UNIVERSITÄT
KOBLENZ · LANDAU

Refinements for Restart Model Elimination

Peter Baumgartner, Ulrich Furbach

24/96

Universität Koblenz-Landau Fachbereich Informatik infko

Fachberichte
INFORMATIK

Universität Koblenz-Landau
Institut für Informatik, Rheinau 1, D-56075 Koblenz

E-mail: researchreports@infko.uni-koblenz.de,
WWW: http://www.uni-koblenz.de/universitaet/fb/fb4/

Refinements for Restart Model Elimination

Peter Baumgartner · Ulrich Furbach
Universität Koblenz · Institut für Informatik
Rheinau 1 · D–56075 Koblenz · Germany
E-mail: {peter,uli}@informatik.uni-koblenz.de

Abstract

We introduce and discuss a number of refinements for restart model elimination (RME). Most of these refinements are motivated by the use of RME as an interpreter for disjunctive logic programming. Especially head selection function, computation rule, strictness and independence of the goal clause are motivated by aiming at a procedural interpretation of clauses. Other refinements like regularity and early cancellation pruning are techniques to handle the tremendous search space. We discuss these techniques and investigate their compatibility. As a new result we give a proof of completeness for RME with early cancellation pruning; we furthermore show that this powerful refinement is compatibel with regularity tests.

Keywords: Theorem proving, restart model elimination, disjunctive logic programming, pruning techniques.

1 Introduction

Restart Model Elimination (RME) has been introduced as a variant of model elimination in [Baumgartner and Furbach, 1994a] as a calculus which avoids contrapositives and which introduces case analysis. The restart modification has been successfully incorporated into high performance theorem provers like PROTEIN [Baumgartner and Furbach, 1994b] and SETHEO. In [Baumgartner *et al.*, 1995] it was demontstrated that variants of this calculus can be used for computing answers for disjunctive logic programming.

In this paper we want to present various refinements of RME and we discuss their interrelationship. It turns out that some refinements very well support the procedural reading of disjunctive program clauses $A_1 \vee \ldots \vee A_m \leftarrow B_1 \wedge \ldots \wedge B_n$, these are e.g. head selection function or strictness. Other refinements, e.g. regularity, independance of goal clause or early cancellation pruning aim at cutting down the search space. Unfortunately some of these refinements are not combinable without loosing completeness.

One result of this paper is a table of completeness results with respect to the combination of refinements for RME (Table 1 below). Another original result is completeness of “early cancellation pruning”[1] in combination with regularity and what we call “independence of the goal clause”.

In the following section we recall basic restart model elimination calculus, and in Section 2 we introduce refinements. The main results of this paper are then presented in Section 3.

1.1 Restart Model Elimination (RME)

In this section we will describe restart model elimination as the basis for refinements in the rest of the paper.

We will state some basic definitions. A *clause* is a multiset of literals, usually written as the disjunction $L_1 \vee \ldots \vee L_n$. Clauses can be alternatively represented with an arrow. $A_1 \vee \ldots \vee A_m \leftarrow B_1 \wedge \ldots \wedge B_n$ is a representation of the clause $A_1 \vee \ldots \vee A_m \vee \neg B_1 \vee \ldots \vee \neg B_n$, where the As and Bs are atoms. Clauses with $m \geq 1$ are called *program clauses* with *head literals* A_i and *body literals* B_i, if present. Clauses of the form $\leftarrow B_1 \wedge \ldots \wedge B_n$ are called negative clauses in the sequel.

From now on we assume our clause sets to be in *Goal normal form*, i.e. there exists only one negative clause which furthermore does not contain variables. Without loss of generality this can be achieved by introducing a new clause $\leftarrow Goal$ where $Goal$ is a new predicate symbol, and by replacing every purely negative clause $\neg B_1 \vee \ldots \vee \neg B_n$ by $Goal \leftarrow B_1 \wedge \cdots \wedge B_n$.

We are now turning towards the calculus. Throughout this paper, we consider a variant of ME which uses so-called ME tableaux as basic proof objects, see [Letz *et al.*, 1994], rather than ME chains [Loveland, 1978].

We consider *literal trees*, i.e. finite, ordered trees, all nodes of which, except the root, are labeled with a literal. The labeling function is denoted by λ. Such a literal tree is also called a *tableau* and it is represented as a set of branches, where a *branch (of length n)* is a sequence $[N_0 \cdot N_1 \cdot \ldots \cdot N_n]$ ($n \geq 0$, written as indicated) of nodes such that N_0 is the root of the tree, N_i is the immediate predecessor of N_{i+1} for $0 \leq i < n$, and N_n is a leaf; the functions *First* and *Leaf* return the first *labeled*, resp. last node of a branch, i.e. $First([N_0 \cdot N_1 \cdot \ldots \cdot N_n]) = N_1$ and $Leaf([N_0 \cdot N_1 \cdot \ldots \cdot N_n]) = N_n$.

Throughout this paper, the letter N is used for nodes, L, K denote literals, and the symbols p, q are used for branches; like branches, branch-valued variables are also written with brackets, as in $[p]$. Branch sets are typically denoted by

[1]Early cancellation pruning was introduced in [Loveland and Reed, 1991] within the context of a nearHorn-Prolog variant, InH-Prolog.

the letters $\mathcal{P}, \mathcal{Q}, \ldots$. We write $\mathcal{P}, \mathcal{Q}$ and mean $\mathcal{P} \cup \mathcal{Q}$ (multiset union is intended here). Similarly, $[p], \mathcal{Q}$ means $\{[p]\}, \mathcal{Q}$. We write $N \in [p]$ iff N occurs in $[p]$. A substitution σ is applied to a branch set $\mathcal{P}$, written as $\mathcal{P}\sigma$, by applying σ to all labels of all nodes in $\mathcal{P}$. We say that branch set $\mathcal{P}$ is *more general* than branch set $\mathcal{P}'$ iff $\mathcal{P}\delta = \mathcal{P}'$ for some substitution δ.

Now let $[p]$ be a branch $[N_0 \cdot N_1 \cdot \ldots \cdot N_n]$. Any contiguous subsequence of $[p]$ (possibly $[p]$ itself) is called a *partial branch* (through $[p]$). The concatenation of partial branches $[p]$ and $[q]$ is denoted by $[p \cdot q]$; similarly, $[p \cdot N]$ means the extension of $[p]$ by the node N. We find it convenient to confuse a node with its label and write, for instance $[p \cdot L]$, where L is a literal, instead of "$[p \cdot N]$, where N is labeled with L"; the meaning of $L \in [p]$ is obtained in the same way; also, we say "node L" instead of the "node labelled with L".

In order to memorize the fact that a branch contains a contradiction, we allow to label a branch with a "$\star$" as *closed*; we insist that if a branch is labelled as *closed* then its leaf is complementary to some of its ancestor nodes. Branches which are not labelled as closed are said to be *open*. A literal tree is *closed* if each of its branches is closed, otherwise it is *open*.

Equality on branch sets is defined wrt. the labels and the "closed" status. More precisely, suppose given branches $[p]$ and $[p']$ stemming from (not neccessarily different) branch sets $\mathcal{P}$ and $\mathcal{P}'$ with respective labeling functions λ and λ'; define $\lambda[N_0 \cdot N_1 \cdot \ldots \cdot N_n] = \langle \lambda(N_1), \ldots, \lambda(N_n) \rangle$, $[p]\star =_{\lambda,\lambda'} [p']\star$ iff $[p] =_{\lambda,\lambda'} [p']$, where $[p] =_{\lambda,\lambda'} [p']$ iff $\lambda[p] = \lambda'[p']$. Equality for branch sets, i.e. $\mathcal{P} = \mathcal{P}'$, is defined as the usual multiset extension of "$=_{\lambda,\lambda'}$".

By the previous definitions literal trees are introduced as static objects. We wish to construct such literal trees in a systematic way. This is accomplished by, for instance, the restart model elimination calculus.

