rules are considered implicitly as universally quantified.*

A simplification rule is labeled as safe *if (1) $\psi = true$ and (2) it is either a rewrite rule or an equivalence rule.*

For instance, ax-2 from Example 1 is an unconditional equivalence rule and ax-07 a conditional rewrite rule. All simplification rules from Example 1 are safe, with the single exception ax-07.

For easy of defining inference rules dealing with simplification rules it is advantageous to have a canonical representation of simplification rules. We call this form *implication normal form*[2] and it is defined as follows:

Definition 3 (Implication normal form). *Let R be a simplification rule; the set $inf(R)$ is defined as follows:*

1. $inf(\psi \to (L \leftrightarrow \chi)) = inf(\psi \to (L \to \chi)) \cup inf(\psi \to (\overline{L} \to \overline{\chi}))$
2. $inf(\psi \to (L \to (K_1 \wedge \cdots \wedge K_m))) = \{\psi \to (L \to K_1), \cdots, \psi \to (L \to K_m)\}$, *where K_i, $1 \le i \le m$ literals*
3. $inf(\psi \to (L \to (K_1 \vee \cdots \vee K_m))) = \{\psi \to (L \to (K_1 \vee \cdots \vee K_m))\}$, *where K_i, $1 \le i \le m$ literals*
4. $inf(\psi \to \tau_1 = \tau_2) = \{\psi \to \tau_1 = \tau_2\}$

The label "safe" is inherited by this transformation. Finally, the implication normal form *of a set N of simplification rules is $inf(N) = \bigcup_{R \in N} inf(R)$.*

Notice that after transformation to implication normal form an implication rule may also be labeled as safe, namely if case 1 was applied as the first step. For example, ax-2 from Example 1 which is a safe replacement rule, results in the two safe implication rules $s(N) = s(M) \to N = M$ and $\overline{(s(N) = s(M))} \to \overline{(N = M)}$ in implication normal form. The name "safe" is explained by the circumstance that if safe rules are applied for rewriting then it is "safe" – i.e. completeness preserving – to delete the clause to be rewritten.

[2] There is another motivation for this transformation: in KIV, simplifier rules with quantifiers are allowed, e.g. $\forall X\ (p(X) \leftrightarrow \exists Y\ q(X,Y))$. Implication normal form and Skolemisation then yields the two simplifier rules $p(X) \to q(X, f(X))$ and $\neg p(X) \to \neg q(X,Y)$, which can not be expressed as one single universally quantified equivalence.

From now on we will only consider sets of simplification rules in implication normal form, and N always denotes such a set.

Next we turn to inference rules of the form $\frac{L\ R}{A\ B}$, where L is a literal and $R \in N$.

Definition 4 (Simplification). *The inference rules* rewriting *and* replacement *are defined as follows:*

$$\frac{L[\tau] \quad \psi \to (\tau_1 = \tau_2)}{\{L[\tau_2\sigma]\} \quad \overline{\psi\sigma}} \quad \text{if } \tau = \tau_1\sigma \qquad\qquad \frac{L \quad \psi \to (I \to \chi)}{\chi\sigma \quad \overline{\psi\sigma}} \quad \text{if } L = I\sigma.$$

If one of these inference rules is applicable to L and R, yielding A and B, where A and B are sets of literals. We say that (A,B) is a simplification *of (L,R). A simplification is* safe *iff R is safe. Literal L is called* maximally simplified (wrt. N) *if there is no simplification of (L,R), for every $R \in N$. Similarly, a clause C is* maximally simplified *if every $L \in C$ is maximally simplified.*

That is, if (A,B) is a simplification of (L,R), A is the "simplified" version of L and B is the condition coming from the simplification rule; the need for A to be a multiset, rather than a literal, is explained by the possibility of simplifications rules with disjunctions in the head (cf. case 3 in Def. 3); semantically, (A,B) is just the clause $A \cup B$.

