Computing Answers with Model Elimination

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

Abstract

We demonstrate that theorem provers using model elimination (ME) can be used as answer-complete interpreters for *disjunctive logic programming*. More specifically, we introduce a mechanism for computing answers into the restart variant of ME. Building on this we develop a new calculus called *ancestry restart ME*. This variant admits a more restrictive regularity restriction than restart ME, and, as a side effect, it is in particular attractive for computing definite answers. The presented calculi can also be used successfully in the context of *automated theorem proving*. We demonstrate experimentally that it is more difficult to compute (non-trivial) answers to goals, instead of only proving the *existence* of answers.

Keywords. Automated reasoning; theorem proving; model elimination; logic programming; computing answers.

In first order automatic theorem proving one is interested in the question whether a given formula follows logically from a set of axioms. This is a rather artificial task; whenever one is interested in solving problems it is mandatory to compute answers for given questions. It appears to us, that in automated theorem proving this aspect has been pushed to the background. For instance, the statements of the puzzle problems in the TPTP library [Sutcliffe *et al.*, 1994] also include their solutions, and the prover has only to *verify* them; however, it is much more interesting (and more difficult) to *find* a solution instead of only proving correctness of a given one.

In the early days, when automated theorem provers were understood as tools for real world problem solving, this problem was apparent: of course the textbook-monkey was not interested whether *there is* a solution of the monkey-and-banana-problem; it was merely interested in finding a way to reach the banana hanging on the ceiling. In [Chang and Lee, 1973] there is a whole chapter on question answering, where the work of C. Green on question answering is reviewed thoroughly. In modern theorem proving literature this aspect is not payed sufficient attention.

This is different however, when automated deduction is investigated with respect to a special domain. For example, in the database context there is a lot of research aiming at deriving answers and even cooperative or intensional answers (see e.g. [Demolombe and Imielski, 1994]). In non-monotonic reasoning there are even philosophical discussions which semantics have to been chosen to allow for intuitive answers.
In the logic programming area the computation of answers was an important aspect from the very beginning. From there we learned that it is much more difficult to prove a calculus answer complete instead of only showing its refutational completeness. Recently there is considerable effort to use full first order logic instead of only Horn clauses as a base for logic programming. This disjunctive logic programming approach is investigated from various directions: From the view of theorem proving one is concerned with the problem of modifying theorem provers such that they can be used as interpreters for logic programming purposes. The non-monotonic reasoning community is working on finding appropriate semantics for disjunctive logic programs with negation and from a database viewpoint one is concerned with finding bottom up approaches to computations.

The aim of this paper is twofold: Firstly, we prove that theorem provers using model elimination (ME) can be used as answer complete interpreters for disjunctive logic programming. Secondly, we demonstrate that in the context of automated theorem proving it is much more difficult to compute (non-trivial) answers to goals, instead of only to prove the existence of answers. We furthermore investigate mechanisms for finding special answers.

Concerning the first aspect, it is important to note that there is a lot of work towards model theoretic semantics of **positive disjunctive logic programs**, and of course there are numerous proposals for non-monotonic extensions. However, with respect to proof theory, the situation is not so clear. At first glance one might be convinced that any first order theorem prover can be used for the interpretation of disjunctive logic programs, since a program clause $\mathbf{A}_1 \vee \ldots \vee \mathbf{A_m} \leftarrow \mathbf{B}_1 \wedge \ldots \wedge \mathbf{B_n}$ is a representation of the clause $\mathbf{A}_1 \vee \ldots \vee \mathbf{A_m} \vee \neg\mathbf{B}_1 \vee \ldots \vee \neg\mathbf{B_n}$. Indeed, in [Lobo *et al.*, 1992] SLI-resolution is used as a calculus for disjunctive logic programming. From logic programming with Horn clauses, however, we learn that for a procedural interpretation of program clauses it is crucial that clauses can only be accessed by the literals $\mathbf{A_i}$, i.e. by the head literals. Technically, this means that only those contrapositives are allowed to be used which contain a positive literal in the head. The approach from [Lobo *et al.*, 1992] completely ignores this aspect by using SLI resolution which requires all contrapositives.