Definition 1.1 (Branch Extension, Connection)
The *extension of a branch* $[p]$ *with clause* C, written as $[p] \circ C$, is the branch set $\{[p \cdot L] \mid L \in C\}$. Equivalently, in tree view this operation extends the branch $[p]$ by $|C|$ new nodes which are labelled with the literals from C. A pair of literals (K, L) is a *connection with MGU* σ iff σ is a most general unifier for K and $\overline{L}$. □

Definition 1.2 (Restart Model Elimination)
Given a clause set S in *Goal* normal form. The inference rules *extension step*, *reduction step* and *restart step* on branch sets are defined as in Figure 1. The branch $[p]$ is called *selected branch* in all three inference rules. A restart step followed immediately by an extension step is also called a *restart extension step*. □

Note that like in the usual tableaux model elimination calculus (cf. [Letz *et al.*, 1994]), an extension or a reduction step can be applicable to a negative leaf. To

Restart Model Elimination (RME)

Extension Step:

$$\frac{[p],\,\mathcal{P} \qquad A_{\mathit{1}} \vee \ldots \vee A_m \leftarrow B_{\mathit{1}} \wedge \ldots \wedge B_n}{([p \cdot A_i]\star, [p] \circ (A_{\mathit{1}} \vee \ldots \vee A_{i-\mathit{1}} \vee A_{i+\mathit{1}} \vee \ldots \vee A_m \leftarrow B_{\mathit{1}} \wedge \ldots \wedge B_n), \mathcal{P})\sigma}$$

if

1. $A_{\mathit{1}} \vee \ldots \vee A_m \leftarrow B_{\mathit{1}} \wedge \ldots \wedge B_n$ (with $m \geq \mathit{1}$, $n \geq \mathit{0}$ and $i \in \{\mathit{1}, \ldots, m\}$) is a new variant (called *extending clause*) of a clause in S, and

2. $(\mathit{Leaf}([p]), A_i)$ is a connection with MGU σ. In this context A_i is called the *extension literal.*

Reduction Step:

$$\frac{[p],\,\mathcal{P}}{([p]\star,\,\mathcal{P})\sigma}$$

if $(L, \mathit{Leaf}([p]))$ is a connection with MGU σ, for some node $L \in [p]$.

Restart Step:

$$\frac{[p],\,\mathcal{P}}{[p] \circ \mathit{First}([p]),\,\mathcal{P}}$$

if $\mathit{Leaf}([p])$ is a positive literal.

Figure 1: Inference rules for RME.

a positive leaf, a reduction step or a restart step can be applicable, but never an extension step. We need one more definition before turning towards derivations:

Definition 1.3 (Computation Rule)
A *computation rule* is a total function c which maps a tableau to one of its open branches. It is required that a computation rule is *stable under lifting*, which means that for any substitution σ, whenever $c(\mathcal{Q}\sigma) = [q]\sigma$ then $c(\mathcal{Q}) = [q]$. $\square$

The role of a computation rule is to determine in a derivation the selected branch for the next inference step:

Definition 1.4 (Derivation)
Let S be a clause set in *Goal* normal form and c be a computation rule. A *restart model elimination derivation (RME derivation) of branch set $\mathcal{P}_n$ with substitution $\sigma_1 \ldots \sigma_n$ via c from S* consists of a sequence $(([\neg Goal] \equiv \mathcal{P}_0), \mathcal{P}_1, \ldots, \mathcal{P}_n)$ of branch sets, where for $i = 1 \ldots n$:

1. $\mathcal{P}_i$ is obtained from $\mathcal{P}_{i-1}$ by means of an extension step with an appropriate variant C of some clause from S and MGU σ_i, or
2. $\mathcal{P}_i$ is obtained from $\mathcal{P}_{i-1}$ by means of a reduction step and MGU σ_i, or
3. $\mathcal{P}_i$ is obtained from $\mathcal{P}_{i-1}$ by means of a restart step.

Any branch set which is derivable by some RME derivation is also called a *RME tableau*. $\square$

In each case the selected branch of the inference is determined by c. Quite often we will omit the term "via c" and mean that c is some arbitrary, given computation rule.

Finally, a *RME refutation* is an RME derivation such that $\mathcal{P}_n$ is closed. The term "RME" is dropped if context allows.

Notice that due to the construction of the inference rules, $\mathcal{P}_1$ is obtained from $\mathcal{P}_0$ by an extension step with some clause $Goal \leftarrow B_1 \wedge \cdots \wedge B_n \in S$ and with empty substitution. This clause is called the *goal clause* of the derivation.

Note that in extension steps we can connect only with the head literals of input clauses. Since in general this restriction is too strong, because it destroys completeness, we have to "restart" the computation with a fresh copy of a negative clause. This is achieved by the restart rule, because refutations of clause sets in *Goal* normal form always start with $First([p]) \equiv \neg Goal$, and thus only extension steps are possible to $\neg Goal$, which in turn introduce a new copy of a negative clause (cf. Figure 2, right side).

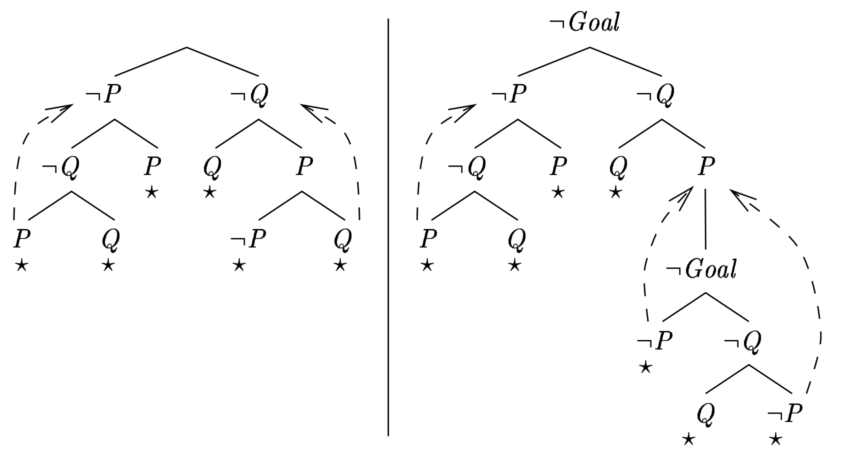


Figure 2: Model Elimination (left side) vs. Restart Model Elimination (right side). In order not to overload the notation, positive *Goal* nodes are not displayed (applies also to the figures below).

2 Refinements of RME

2.1 Refinement: Head Selection Function

We have mentioned that the restart model elimination calculus can be used as a basis for logic programming with clauses which contain disjunctions in their head. By dissallowing extension steps at positive leave literals we assure that program clauses can only be used for an extension step such that one of the head literals form the connection. Hence our calculus supports the distinction into head- and body literals: only head literals are used for "calling a program clause". This was one important step towards a procedural reading of disjunctive program clauses. We now go one step further, by introducing a *head selection function*. This is a means to distinguish one single head literal, which is then the only one allowed to use for an extension step (this concept is also present in Plaisted's Problem Reduction Formats [Plaisted, 1988], but not for the nearHorn Prolog family). In our example restart model elimination refutation from Figure 2 we used the clause $P \vee Q \leftarrow$ for an extension step at a leaf literal $\neg Q$. Now, if the P literal is to be the distinguished literal, then this extension step would be impossible. This is a severe restriction of the calculus, but it can be applied and combined with some other refinements (but not all) to still yield a complete calculus.

Definition 2.1 (Head Selection Function)
A *head selection function* f is a function that maps a clause $A_1 \vee \ldots \vee A_n \leftarrow B_1 \wedge \ldots \wedge B_m$ 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 head selection function f is required to be *stable under lifting*

which means that if f selects $L\gamma$ in the instance of the clause $(A_1 \vee \ldots \vee A_n \leftarrow B_1 \wedge \ldots \wedge B_m)\gamma$ (for some substitution γ) then f selects L in $A_1 \vee \ldots \vee A_n \leftarrow B_1 \wedge \ldots \wedge B_m$. $\square$

Note that this head selection function has nothing to do with the selection function from SLD-resolution which selects subgoals. The latter is called in our setting the computation rule (Def. 1.3).

