Model Elimination Without Contrapositives and its Application to PTTP*

Peter Baumgartner Ulrich Furbach

University of Koblenz, Rheinau 1,
D 56075 Koblenz, Germany
{peter,uli}@informatik.uni-koblenz.de

Abstract

We give modifications of model elimination which do not necessitate the use of contrapositives. These restart model elimination calculi are proven sound and complete and their implementation by PTTP is depicted. The corresponding proof procedures are evaluated by a number of runtime experiments and they are compared to other well known provers. Finally we relate our results to other calculi, namely the connection method, modified problem reduction format and Near-Horn Prolog.

1 Introduction

This paper demonstrates that model elimination and hence PTTP can be defined such that it is complete without the use of contrapositives. We believe that this result is interesting in at least two respects: it makes model elimination available as a calculus for non-Horn logic programming and it enables model elimination to perform proofs of mathematical theorems by case analysis.

Let us first explain what we mean by the term "without the use of contrapositives". In implementations of theorem proving systems usually n procedural counterparts $L_i \leftarrow \overline{L}_1 \wedge \cdots \wedge \overline{L}_{i-1} \wedge \overline{L}_{i+1} \wedge \cdots \wedge \overline{L}_n$ for a clause $L_1 \vee \cdots \vee L_n$ have to be considered. Each of these is referred to as a *contrapositive* of the given clause and represents a different entry point during the proof search into the clause. It is well-known that for Prolog's SLD-resolution one single contrapositive suffices, namely the "natural" one, selecting the head of the clause as entry point. For full first-order systems the usually required n contrapositives are either given more *explicitly* (as in the SETHEO prover [Letz *et al.*, 1992]) or more *implicitly* (as in the connection method by allowing to set up a connection with *every* literal in a clause [Eder, 1992]). The distinction is merely a matter of presentation and will be given up for this paper. Now, by a system "without contrapositives" we mean

*A shorter version will appear under the title " Model Elimination Without Contrapositives" in *Proc. of the CADE'94*.

more precisely a system which does not need all n contrapositives for a given n-literal clause.

Model elimination [Loveland, 1968] is a calculus, which is the base of numerous proof procedures for first order deduction. There are high speed theorem provers, like METEOR [Astrachan and Stickel, 1992] or SETHEO [Letz *et al.*, 1992]. The implementation of Model elimination provers can take advantage of techniques developed for Prolog. For instance, SETHEO compiles the input clause set into a generalized WAM architecture. Stickel's Prolog technology theorem proving system (PTTP, [Stickel, 1988]) uses Horn clauses as an intermediate language, which can even be processed by conventional Prolog systems [Stickel, 1989]. Hence, it should be straightforward to use model elimination and PTTP in the context of **non-Horn logic programming**. Indeed this possibility is discussed in various papers; however it is discarded by some authors because of the necessity to use contrapositives (e.g. [Loveland, 1991, Plaisted, 1988]). The argument is given by Plaisted [Plaisted, 1988] explicitly: "In general, however, we feel that the need for contrapositives makes it difficult to view model elimination as a true programming language in the style of Prolog, since the user has less control over the search." Suppose, for example we are given an input clause[1]

$$prove(and(X, Y)) \leftarrow prove(X) \wedge prove(Y)$$

which can be used within a formalization of propositional calculus. A possible contrapositive is

$$\neg prove(X) \leftarrow \neg prove(and(X, Y)) \wedge prove(Y)$$

The procedural reading of this contrapositive is somewhat strange and leads to an unnecessary blowing-up of the search space; in order to prove $\neg prove(X)$ one has to prove $prove(Y)$ – a goal which is totally unrelated to $\neg prove(X)$ by introducing a new variable. Such observations had been the motivation for the development of calculi which need no contrapositives, e.g. Loveland's NearHorn-Prolog. Gabbay's N-Prolog [Gabbay, 1985] when restricted to clause logic is general enough to be instantiated to both NearHorn-Prolog and problem reduction formats ([Plaisted, 1988], see also Section 5 below; [Reed and Loveland, 1992] contains a comparison of these).

Another motivation for the new calculi is as follows: in proving theorems such as "if $x \neq 0$ then $x^2 > 0$" a human typically uses case analysis according to the axiom $X < 0 \vee X = 0 \vee -X < 0$. This seems a very natural way of proving the theorem and leads to a well-understandable proof. Our modified model elimination procedure carries out precisely such a proof by case analysis. Experimental results with similar examples from calculus and a comparison with other proof procedures are presented in this paper.

The calculi we derive in this paper are a very small modification of model elimination and hence allow for Prolog implementation techniques. They are complete without the use of contrapositives and hence well-suited for logic programming and for theorem proving by "case analysis" or "splitting".

[1] Taken from [Plaisted, 1988].

As a more theoretical contribution we will show that one of the NearHorn-Prologs, namely InH-Prolog ([Loveland and Reed, 1989]), can be seen as one of our modified model elimination procedures, i.e. NearHorn-Prolog is nothing but a variant of model elimination.

As a final point, we discovered that the connection method ([Bibel, 1987, Eder, 1992], [Baumgartner and Furbach, 1993] contains a comparative study) is complete without contrapositives and *without any change to the calculus*. This surprising result is due to a relaxed complementary-literal condition which subsumes the above-mentioned small change in model elimination.

This paper is organised as follows: in the following section we review the model elimination calculus we use as a starting point of our investigation. In section 3 we define various variants of this calculus and give soundness and completeness proofs of the weakest one. In section 4 we describe the corresponding PTTP-procedures and give a comparative study of various implementations.

2 Review of Tableau Model Elimination

As a starting point we use a model elimination calculus that differs from the original one presented by [Loveland, 1968]; it is described in [Letz *et al.*, 1992] as the base for the prover SETHEO. In [Baumgartner and Furbach, 1993] this calculus is discussed in detail by presenting it in a consolution style [Eder, 1991] and comparing it to various other calculi. This model elimination manipulates trees by extension- and reduction-steps. In order to recall the calculus and to state a running example consider the clause set

$$\{\{P, Q\}, \{\neg P, Q\}, \{\neg Q, P\}, \{\neg P, \neg Q\}\},$$

A model-elimination refutation is depicted in Figure 1. It is obtained by successive fanning with clauses from the input set (*extension steps*. Additionally, it is required that every inner node (except the root) is complementary to one of its sons. An arc indicates a *reduction step*, i.e. the closing of a branch due to a path literal complementary to the leaf literal.

In the following we use a formal presentation of the calculus along the lines of [Baumgartner and Furbach, 1993]. Instead of trees we manipulate multisets of paths, where paths are sequences of literals.

A clause is a multiset of literals, usually written as the disjunction $L_1 \vee \ldots \vee L_n$. A *connection* in a set of clauses is a pair of literals, written as (K, L), which can be made complementary by application of a substitution. A *path* is a sequence of literals, written as $p = \langle L_1, \ldots, L_n \rangle$. L_n is called the *leaf* of p, which is also denoted by $\mathit{leaf}(p)$; similarly, the first element L_1 is also denoted by $\mathit{first}(p)$. '$\circ$' denotes the append function for literal sequences. Multisets of paths are written with caligraphic capital letters.