For illustration take ax-07 in Example 1 as a simplification rule. Then, $(s(\#\emptyset) \neq s(0),\ d \in \emptyset)$ is a simplification of $(\#(\emptyset \oplus d) \neq s(0),\ D \notin E \to \#(E \oplus D) = s(\#E))$.

3.1 Static simplification

Static simplification applies simplification rules to input clauses as long as possible, modulo subsumption. Thereby, the conditions of the simplification rules are added to the simplified clauses. This yields a maximally simplified clause set.

Definition 5 (Static simplification step). *The inference rule* static simplification step *takes a clause C and a simplification rule R and is defined as follows:*

$$\frac{L \vee C \quad R}{C \cup A \cup B} \quad \text{if } (A,B) \text{ is a simplification of } (L,R)$$

We apply simplification to clauses by replacing the literal to simplify by its simpler set of literals A and the negated condition B. We write $C \overset{R}{\Longrightarrow} C_S$ to indicate that a static simplification step is applicable to C and R and yields C_S. If a clause C_S is derived from a clause C by a chain of simplification steps $C \overset{R_1}{\Longrightarrow} C_1 \overset{R_2}{\Longrightarrow} \cdots \overset{R_n}{\Longrightarrow} C_S$ with $R_i \in N$ we write $C \overset{N}{\Longrightarrow} C_S$. Such a chain is called safe *iff every underlying simplification is safe. In the case that C_S in $C \overset{N}{\Longrightarrow} C_S$ is maximal simplified we write $C \overset{N}{\underset{max}{\Longrightarrow}} C_S$.*

For instance, when taking the axioms in Example 1 as simplification rules and converting them to implication normal form[3], the maximal simplification of the query $\{\#(\emptyset \oplus d) \neq s(0)\}$ results in $\{0 \neq 0\}$. The underlying chain of simplification steps is depicted to the right. With the new query $0 \neq 0$ model elimination would find the proof in one step now using the reflexivity axiom $X = X$.

$$\begin{array}{ll} \#(\emptyset \oplus d) \neq s(0) & \stackrel{ax-07}{\Longrightarrow} \\ s(\#\emptyset) \neq s(0) \vee d \in \emptyset & \stackrel{ax-06}{\Longrightarrow} \\ s(0) \neq s(0) \vee d \in \emptyset & \stackrel{ax-03}{\Longrightarrow} \\ s(0) \neq s(0) & \stackrel{ax-2}{\Longrightarrow} \\ 0 \neq 0 & \end{array}$$

Based on this, static simplification on a set of clauses with certain properties will be defined.

Definition 6 (Static simplification). *By $C \sqsubseteq D$ we denote subsumption among clauses, i.e. $C \sqsubseteq D$ iff $\exists\sigma\ C\sigma \subseteq D$. For a clause C we mean by S_C (static simplification of C wrt. N) any clause set satisfying the following conditions:*

1. $S_C \subseteq (\{C_S \mid (C \stackrel{N}{\underset{max}{\Longrightarrow}} C_S)\} \cup \{C\})$
   (S_C consists only of clauses obtained by applying maximal static simplification steps to C and optionally includes C)
2. *For all C_S such that $C \stackrel{N}{\underset{max}{\Longrightarrow}} C_S$ there is a $C'_S \in S_C$ such that $C'_S \sqsubseteq C_S$*
   (all maximal simplified clauses C_S of C are included in S_C)
3. *For all $C_S \in S_C$ there is no $C'_S \in S_C$ such that $C'_S \sqsubseteq C_S$*
   (S_C is minimal wrt subsumption)
4. *For all C_S such that $C \stackrel{N}{\underset{max}{\Longrightarrow}} C_S$: if $C \stackrel{N}{\underset{max}{\Longrightarrow}} C_S$ is safe, then $C \notin S_C$*
   (clauses with only safe simplifications are deleted from S_C)

Now let M be a set of clauses. Its static simplification (wrt. N) *is $M_S = \bigcup_{C \in M} S_C$.*

We are going to discuss and motivate Definition 6 now. Notice that the static simplification S_C of clause C is not uniquely defined; the definition states only neccessary conditions (which we found quite natural) of what a static simplification is. In particular, it gives some freedom whether to include the original clause C in S_C or not. This non-uniqueness allows to define various concrete static simplification procedures, as long as they satisfy the requirements stated in the definition.