Definition 2.2
Let f be a head selection function. A RME derivation is called a *derivation with selection function f* if it is a RME derivation such that in every extension step only the selected literal A_i from an input clause $A_1 \vee \ldots \vee A_n \leftarrow B_1 \wedge \ldots \wedge B_m$ is used for a connection $(\mathit{leaf}(p), A_i)$. $\square$

A head selection function oviously allows to distinguish one single head literal to be used as the only entry point during the whole derivation. Hence we arrived at a calculus which does not need contrapositives at all: from the restart property we know that the contrapositives which have a negative clause in its head can be discarded and in a derivation with selection function we know that only the selected single positive literal has to be used. Our small example derivation from the right part of Figure 2 uses the clause $P \vee Q \leftarrow$ two times for an extension step. In both cases Q was used for the connection; hence if we assume a head selection function which gives Q as a sected literal this tableau is a refutation with selection function.

2.2 Refinement: Strictness

RME with selection function allows a procedural reading of a single program clause, since extension steps use a clause $A_1 \vee \ldots \vee A_i \vee \ldots \vee A_n \leftarrow B_1 \wedge \ldots \wedge B_m$ as a procedure via the head A_i which is determined by the selection function. There is however still a problem with the procedural interpretation, namely to explain how the remaining head literals are treated throughout a derivation. There are two possibilities to further derive from a positive leaf literal: either the branch can be closed by performing a reduction step or a restart has to be done. In the right part of Figure 2 both possibilities are contained. The literal P is used in an reduction step, hence it is the leaf of a closed branch and the rightmost branch containing the P was extended by a restart step. In *strict* restart derivations we forbid reduction steps at positve leaf literals. This refinement has been discussed in [Baumgartner and Furbach, 1994a] and hence we ommit a formal treatment.

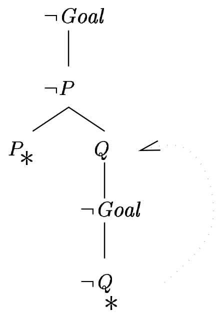
In this strict setting we can assume that reduction steps only occur that close a branch with a negative literal. These reduction steps are interpreted very naturally by the following view to the restart concept. Let $A_1 \vee A_2 \leftarrow B$ be a program clause A_1 be the selected literal. Then a call to this clause can be done via A_1 within a refutation and in addition one has to prove the original goal with the extra assumption A_2. Instead of adding the fact A_2 to the clause set for this additional prove, strict restart model eleimination allows the closing of branch by reduction steps to the branch literal A_2 which obviously has the same effect.

2.3 Refinement: Independence of the Goal Clause

In order to arrive at a really *goal*-oriented calculus, one wants to restrict the starting branch set to be derived from a negative clause. In the case of logic programming this is well known and straightforward: there is one goal, namely the query, from which the proof prodedure works backwards. Note that in our *Goal* normal form this is also the case; there is $\leftarrow Goal$ as the only negative clause and we required a derivation $(\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_n)$ to start with a branch set $\mathcal{P}_0$ which corresponds to this $\leftarrow Goal$.

However this *Goal* is a new predicate symbol, which was introduced to replace every purely negative clause $\neg B_1 \vee \ldots \vee \neg B_n$ by $Goal \leftarrow B_1 \wedge \cdots \wedge B_n$. Remember that due to definition of a derivation and the construction of the inference rules, $\mathcal{P}_1$ is obtained from $\mathcal{P}_0 \equiv \neg Goal$ by an extension step with some clause $Goal \leftarrow B_1 \wedge \cdots \wedge B_n \in S$ and with empty substitution. This clause is called the *goal clause* of the derivation. What we really want, is to be independant of this goal clause:

Definition 2.3
A (refinement of the) RME caculus is called *independent of goal clause* if every derivation $(([\neg Goal] \equiv \mathcal{P}_0), \mathcal{P}_1)$ with goal clause from a minimal unsatisfiable subset of the clause set S can be extended to a refutation, if a refutation exists. □



Take as an example the RME refutation in the figure on the right; once we extended with the goal clause $Goal \leftarrow P$, we can be sure that this tableau can be extended to a closed one. Note that for the subsequent restart steps it may be necessary to use other goal clauses in order to ensure completeness of the calculus. Still, this restriction may prune down the search space significantly: assume you have a consistent set of clauses S^+, including negative clauses as well. The latter can be understood as integrity constraints. If you now add the negation of a query as a further negative clause to yield S, you can be sure , that every minimal unsatsfiable subset of S contains this negated query and you know that you can start the derivation with this goal clause. If your calculus is independant of goal clause you never need backtracking over this point.

2.4 Refinement: Regularity

The *regularity check* for model elimination says that it is never necessary to construct a tableau where a literal occurs twice (or even more often) along a branch. Expressed operationally, it says that it is never necessary to repeat a previously derived subgoal (viewing open leaves as subgoals). For a semantic interpretation take the view that a branch constitutes a partial interpretation, and any clause containing a literal from the branch would be satisfied by this interpretation. Hence this clause need not be considered for "eliminating" the interpretation given by the branch. From a logic programming perspective this can be seen as a very simple loop check.

Regularity is easy to implement, at least approximately, and it is one of the most effective restrictions for model elimination procedures. Unfortunately, the regularity check is *not* compatible with RME. This is rather easy to see since after a restart step it might be necessary to repeat —in parts— a refutation derived so far up to the restart step. However, what can be achieved is *blockwise regularity*.

Definition 2.4 (Blockwise Regularity)
Let $[p]$ be branch written as follows, where the As and Bs are atoms:

$$[p] = [\neg B^1_1 \cdots \neg B^1_{k_1} \cdot A^1 \cdot \neg B^2_1 \cdots \neg B^2_{k_2} \cdot A^2 \cdots A^{n-1} \cdot \neg B^n_1 \cdots \neg B^n_{k_n}]$$

Then $[p]$ is called *blockwise regular* iff

1. $A^i \neq A^j$ for $1 \leq i, j \leq n-1$, $i \neq j$ *(Regularity wrt. positive literals)*, and
2. $B^l_i \neq B^l_j$ for $1 \leq l \leq n$, $1 \leq i, j \leq k_l$, $i \neq j$ *(Regularity inside blocks).*

A branch set is called *blockwise regular* iff every branch in it is blockwise regular. Similarly, a derivation is called *blockwise regular* iff each of its branch sets is blockwise regular. □

For example the restart model elimination derivation in Figure 2 is blockwise regular.

2.5 Refinement: Early Cancellation Pruning

This refinement is motivated as a kind of relevancy test for restart steps. Recall that a restart step occurs at a positive leaf, say A. If below this leaf in the rest of the derivation another restart step could be applied, early cancellation pruning allows this step only if the reason for the previous restart — the literal A – was used for an reduction step until then. If this is not the case, one has to backtrack. In

Figure 3, the derivation c) is an example: early cancellation pruning would forbid a restart step at leaf D because the literal B was not used.

The easiest way to define this refinement is to use a "static" property of RME tableaux:

Definition 2.5 (Weakly Connected RME)
A node L_k in a branch $[L_1 \cdots L_k \cdot L_{k+1} \cdots L_n]$ with $n \geq k$ is called the *leafmost positive node* of that branch iff L_k is positive and $L_{k+1}, \ldots, L_n$ all are negative. An inner positive node L in a branch $[p]$ is *weakly connected in* $[p]$[2] iff it is the leafmost positive node of $[p]$ and $\mathit{Leaf}([p]) = \overline{L}$.

A RME tableau $\mathcal{P}$ is called *weakly connected* iff every inner positive node L is weakly connected in some branch in $\mathcal{P}$.

□

We are interested in a *calculus* refinement corresponding to the just defined weakly connected RME tableaux. This is done as follows:

Definition 2.6 (RME with Early Cancellation Pruning)
We allow to label positive nodes in RME tableaux by the symbol "r" (meaning: used for $r$eduction steps). If node L is labeled in this way we will write L^r. The calculus *RME with early cancellation pruning (RMEP)* consists of the inference rule "extension step" of Def. 1.1 and the following inference rules:

Labeling Reduction Step:

$$\frac{[p \cdot L \cdot q],\ \mathcal{P}}{([p \cdot L^r \cdot q]\star,\ \mathcal{P})\sigma}$$

if $(L, \mathit{Leaf}([p \cdot L \cdot q]))$ is a connection with MGU σ.

Restricted Restart Step:

$$\frac{[p],\ \mathcal{P}}{[p \cdot \mathit{First}([p])],\ \mathcal{P}}$$

if $\mathit{Leaf}([p])$ is a positive literal, and the leafmost positive inner node of $[p]$, if it exists, is labeled with r.