There are proposals for first order proof calculi using program clauses only in this procedural reading, e.g. Plaisted's problem reduction formats [Plaisted, 1988], or the near-Horn-Prolog family introduced by Loveland and his co-workers [Loveland, 1991]. These approaches introduce new calculi or proof procedures, for which efficient implementations still have to be developed. (For a thorough discussion we refer to [Baumgartner and Furbach, 1994a].) Our aim was to modify ME such that it can be used for logic programming in the above sense. This gives us the possibility to use existing the-

orem provers for disjunctive logic programming. As a first step towards this goal, we introduced in [Baumgartner and Furbach, 1994a] the restart variant of ME and proved its refutational completeness. In this paper, we introduce an answer computing mechanism into restart model elimination . Furthermore we define a variant called *ancestry restart ME* which allows extended regularity checking (i.e. loop checking) wrt. the ordinary restart ME. Additionally this variant prefers proofs which permit definite answers.

For the second aspect, namely **computing answers**, we accommodated our PROTEIN system [Baumgartner and Furbach, 1994b] for answer computing as described below.

We demonstrate with some of Smullyan's puzzles [Smullyan, 1978] that it is much more difficult to compute answers instead of only to prove unsatisfiability. We show how the model elimination calculus can be modified such that it preferably computes definite answers.

Finally we give a comparative study, of high performance theorem provers, including OTTER, SATCHMO, SETHEO and our PROTEIN system.

A short version of this work has appeared in [Baumgartner *et al.*, 1995].

1 From Tableau to Restart Model Elimination

1.1 Tableau Model Elimination

In this subsection we use the clause notation, mirroring the fact that we review a calculus which is, as it stands, not suited for programming purposes. We use a ME calculus that differs from the original one presented in [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 compared to various other calculi. ME (in this sense) manipulates trees by extension and reduction steps. In order to recall the calculus consider the clause set

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

A model elimination refutation is depicted in Figure 1 (left side). It is obtained by successive fanning with clauses from the input set (*extension steps*). Additionally, it is required that every inner node is complementary to one of its sons. Such sons are decorated with a "$\star$" in Figure 1. A dashed arrow indicates a *reduction step*, i.e. the closing of a branch due to a path literal complementary to the leaf literal. Extension and reduction steps are allowed at any leaf of the tree and for extension steps any literal from an input clause can be used to form a complementary pair of literals. For example, in the right subtree of Figure 1 (left side) the clause $\{\neg\mathbf{P}, \mathbf{Q}\}$ was used to extend the positive leaf $\mathbf{P}$, i.e. we used the program clause $\mathbf{Q} \leftarrow \mathbf{P}$ via the body literal $\mathbf{P}$ and hence disposed with a procedural reading of the clause.