Definition 2.1 (Tableau Model Elimination) Given a set of clauses S.

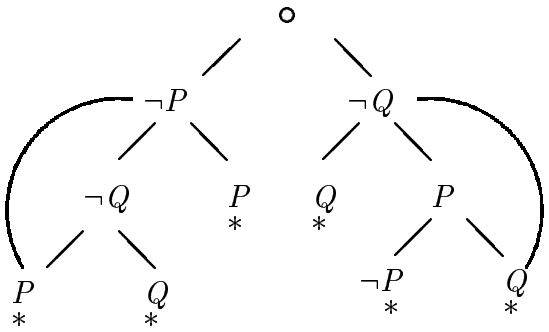


Figure 1: A Model Elimination Refutation

- The inference rule *extension* is defined as follows:

$$\frac{\mathcal{P} \cup \{p\} \qquad\qquad L \vee R}{\mathcal{R}}$$

  where all the following holds:

  1. $\mathcal{P} \cup \{p\}$ is a path multiset, and $L \vee R$ is a variable disjoint variant of a clause in S; L is a literal and R denotes the rest literals of $L \vee R$.
  2. $(\mathit{leaf}(p), L)$ is a connection with MGU σ.
  3. $\mathcal{R} = (\mathcal{P} \cup \{p \circ \langle K \rangle \mid K \in R\})\sigma$.

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

$$\frac{\mathcal{P} \cup \{p\}}{\mathcal{P}\sigma}$$

  where

  1. $\mathcal{P} \cup \{p\}$ is a path multiset, and
  2. there is a literal L in p such that $(L, \mathit{leaf}(p))$ is a connection with MGU σ.

- A sequence $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ is called a *model elimination derivation* iff

  $\mathcal{P}_1$ is a path multiset $\{\langle L_1 \rangle, \ldots, \langle L_n \rangle\}$ consisting of paths of length 1, with $L_1 \vee \ldots \vee L_n$ in S (also called the *goal clause*), and

  $\mathcal{P}_{i+1}$ is obtained from $\mathcal{P}_i$ by means of an extension step with an appropriate clause C, or

  $\mathcal{P}_{i+1}$ is obtained from $\mathcal{P}_i$ by means of a reduction step.

The path p is called *selected path* in both inference rules. Finally, a *refutation* is a derivation where $\mathcal{P}_n = \{\}$. □

Note that this calculus does not assume a special selection function which determines which path is to be extended or reduced next. Correctness and completeness of this calculus follows immediately from a result proved in [Baumgartner, 1992].

3 Restart Model Elimination Calculi

Let us now modify the calculus given above, such that no contrapositives are needed.

In order to get a complete calculus, we have to assume that there exists only one goal, i.e. a clause containing only negative literals, which furthermore does not contain variables. Without loss of generality this can be achieved by introducing a new clause *goal* where *goal* is a new predicate symbol and by modifying every purely negative clause $B_1 \cdots B_n$ to $goal B_1 \cdots B_n$. In the following we will refer to clause sets S satisfying that property as clause sets in *goal-normal form*. Note that since the *goal* literal is attached only to purely negative clauses the Horn status of the given clause set is not affected by the transformation. Furthermore, besides the *goal*-normal form we will only allow derivations which start with the goal clause *goal*.

Soundness of this transformation is evident. Completeness holds as follows:

Theorem 3.1 (Completeness of Model Elimination) *Let S be an unsatisfiable clause set in goal-normal form. Then there exists a tableau model elimination refutation of S with goal clause goal. Furthermore, if S is Horn then no reduction steps are required in this refutation.*

Proof. From [Baumgartner, 1992] we learn that model elimination is complete if the goal clause is contained in some minimal unsatisfiable subset of the input clauses. Since every clause in $S \setminus \{goal\}$ contains at least one positive literal, this set is satisfiable (take the interpretation that assigns *true* to every positive literal). Hence *goal* is contained in every minimal unsatisfiable subset of S. Thus tableau model elimination with the above assumptions is complete.
For the second part observe that every literal along every path in a refutation of a Horn clause set is negative. This is due to the fact that in extension steps the single positive literal of the extending clause is not considered in the formation of the resulting path multiset. Hence reduction steps are impossible for syntactical reasons. □

We are now ready to modify the calculus, such that no contrapositives are necessary. This will be done in three steps:

- As a base we define a *restart model elimination* by a small modification in the definition of tableau model elimination, with the result, that no contrapositives are needed.
- We then weaken this calculus by introducing a *selection function*, which determines which positive head literal can be used for an extension step and finally,
- as a further weakening we introduce *strict restart model elimination* ; by disallowing reduction steps with a positive leaf literal.

A completeness proof is given for the weakest variant, i.e. strict restart model elimination with selection function. Completeness of the stronger variants follows from this result as a simple corollary.

Definition 3.2 (Restart Model Elimination) Assume the following line additionally given between conditions 1 and 2 in the definition of *extension* (Def. 2.1).

1a. if $\mathit{leaf}(p)$ is positive then let $p := p \circ \langle \mathit{first}(p) \rangle$ in the following conditions 2 and 3.

An extension step with lengthening according to 1a above is called a *restart*. The calculus of restart model elimination consists of the inference rules *restart* and *reduction*. □

If a clause $A_1 \cdots A_n B_1 \cdots B_m$ is written in a sequent style (this notation will also be used from now on)

$$A_1, \ldots, A_n \leftarrow B_1, \ldots, B_m$$

then it is clear that, for syntactical reasons extension steps are possible only with head literals A's and not with B's from the body. Thus it is possible to represent clauses as above *without* the need of augmenting them with all contrapositives; only contrapositives with conclusions (i.e. entry points) stemming from the positive literals are necessary.

The price of the absence of contrapositives is that whenever a path ends with a positive literal, the root of the tree, i.e. the clause $\neg goal$ has to be copied and the path has to be lengthened with that literal. Note that there is only a restriction on the applicability of extension steps – reductions are still allowed when a path ends with a positive literal.

In Figure 1 there is one extension step, which is no longer allowed in restart model elimination, namely the extension of the path $\langle \neg Q, P \rangle$. Note that with our assumptions on goals, this path becomes $p = \langle \neg goal, \neg Q, P \rangle$ in restart model elimination. There is no reduction step possible and since $\mathit{leaf}(p)$ is positive, we lengthen p to $p' = \langle \neg goal, \neg Q, P, \neg goal \rangle$ in a restart step, which can finally be extended to the path multiset

$$\{\langle \neg goal, \neg Q, P, \neg goal, \neg P \rangle, \langle \neg goal, \neg Q, P, \neg goal, \neg Q \rangle\}$$

The complete restart model elimination refutation is depicted in Figure 2.

It is obvious that for reasons of efficiency a proof procedure based on this calculus must provide some refinements. For example, the use of lemmata might reduce the amount of redundancy introduced by restart steps. In the example from Figure 2 the restart led to the newly introduced paths ending with $\neg P$ and $\neg Q$, respectively. When processing the tree from left to right it is obvious that solving $\neg P$ would be unnecessary, since there is already a closed subtree containing $\neg P$ as a root; thus a proof procedure would benefit extremely from the possibility of using lemmata and/or caching [Astrachan and Stickel, 1992]. In our example, however, we are lucky, the same effect could be achieved by a reduction step. These and other topics concerning proof procedures are discussed in the following section.