The notion of *derivation* is taken from Def. 1.4. A RMEP refutation is a derivation of closed RME tableau where every inner positive node is labeled with r. □

[2] As opposed to the "fully" connected nodes in ME tableaux.

This refinement is essentially due to [Loveland and Reed, 1991] and was called *strong early cancellation pruning rule* in Inh-Prolog. Since in our framework it would be difficult to formulate the *rejection* of a derivation — which is what a *pruning* rule is about — we defined a positive formulation. As will be shown below, this restriction drastically changes the calculus' properties.

The idea of the early cancellation pruning is to achieve a relevance check: a new "case" by means of a restart step applied to $[\cdots L' \cdots L]$ may only be examined if the previous case L' turned out to be "relevant" for the derivation of the new case L. Here, "relevant" means that L' is the target for a reduction step. The term "early" means that the reduction step targetting at L' can be required to come *before* the next restart step. In other words, L' must be an inner leafmost positive literal in some branch with complementary leaf.

Notice that in the definition of RMEP we do not use the definition of weakly connected RME tableau. Instead we rely on a simple and cheaply implementable labeling mechanism. It is thus not immediately clear that RMEP is correct wrt. the construction of the intended weakly connected RME tableau. Therefore we state:

Proposition 2.7 (RMEP constructs Weakly Connected RME Tableaux) *Any strict RMEP refutation of S ends in a closed, strict and weakly connected RME tableaux for S.*

Proof. Let $\mathcal{P}$ be the closed RME tableaux constructed in a strict RMEP refutation of S. Assume, to the contrary, that $\mathcal{P}$ is not weakly connected. Hence there is an inner positive node L which is not the leafmost positive node of some branch $[p]$ with $\mathit{Leaf}([p]) = \overline{L}$. Since $\mathcal{P}$ is a refutation all branches are closed, and, by definition of RMEP refutation, all inner positive nodes are labeled as r. The subtree below L is non-empty (because L is an *inner* node) and can be written as the branch set

$$\mathcal{P}_L = \{[q \cdot L^r \cdot q_1]\star, \ldots , [q \cdot L^r \cdot q_n]\star\} \subseteq \mathcal{P}$$

where each $[q_i]$ $(i = 1, \ldots , n)$ is a non-empty sequence of nodes. Let

$$\mathcal{P}'_L = \{[q \cdot L^r \cdot q_i]\star \in \mathcal{P}_L \mid [q_i] = [q'_i \cdot \neg L], \text{ for some } q'_i \text{ and } i = 1, \ldots , n\}$$

be those branches from $\mathcal{P}_L$ which use the node L as a target for a reduction step. This set must be non-empty, because the node L is labeled with r. Each $[q'_j]$, where $[q \cdot L^r \cdot q'_j \cdot \neg L] \in \mathcal{P}'_L$, contains a positive node K_j, because otherwise there is no contradiction to "weakly connected". That is, $[q'_j]$ takes the form

$$[q'_j] = [s_j \cdot K_j \cdot s'_j], \text{ for some } [s_j], [s'_j].$$

Since K_j is positive, the refutation would have had to restart at the intermediately derived branch $[q \cdot L^r \cdot s_j \cdot K_j]$. Recall that the RMEP restricted restart step requires that the leafmost positive inner node of this branch is labeled with r, which is L^r here. Hence, prior to this restart step there must be some reduction step targeting at the node L. Let $[q \cdot L^r \cdot q'_k \cdot \neg L]\star \in \mathcal{P}'_L$ be that branch to which the first reduction step to L is executed. This reduction step closes the branch $[q \cdot L \cdot q'_k \cdot \neg L]$. However, as derived above for all members of $\mathcal{P}'_L$, the branch $[q'_k]$ must contain a positive node K_k and thus is of the form

$$[q'_k] = [s_k \cdot K_k \cdot s'_k], \text{ for some } [s_k],\ [s'_k].$$

Hence, a restricted restart step must have been previously occured to $[q \cdot L \cdot s_k \cdot K_k]$. However, this is impossible, because the leafmost inner positive node — L — is not labeled with r. Hence, the assumption about the existence of the inner node L contradicting the "weakly connected" property must have been wrong. Therefore, $\mathcal{P}$ is weakly connected. ■

The converse of Proposition 2.7 does not hold in full generality. The problem is the usage of a possibly incompatible computation rule. Most of our calculi variants allow for *arbitrary* computation rules. The notable exception are the variants which employ the early cancellation pruning. In these cases we have to restrict to the following class of computation rules.

Definition 2.8 (Negative Preferrence Computation Rule)
A *negative preferrence computation rule* is a computation rule c such that whenever a positive inner node L is contained in an open branch $[p]$ with positive leaf, and L is contained in an open branch $[p']$ with negative leaf, then c does not select $[p]$. □

In other words: as long as possible, this computation rule selects an open branch with a negative leaf. *But:* this applies only "locally", below a restart point N_k. It may well be possible that a restart occurs, although there are still open branches with negative literals left.

Of course, any computation rule selecting an open branch with negative leaf as long as one is present, is also negative preferrence computation rule. Such a computation rule is implicitly used in Loveland's InH-Prolog.

For the converse of Proposition 2.7 it suffices for our purposes to restrict to the ground case, because we give a lifting lemma for derivations below and require that the computation rules are stable under lifting, cf. Def. 1.3.

Proposition 2.9
For any closed, weakly connected RME tableau $\mathcal{P}$ for a ground clause set S and any negative preferrence computation rule there is a RMEP refutation of S ending in $\mathcal{P}$. Furthermore, if $\mathcal{P}$ is strict (and/or blockwise regular), then the RMEP refutation is also strict (and/or blockwise regular).

Together with a proof of the existence of closed, strict and weakly connected tableaux (for unsatisfiable clause sets) we get a completeness result[3].

Proof. We construct a derivation $\mathcal{P}_0, \ldots , (\mathcal{P}_n \equiv \mathcal{P})$, for some n. The top clause in $\mathcal{P}_0$ consists of the corresponing rootmost tableau clause in $\mathcal{P}$. The transition from $\mathcal{P}_i$ to $\mathcal{P}_{i+1}$ (for $i = 0, \ldots , n-1$) is done according to the following procedure.

Case 1: If the given computation rule c selects a branch of the form $[p \cdot \neg A] \in \mathcal{P}_i$ and $\mathcal{P}$ contains a branch of the form $[p \cdot \neg A]\star$, then this branch is to be closed in a labeling reduction step, yielding $\mathcal{P}_{i+1}$.

Case 2: If c selects a branch of the form $[p \cdot \neg A] \in \mathcal{P}_i$ and the previous case does not apply, then $\mathcal{P}$ contains a branch with prefix of the form $[p \cdot \neg A]$ and a tableaux clause of the form $A \vee R$ (for some clause R) below the node $\neg A$. Further, $\mathcal{P}$ contains a branch $[p \cdot \neg A \cdot A]\star$. Hence perform an extension step to $[p \cdot \neg A]$ with clause $A \vee R$ to obtain $\mathcal{P}_{i+1}$.

If neither of the previous cases applies, then c selects a branch of the form $[p \cdot A] \in \mathcal{P}_i$.

Case 3: If $\mathcal{P}$ contains a branch of the form $[p \cdot A]\star$, then this branch is to be closed in a labeling reduction step, yielding $\mathcal{P}_{i+1}$. If $\mathcal{P}$ is strict, then this case cannot apply, and hence the strict version of the reduction step suffices.