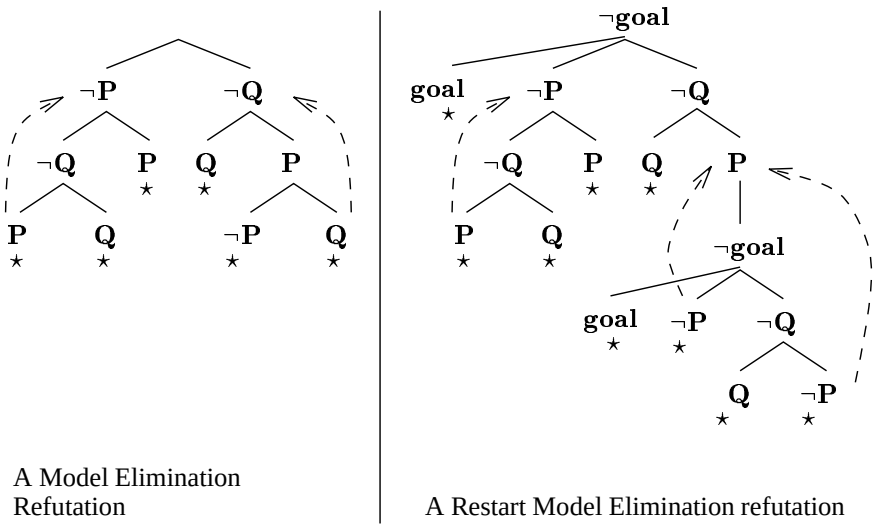

Figure 1: Model Elimination (left side) vs. Restart Model Elimination (right side).

Now we show how restart model elimination treats this example. As a preliminary step, the input clause set has to be transformed into what we call *goal normal form:* every purely negative clause is conjoined with the new literal $\mathbf{goal}$. Thus, in the example, we replace $\{\neg\mathbf{P}, \neg\mathbf{Q}\}$ by $\{\mathbf{goal}, \neg\mathbf{P}, \neg\mathbf{Q}\}$. Further, as query the unit clause $\neg\mathbf{goal}$ is used. Figure 1 (right side) displays a restart model elimination refutation. Besides using the goal normal form, the only difference is that extension steps at positive literals are not allowed; instead either a reduction step is carried out, or else the root literal — which is always $\neg\mathbf{goal}$ — is copied, and then an extension follows.

In a variant called *strict* restart model elimination not even reduction steps are allowed at positive leaves. Hence the calculus is forced to apply restart steps wherever possible. Note that the purpose of the goal normal form transformation is simply to "restart" derivations at positive literals with any negative clause from the input set.

These simple modifications obviously allow only extension steps with a positive, i.e. a head literal of a clause, and hence support a procedural reading of program clauses. In the following subsection we give a formal presentation of the calculus along the lines of [Baumgartner and Furbach, 1993].

1.2 Restart Model Elimination

Instead of trees we now manipulate multisets of paths, where paths are sequences of literals. We will state some basic definitions.

A *clause* is a multiset of literals, usually written as the disjunction $\mathbf{L}_1 \vee \ldots \vee \mathbf{L_n}$. A *program* is a consistent set of clauses (thus possibly including negative clauses).

A *connection* is a pair of literals, written as $(\mathbf{K}, \mathbf{L})$, which can be made complementary by an application of a substitution. A *path* is a sequence of literals, written as $\mathbf{p} = \langle \mathbf{L}_1, \ldots, \mathbf{L_n} \rangle$. $\mathbf{L_n}$ is called the *leaf* of $\mathbf{p}$, which is also denoted by $\mathbf{leaf}(\mathbf{p})$; similarly, the first element $\mathbf{L}_1$ is also denoted by $\mathbf{first}(\mathbf{p})$. The symbol "$\circ$" denotes the append function for literal sequences.

In the sequel both path sets and sets of literals are always understood as *multi*sets, and usual set notation will be used. Multisets of paths are written with caligraphic capital letters.

From now on we use the notation $\mathbf{A}_1 \vee \ldots \vee \mathbf{A_m} \leftarrow \mathbf{B}_1 \wedge \ldots \wedge \mathbf{B_n}$ as a representation of the clause $\mathbf{A}_1 \vee \ldots \vee \mathbf{A_m} \vee \neg \mathbf{B}_1 \vee \ldots \vee \neg \mathbf{B_n}$. Such clauses are called *program clauses* with *head literals* $\mathbf{A_i}$ (if present) and *body literals* $\mathbf{B_i}$.

We assume our clause sets to be in *goal normal form*, i.e. there exists only one goal clause (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 $\leftarrow \mathbf{goal}$ where $\mathbf{goal}$ is a new predicate symbol, and by modifying every purely negative clause $\neg \mathbf{B}_1 \vee \ldots \vee \neg \mathbf{B_n}$ to $\mathbf{goal} \leftarrow \mathbf{B}_1, \ldots, \mathbf{B_n}$.