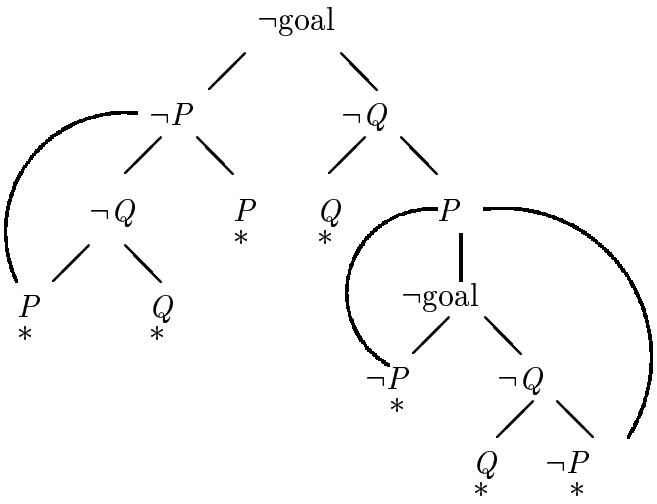


Figure 2: A Restart Model Elimination Refutation (positive *goal*-nodes are not displayed).

Selection Function

Now we weaken the calculus by introducing a selection function on head literals.

Definition 3.3 (Selection Function) A *selection function* f is a function that maps a clause $A_1, \ldots, A_n \leftarrow B_1, \ldots, B_m$ with $n \geq 1$ to an atom $L \in \{A_1, \ldots, A_n\}$. L is called the *selected literal* of that clause by f. The selection function f is required to be *stable under lifting* which means that if f selects $L\gamma$ in the instance of the clause $(A_1, \ldots, A_n \leftarrow B_1, \ldots, B_m)\gamma$ (for some substitution γ) then f selects L in $A_1, \ldots, A_n \leftarrow B_1, \ldots, B_m$.
Now let f be a selection function, and assume the following line additionally given between conditions 2 and 3 in the definition of *restart* (Def. 3.2).

2a L is selected in C by f.

This modified calculus is called *(restart) model elimination with selection function.* □

Assume there is a path p in our running example with $leaf(p) = \neg P$ and the selection function gives $f(P, Q \leftarrow) = Q$, then it is not allowed to perform an extension step with $P, Q \leftarrow$. Positive literals not selected by f can only be used within reduction steps. Note that the proof from Figure 2 is a proof with the above assumed selection function.

Strict Restart Model Elimination

As a further restriction we force the calculus to perform restarts whenever they are possible, i.e. if a leaf of a path is a positive literal it may not be used for a reduction step. Since for these leaves extension steps are possible only through a restart, we call this calculus strict restart model elimination. This restriction is motivated in several ways:

first, a comparable restriction is formulated within Plaisted's modified problem reduction format ([Plaisted, 1988], see also Section 5 below) and we would like to evaluate it within our framework; second, strict restart model elimination minimizes the search in ancestor lists, which occasionally results in shorter runtimes to find a proof. See also [Plaisted, 1990] for restrictions on accessing ancestor lists within a non-restart calculus.

Definition 3.4 (Strict Restart Model Elimination) The inference rule *reduction* is modified by adding the following line after condition 2 to the definition of *reduction* (Def. 2.1).

3. and $leaf(p)$ is a negative literal.

Such reduction steps are called *strict reduction steps*. *Strict restart model elimination* is defined to be the same as restart model elimination, except that "reduction step" is replaced by "strict reduction step". □

Alltogether we get the following inference rules in strict restart model elimination:

The inference rule *extension* is defined as follows:

$$\frac{\mathcal{P} \cup \{p\} \qquad L \vee R}{\mathcal{R}}$$

where

1. $\mathcal{P} \cup \{p\}$ is a path multiset, and $L \vee R$ is a variable disjoint variant of a clause in S; L is a literal and R denotes the rest literals of $L \vee R$.

1a. if $leaf(p)$ is positive then let $p := p \circ \langle first(p) \rangle$ in the following conditions 2 and 3.

2. $(leaf(p), L)$ is a connection with MGU σ.

3. $\mathcal{R} = (\mathcal{P} \cup \{p \circ \langle K \rangle \mid K \in R\})\sigma$.

The inference rule *reduction* is defined as follows:

$$\frac{\mathcal{P} \cup \{p\}}{\mathcal{P}\sigma}$$

where

1. $\mathcal{P} \cup \{p\}$ is a path multiset, and

2. there is a literal L in p such that $(L, leaf(p))$ is a connection with MGU σ, and

3. $leaf(p)$ is a negative literal.

For the rest of this section we deal with soundness and completeness of restart model elimination. For the soundness we refer to soundness of clausal tableau calculus, while completeness is proven directly.

Theorem 3.5 (Soundness) *Restart model elimination is sound.*

Proof. Soundness of the calculus follows immediately by showing that every restart model elimination proof can be mapped to a proof in the free variable semantic tableau calculus ([Fitting, 1990]). In particular, an *extension step* with a clause C is mapped to a β tableau extension step with clause C and subsequent application of an MGU for branch closure. For the *restart step* note first that a restart step must always be followed by an extension step for syntactical reasons. But then, if C is the clause used for that extension step, we can also carry out a β tableau extension step with C, also followed by closure with an MGU. Finally, *reduction steps* have a direct counterpart by closure with an MGU. □

Theorem 3.6 (Completeness) *Let f be a selection function and S be a clause set in goal-normal form. Then there exists a strict restart model elimination refutation of S with goal $\leftarrow$ goal and selection function f.*

Since a strict reduction step is by definition also a reduction step we obtain as a corollary the completeness of the non-strict restart model elimination.

Since a selection function restricts the set of permissible derivations, completeness without selection function follows immediately from completeness with selection function.

The proof of the theorem proceeds by assuming a set S^g of ground instances of clauses of S which is unsatisfiable. Such a set exists according to the Skolem-Herbrand-Löwenheim theorem (see e.g. [Gallier, 1987] for a proof). Then by ground completeness (which is to be proven below) a refutation on the ground level exists. Although not quite trivial, lifting can be carried out by using standard techniques (see e.g. [Chang and Lee, 1973, Lloyd, 1987]). In particular, by stability under lifting (Def. 3.3) it is guaranteed that the selection function will select on the first order level a literal whose ground instance was selected at the ground level.

Ground completeness reads as follows:

Lemma 3.7 (Ground Completeness) *Let f be a selection function and S be an unsatisfiable ground clause set in goal-normal form. Then there exists a strict restart model elimination refutation with selection function f of S with goal clause $\leftarrow$ goal.*

Proof. Informally, the proof is by splitting the non-Horn clause set into Horn sets, assuming by completeness of model elimination refutations without reduction steps, and then assembling these refutations into the desired restart model elimination refutation. There, reduction steps come in by replacing extension steps with split unit clauses by reduction steps to the literals where the restart occurred.