What about termination? We tacitly assume that the simplifier rules are given such that static simplification terminates! In all our examples from the KIV domain this was the case. Clearly, a more systematic approach should be taken in the future.

For a large number of simplification rules or clauses the operation can be restricted to unconditional simplification rules or to simplification of the query clause, only.

Subsumption is *not* carried out across all clauses handled during simplification, because we observed that more exhaustive subsumption tests would be too time consuming.

Notice that we keep the maximally simplified clauses only, but not the intermediate stages. But the reader might wonder why according to 2 *all* maximal simplifications of a clause have to be kept. For example, the unit clause A would be simplified to

[3] Units, such as $D \notin \emptyset$ (ax-03) are treated as $D \notin \emptyset \leftrightarrow \mathit{true}$.

the *two* clauses B and C in presence of the simplification rules $A \leftrightarrow B$ and $A \leftrightarrow C$. Any resolution based system (appropriate ordering presupposed) would rewrite A to B (or C) and delete A afterwards. Resolution can afford this due to saturation of the simplification rules towards a *confluent* system. However, our situation is different: we can neither assume in our KIV domain that the simplification rules form a confluent system, nor do we want to do a resolution-like saturation (it is too time consuming for a preprocessing step, possibly nonterminating). Hence, for completeness reasons, we have to do all simplifications and can delete the simplified clause C only in special "safe" situations (cf. Condition 4).

Alltogether, we get the following important property:

Theorem 1 (Completeness of static simplification). *Let M be a clause set and N be a set of simplification rules. Suppose that M_S is the static simplification of M wrt. N. If $M \cup N$ is unsatisfiable, then $M_S \cup N$ is also unsatisfiable.*

In some situations the theorem can be strengthened by replacing "$M_S \cup N$" with "M_S", e.g. in case of "definitions", when all occurrences of a predicate symbol are eliminated by a safe simplification, and, furthermore, the rewrite rules do not overlap (i.e. there are no critical pairs among the lefthand sides). Details, as well as proofs, are contained in [Schäfer, 1998].

Static simplification sometimes is quite effective to speed up proofs signifcantly. For example, proving that set union is associative from the axioms $\forall x,y,z:\ x \in y \cup z \leftrightarrow (x \in y \vee x \in z)$ and $\forall y,z:\ ((y = z) \leftrightarrow \forall u:\ (u \in y \leftrightarrow u \in z))$, static simplification will eliminate equality when the second axiom is turned into rewrite rules. The proof can be found then in a few steps, while it is a hard problem without simplification (more than 5 hours for PROTEIN, other provers have difficulties as well).

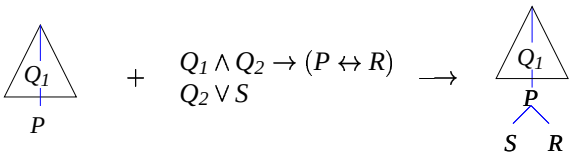
3.2 Dynamic simplification

In order to take advantage of simplification rules during proof time, we extend ME (Def. 1) by the following inference rule:

Definition 7 (Simplification inference rule). *The inference rule* (model elimination) simplification step *transforms a path multiset, a simplification rule R and n clauses ($n \geq 0$) into a path multiset:*

$$\frac{\mathcal{P} \cup \{p\} \quad R \quad L_1 \vee C_1 \quad \cdots \quad L_n \vee C_n}{(\mathcal{P} \cup \{p \circ \langle L \rangle \mid L \in C_1 \vee \cdots \vee C_n \cup A\})\sigma} \quad \textit{if} \begin{cases} 1.\ (A,B) \textit{ is a simplification of} \\ \quad (\textit{leaf}(p), R), \textit{ and} \\ 2.\ B\sigma = (\{\overline{L_1}, \cdots, \overline{L_n}\} \cup X)\sigma, \\ \quad \textit{for some } X \subseteq \{\overline{L} \mid L \dot{\in} p\} \\ \quad \textit{and MGU } \sigma. \end{cases}$$