If $\mathbf{C} = \mathbf{A}_1 \vee \ldots \vee \mathbf{A_m} \leftarrow \mathbf{B}_1 \wedge \ldots \wedge \mathbf{B_n}$ is a clause then its *path set* $\mathcal{P}_\mathbf{C}$ is $\{\langle \mathbf{L} \rangle \mid \mathbf{L} \in \{\mathbf{A}_1, \ldots, \mathbf{A_m}, \neg \mathbf{B}_1, \ldots, \neg \mathbf{B_n}\}\}$.

The inference rule *extension* from the restart ME calculus,will be defined in such a way that one is free in selecting any head literal as part of a connection. For this we introduce a head selection function.

DEFINITION 1.1
(Head selection Function) A *head selection function* $\mathbf{f}$ is a function that maps a clause $\mathbf{A}_1 \vee \ldots \vee \mathbf{A_n} \leftarrow \mathbf{B}_1 \wedge \ldots \wedge \mathbf{B_m}$ with $\mathbf{n} \geq 1$ to an atom $\mathbf{L} \in \{\mathbf{A}_1, \ldots, \mathbf{A_n}\}$. $\mathbf{L}$ is called the *selected literal* of that clause by $\mathbf{f}$. The head selection function $\mathbf{f}$ is required to be *stable under lifting* which means that if $\mathbf{f}$ selects $\mathbf{L}\gamma$ in the instance of the clause $(\mathbf{A}_1 \vee \ldots \vee \mathbf{A_n} \leftarrow \mathbf{B}_1 \wedge \ldots \wedge \mathbf{B_m})\gamma$ (for some substitution γ) then $\mathbf{f}$ selects $\mathbf{L}$ in $\mathbf{A}_1 \vee \ldots \vee \mathbf{A_n} \leftarrow \mathbf{B}_1 \wedge \ldots \wedge \mathbf{B_m}$. (END DEFINITION)

Note that this head selection function has nothing to do with the selection function from SLD-resolution which selects subgoals. This will be discussed later.

DEFINITION 1.2
(Strict Restart Model Elimination) Given a set of clauses $\mathbf{S}$ and a head selection function.

The inference rule *extension* is defined as follows:

$$\frac{\mathcal{P} \cup \{\mathbf{p}\} \quad \mathbf{A}_1 \vee \ldots \vee \mathbf{A_i} \vee \ldots \vee \mathbf{A_m} \leftarrow \mathbf{B}_1 \wedge \ldots \wedge \mathbf{B_n}}{\mathcal{R}}$$

where

1. $\mathcal{P} \cup \{\mathbf{p}\}$ is a path multiset, and $\mathbf{A}_1 \vee \ldots \vee \mathbf{A_i} \vee \ldots \vee \mathbf{A_m} \leftarrow \mathbf{B}_1 \wedge \ldots \wedge \mathbf{B_n}$ is a variable disjoint variant of a clause in $\mathbf{S}$; $\mathbf{A_i}$ is the selected literal, and

2. $(\mathbf{leaf}(\mathbf{p}), \mathbf{A_i})$ is a connection with MGU σ, and

3. $\mathcal{R} = (\mathcal{P} \cup \{\mathbf{p} \circ \langle \mathbf{K} \rangle \mid \mathbf{K} \in \{\mathbf{A}_1, \ldots, \mathbf{A_{i-1}}, \mathbf{A_{i+1}}, \ldots, \mathbf{A_m}, \neg\mathbf{B}_1, \ldots, \neg\mathbf{B_n}\}\})\sigma$

The inference rule *reduction* is defined as follows:

$$\frac{\mathcal{P} \cup \{\mathbf{p}\}}{\mathcal{P}\sigma} \quad \text{where}$$

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

2. there is a positive literal $\mathbf{L}$ in $\mathbf{p}$ such that $(\mathbf{L}, \mathbf{leaf}(\mathbf{p}))$ is a connection with MGU σ.