For the formal proof some terminology is introduced: we say that a path multiset $\mathcal{P}$ "contains (an occurrence of) a clause $A_1, \ldots, A_n \leftarrow B_1, \ldots, B_m$" iff for some path p it holds

$$\{p \circ \langle A_1 \rangle, \ldots, p \circ \langle A_n \rangle, p \circ \langle \neg B_1 \rangle, \ldots, p \circ \langle \neg B_m \rangle\} \subseteq \mathcal{P}$$

If we speak of "replacing a clause C in a derivation by a clause $C \cup D$" we mean the derivation that results when using the clause $C \cup D$ in place of C in extension steps. Also, the same literal $L \in C$ must be used to constitute the connection.

By a "derivation of a clause C" we mean a derivation that ends in a path multiset which contains (several occurrences of) the clause C.

Let $k(S)$ denote the number of occurrences of positive literals in S minus the number of definite clauses[2] in S ($k(S)$ is related to the well-known *excess literal parameter*). Now we prove the claim by induction on $k(S)$.

Induction start ($k(S) = 0$): M must be a set of Horn clauses. By Theorem 3.1 there exists a model elimination refutation of S with goal $\leftarrow$ *goal* without reduction steps. Furthermore, for syntactical reasons, in every extension step only the single positive literal (and never a negative literal) of the extending clause can be selected. Thus, this refutation is also a strict restart model elimination refutation.

Induction step ($k(S) > 0$): As the induction hypothesis suppose the result to hold for unsatisfiable ground clause sets S' in goal-normal form with $k(S') < k(S)$.

Since $k(S) > 0$, S must contain a non-Horn clause

$$C = A_1, A_2, \ldots, A_n \leftarrow B_1, \ldots, B_m$$

with $n \geq 2$. W.l.o.g. assume that A_1 is the literal selected by f in C. Now define n sets

$$\begin{aligned} S_1 &:= (S \setminus C) \cup \{A_1 \leftarrow B_1, \ldots, B_m\} \\ S_2 &:= (S \setminus C) \cup \{A_2\} \\ &\vdots \\ S_n &:= (S \setminus C) \cup \{A_n\} \end{aligned}$$

Every set S_i ($i = 1 \ldots n$) is unsatisfiable (because otherwise, a model for one of them would be a model for S). Furthermore, it holds $k(S_i) = k(S) - n + 1 < k(S)$. Thus, by the induction hypothesis there exist a strict restart model elimination refutations R_i with goal clauses $\leftarrow$ *goal* of S_i, respectively.

Now consider R_1 and replace in R_1 every occurrence of the clause $A_1 \leftarrow B_1, \ldots, B_m$ by C. Call this derivation R_1'. Since A_1 is the sole positive literal in $A_1 \leftarrow B_1, \ldots, B_m$, A_1 must have been selected in the extension steps with that clause in R_1. Thus the corresponding extension steps in R_1' with C are legal in the sense of the restriction to the selection function.

R_1' is a derivation of, say, k_j occurrences of the positive unit clauses A_j ($j = 2 \ldots n$) from the input set S. Now every A_j can be eliminated from the input set S according to the following procedure: for $j = 2 \ldots n$ append R'_{j-1} k_j times with the refutation R_j, thus introducing k_j restart steps at the paths ending in A_j. Note here that the restart step produces exactly the same goal clause $\leftarrow$ *goal* as is in R_j. Let R_j'' be the refutation resulting from these k_j restart steps at leaf A_j. R_j'' is a restart model elimination refutation of $S \cup \{A_j, \ldots, A_n\}$. In order to turn R_j'' into a refutation of $S \cup \{A_{j+1}, \ldots, A_n\}$, replace every extension step with A_j in the appended refutations R_j by a reduction step to the positive path literal A_j. The resulting refutation R_j' is a desired strict model elimination refutation of $S \cup \{A_{j+1}, \ldots, A_n\}$.

Finally, R_n' is the desired strict restart model elimination refutation of S alone. □

[2] A *definite clause* is a clause containing exactly one positive literal.

Regularity in Restart Model Elimination.

Regularity means for ordinary model elimination that it is never necessary to construct a tableau where a literal occurs more than once along a path. Expressed more semantically, it says that it is never necessary to repeat in a derivation a previously derived subgoal (viewing open leaves as subgoals).

Regularity, or an approximative implementation of it, tends to be one of the more useful refinements of model elimination. Unfortunately, regularity is *not* compatible to restart model elimination. This can be seen easily as the *goal*-literal is copied in restart steps, thus violating the regularity restriction. Hence at least the goal literal has to be excluded from the regularity restriction, because otherwise restart steps are impossible! But even with this exception completeness is lost in general, since after a restart step it might be necessary to repeat – in parts – a proof found so far up to the restart step.

However the following observations allows to define a somewhat weaker notion of regularity: First, the proof of Lemma 3.7 proceeds by splitting the input clause set into Horn sets and then assembles the existing non-restart refutations $R_1, \ldots R_n$ into a restart refutation R'_n. Since this assembling is done "blockwise" (the R_is are not interleaved among each other and keep their structure) some properties of the R_is carry over to their respective occurrence in R'_n. In particular, the *regularity* of the R_is carries over in this way. Hence we define a path as *blockwise regular (version 1)* iff every pair of occurrences of identical literals (unequal to $\neg goal$) is separated by at least one occurrence of the literal $\neg goal$. A derivation is is called *blockwise regular* iff every path in every of its path multisets is blockwise regular. From these considerations and definitions it follows with Lemma 3.7:

Lemma 3.8 *Restart model elimination is complete when restricted to blockwise regular refutations (version 1).*

Since all the R_is are refutations of *Horn* clause sets this regularity restriction applies only to *negative* literals along a path. Thus we might derive a path $p = \neg goal \cdots A \cdots \neg goal \cdots A$ which is blockwise regular. We wish to extend blockwise regularity to forbid such duplicate occurrences of positive literals, and say that a branch is *positive regular* iff all the positive literals occurring in it are pairwise distinct (not identical). Extending the preceding definition, we define a branch to be *blockwise regular (final version)* iff it is blockwise regular (version 1) and positive regular. Fortunately it holds:

Theorem 3.9 *Restart model elimination is complete when restricted to blockwise regular refutations (final version).*

Proof. By the preceding lemma it remains to prove the restriction to positive regular paths. Again, this can be done by analyzing the proof of Lemma 3.7: while for the induction start positive regularity is trivial, for the induction step the following observation can be used: consider the splitted sets $S_2, \ldots, S_n$ and the associated unit clauses $A_2 \in S_2, \ldots, A_n \in S_n$. Without loss of generality the S_is can be supposed to be *minimal* unsatisfiable. Now, if $A_i \in S_i$ then S_i does not contain a disjunctive clause of the form $C = A_i, B_1, \ldots, B_k \leftarrow C_1, \ldots, C_n$ where $k \geq 1$ (because otherwise S_i were not minimal

unsatisfiable). By a similar line of reasoning it holds that if $A_i \notin S_i$ then S_i does also not contain such a clause C. In any case, for this syntactical reason, a branch containing A_i cannot come up in the refutation R_i of S_i (which can be supposed to be regular by the induction hypothesis). Thus, in the assembling of the R_is into the final refutation R'_n, the occurrence of A_i which caused the restart step remains the *only* occurrence of A_i. In other words, positive regularity holds for this branch, and hence more generally also for the whole refutation R'_n. □

4 PTTP without Contrapositives

As exemplified by PTTP ("Prolog Technology Theorem Prover") [Stickel, 1988, Stickel, 1989], Prolog can be viewed as an "almost complete" theorem prover, which has to be extended by only a few ingredients in order to handle the non-Horn case. By this technique the benefits of optimizing Prolog compilers are accessible to theorem proving. First we will briefly review the standard approach, and then we will describe the necessary modifications to obtain restart model elimination.