Illustration, at the ground level:



A leaf is extended by its simplification. The condition has to be fulfilled. Therefore path literals or literals from input clauses may be used. For the latter the leaf is additionally extended by the rest literals of the clauses.

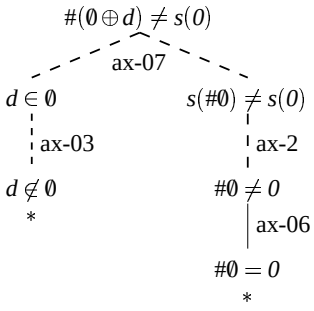
The case $n = 0$ is called strict*, else it is called* nonstrict*. We extend the notion of* derivation *(Def. 1) in a natural way, namely by assuming as given a set N of simplification rules in implication normal form, and adding this case to the definition of derivation:*

3. *$\mathcal{P}_{i+1}$ is obtained from $\mathcal{P}_i$ by an ME simplification step applied to $\mathcal{P}_i$, some new variant R of a simplification rule from N and $n \geq 0$ new variants $L_1 \vee C_1, \cdots, L_n \vee C_n$ of clauses from M.*

The new calculus is called ME with dynamic simplification (SimME)*. The term* ME with simplification *refers to SimME applied to a clause set which was obtained from the original clause set and some set of simplification rules by static simplification (cf. Def. 6).*

An operational description: the ME simplification step first simplifies (cf. Def. 4) the leaf literal $leaf(p)$ using simplification rule R, yielding the simpler set of literals A. The condition of R, a literal set, has to be resolved away by taking a combination X of literals from p, and literals $L_1, \ldots, L_n$ from input clauses. The rationale for this strategy is to restrict application of rewrite rules more than it would be the case when the conditions would be taken without resolving them away immediately[4].

We continue on Example 1 and take the axioms as simplification rules, just as was done for static simplification above (cf. the text after Definition 6). The figure on the right depicts the refutation in a tableau notation. Dashed lines indicate simplification steps. The first (topmost) simplification step with ax-07 as simplification rule branches to the right with the simplified leaf literal, and it branches to the left with the (instantiated) condition $d \in \emptyset$, which is closed within this step by the input clause ax-03 (thus, this is a nonstrict inference). Since ax-03 is a unit clause, no more proof obligations arise here.



Due to the simplification rules no equality axioms are needed to find the refutation. In this example simplification directs the proof process immediately into the right direction. In this example PROTEIN needed only 5 inferences for the whole proof search. This means that the prover did not have to backtrack.

Why do we need both, static and dynamic simplification? The static version works bottom up, whereas dynamic simplification works top down. Because of the goal-orientedness of model elimination we need both. Consider the following example $\forall X$:

[4] This resembles the situation of hyper resolution vs. binary resolution.

$male(X) \leftrightarrow \neg female(X)$. The dynamic simplification inference rule translates every positive (negative) *male*-leaf immediately into a negative (positive) *female* literal. Hence, extension steps with clauses with negative (or positive) *male* literals would no longer exist. On the other hand, with static simplification as preprocessing, the situation can be repaired by replacing the *male*-literals in the input clauses by complementary *female*-literals.

We tested PROTEIN with SimME with examples from the TPTP library ([Sutcliffe *et al.*, 1994]). We manually scanned the input specification for formulas which seemed to be suitable for simplification rules. For many puzzles we used formulas specifying something belonging exactly to one of two groups. Like $\forall X : male(X) \leftrightarrow \neg female(X)$ mentioned above. Using this formulae as the sole simplification rule prunes the search space dramatically. SimME with this single simplification rule had a much better performance than plain model elimination. For example, the TPTP-Example PUZ006-1 could not be solved in reasonable time by PROTEIN in its model elimination setting but PROTEIN with SimME found a proof within 13 seconds.