The inference rule *restart* is defined as follows:

$$\frac{\mathcal{P} \cup \{\mathbf{p}\}}{\mathcal{P} \cup \{\mathbf{p} \circ \langle \mathbf{L} \rangle\}} \quad \text{where}$$

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

2. $\mathbf{leaf}(\mathbf{p})$ is a positive literal, and

3. $\mathbf{L} = \mathbf{first}(\mathbf{p})$.

A *strict restart ME derivation* from the clause set $\mathbf{S}$ consists of a sequence $(\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P_n})$ and a substitution $\sigma_1 \cdots \sigma_\mathbf{n}$, where

1. $\mathcal{P}_0$ is a path multiset $\{\langle \mathbf{L}_1 \rangle, \ldots, \langle \mathbf{L_n} \rangle\}$ consisting of paths of length 1, with $\mathbf{L}_1 \vee \ldots \vee \mathbf{L_n}$ in $\mathbf{S}$ (also called the *goal clause*), and for $\mathbf{i} = 1 \ldots \mathbf{n}$

2. $\mathcal{P_i}$ is obtained from $\mathcal{P}_{\mathbf{i}-1}$ by means of an extension step with an appropriate clause $\mathbf{C}$ from $\mathbf{S}$ and MGU $\sigma_\mathbf{i}$, or

3. $\mathcal{P_i}$ is obtained from $\mathcal{P}_{\mathbf{i}-1}$ by means of a reduction step and MGU $\sigma_\mathbf{i}$, or

4. $\mathcal{P_i}$ is obtained from $\mathcal{P}_{\mathbf{i}-1}$ by means of a restart step.

The path $\mathbf{p}$ is called *selected path* in all three inference rules. A restart step followed immediately by an extension step is also called a *restart extension step*. Finally, a *refutation* is a derivation where $\mathcal{P_n} = \{\}$. (END DEFINITION)

Note that in extension steps we can connect only with the head literals of input clauses. Since in general this restriction is too strong, we have to "restart" the computation with a fresh copy of a negative clause. This is achieved by the restart rule, because refutations of programs in goal normal form always start with $\mathbf{first(p)} \equiv \neg\mathbf{goal}$, and thus only extension steps are possible to $\neg\mathbf{goal}$, which in turn introduce a new copy of a negative clause (cf. Figure 1, right side).

The reduction operation is permitted from negative leaf literals to positive ancestor literals only. This condition can be relaxed towards disregarding the sign, which then yields the *non-strict* calculus version. See [Baumgartner and Furbach, 1994a] for a discussion of the differences. The reader acquainted of this work will notice that in the present text we define the calculus slightly differently. This happens in order to conveniently express another calculus variant defined below.

Note that the restart ME calculus does not assume a special selection function for determining which path is to be extended or reduced next. Correctness and completeness of this calculus follows immediately from a result in [Baumgartner, 1994]. From the definition of the inference rule extension, it follows immediately, that this calculus only needs those contrapositives of clauses which contain a positive literal in their heads.

The following result is due to [Baumgartner and Furbach, 1994a]:

THEOREM 1.3
(Ground completeness of Strict Restart Model Elimination)
Let $\mathbf{f}$ *be a head selection function and* $\mathbf{S}$ *be an unsatisfiable ground clause set in* $\mathbf{goal}$*-normal form. Then there exists a strict restart model elimination refutation of* $\mathbf{S}$ *with goal* $\leftarrow \mathbf{goal}$ *and selection function* $\mathbf{f}$*.*

In [Baumgartner and Furbach, 1994a] we also gave the lifting arguments, but we did not carry out the proof explicitly. In this paper, this is done.