Case 4: If case 3 does not apply, then $\mathcal{P}$ contains a branch of the form $[p \cdot A \cdot \neg Goal]$. In order to derive this branch by means of a restart step, recall that we have to have that the leafmost positive ancestor node of A in $[p \cdot A]$, if it exists, is labeled with r. This, however, must hold for the following reason: suppose, to the contrary, $[p \cdot A] = [q \cdot B \cdot \neg Goal \cdot q' \cdot A]$ (for some $[q]$ and $[q']$), where node B is the leafmost positive ancestor of node A, and B is not labeled with r. Recall that $\mathcal{P}$ is weakly connected. This means that $\mathcal{P}$ contains a branch of the form $[q \cdot B \cdot \neg Goal \cdot \neg B_1 \cdots \neg B_n \cdot \neg B]\star$. With $[q \cdot B \cdot \neg Goal \cdot q' \cdot A] \in \mathcal{P}_i$ it follows that some $\mathcal{P}_j$ with $j < i$ contains $[q \cdot B \cdot \neg Goal \cdot \neg B_1]$. Due to the negative preferrence computation rule, this branch must be selected before $[q \cdot B \cdot \neg Goal \cdot q' \cdot A]$ is selected. Hence we get eventually, using the appropriate extension step (cf. case 2) $[q \cdot B \cdot \neg Goal \cdot \neg B_1 \cdot \neg B_2]$. Repeat application of this argument gives us $[q \cdot B \cdot \neg Goal \cdot \neg B_1 \cdots \neg B_n \cdot \neg B]$. Also this branch must be selected before $[q \cdot B \cdot \neg Goal \cdot q' \cdot A]$ is selected. Hence by application of a labeling reduction step (cf. case 1) we end up with $[q \cdot B^r \cdot \neg Goal \cdot \neg B_1 \cdot \neg B_n \cdot \neg B]\star$, which is also contained in $\mathcal{P}_i$. Thus B, which is also the leafmost positive ancestor of A in $[p \cdot A] \in \mathcal{P}_i$ is labeled with r. This, however, plainly contradicts the assumption that B is not labeled with r. Therefore the restart step can be applied to $[q \cdot A]$ as suggested above, which gives us $\mathcal{P}_{i+1}$.

[3]All proofs are contained in the long version of this paper.

In sum, all four cases advance the construction of the derived tableau so far towards the given tableau $\mathcal{P}$ (we feel no need to make the induction explicit here). Hence we end up with $\mathcal{P}$ itself. ■

3 Properties

The purpose of this section is to discuss the various refinements described above. Each refinement was motivated in some way, so the question arises to what extent they can be combined, Unfortunately, not all of them are compatible to each other. That is, some combinations cause incompleteness. Further, these incompatibilities are "inherent", in the sense that completeness cannot be recovered by relaxing other refinements, such as giving up "regularity" or "strictness". Table 1 contains a summary of both, these negative results, and the positive results.

	Calculus	*Selection function*	*Regularity*	*Indep. of goal clause*	*Completeness*
(0)	ME	–	Full	Yes	Yes
(1)	RME	with	Blockwise	No	Yes
(2)		without	Blockwise	Yes	Yes
(3)	RMEP	with	–	–	No
(4)		without	Blockwise	Yes	Yes

Table 1: Summary of properties of model elimination variants. "–" means "does not apply".

3.1 Negative Results

The negative results (the "no" entries in Table 1) are shown by appropriate counterexamples to the assumption that the combination of the indicated features would yield a complete calculus.

"Head selection function" is incompatible to "early cancellation pruning". This addresses line (3) in Table 1. Consider the following clause set:

$$M_1 = \{ \quad \leftarrow A, \quad \underline{A} \vee D \leftarrow, \quad A \leftarrow B \wedge D, \quad B \vee \underline{C} \leftarrow, \quad A \leftarrow C\} \ .$$

There is no RMEP refutation of the *Goal* normal form of M_1 with a head selection function which selects in M_1 the underlined atoms in the clause heads. Figure 3 shows an exhaustive case analysis: either the derivation contains a negative leaf $\neg D$ and gets stuck because the sole clause containing D in the head is $\underline{A} \vee D$, but

D is not selected. Or, the other derivations are not weakly connected, so that a restart step at the positive branches is not permitted. There are some variations of these derivations by re-using clauses on a branch, but all of these run into the same problems. On the other side, there is a RMEP refutation of the *Goal* normal form of M_1 without selection function. Hence, in sum, the concept "head selection function" is not compatible to RMEP.

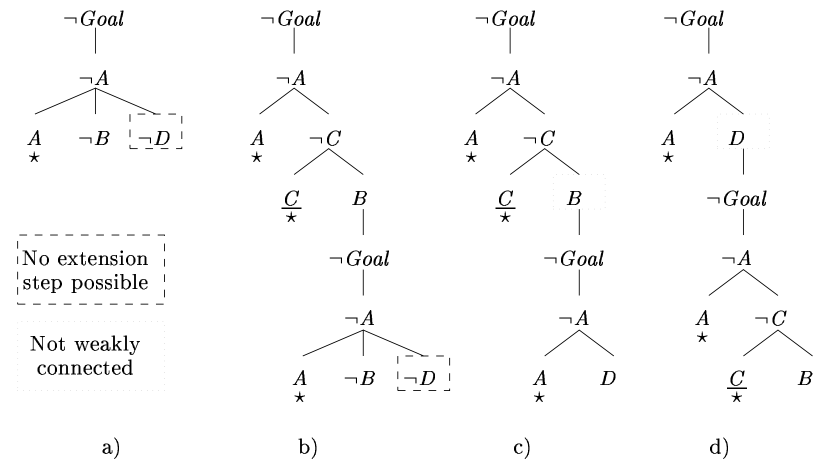


Figure 3: "Selection Function" is incompatible to "Early Pruning".

"Head selection function" is incompatible to "independence of the goal clause". This addresses line (1) in Table 1. Consider the following clause set, which was also used in Section 2.

$$M_2 = \{ \quad \leftarrow P, \quad \leftarrow Q, \quad \underline{P} \vee Q \leftarrow \} \ .$$

It is easy to see that there is no RME refutation (and hence no RMEP refutation either) of the *Goal* normal form of M_2 with goal clause $Goal \leftarrow Q$ and a head selection function which selects P in $P \vee Q \leftarrow$. However, chosing $Goal \leftarrow P$ as the goal clause admits a RME refutation. In sum, the concepts "head selection function" and "independence of the goal clause" are incompatible for all considered RME variants.

3.2 Positive Results

A standard strategy for completeness proofs of related calculi is to prove completeness of the *weakest* variants only, i.e. the variants, the refutations of which can stepwisely be simulated by the other variants. For instance, strict RME is weaker than non-strict RME. For the case of restart model elimination there are the following two weakest variants:

- RME with "head selection function", but without "independence of the goal clause. This addresses the "completeness" entry in line (1) in Table 1. See [Baumgartner *et al.*, 1995] for a proof.
- RME without "selection function" but with "independence of the goal clause" and with "early cancellation pruning". This addresses the "completeness" entries in lines (2) and (4) in Table 1. The result is as follows[4]:

Theorem 3.1 (Ground completeness of Blockwise Regular Strict RMEP)
Let S be a minimal unsatisfiable ground clause set in Goal normal form, c be a negative preferrence computation rule. Then, for any clause $G = (Goal \leftarrow B_1 \wedge \cdots \wedge B_n) \in S$ there is a strict RMEP refutation via c with top clause $\leftarrow Goal$, goal clause G which derives a closed, regular, strict and weakly connected RME tableau.

3.3 Proof of Theorem 3.1

It turns out to be advantageous to first prove the existence of the claimed RME *tableau* in a slightly more specific setting than in the statement of Theorem 3.1. Later, we will prove that such RME tableaux can be constructed in *refutations*.

Lemma 3.2 (Existence of Weakly Connected RME Tableaux)
Let $S \cup \{A \leftarrow\}$ be a minimal unsatisfiable ground clause set in Goal normal form. Then there is a closed, regular, strict and weakly connected RME tableau $\mathcal{P}$ with top clause $\leftarrow Goal$ such that the unit clause $A \leftarrow$ is used within the first block. More precisely, $\mathcal{P}$ contains a branch $[L_1 \cdots L_n \cdot \neg A \cdot A]\star$ such that $L_1, \ldots, L_n$ all are negative.

Proof. Let $k(S)$ denote the number of occurrences of positive literals in S minus the number of non-negative clauses[5] in S ($k(S)$ is a measure for the "Hornness" of S; it is related to the well-known *excess literal parameter*). Now we prove the claim by induction on $k(S)$.