Typically, simplification techniques such as term rewriting, are not compatible to goal-oriented, linear calculi like model elimination. Hence, special care must be taken not to loose completeness (see e.g. [Brüning, 1995]). It is in general not even complete to rewrite a leaf, say $P(f(a))$, using $f(X) = X$ to $P(a)$. However, in our case, the dynamic simplification inference rule does not preclude the other inference rules from being applied. One might be tempted to think that SimME is not useful at all then. However, simplification can be used as a *preference* strategy which allows to find shorter proofs first (see the experimental results in Section 4 below). We conclude this section with the following trivial, nethertheless important theorem:

Theorem 2 (Completeness of SimME). *Let M be a clause set and N be a set of simplification rules. If M is unsatisfiable then there is a SimME refutation of M and N.*

4 The ENUM case study

The new SimME calculus was mainly tested with the KIV "Enumeration" (enum) series, which is described in detail in [Schellhorn and Reif, 1997]. The goal of that series is to prove 52 consequences of a specification of finite enumerations. These arose during an interactive session with KIV. They are formulated in first order logic with equality and sorts[5]. Hence they can be passed to any suitable first order prover. Before that, KIV performs an axiom reduction, which deletes many irrelevant axioms from the overall specification.

In [Schellhorn and Reif, 1997], results for the non-inductive theorems of the enum series are reported for the tableaux prover 3TAP [Beckert *et al.*, 1996], the resolution prover Otter [McCune, 1994] and the model elimination prover SETHEO [Letz *et al.*, 1994]. The results for Otter and SETHEO differ to some degree, as they are based on very different calculi, but the overall performance is comparable. 3TAP seems not yet to

[5] In PROTEIN, sorts are handled by transforming them away following the approach in [Schmitt and Wernecke, 1989]. To treat equality, adding the equality axioms turned out to be best.

be fully optimized. The SETHEO results are quite comparable to the ones we obtained with our PROTEIN prover.

Table 1 summarizes our results. Missing "Thm." numbers indicate inductive theorems, which cannot be proven by first-order provers. The first column *KIV* gives the number of interactions needed by an experienced KIV user to direct the built-in simplifier to a proof. In sum, these are 51. The next column contains the results for PROTEIN in its default setting without simplification (see [Baumgartner and Furbach, 1994] for a system description); *Total* is the overall time in seconds and includes reading in and preprocessing the source file. Blank entries mean that no proof was found within the time limit of two minutes (we used a SUN Ultra 1 for our experiments).

Similarly to the PROTEIN entries, the last two columns S-PROTEIN describe PROTEIN with its simplification extension as described in Section 3.1 and Section 3.2. To state it explicitly, we first applied static simplification to the input clause set, and then used also dynamic simplification during proof search (theorems 1 and 2 guarantee the completeness of the approach). As simplification rules we used those given from the KIV system.

	KIV	PROTEIN	S-PROTEIN	
Thm	*#Int.*	*Total*	*Proof*	*Total*
th-01	*0*	*2.2*	*0.0*	*4.5*
th-02	*0*	*0.9*	*0.0*	*1.5*
th-04	*1*	*20*	*6.4*	*8.0*
th-05	*2*	*0.9*	*0.1*	*1.7*
th-06	*2*	*0.9*	*0.0*	*1.7*
th-09	*4*			
th-10	*0*	*1.1*	*0.0*	*2.2*
th-11	*0*	*1.7*	*0.3*	*2.4*
th-12	*0*	*1.3*	*0.3*	*2.6*
th-14	*1*			
th-16	*3*			
th-17	*1*		*0.9*	*3.5*
th-18	*0*	*2.1*	*0.0*	*4.3*
th-19	*2*	*106*	*8.6*	*18*
th-20	*1*	*33*	*5.8*	*9.6*
th-21	*0*	*2.3*	*0.0*	*4.7*
th-24	*0*	*3.9*	*3.2*	*12*
th-25	*0*	*4.3*	*0.1*	*8.3*
th-26	*0*	*4.3*	*1.8*	*10*
th-27	*0*	*2.0*	*0.0*	*3.4*
th-28	*0*	*4.1*	*0.1*	*8.6*
th-29	*0*	*4.3*	*0.0*	*9.0*
th-30	*0*		*1.1*	*40*