The PTTP-approach transforms a given clause set into a Prolog program. The transformed Prolog program must execute the clauses according to some complete proof procedure. *Model elimination* turns out to be particularly useful for this, since it is, like Prolog, an input proof procedure. In particular, the transformation from the input clauses to Prolog works as follows (see [Stickel, 1988] again for details):

- An input clause such as

  $$C \leftarrow A \wedge B$$

  is transformed into a Prolog clause

  ```
  (1)     c :- a, b.
  ```

  Additionally, since in the model elimination calculus every literal in a clause can equally well serve as an entry point into the clause, all contrapositives are needed. In this case these are

  ```
  (2)     not_a :- not_c, b.
  (3)     not_b :- a, not_c.
  ```

  This example also shows how negation is treated, namely by making it part of the predicate name.

- Prolog's unsound unification has to be replaced by a sound unification algorithm. This can either be done by directly building-in sound unification into the Prolog implementation, or by reprogramming sound unification in Prolog and calling this code instead of Prolog's unsound unification.

- A complete search strategy is needed. Usually depth bounded iterative deepening is used. The strategy can be compiled into the prolog program by additional parameters, being used as "current depth" and "limit depth". The cost of an extension

step can be uniformly 1 (depth bounded search), or can be proportional to the length of the input clause (inference bounded search).

- The model elimination reduction operation has to be implemented. This can be realized by memorizing the subgoals solved so far (the A-literals) as a list in an additional argument, and by Prolog code that checks a goal for a complementary member of that list. Of course, this check has to be carried out with sound unification.

  The Prolog clause (1) from above then looks like

  ```
  (1')     c(Anc) :- a([-a|Anc]), b([-b|Anc]).
  ```

  where `Anc` is a Prolog list which contains the ancestor literals (called A-literals in [Loveland, 1968]); code for reduction steps then looks like

  ```
  (Red-C)        c(Anc) :- member(c,Anc).
  (Red-notC)     not_c(Anc) :- member(-c,Anc).
  ```

Next we will turn to restart model elimination. We assume the clause set to be in *goal*-normal form. In the following transformation the same precautions regarding sound unification and complete search strategy have to be obeyed. The rest of the transformation then works as follows:

- For *restart model elimination with selection function* every input clause

  $$A_1, \ldots, A_n \leftarrow B_1, \ldots, B_m$$

  with selected literal A_i is transformed into the Prolog clause

  ```
  (C_i)     a_⟨i⟩(Anc)  :-  b_1([-b_1|Anc]),...,b_m([-b_m|Anc]),
                            solve_pos(a_1,Anc),
                              ⋮
                            solve_pos(a_⟨i − 1⟩,Anc),
                            solve_pos(a_⟨i + 1⟩,Anc),
                              ⋮
                            solve_pos(a_n,Anc).
  ```

  Here, the call to `solve_pos(`*Literal*`,`*AncestorList*`)` means to solve a positive literal in the context of the given ancestor list. This can be done according to the inference rules of restart model elimination either by a reduction step or by a restart step. The following code mirrors this:

  ```
  (Reduction)   solve_pos(Lit,AncList)  :-  member(-Lit,AncList).
  (Restart)     solve_pos(Lit,AncList)  :-  goal([Lit|AncList]).
  ```

  Since we assume *goal*-normal form, the original goal clause – namely $\leftarrow$ *goal* – is known a priori. Thus, instead of copying in restart steps the root literal, it is equivalent to hard-code a call to `goal` into the body of the Prolog clause. Note that

for Horn clauses only the single "natural" contrapositive is produced. In the above example only the contrapositive (1) is produced, while (2) and (3) are not.

Obviously, the ordering of subgoals in this transformation can be altered, thus allowing for different search strategies.

Furthermore, code for reduction steps has to be added as in the above transformation.

- If *strict* restart model elimination is to be implemented, only reduction steps from negative literals towards positive ancestor literals are required. Consequently, negative ancestor literals need not to be stored. Since reduction steps from positive literals are not carried out, the (Reduction) clause can be deleted. But then the calls to the (Restart) clause can be unfolded. Altogether, this simplifies the transformation to the following format (for the same clause as above, and also with A_i being selected):

```
(C_i)    a_⟨i⟩(Anc)  :-  b_1(Anc),..., b_m(Anc),
                         goal([a_1|Anc]),
                           ⋮
                         goal([a_⟨i − 1⟩|Anc]),
                         goal([a_⟨i + 1⟩|Anc]),
                           ⋮
                         goal([a_n|Anc]).
```

- Finally, for both versions the Prolog program gets its query

  ```
  (Q)    ?- goal([]).
  ```

  Even in the case of strict restart model elimination it is not necessary to include `-goal` in the ancestor list, since due to the construction it cannot be subject to a reduction step.

Experimental Results

The aim of this section is just to demonstrate that restart model elimination is more than a theoretical concept. We want to present some first experiments, which we interpret to be encouraging enough to follow this way.

Theorem	Restart Model Elimination w/o Selection	Restart Model Elimination w. Selection	Strict Restart Model Elimination w/o Selection	Strict Restart Model Elimination w. Selection	MPRF (Normalized)
Non-Obvious	**13.6**	21.7	> 1000	> 1000	257.7
	0.7 [1]				6.5 [3]
Eder45	3.3	2.0 [4]	3.2	14.7 [4]	
		19.4 [4]		19.7 [4]	
	1.7 [2]	1.7 [2]	**1.6** [2]	8.8 [2]	
		0.88 [1,2]		8.2 [1,2]	
Eder58	> 1000	> 1000	> 1000	> 1000	
	152.8 [2]	**45.9** [2]	428.9 [2]		
	45.9 [1,2]	11.0 [1,2]			
Steamroller	**4.9**	6.8	> 1000	> 1000	4492
					29.0 [3]
$x \neq 0 \rightarrow$	0.21	**0.083** [4]	0.20	0.084 [4]	
$x^2 > 0$		> 1000 [4]		> 1000 [4]	
Bledsoe1	132.8	12.0	> 1000	> 1000	
		11.5 [2]			
Bledsoe2	> 1000	45.8	> 1000	> 1000	
		43.1 [2]			
Natnum3	18.0	0.38	11.2	**0.26**	
	0.08 [1]				
Wos4	**109.3**	> 1000	> 1000	> 1000	
	23.8 [2a]				
Wos 11 (Horn)				**10.6**	214.5
					205.7 [3]
Lukasiewitz (Horn)				> 1000	≈ 285
				73.8 [2]	

Remarks: 1 – With (incomplete) regularity restriction . 2 – With lemmas. 2a – With anti-lemmas. 3 – "nosave" flag cleared. 4 – Depends from selected literal.