Base case: $k(S \cup \{A \leftarrow\}) = 0$. Then $S \cup \{A \leftarrow\}$ must be a set of Horn clauses. By well-known completeness results for ME (see e.g. [Loveland, 1978; Baumgartner, 1992; Letz *et al.*, 1994]) there exists a regular ME refutation of $S \cup \{A \leftarrow\}$ with top clause $\leftarrow Goal$. The claimed properties trivially hold for the tableau $\mathcal{P}$ constructed in this refutation: observe that all open branches of any derivable tableau consist of negative literals only, and every closed branch ends in a positive leaf. In particular,

[4]For space reasons we can only cite the relevant results. We will state the ground version only. Lifting to the first-order case can be done along the lifting proof in [Baumgartner *et al.*, 1995]. We recall only that both the "computation rule" and the "head selection function" were defined to be stable under lifting (Defs. 1.3 and 2.1), which enables lifting them to the first-order level.

[5]A *non-negative clause* is a clause containing at least one positive literal.

$\mathcal{P}$ contains a branch $[L_1 \cdots L_n \cdot \neg A \cdot A]\star$ as desired (the fact that $A \leftarrow$ must be used in the refutation at all follows from the precondition that $S \cup \{A \leftarrow\}$ is *minimal* unsatisfiable: if $A \leftarrow$ would not have been used, the refutation would witness that S *alone* is unsatisfiable, which is by soundness of ME a contradiction to the minimal unsatisfiablity of $S \cup \{A \leftarrow\}$).

Induction step: $k(S \cup \{A \leftarrow\}) > 0$**.** As the induction hypothesis assume the result to hold for clause sets $S' \cup \{A' \leftarrow\}$ satisfying the preconditions and $k(S' \cup \{A' \leftarrow\}) < k(S \cup \{A \leftarrow\})$.

Since $k(S \cup \{A \leftarrow\}) > 0$ there is in S a disjunctive clause

$$A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n \qquad (m \geq 2,\ n \geq 0) \ .$$

By Lemma 3.4 there is a $i \in \{1, \ldots , m\}$ and a set

$$S' \subseteq ((S \setminus \{A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n\}) \cup \{A_i \leftarrow B_1 \wedge \cdots \wedge B_n\})$$

such that $S' \cup \{A \leftarrow\}$ is minimal unsatisfiable. $S' \cup \{A \leftarrow\}$ still is in *Goal* normal form. Hence, by the induction hypothesis there is a closed, regular, strict and weakly connected RME tableau $\mathcal{P}'$ for $S' \cup \{A \leftarrow\}$ such that $\leftarrow A$ is used in the first block. Now replace every occurrence of the tableau clause $A_i \leftarrow B_1 \wedge \cdots \wedge B_n$ in $\mathcal{P}'$ by $A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n$. This leaves us with an open tableau for $S \cup \{A \leftarrow\}$ whose open branches all end in a literal from $\{A_1, \ldots , A_{i-1}, A_{i+1}, \ldots , A_m\}$. Now we delete from this tableau all subtrees below all rootmost positive inner nodes. More formally, replace every branch of the form $[p \cdot A' \cdot q]\star$ by $[p \cdot A']$, where $[p]$ consists of negative nodes only, A' is a positive node and $[q]$ is non-empty. Let $\mathcal{P}''$ be the resulting open tableau. $\mathcal{P}''$ can be thought of to be constructed in a RME derivation of $S \cup \{A \leftarrow\}$ which stops at the first positive nodes in each branch. Notice that $A \leftarrow$ still is used in the first block (in the sense given in the lemma statement) in $\mathcal{P}''$.

Every open branch in $\mathcal{P}''$ now takes the form $[p \cdot A']$, for some $[p]$ consisting of negative nodes only, and A' is a positive node, stemming from some disjunctive clause from S.

We will define a suitable literal tree to be put below each $[p \cdot A']$ in $\mathcal{P}''$ such that the desired weakly connected RME tableau $\mathcal{P}$ comes up. Let $[p \cdot A']$ be one of those branches. It suffices to show the construction for this branch. A' stems from a clause

$$A'_1 \vee \cdots \vee A'_{m'} \leftarrow B'_1 \wedge \cdots \wedge B'_{n'} \in S \qquad (m' \geq 2,\ n' \geq 0) \ ,$$

where $A' \equiv A'_j$ for some $j \in \{1, \ldots , m'\}$. Since $S \cup \{A \leftarrow\}$ is given as minimal unsatisfiable, we can find by Lemma 3.3 (setting $C = A' \leftarrow$ there) a set

$$S'' \subseteq (S \cup \{A \leftarrow\}) \setminus \{A'_1 \vee \cdots \vee A'_{m'} \leftarrow B'_1 \wedge \cdots \wedge B'_{n'}\}$$

such that $S'' \cup \{A' \leftarrow\}$ is minimal unsatisfiable. Clearly, $k(S'' \cup \{A' \leftarrow\}) < k(S \cup \{A \leftarrow\})$ (we replace a disjunctive clause by definite unit clause and possibly delete clauses). Since $S'' \cup \{A' \leftarrow\}$ still is *Goal* normal form, we can find by the induction hypothesis a closed, regular, strict and weakly connected RME tableau $\mathcal{P}'''$ for $S'' \cup \{A' \leftarrow\}$ with top clause $\leftarrow$ *Goal* such that the unit clause $A' \leftarrow$ is used within the first block. The plan of attack is to put $\mathcal{P}'''$ below $[p \cdot A']$ in $\mathcal{P}''$ and remove applications of $A' \leftarrow$.

More precisely, define first

$$Q = \{[p \cdot A' \cdot q]\star \mid [q]\star \in \mathcal{P}'''\}$$

as the literal tree to be obtained from $\mathcal{P}'''$ by putting $[p \cdot A']$ on top of it. The unit clause $A' \leftarrow$ is used in $\mathcal{P}'''$, not neccessarily only in the first block. But any usage of $A' \leftarrow$ is limited to leaf positions (this follows from the definition of RME tableau). Thus Q contains branches of the form $[p \cdot A' \cdot q' \cdot \neg A' \cdot A']\star$ for some branch $[q' \cdot \neg A' \cdot A']\star \in \mathcal{P}'''$. Next replace each of these branches in Q by $[p \cdot A' \cdot q' \cdot \neg A']\star$ and obtain Q'.

Since by the induction hypothesis $A' \leftarrow$ is used in the first block in $\mathcal{P}'''$, i.e. $\mathcal{P}'''$ contains a branch of the form $[q' \cdot \neg A' \cdot A']\star$ where $[q']$ consists of negative nodes only, the node A' is weakly connected in the corresponding branch $[p \cdot A' \cdot q' \cdot \neg A']\star$ in Q'. From the induction hypothesis we further learn that $\mathcal{P}'''$ is weakly connected. Thus, together, Q' is weakly connected as well.

Furthermore, every branch in Q' is blockwise regular, because the blockwise regularity of $\mathcal{P}'''$ carries over to $\mathcal{Q}'$. The critical case is the node A' which is with respect to positive nodes the only difference between the branches in $\mathcal{P}'''$ and Q'. However, $\mathcal{P}'''$ is a tableau for $S'' \cup \{A' \leftarrow\}$, which is *minimal* unsatisfiable. This implies that S'' cannot contain a disjunctive clause with a head literal A'. But then, A' can occur *only* in a branch in $\mathcal{P}'''$ at leaf position (as observed above). These usages, however, were eliminated in the transition from Q to Q'. Hence Q' is blockwise regular.

The final step is to replace $[p \cdot A']$ in $\mathcal{P}''$ by the branch set Q'. When this construction is carried out for all branches of this form, we arrive at the claimed tableau $\mathcal{P}$ for $S \cup \{A \leftarrow\}$. ■

Lemma 3.3
Let S be a minimal unsatisfiable ground clause set with $L_1 \vee \cdots \vee L_n \in S$ $(n \geq 1)$. Then for every subclause $C \subset L_1 \vee \cdots \vee L_n$ there is a set

$$S_C \subseteq (S \setminus \{L_1 \vee \cdots \vee L_n\})$$

such that $S_C \cup \{C\}$ is minimal unsatisfiable.

Proof. It is clear that $(S \setminus \{L_1 \vee \cdots \vee L_n\}) \cup \{C\}$ is unsatisfiable, because otherwise a model for this set would constitute a model for S itself. Let $S'_C \subseteq (S \setminus \{L_1 \vee \cdots \vee$

$L_n\}) \cup \{S_C\}$ be any minimal unsatisfiable subset. It suffices to show that $C \in S'_C$, because then $S_C := S'_C \setminus \{C\}$ proves the lemma.