	KIV	PROTEIN	S-PROTEIN	
Thm	*#Int.*	*Total*	*Proof*	*Total*
th-31	*1*	*4.1*	*0.0*	*8.7*
th-32	*0*	*4.2*	*0.0*	*8.9*
th-33	*0*	*4.4*	*0.1*	*8.7*
th-34	*3*	*3.0*	*0.0*	*6.2*
th-35	*1*	*4.1*	*0.0*	*9.0*
th-36	*0*	*4.7*	*0.1*	*9.0*
th-37	*1*	*4.6*	*0.1*	*9.1*
th-38	*11*			
th-39	*0*	*3.1*	*0.0*	*6.3*
th-40	*0*	*2.0*	*0.0*	*3.7*
th-41	*0*	*4.4*	*0.0*	*9.6*
th-42	*4*		*83*	*92*
th-43	*0*	*4.4*	*0.0*	*9.6*
th-44	*1*	*2.3*	*1.2*	*4.9*
th-45	*1*			
th-46	*0*	*2.7*	*2.9*	*6.7*
th-47	*0*	*2.3*	*1.2*	*5.1*
th-48	*0*	*2.1*	*0.0*	*4.2*
th-49	*1*	*5.8*	*7.5*	*16*
th-50	*5*			
th-51	*0*	*2.8*	*1.4*	*5.6*
th-52	*5*			
Σ #Int.	*51*	*35*	*30*	

Table 1. Results for PROTEIN and S-PROTEIN for the enum series. See the text for explanations of the entries.

The S-PROTEIN values for very easy problems are higher than those for PROTEIN. The simple explanation is that reading in and preparing the simplification rules takes some time. This time, however, is not prohibitively high. The extreme case is *9.6* seconds for the large specification th-41 (85 first-order formulae and 77 simplifier rules) with almost zero proof search time.

Some of the easy problems could be almost solved during preprocessing by static simplification, but in most cases dynamic simplification steps are also applicable. However, during the experiments we found that one better does not overemphasize the role of simplification. For the static simplification we applied simplification only to the theorem to be proven, because otherwise it would be too time consuming. Furthermore, only unconditional rules were allowed. In this setting, the time for static simplification alone was neglectable (< 1 sec.).

For dynamic simplification we added all simplification rules as "ordinary" formulae to the input clause set (some few cases, however, had to be treated slightly differently). In order to make this meaningful, dynamic simplification inferences have a strong preference over "usual" extension inferences (cf. Def. 1).

Throughout the experiments we used the *strict* version of dynamic simplification (cf. Def. 7), and only rewrite rules (i.e. rules with equality in the head, cf. Def. 2) with an empty condition or one literal condition were allowed. This was made in order not to spend too much time for the simplification inferences. Replacement rules obviously cannot be applied at term positions and thus do not tend to broaden the search space as much as rewrite rules. Hence we used no restriction on the length of the conditions for such rules.

We tried various flag combinations. As an outcome of these experiments, the just described flag settings was identified as the most successful one in average.

With this setup, we draw the conclusion that simplification pays off. This holds in particular for the more difficult problems (th-04, th-19, th-20 and th-42), while no simpler ones are lost by the overhead (there are some cases listed in the table where proof time increases for S-PROTEIN, but not in an unacceptable way).