2 Computing Answers

In this section we introduce the notion of computed answers and we prove an answer completeness result for restart ME. We assume as given a program $\mathbf{P}$ together with one single *query* $\leftarrow \mathbf{G_1} \wedge \ldots \wedge \mathbf{G_n}$, where the $\mathbf{G_i}$s are positive literals. We will often abbreviate such a query as $\leftarrow \mathbf{Q}$, where $\mathbf{Q}$ stands for the conjunction of $\mathbf{G_i}$s. The clause set $\mathbf{S}$ is the transformation of $\mathbf{P} \cup \{\leftarrow \mathbf{Q}\}$ into goal normal form. In the following definition of computed answer we collect applications of the query clause, but not applications of negative clauses from the program $\mathbf{P}$.

DEFINITION 2.1
(Answers) If $\leftarrow \mathbf{Q}$ is a query and $\theta_1, \ldots, \theta_\mathbf{m}$ are substitutions for the variables from $\mathbf{Q}$, then $\mathbf{Q}\theta_1 \vee \ldots \vee \mathbf{Q}\theta_\mathbf{m}$ is an *answer* (for $\mathbf{S}$). An answer $\mathbf{Q}\theta_1 \vee \ldots \vee \mathbf{Q}\theta_\mathbf{m}$ is a *correct*

answer if $\mathbf{P} \models \forall(\mathbf{Q}\theta_1 \vee \ldots \vee \mathbf{Q}\theta_\mathbf{m})$. Now let a restart ME refutation of $\mathbf{S}$ with goal clause $\leftarrow \mathbf{goal}$ and substitution σ be given. Assume that this refutation contains $\mathbf{m}$ extension steps with the query, i.e. it contains $\mathbf{m}$-times an extension step with the clause $\mathbf{goal} \leftarrow \mathbf{Q}\rho_\mathbf{i}$, where $\rho_\mathbf{i}$ is the renaming substitution of this step. Let $\sigma_\mathbf{i} = \rho_\mathbf{i}\sigma|_{\mathbf{dom}(\rho_\mathbf{i})}$. Then $\mathbf{Q}\sigma_1 \vee \ldots \vee \mathbf{Q}\sigma_\mathbf{m}$ is a *computed answer* (for $\mathbf{S}$). (END DEFINITION)

THEOREM 2.2

(Lifting Theorem for Restart Model Elimination) *Let $\mathbf{S}'$ be a set of ground instances of clauses taken from a clause set $\mathbf{S}$. Assume there exists a restart ME derivation $\mathbf{D}' \equiv \mathbf{P}'_0, \mathbf{P}'_1, \ldots, \mathbf{P}'_\mathbf{n}$ from $\mathbf{S}'$ with goal clause $\mathbf{C}'_0 \in \mathbf{S}'$. Then there exists a restart ME derivation $\mathbf{D} \equiv \mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_\mathbf{n}$ from $\mathbf{S}$ with some goal clause $\mathbf{C}_0 \in \mathbf{S}$ and substitution σ such that $\mathbf{P}_\mathbf{n}$ is more general than $\mathbf{P}'_\mathbf{n}$. (A path set $\mathbf{P}$ is* more general *than a path set $\mathbf{Q}$ iff for some substitution δ we have $\mathbf{P}\delta = \mathbf{Q}$.)*

Furthermore, there exists a substitution δ such that $\mathbf{P}'_\mathbf{i}$ is obtained from $\mathbf{P}'_{\mathbf{i}-1}$ by an extension step with clause $\mathbf{C}' \in \mathbf{S}'$ if and only if $\mathbf{P}_\mathbf{i}$ is obtained from $\mathbf{P}_{\mathbf{i}-1}$ by an extension step with a clause $\mathbf{C} \in \mathbf{S}$ such that $\mathbf{C}\rho_\mathbf{i}\sigma\delta = \mathbf{C}'$, where $\rho_\mathbf{i}$ is the renaming substitution applied in that extension step.