Figure 4: Runtime results (in seconds) for various provers.

We implemented the restart model elimination calculus in the way described above in the theorem proving system PROTEIN ([Baumgartner and Furbach, 1994]). Both the implementation language and the target language for the compiled code is ECL^{i}PS^e, an enhanced Prolog-dialect, running on a SPARC/2. In the implementation we made use of only one non-standard Prolog feature of ECL^{i}PS^e, namely sound unification.

We ran several examples known from the literature, and some new ones. We compared several versions of the prover, varying in strict restart model elimination vs. (non-strict) restart model elimination, and selection function vs. no selection function; the first four

Theorem	**Restart Model Elimination** (No Contrapositives!)	**ME** PTTP (Contrapositives)	**ME** Setheo (Contrapositives)
Eder58	45.9	11.5	50.0
Steamroller	4.9	0.93	0.43
$x \neq 0 \rightarrow x^2 > 0$	0.083	15.2	2.87
Bledsoe1	11.5	> 1000	130.0
Bledsoe2	43.1	> 1000	22.7
Natnum3	0.26	0.08	0.08

Figure 4: No-Contrapositive Provers vs. Contrapositive Provers

columns in Figure 4 contain the runtime results. Column 5 (**MPRF**) contains the results for the *Modified Problem Reduction Format* prover (see also the section on related work below). The option "nosave flag cleared" means that caching is enabled. The data, taken from [Plaisted, 1988], were obtained on a SUN 3 workstation, whereas the other provers ran on a SUN Sparc Station 2. Hence, for normalisation the times for the MPRF prover were divided by 7.

Optionally, the provers written by us extend the base calculi by the following features: *(Unit)-lemmas* (currently *all* lemma candidates are stored), *Anti-Lemmas* (i.e. failure to prove a goal within a given depth bound is recorded), *ground reduction steps* (in reduction steps where no substitution is involved no further proof alternative needs to be explored), *blockwise regularity* as defined at the end of Section 3.

Unless otherwise noted in Figure 4, only the ground-reduction flag was set in our provers; for iterative deepening, the threshold was increased in each iteration by 1, and each extension step was uniformly charged with a cost of 1.

Furthermore, we also found it interesting to run standard model elimination provers which use contrapositives. These experiments given in Figure 4 demonstrate that in some cases, namely the examples from real-analysis, restart model elimination results in better performance, whereas in the usual benchmarks the results with restart based procedures are in the same order of magnitude.

The one referred to as **ME-PTTP** in column 2 is a PTTP-prover closely following the ideas from [Stickel, 1988, Stickel, 1989] implemented in the same environment as our restart provers (in fact, it requires only setting a flag to switch calculi), and hence runtime results are easily comparable. Column 3 (**Setheo**) is the well-known Setheo prover [Letz *et al.*, 1992] in its latest version (Version 3.0). Setheo was run in its default mode, which then makes use of the following refinements and constraints: subgoal reordering, purity, antilemmas, regularity, tautology and subsumption,

The examples run were the following:
The one referred to as *Non-Obvious* is taken from the October 1986 Newsletter of the

Association of Automated Reasoning and consists of the following clauses:

$$\begin{array}{rcl} & \leftarrow & p(a,b) \\ & \leftarrow & q(c,d) \\ p(X,Y), q(X,Y) & \leftarrow & \\ p(X,Z) & \leftarrow & p(X,Y), p(Y,Z) \\ q(X,Z) & \leftarrow & q(X,Y), q(Y,Z) \\ q(X,Y) & \leftarrow & q(Y,X) \end{array}$$

The paper [Astrachan and Stickel, 1992] investigates caching and lemmaizing techniques within Horn problems. This example, however, is included as one of a few non-Horn problems. The report is 653 secs for the METEOR prover, and 1.42 secs with lemmaizing.

The *Eder*$\langle m \rangle\langle n \rangle$ examples consist of the clauses

$$\begin{array}{rcl} & \leftarrow & p(X), p(f^m(X)) \\ p(X), p(f^n(X)) & \leftarrow & \end{array}$$

For the *steamroller* example we used the formulation from [Manthey and Bry, 1988]. Although this example seems to be a domain of the standard model elimination provers, restart model elimination behaves quite well.

The example referred to by $x \neq 0 \rightarrow x^2 > 0$ is to prove this theorem (x is universally quantified) from calculus, partially given by the formulas

$$\begin{array}{llll} X > 0 \quad \vee \quad X = 0 \quad \vee \quad -X > 0 & & & \text{Axiom.} \\ X > 0 \quad \wedge \quad Y > 0 \quad \rightarrow \quad X + Y > 0 & & & \text{Axiom.} \\ X > 0 \quad \wedge \quad Y > 0 \quad \rightarrow \quad X * Y > 0 & & & \text{Axiom.} \\ -X * -Y = X * Y & & & \text{Lemma.} \end{array}$$

where $=$ is equality and $>$ is a strict ordering. The transformation to clauses was done in a way avoiding the use of equality as much as possible, much as in [Wos, 1988].

The *Bledsoe* examples are the first two of the five given in [Bledsoe, 1990].

Natnum3 is more relevant to logic programming; it computes in simple integer arithmetic:

$$\begin{array}{rcl} & \leftarrow & even(s(s(s(s(s(s(s(s(s(0)))))))))) \\ & \leftarrow & odd(0) \\ even(s(X)) & \leftarrow & odd(X) \\ odd(s(X)) & \leftarrow & even(X) \\ even(X), odd(X) & \leftarrow & \end{array}$$

The *Wos4* example is a well known example from group theory [Wos, 1988]. Finally, *Wos11* and *Łukasiewicz* are two Horn examples. The latter is:

$$\begin{array}{rcl} & \leftarrow & th(i(i(i(a,b),a),a)) \\ th(Q) & \leftarrow & th(P), th(i(P,Q)) \\ th(i(i(i(X,Y),Z),i(i(Z,X),i(U,X)))) & \leftarrow & \end{array}$$

Now let us summarize the results. Compared among each other each of the 4 versions of restart model elimination has its justification by dedicated examples. In the parameter

space *non-strict restart* vs. *strict restart* model elimination we prefer as the default strategy the *non-strict* version, since whenever the *strict* version found a proof in reasonable time, the *non-strict* version did as well; but on most examples the *strict* version failed or behaved poorly, while the other version found a proof (*Non-Obvious*, *Eder58*, *Bledsoe1*, *Bledsoe2*, *Steamroller*, *Wos4*).

In the parameter space *selection function* vs. *no selection function* we will not strictly prefer the one to the other. Note the extreme dependence on the "right" choice in the $x \neq 0 \rightarrow x^2 > 0$-example, while in the Eder-examples the right choice is not that crucial. On the other side, the Wos4 and Steamroller example obviously require the use of several contrapositives.

From this results we learn that the selection function should be carefully determined. Here, heuristics are conceivable such as "always select a biggest (in some ordering) head literal" in order to work in a decreasing direction. The selection function can even be determined dynamically within the bounds of Definition 3.3.

Currently, the user has to supply the selection function for a given input clause. This function is inherited to all instances of that clause. Here we see potential for further improvements.