Hence suppose, to the contrary, that $C \notin S'_C$. But then $S'_C \subset S$, and the strict inclusion contradicts the given fact that S is *minimal* unsatisfiable. ■

Lemma 3.4
Let $S \cup \{A \leftarrow\}$ be a minimal unsatisfiable ground clause set containing a clause

$$A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n \qquad (m \geq 2,\ n \geq 0)\ .$$

Then there is a $i \in \{1, \ldots, m\}$ and a set

$$S' \subseteq ((S \setminus \{A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n\}) \cup \{A_i \leftarrow B_1 \wedge \cdots \wedge B_n\})$$

such that $S' \cup \{A \leftarrow\}$ is minimal unsatisfiable.

Proof. Let

$$S_j \subseteq (((S \cup \{A \leftarrow\}) \setminus \{A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n\}) \cup \{A_j \leftarrow B_1 \wedge \cdots \wedge B_n\}) \qquad \text{(for } j = 1, \ldots, m) \quad (1)$$

be any m minimal unsatisfiable sets. Such sets exists by Lemma 3.3. We show that $A \leftarrow \in S_i$, for some $i \in \{1, \ldots, m\}$. Setting then $S' = S_i \setminus \{A \leftarrow\}$ proves the lemma.

Hence suppose, to the contrary, $A \leftarrow \notin S_j$, for $j = 1, \ldots, m$. It holds that

$$S' = (\bigcup_{j=1,\ldots,m} S_j \setminus \{A_j \leftarrow B_1 \wedge \cdots \wedge B_n\}) \cup \{A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n\}$$

is unsatisfiable. Proof: suppose, to the contrary, S' is satisfiable. Let $\mathcal{I}$ be a model. Since $\mathcal{I} \models A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n$ we distinguish two cases:

Case 1: $\mathcal{I} \not\models B_k$, for some $k \in \{1, \ldots, n\}$: But then

$$\mathcal{I} \models (S' \cup \{A_j \leftarrow B_1 \wedge \cdots \wedge B_n\}) \supseteq S_j \qquad \text{(for any } j \in \{1, \ldots, m\}).$$

In other words, S_j would be satisfiable, which plainly contradicts the unsatisfiability of S_j.

Case 2: $\mathcal{I} \models A_j$, for some $j \in \{1, \ldots, m\}$: But then

$$\mathcal{I} \models (S' \cup \{A_j \leftarrow B_1 \wedge \cdots \wedge B_n\}) \supseteq S_j\ .$$

As in the previous case, S_j would be satisfiable, which plainly contradicts the unsatisfiability of S_j.

Thus, together, S' is unsatisfiable. Now, if $A \leftarrow \notin S_j$, for $j = 1, \ldots, m$, as supposed above, then also $A \leftarrow \notin S'$ by construction of S'. Also by construction, $S' \subseteq S \cup \{A \leftarrow\}$. But with $A \leftarrow \notin S'$ we have even $S' \subseteq S$, and since S' is unsatisfiable we have a contradiction to the minimality assumption of $S \cup \{A \leftarrow\}$. Hence the assumption must have been wrong and thus $A \leftarrow \in S_i$, for some $i \in \{1, \ldots, m\}$, as remained to be shown. ■

Now we can turn to the proof of the completeness result.

Proof.(Theroem 3.1) With respect to the previous Lemma 3.2 the crucial point is to find an argument for the "independence of the goal clause", i.e. that the claimed refutation exists for the desired goal clause $G \equiv (\mathit{Goal} \leftarrow B_1 \wedge \cdots \wedge B_n) \in S$. For this, let *Top* be a new predicate symbol (wrt. the given signature) and define

$$\begin{aligned} G' &= \mathit{Goal} \leftarrow B_1 \wedge \cdots \wedge B_n, \mathit{Top} \\ S' &= (S \setminus \{G\}) \cup \{G'\} \cup \{\mathit{Top} \leftarrow\} \ . \end{aligned}$$

It is easy to see that S is minimal unsatisfiable iff S' is minimal unsatisfiable. Furthermore, S' is in *Goal* normal form and thus meets the requirement for Lemma 3.2. Hence let $\mathcal{P}'$ be the closed tableaux according to Lemma 3.2. The topmost clause in $\mathcal{P}'$ is, by definition $\leftarrow \mathit{Goal}$, and the clause immediately below it must be G'. Reason: G' is the *sole* *Goal*-clause which is extended with the *Top*-literal. If any other *Goal*-clause would have been used, it would be impossible to use $\mathit{Top} \leftarrow$ in the first block (cf. Lemma 3.2), a requirement which, however, we can insist on by Lemma 3.2.

In order to turn $\mathcal{P}'$ into a closed tableau for S (but not for S') one simply has to replace in $\mathcal{P}'$ every tableau clause G' by G. In the resulting tableau $\mathcal{P}$ the *Goal*-clause is G, as desired. Finally apply Proosition 2.9 to obtain the claimed refutation. ■

4 Conclusions

We turn to related work. Among the various calculi sharing the idea of avoiding contrapositives, the closest relative to RME is certainly inheritance near-Horn Prolog (InH-Prolog) [Loveland and Reed, 1991]. See [Baumgartner and Furbach, 1994a] for an anchor and a comparison of RME to InH-Prolog. In the present paper, RME and InH-Prolog even get closer due to the "early cancellation pruning" refinement (RMEP, as we call our calculus); the early cancellation pruning refinement was introduced for InH-Prolog in [Loveland and Reed, 1991] and was termed "strong" there. Nevertheless, we think that our RMEP variant is an original contribution, as it can be seen as a solution to some open issues for InH-Prolog. These are the following refinements which were not considered for InH-Prolog: the first is the possibility to use any computation rule. Currently, this is more a theoretical device and we did not yet explore its practical relevance. The second is the *block-wise regularity* refinement. Regularity is known as one of the most effective pruning methods for ME calculi and should thus be included in the strongest possible way. The third is the *independence of the goal clause*. At a first glance it might seem that not much is gained by it, because at restart extension steps any negative input clause must be considered for a restart step anyway. While this might be true to some extent for typical, minimal unsatisfiable benchmark examples, we experienced the value of the refinement for more real-world like examples. We tried clause sets containing about 1200 clauses, stemming from a program verification task of the correctness of a WAM compiler. Thereof, 85 clauses are negative, and the theorem to be proven is a negative clause as well. The clause set is highly *non*-minimal unsatisfiable, and considering each of the 85 negative clauses as a goal clause makes it impossible to find refutations in acceptable time (about half an hour). On the other side, taking advantage of the independence refinement, the refutations show up immediately in many cases by taking the theorem as the only goal clause.

There are many more refinements conceivable. We indicate some of those. In [Baumgartner *et al.*, 1995] we investigated RME as a calculus for *answer computation*. In particular, we showed that a certain variant called *ancestry* RME has nice properties for the computation of definite answers[6]. It should be possible to further refine RMEP in this way.

Another refinement could be called *local proof confluence:* whenever a RME tableau T is derived such that all open branches end in positive leaves, it should be possible to continue *this* tableau T to a refutation (if one exists at all). The benefit would be to not backtrack over T, thus saving a lot of search.

References

[Baumgartner and Furbach, 1994a] P. Baumgartner and U. Furbach. Model Elimination without Contrapositives and its Application to PTTP. *Journal of Automated Reasoning*, 13:339–359, 1994. Short version in: Proceedings of CADE-12, Springer LNAI 814, 1994, pp 87–101.

[Baumgartner and Furbach, 1994b] 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. Available in the WWW, URL: `http://www.uni-koblenz.de/ag-ki/Systems/PROTEIN/`.

[6] I.e. the solution to a posed problem is *not* a disjunction such as $apply(blood - letting) \vee apply(hot\_pack)$.

[Baumgartner *et al.*, 1995] P. Baumgartner, U. Furbach, and F. Stolzenburg. Model Elimination, Logic Programming and Computing Answers. In *14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, volume 1, 1995. (Long version in: Research Report 1/95, University of Koblenz, Germany. To appear in *Artificial Intelligence*).

[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.

[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 and Reed, 1991] D.W. Loveland and D.W. Reed. A near-Horn Prolog for Compilation. In Jean-Luis Lassez and Gordon Plotkin, editors, *Computational Logic — Essays in Honor of Alan Robinson*, chapter III/16, pages 542–564. MIT Press, 1991.