The first part of the theorem will be used in the proof of *refutational completeness* (because for a refutation on the ground level, i.e. a derivation of $\mathbf{P}'_\mathbf{n} = \{\}$, only the empty path set $\mathbf{P}_\mathbf{n} = \{\}$ can be more general), while the second part will be used in the proof of *answer completeness* (Theorem 2.3). In particular, to obtain this we have to demand *one single* substitution δ which maps any of the clauses $\mathbf{C}\rho_\mathbf{i}\sigma$ used in extension steps to the respective clause on the ground level.

Clearly, this result is harder to establish and more relevant than a lifting result for SLI-resolution in [Lobo *et al.*, 1992] which "moves the $\exists$ quantification inside": in our words, they state that for every application of an input clause at the ground level there exists an application at the first-order level, and there exists a substitution to map *this instance* to the ground level. We may even conclude that the approach of [Lobo *et al.*, 1992] cannot handle the case correctly if there are variable interdependencies in disjunctive answers.

Let us consider the program $\mathbf{P} = \mathbf{p}(\mathbf{f}(\mathbf{x}))$ and the query $\mathbf{Q} = ?\text{-}\mathbf{p}(\mathbf{z})$. Then according to [Lobo *et al.*, 1992] $\mathbf{A} = \mathbf{p}(\mathbf{z}) \vee \mathbf{p}(\mathbf{f}(\mathbf{z}))$ is a correct and complete answer set, although $\mathbf{A}$ does not subsume $\mathbf{p}(\mathbf{f}(\mathbf{x}))$. Thus we would be incomplete. On the other hand, if we permit each literal to be substituted separately, in order to fix this problem, then we would be incorrect. For this let us consider the program $\mathbf{P}' = \mathbf{p}(\mathbf{x}) \vee \mathbf{p}(\mathbf{f}(\mathbf{x}))$. Then we could get also the answer $\mathbf{A}$ with respect to the query $\mathbf{Q}$ from above. But we could factor it to $\mathbf{p}(\mathbf{f}(\mathbf{z}))$ which is not entailed by $\mathbf{P}'$.

Now follows the proof of the theorem.

PROOF. The basic proof plan is to show by induction on $\mathbf{n}$ that $\mathbf{P}_\mathbf{n}$ can be mapped by application of a certain substitution $\delta_\mathbf{n}$ to $\mathbf{P}'_\mathbf{n}$. This approach would suffice to lift a derivation to the first-order level. However, as stated in the second part of the theorem, we need moreover a lifting result for the clauses used in extension steps. The difficulty in

proving this is that in $\mathbf{P'_n}$ a clause used in a former extension step need no longer be present in $\mathbf{P'_n}$ (for instance, in a refutation $\mathbf{P'_n} = \{\}$). This is due to the fact that both extension and reduction steps delete paths. We thus have to explicitly keep track of the used clauses.

In order to make things technically manageable we first define a clause set $\mathbf{S_v}$ as follows: $\mathbf{S_v}$ is a set of, say $\mathbf{l}$, pairwise variable disjoint clauses, and $\mathbf{S_v}$ contains for every ground instance $\mathbf{C_k}\gamma_\mathbf{k} \in \mathbf{S'}$ ($\mathbf{k} = 1, \ldots, \mathbf{l}$) of a clause $\mathbf{C_k} \in \mathbf{S}$ a variant $\mathbf{C_k}\tau_\mathbf{k}$. Furthermore $\mathbf{C_k}\tau_\mathbf{k}$ is supposed to be variable disjoint from $\mathbf{S}$.