But even at the moment our restart provers can well compete with the MPRF prover, which is according to our classification, closest to the strict restart prover with selection function (see also the next section). Since caching helps a lot in MPRF as in *Non-Obvious* and *Steamroller*, we also intend to include it in our provers (see also [Astrachan and Stickel, 1992] for practical results on caching). On the other hand, as our experiments suggest, the MPRF prover might benefit a lot from relaxing the calculus towards the absence of a selection function and away from strict reduction steps.

Ordinary model elimination as implemented by ME-PTTP and Setheo is sometimes faster than restart model elimination. This also holds for some examples from [Chang and Lee, 1973] and surely several others. In the case of Setheo this may especially be due to the numerous refinements not present in the other provers. However there are enough interesting cases where restart model outperforms ordinary model elimination. Note in particular the dramatic speedup for the example from calculus.

5 Related Work

Connection Method. The *connection method* [Bibel, 1987] is an analytic calculus closely related to model elimination. Clause sets are called *matrices* there, and a *path through a matrix* is obtained by taking exactly one literal from every clause in the matrix. The method proceeds by systematically checking all paths through the matrix to contain complementary literals. If this is the case, a refutation has been found.

A somewhat higher-level formulation of the connection method can be found in [Eder, 1992], and in [Baumgartner and Furbach, 1993] we showed that this connection method can, in steps, simulate model elimination. The converse, however, is not true for the following essential difference between the connection method and model elimination: in model elimination in extension steps a complementary pair of literals (called *connection*)

must be established between the *leaf* literal where the extension occurred and some literal of the extending clause. In the connection method this restriction is dropped, and so every literal along the path (or even none) may be part of the connection.

This property is also the key for the observation stated in the introduction, namely that the connection method is complete without the use of contrapositives. In order to see this, recall that a restart step consists of copying the first literal of the path, followed by an extension step. Thus, copying is not necessary if the first literal in the path is accessible for the connection — as is the case in the connection method. Hence we get as a corollary to theorem 3.6, the completeness of strict restart model elimination with selection function:

Corollary 5.1 *The connection method is complete for input sets in goal-normal form, even if no contrapositives are used.*

Problem Reduction Formats. In [Plaisted, 1988] two calculi named *simplified problem reduction format* and *modified problem reduction format* are described. They are goal-oriented, and neither of these needs contrapositives. We will discuss both of them.

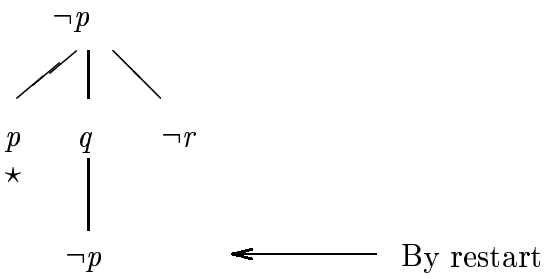
The *simplified problem reduction format* (SPRF) is a variant of the Gentzen sequent calculus (see e.g. [Gallier, 1987]). A sequent is pair, written as $\Gamma \rightarrow L$ where Γ is a list of literals, and L is a literal. From the model elimination point of view a sequent $\Gamma \rightarrow L$ corresponds to the path $\Gamma \circ \langle \neg L \rangle$, i.e. the goal L is to be proven in the context (ancestor list) Γ.

Clauses are translated to inference rules operating on sequents; a clause $L \leftarrow L_1, \ldots, L_n$ is translated into the inference rule

$$\frac{\Gamma \rightarrow L_1 \qquad \cdots \qquad \Gamma \rightarrow L_n}{\Gamma \rightarrow L}$$

where Γ is a variable.

The interesting case is to see how model elimination restart steps can be mapped to derivations in SPRF. Suppose we have in a restart model elimination derivation a leaf $\neg p$ and wish to extend with the clause $p, q \leftarrow r$. After copying, the situation looks as follows:



This situation can be mirrored in SPRF by the following partial proof:

$$\dfrac{\dfrac{not(q) \rightarrow not(q) \qquad \mathbf{not(q) \rightarrow r}}{not(q) \rightarrow p}\, R_1 \qquad \mathbf{q \rightarrow p}}{\rightarrow p}\, Split .$$

The *Split* rule is in effect the cut-rule, and R_1 stems from the clause $p, q \leftarrow r$. While the sequent $not(q) \rightarrow not(q)$ is an instance of an axiom, the boldface sequents are unproved. Note the close relationship to restart model elimination: the sequent $q \rightarrow p$ immediately corresponds to the goal $\neg p$ with ancestor list $\neg p \circ q$ in restart model elimination; it is even identical in strict restart model elimination, as negative ancestors need not be stored. For the other sequent $not(q) \rightarrow r$ note that the corresponding goal $\neg r$ in restart model elimination does not have the ancestor $not(q)$. If additional information – such as $not(q)$ – is considered as an advantage for proof finding, this is a shortcoming of restart model elimination. The situation however can easily be repaired either by an explicit change to the calculus, or by incorporating a more general *factorization* rule[3].

In this way, restart model elimination steps can be mapped to partial SPRF proofs. The converse, however, is not true. This is due to the fact that the splitting rule can be applied in every proof situation, i.e. to every sequent derived along a proof. In other words, a case analysis p or $\neg p$ can be carried even to goals totally unrelated to p.

Thus, in sum, the restart model elimination is more restricted than SPRF.

The *modified problem reduction format* (MPRF) avoids the problem of uncontrolled application of the splitting rule. This is formally carried out by an additional syntactical layer between "sequents" and "inference rules". An alternate presentation in tree format was defined by [Mellouli, 1990]. Similar mappings as carried out for the SPRF convince us that the MPRF is operationally closely related to strict restart model elimination. More precisely, MPRF explained in our terminology means that a restart step may occur with *any* negative literal along the current path, not just the topmost *goal* literal. Plaisted suggested to try for the proof search all negative literals, beginning from the current leaf and processing towards the root node. The advantage of doing so is to avoid useless re-solving of a goal too near to the root. But, in general, it cannot be predicted which of these goals results in a succesfull proof. Hence, in the worst case the topmost *goal* literal has to be used for a restart step eventually, and all previous proof attempts are useless.

Another notable difference between restart model elimination and MPRF is that restart model elimination enables the use of the negative literals along the paths for reduction steps. As our experiments show this is quite valuable.

Near-Horn Prolog. As already mentioned in the introduction, there is a close relation to Loveland's Near-Horn Prolog, especially to the InH-Prolog variant from [Loveland and Reed, 1989]. Instead of one tableau in our model elimination calculi, Inh-Prolog deductions consist of a sequence of Prolog-like computations, called blocks. The activation of

[3]Factorization means that a branch may be closed if its leaf is identical to some brother node of a predecessor of this leaf.

such blocks corresponds to our restart extension steps. If we agree that Prolog stepwisely transforms a goal set `G` into the empty goal set, then the Prolog-like computations in InH-Prolog deal with triples of the form `G # A { D }`. Here, the list `A` is called *active heads* and the list `D` is called *deferred heads*. These components can easily be explained from the viewpoint of restart model elimination: the active heads `A` corresponds to the *positive* literals of the path in restart model elimination which was most recently selected for a restart step; consequently, since `A` is a left-ended stack the leftmost literal in `A` is the literal which caused the restart step. In the Prolog-like restart blocks every literal in `A` may be used in the role of a unit input clause ("cancellation step) in order to get rid of a goal literal. The deferred heads `D` correspond to the remaining positive leaf literals of the path multiset; they will cause new restart blocks (or restart steps) at a later time.