The last line in Table 1 (right) sums up interactions. In the *KIV* column it is the total number of interactions needed by the KIV user to solve all 52 problems by using the built-in simplifier. The other two values for PROTEIN, (resp. S-PROTEIN) sum up the *remaining* number of interactions needed by the KIV user, under the assumption that PROTEIN (resp. S-PROTEIN) was tried on the theorems. With S-PROTEIN, the number of interactions decreases from 51 to 30.

5 Conclusions

In this paper we extended model elimination (ME) in a new way to take advantage of "simplification by rewriting", as it is used so successfully in the resolution paradigm. Unfortunately, rewriting cannot be incorporated to such a high degree in ME as in resolution calculi. This is not specific to ME, it is rather more generally the price to be paid for goal-oriented linear calculi. Nevertheless, from our practical experiments we conclude that simplification pays off.

Quite often, the overhead of simplification and the breadthening of the local search space was overcompensated by shorter proofs and shorter proof times. In three cases, the prover could even find a proof where ordinary model elimination had to give up.

We understand this paper as a first investigation into the potential of simplification in the context of software verification domains. The implementation should be improved by making rewriting operations faster (they are unnecessarily slow at the moment). This will allow us to stronger emphasize the role of simplification then.

In general, more improvements and fine tuning will be investigated in the future, and the coupling of PROTEIN and KIV will be fully implemented. This then would allow to apply the method in real verification scenarios.

Acknowledgments: We would like to thank the reviewers for their helpful comments.

References

[Astrachan, 1992] Owen L. Astrachan. *Investigations in Model Elimination based Theorem Proving*. PhD thesis, Duke University, 1992. Technical Report CS-1992-21.

[Bachmair and Ganzinger, 1998] Leo Bachmair and Harald Ganzinger. Chapter 11: Equational Reasoning in Saturation-Based Theorem Proving. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction. A Basis for Applications*, volume I: Foundations. Calculi and Refinements, pages 353–398. Kluwer Academic Publishers, 1998.

[Baumgartner and Furbach, 1994] Peter Baumgartner and Ulrich Furbach. PROTEIN: A *PRO*ver with a *T*heory *E*xtension *I*nterface. In A. Bundy, editor, *Automated Deduction – CADE-12*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 769–773. Springer, 1994. Available in the WWW, URL: `http://www.uni-koblenz.de/ag-ki/Systems/PROTEIN/`.

[Baumgartner *et al.*, 1997] Peter Baumgartner, Ulrich Furbach, and Frieder Stolzenburg. Computing Answers with Model Elimination. *Artificial Intelligence*, 90(1–2):135–176, 1997.

[Beckert and Posegga, 1995] Bernhard Beckert and Joachim Posegga. leanTAP: Lean tableau-based deduction. *Journal of Automated Reasoning*, 15(3):339–358, 1995.

[Beckert *et al.*, 1996] Bernhard Beckert, Reiner Hähnle, Peter Oel, and Martin Sulzmann. The tableau-based theorem prover 3TAP, version 4.0. In M.A. McRobbie and J.K. Slaney, editors, *Automated Deduction — CADE 13*, LNAI 1104, pages 303–307, New Brunswick, NJ, USA, July 1996. Springer-Verlag.

[Bjoerner *et al.*, 1996a] Nikolaj Bjoerner, Ance Brown, Eddie Chang, Michael Colon, Arjun Kapur, Zohar Manna, Henny Sipma, and Tomas Uribe. Step: Deductive-algorithmic verification of reactive and real-time systems. In *8th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science. Springer, 1996.

[Bjoerner *et al.*, 1996b] Nikolay Bjoerner, Mark Stickel, and Tomás Uribe. A Practical Integration of First-Order Reasoning and Decision Procederes. In M.A. McRobbie and J.K. Slaney, editors, *Automated Deduction — CADE 13*, LNAI 1104, New Brunswick, NJ, USA, July 1996. Springer-Verlag.

[Boyer and Moore, 1988] R.S. Boyer and J.S. Moore. *A Computational Logic*. Academic Press, 1988.