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

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

Available Research Reports (since 1994):

1996

24/96 *Peter Baumgartner, Ulrich Furbach.* Refinements for Restart Model Elimination.

23/96 *Peter Baumgartner, Peter Fröhlich, Ulrich Furbach, Wolfgang Nejdl.* Tableaux for Diagnosis Applications.

22/96 *Jürgen Ebert, Roger Süttenbach, Ingar Uhe.* Meta-CASE in Practice: a Case for KOGGE.

21/96 *Harro Wimmel, Lutz Priese.* Algebraic Characterization of Petri Net Pomset Semantics.

20/96 *Wenjin Lu.* Minimal Model Generation Based on E-Hyper Tableaux.

19/96 *Frieder Stolzenburg.* A Flexible System for Constraint Disjunctive Logic Programming.

18/96 *Ilkka Niemelä (Ed.).* Proceedings of the ECAI'96 Workshop on Integrating Nonmonotonicity into Automated Reasoning Systems.

17/96 *Jürgen Dix, Luis Moniz Pereira, Teodor Przymusinski.* Non-monotonic Extensions of Logic Programming: Theory, Implementation and Applications (Proceedings of the JICSLP '96 Postconference Workshop W1).

16/96 *Chandrabose Aravindan.* DisLoP: A Disjunctive Logic Programming System Based on PROTEIN Theorem Prover.

15/96 *Jürgen Dix, Gerhard Brewka.* Knowledge Representation with Logic Programs.

14/96 *Harro Wimmel, Lutz Priese.* An Application of Compositional Petri Net Semantics.

13/96 *Peter Baumgartner, Ulrich Furbach.* Hyper Tableaux and Disjunctive Logic Programming.

12/96 *Klaus Zitzmann.* Physically Based Volume Rendering of Gaseous Objects.

11/96 *J. Ebert, A. Winter, P. Dahm, A. Franzke, R. Süttenbach.* Graph Based Modeling and Implementation with EER/GRAL.

10/96 *Angelika Franzke.* Querying Graph Structures with G^{2}QL.

9/96 *Chandrabose Aravindan.* An abductive framework for negation in disjunctive logic programming.

8/96 *Peter Baumgartner, Ulrich Furbach, Ilkka Niemelä .* Hyper Tableaux.

7/96 *Ilkka Niemelä, Patrik Simons.* Efficient Implementation of the Well-founded and Stable Model Semantics.

6/96 *Ilkka Niemelä .* Implementing Circumscription Using a Tableau Method.

5/96 *Ilkka Niemelä .* A Tableau Calculus for Minimal Model Reasoning.

4/96 *Stefan Brass, Jürgen Dix, Teodor. C. Przymusinski.* Characterizations and Implementation of Static Semantics of Disjunctive Programs.

3/96 *Jürgen Ebert, Manfred Kamp, Andreas Winter.* Generic Support for Understanding Heterogeneous Software.

2/96 *Stefan Brass, Jürgen Dix, Ilkka Niemelä, Teodor. C. Przymusinski.* A Comparison of STATIC Semantics with D-WFS.

1/96 *J. Ebert (Hrsg.).* Alternative Konzepte für Sprachen und Rechner, Bad Honnef 1995.

1995

21/95 *J. Dix and U. Furbach.* Logisches Programmieren mit Negation und Disjunktion.

20/95 *L. Priese, H. Wimmel.* On Some Compositional Petri Net Semantics.

19/95 *J. Ebert, G. Engels.* Specification of Object Life Cycle Definitions.

18/95 *J. Dix, D. Gottlob, V. Marek.* Reducing Disjunctive to Non-Disjunctive Semantics by Shift-Operations.

17/95 *P. Baumgartner, J. Dix, U. Furbach, D. Schäfer, F. Stolzenburg.* Deduktion und Logisches Programmieren.

16/95 *Doris Nolte, Lutz Priese.* Abstract Fairness and Semantics.

15/95 *Volker Rehrmann (Hrsg.).* 1. Workshop Farbbildverarbeitung.

14/95 *Frieder Stolzenburg, Bernd Thomas.* Analysing Rule Sets for the Calculation of Banking Fees by a Theorem Prover with Constraints.

13/95 *Frieder Stolzenburg.* Membership-Constraints and Complexity in Logic Programming with Sets.

12/95 *Stefan Brass, Jürgen Dix.* D-WFS: A Confluent Calculus and an Equivalent Characterization..

11/95 *Thomas Marx.* NetCASE — A Petri Net based Method for Database Application Design and Generation.

10/95 *Kurt Lautenbach, Hanno Ridder.* A Completion of the S-invariance Technique by means of Fixed Point Algorithms.

9/95 *Christian Fahrner, Thomas Marx, Stephan Philippi.* Integration of Integrity Constraints into Object-Oriented Database Schema according to ODMG-93.

8/95 *Christoph Steigner, Andreas Weihrauch.* Modelling Timeouts in Protocol Design..

7/95 *Jürgen Ebert, Gottfried Vossen.* I-Serializability: Generalized Correctness for Transaction-Based Environments.

6/95 *P. Baumgartner, S. Brüning.* A Disjunctive Positive Refinement of Model Elimination and its Application to Subsumption Deletion.

5/95 *P. Baumgartner, J. Schumann.* Implementing Restart Model Elimination and Theory Model Elimination on top of SETHEO.

4/95 *Lutz Priese, Jens Klieber, Raimund Lakmann, Volker Rehrmann, Rainer Schian.* Echtzeit-Verkehrszeichenerkennung mit dem Color Structure Code — Ein Projektbericht.

3/95 *Lutz Priese.* A Class of Fully Abstract Semantics for Petri-Nets.

2/95 *P. Baumgartner, R. Hähnle, J. Posegga (Hrsg.).* 4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods — Poster Session and Short Papers.

1/95 *P. Baumgartner, U. Furbach, F. Stolzenburg.* Model Elimination, Logic Programming and Computing Answers.

1994

18/94 *W. Hower, D. Haroud, Z. Ruttkay (Eds.).* Proceedings of the AID'94 workshop W9 on Constraint Processing in Computer-Aided Design.

17/94 *W. Hower.* Constraint satisfaction — algorithms and complexity analysis.

16/94 *S. Brass, J. Dix.* A Disjunctive Semantics Based on Unfolding and Bottom-Up Evaluation.

15/94 *S. Brass, J. Dix.* A Characterization of the Stable Semantics by Partial Evaluation.

14/94 *Michael Möhring.* Grundlagen der Prozeßmodellierung.

13/94 *D. Zöbel.* Program Transformations for Distributed Control Systems.

12/94 *Martin Volk, Michael Jung, Dirk Richarz, Arne Fitschen, Johannes Hubrich, Christian Lieske, Stefan Pieper, Hanno Ridder, Andreas Wagner.* GTU – A workbench for the development of natural language grammars.

11/94 *S. Brass, J. Dix.* A General Approach to Bottom-Up Computation of Disjunctive Semantics.

10/94 *P. Baumgartner, F. Stolzenburg.* Constraint Model Elimination and a PTTP Implementation.

9/94 *K.-E. Großpietsch, R. Hofestädt, C. Steigner (Hrsg.).* Workshop Parallele Datenverabeitung im Verbund von Hochleistungs-Workstations.

8/94 *Baumgartner, Bürckert, Comon, Frisch, Furbach, Murray, Petermann, Stickel (Hrsg.).* Theory Reasoning in Automated Deduction.

7/94 *E. Ntienjem.* A descriptive mode inference for normal logic programs.

6/94 *A. Winter, J. Ebert.* Ein Referenz-Schema zur Organisationsbeschreibung.

5/94 *F. Stolzenburg.* Membership-Constraints and Some Applications.

4/94 *J. Ebert (Hrsg.).* Alternative Konzepte für Sprachen und Rechner – Bad Honnef 1993.

3/94 *J. Ebert, A. Franzke.* A Declarative Approach to Graph Based Modeling.

2/94 *M. Dahr, K. Lautenbach, T. Marx, H. Ridder.* NET CASE: Towards a Petri Net Based Technique for the Development of Expert/Database Systems.

1/94 *U. Furbach.* Theory Reasoning in First Order Calculi.