Let us compare our refutation from Figure 2 with the following InH-Prolog refutation. In this example no deferred head occurs.

```
?- GOAL
:- P,Q
:- Q,Q            % factorisation to simplify presentation!
:- Q # P          % P from disjunctive clause is deferred
:-   # P          % Block finished

restart
?-GOAL # P
:- P,Q # P        % cancellation (reduction)
:- Q # P
:- P # P          % cancellation
:-   # P          %
```

The cancellation steps in this derivation correspond to the two reduction steps in the right subtree of Figure 2. The derivation from the left subtree does not have a counterpart in the above InH-Prolog refutation, because of the factorisation step we performed in the first block; this, of course, would have been possible in the restart model elimination refutation.

The reduction steps starting from positive leaf-literals have no counterpart in InH-Prolog - within a block there are only extension or cancellation steps. The latter correspond to reductions with a negative leaf-literal.

On the other side, the concept of a *strong cancellation pruning rule* of InH-Prolog has (so far) no counterpart in restart model elimination. By this rule, a certain class of refutations is discarded. Stated positively, and in the terminology of restart model elimination, only those refutations are acceptable in which a literal which caused a restart step is used in a (any) subsequent reduction step. Thus restart steps not relevant for the proof are filtered out. The completeness of this restriction can be seen again by analyzing the completeness proof of Lemma 3.7. In brief, a restart step applied to $A, B \leftarrow C$ causes by the splitting rule Horn refutations with $A \leftarrow C$ and B. Now, if the given clause set is supposed (without loss of generality) to be *minimal* unsatisfiable, then also the splitted

sets contain minimal unsatisfiable subsets containing $A \leftarrow C$ and B, respectively. Hence these clauses must be used in the Horn refutations, and consequently, the restart step occurring at B must be followed by a reduction step to B.

Summarizing on all these considerations we conclude that InH-Prolog is a variant of strict restart model elimination without selection function. As a consequence we see that our PTTP implementation of strict restart model elimination is an implementation of InH-Prolog.

SLWV-Resolution. In [Pereira *et al.*, 1991] a theorem prover that retains the procedural aspects of logic programming is defined. This so-called SLWV resolution system is based on SL-resolution, a linear resolution format. SLWV saves contrapositives and uses case analysis as an additional inference rule. To this end the usual resolution step from SL-resolution is modified such that besides the current goal any ancestor is allowed to be expanded. In our terminology this would mean that every negative literal along a path can be copied in a restart step. As the authors of [Pereira *et al.*, 1991] explain, this freedom clearly increases the search space when compared to Near-Horn Prolog in the case of near-Horn problems. As a further difference to our restart model elimination, SLWV-Resolution needs a completely new framework. Pereira et.al. had to redesign the PTTP-implementation technique for their prover, whereas we were able to implement restart model elimination by a small change of our existing prover.

References

[Astrachan and Stickel, 1992] Owen L. Astrachan and Mark E. Stickel. Caching and Lemmaizing in Model Elimination Theorem Provers. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction (CADE-11)*, pages 224–238. Springer-Verlag, June 1992. LNAI 607.

[Baumgartner and Furbach, 1993] P. Baumgartner and U. Furbach. Consolution as a Framework for Comparing Calculi. *Journal of Symbolic Computation*, 16(5), 1993.

[Baumgartner and Furbach, 1994] P. Baumgartner and U. Furbach. PROTEIN: A *PRO*ver with a *T*heory *E*xtension *I*nterface. In A. Bundy, editor, *Automated Deduction – CADE-12*, volume 814 of *LNAI*, pages 769–773. Springer, 1994.

[Baumgartner, 1992] P. Baumgartner. A Model Elimination Calculus with Built-in Theories. In H.-J. Ohlbach, editor, *Proceedings of the 16-th German AI-Conference (GWAI-92)*, pages 30–42. Springer, 1992. LNAI 671.

[Bibel, 1987] W. Bibel. *Automated Theorem Proving.* Vieweg, 2nd edition, 1987.

[Bledsoe, 1990] W. W. Bledsoe. Challenge Problems in Elementary Calculus. *Journal of Automated Reasoning*, 6:341–359, 1990.

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

[Eder, 1991] E. Eder. Consolution and its Relation with Resolution. In *Proc. IJCAI '91*, 1991.

[Eder, 1992] E. Eder. *Relative Complexities of First Order Languages.* Vieweg, 1992.

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

[Gabbay, 1985] D. M. Gabbay. N-Prolog: An extension of Prolog with hypothetical implication II. logical foundations, and negation as failure. *The Journal of Logic Programming*, 2(4):251–284, December 1985.

[Gallier, 1987] J. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving.* Wiley, 1987.

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

[Lloyd, 1987] J. Lloyd. *Foundations of Logic Programming.* Symbolic Computation. Springer, second, extended edition, 1987.

[Loveland and Reed, 1989] D.W. Loveland and D.W. Reed. A near-horn prolog for compilation. Technical Report CS-1989-14, Duke University, 1989.

[Loveland, 1968] D. Loveland. Mechanical Theorem Proving by Model Elimination. *JACM*, 15(2), 1968.

[Loveland, 1991] D. Loveland. Near-Horn Prolog and Beyond. *Journal of Automated Reasoning*, 7:1–26, 1991.

[Manthey and Bry, 1988] R. Manthey and F. Bry. SATCHMO: a theorem prover implemented in Prolog. In *Proc. 9th CADE.* Argonnee, Illinois, Springer LNCS, 1988.

[Mellouli, 1990] Taïeb Mellouli. A Tree Representation of the Modified Problem Reduction and its Extension to Three-valued Logic. KI-NRW 90-19, Universität Duisburg, FB 11 - Praktische Informatik, 1990.

[Pereira *et al.*, 1991] Luis Moniz Pereira, Luis Caires, and Jos'e Alferes. SLWV – A Theorem Prover for Logic Programming. AI Centre, Uninova, Monte da Caparica, Portugal, July 1991.

[Plaisted, 1988] D. Plaisted. Non-Horn Clause Logic Programming Without Contrapositives. *Journal of Automated Reasoning*, 4:287–325, 1988.

[Plaisted, 1990] D. Plaisted. A Sequent-Style Model Elimination Strategy and a Positive Refinement. *Journal of Automated Reasoning*, 4(6):389–402, 1990.

[Reed and Loveland, 1992] D. W. Reed and D. W. Loveland. A Comparison of Three Prolog Extensions. *Journal of Logic Programming*, 12:25–50, 1992.

[Stickel, 1988] M. Stickel. A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.

[Stickel, 1989] M. Stickel. A Prolog Technology Theorem Prover: A New Exposition and Implementation in Prolog. Technical note 464, SRI International, 1989.

[Wos, 1988] L. Wos. *Automated Reasoning: 33 Basic Research Problems.* Prentice Hall, 1988.