[Bronsard and Reddy, 1992] Francois Bronsard and Uday S. Reddy. Reduction Techniques for First-Order Reasoning. In M. Rusinowitch and J.L. Rémy, editors, *Proceedings of the Third International Workshop on Conditional Term Rewriting Systems*, pages 242–256. Springer-Verlag, July 1992. LNCS 656.

[Brüning, 1995] S. Brüning. Exploiting Equivalences in Connection Calculi. *Journal of the IGPL*, 3(6):857–886, 1995.

[Chang and Lee, 1973] C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[Dahn *et al.*, 1997] B.I. Dahn, J. Gehne, Th. Honigmann, and A. Wolf. Integration of Automated and Interactive Theorem Proving in ILF. In W. McCune, editor, *Automated Deduction — CADE 14*, LNAI 1249, pages 57–60, Townsville, North Queensland, Australia, July 1997. Springer-Verlag.

[Fitting, 1990] M. Fitting. *First Order Logic and Automated Theorem Proving*. Texts and Monographs in Computer Science. Springer, 1990.

[Gordon and Melham, 1993] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[Kaufmann and Moore, 1996] M. Kaufmann and J.S. Moore. Acl2: An industrial strength version of nqthm. In *Proceedings of Eleventh Annual Conference on Computer Assurance (COMPASS-96)*, pages 23–34. IEEE Computer Society Press, 1996.

[Kromodimoeljo *et al.*, 1992] Sentot Kromodimoeljo, Bill Pase, Mark Saaltink, Dan Craigen, and Irwin Meisels. The eves system. In *Proceedings of the International Lecture Series on Functional Programming, Concurrency, Simulation and Automated Reasoning (FPCSAR)*. McMaster University, 1992.

[Lee and Plaisted, 1989] Shie-Jue Lee and David A. Plaisted. Reasoning with Predicate Replacement, 1989.

[Letz *et al.*, 1994] R. Letz, K. Mayr, and C. Goller. Controlled Integrations of the Cut Rule into Connection Tableau Calculi. *Journal of Automated Reasoning*, 13, 1994.

[Loveland, 1969] D. Loveland. A Simplified Version for the Model Elimination Theorem Proving Procedure. *JACM*, 16(3), 1969.

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

[Nipkow and Paulson, 1992] Tobias Nipkow and Lawrence C. Paulson. Isabelle-91. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, pages 673–676, Saratoga Springs, NY, 1992. Springer-Verlag LNAI 607. System abstract.

[Owre *et al.*, 1992] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction*, LNAI 607, pages 748–752. Springer-Verlag, 1992.

[Reif *et al.*, 1997] Wolfgang Reif, Gerhard Schellhorn, and Kurt Stenzel. Proving System Correctness with KIV 3.0. In W. McCune, editor, *Automated Deduction — CADE 14*, LNAI 1249, pages 69–72, Townsville, North Queensland, Australia, July 1997. Springer-Verlag.

[Schäfer, 1998] Dorothea Schäfer. Simplification in model elimination. Master's thesis, Universität Koblenz, 1998. To appear.

[Schellhorn and Reif, 1997] Gerhard Schellhorn and Wolfgang Reif. Proving properties of finite enumerations: A problem set for automated theorem provers. Technical report, University of Ulm, Dept. of Computer Science, 1997. URL: http://www.informatik.uni-ulm.de/pm/kiv/setheo/enum.ps.

[Schmitt and Wernecke, 1989] P.H. Schmitt and W. Wernecke. Tableau calculus for sorted logics. In *Sorts and Types in Artificial Intelligence*, volume 418 of *Lecture Notes in Artificial Intelligence*, pages 49–60. Springer, 1989.

[Sutcliffe *et al.*, 1994] G. Sutcliffe, C. Suttner, and T. Yemenis. The TPTP problem library. In Alan Bundy, editor, *Automated Deduction — CADE 12*, LNAI 814, Nancy, France, June 1994. Springer-Verlag.