At first we will show that there exists one single ground substitution γ which can be used instead of the individual $\gamma_\mathbf{k}$'s. More precisely, we define first $\gamma'_\mathbf{k} = \tau_\mathbf{k}^{-1}\gamma_\mathbf{k}|_{\mathbf{vcod}(\tau_\mathbf{k})}$, because then we have

$$(\mathbf{C_k}\tau_\mathbf{k})\gamma'_\mathbf{k} = \mathbf{C_k}\gamma_\mathbf{k} \tag{1}$$

Moreover, since the clauses in $\mathbf{S_v}$ are assumed to be pairwise variable disjoint (by means of the $\tau_\mathbf{k}$s) and because of the domain restriction of the $\gamma'_\mathbf{k}$s it follows $(\mathbf{C_k}\tau_\mathbf{k})\gamma'_\mathbf{k} = (\mathbf{C_k}\tau_\mathbf{k})\gamma'_1 \cdots \gamma'_\mathbf{l}$. But then, defining $\gamma = \gamma'_1 \cdots \gamma'_\mathbf{l}$ and using (1) we recognise more generally that $\mathbf{S_v}\gamma = \mathbf{S'}$.

It follows with $\mathbf{D'}$ being a derivation from $\mathbf{S'}$ that $\mathbf{D'}$ is also a derivation from $\mathbf{S_v}\gamma$. We will show how to lift $\mathbf{D'}$ from $\mathbf{S_v}\gamma$ to the first-order level. In order to do so we have to define a slightly more general induction invariant $\mathbf{I(n)}$ than the theorem gives us. In present notation it reads as follows:

> $\mathbf{I(n)}$: there exists a restart ME derivation $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P_n}$ from $\mathbf{S}$ with substitution $\sigma_1 \cdots \sigma_\mathbf{n}$, and there exists a substitution $\delta_\mathbf{n}$ such that
>
> **Invariant 1:** $\mathbf{P_n}\delta_\mathbf{n} = \mathbf{P'_n}$ and
>
> **Invariant 2:** whenever $\mathbf{P'_i}$ ($\mathbf{i} = 1 \ldots \mathbf{n}$) is obtained from $\mathbf{P'_{i-1}}$ by an extension step with a clause $\mathbf{C_v}\gamma \in \mathbf{S_v}\gamma$, then $\mathbf{P_i}$ is obtained from $\mathbf{P_{i-1}}$ by an extension step with some clause $\mathbf{C} \in \mathbf{S}$ such that
>
> $$\mathbf{C}\rho_\mathbf{i}\sigma_1 \cdots \sigma_\mathbf{n}\delta_\mathbf{n} = \mathbf{C_v}\gamma$$
>
> where $\rho_\mathbf{i}$ is the renaming substitution used in the $\mathbf{i}$-th step to obtain a new variant of $\mathbf{C}$.

Clearly, $\mathbf{I(n)}$ proves the theorem using the identity $\mathbf{C'} = \mathbf{C_v}\gamma$, and defining $\sigma := \sigma_1 \cdots \sigma_\mathbf{n}$ and $\delta := \delta_\mathbf{n}$. The if-direction of the theorem's statement (i.e. that $\mathbf{D}$ does not need more clauses for extension steps than $\mathbf{D'}$) follows from the construction given below.

It thus remains to prove $\mathbf{I(n)}$ (induction on $\mathbf{n}$):
$\boxed{\mathbf{n = 0}}$: $\mathbf{P'_0}$ is a path set for the query $\mathbf{C}_0\gamma \in \mathbf{S_v}\gamma$. By construction of $\mathbf{S_v}$, $\mathbf{C}_0$ is obtained by renaming with τ_0 a certain clause in $\mathbf{S}$, i.e. $\mathbf{C}_0\tau_0^{-1} \in \mathbf{S}$. Now take $\mathbf{P}_0$ as the path set for the query $\mathbf{C}_0\tau_0^{-1}$, and define the desired substitution $\delta_0 = \tau_0\gamma$. It follows

immediately that $\mathbf{P}_0\delta_0 = \mathbf{P}'_0$, which proves the invariant 1. With $\mathbf{n} = 0$ the invariant 2 holds vacuously.

$\boxed{(\mathbf{n}-1) \to \mathbf{n}}$: Let $\mathbf{n} > 0$ and suppose $\mathbf{I}(\mathbf{n}-1)$ to hold for the derivation $\mathbf{P}'_0, \mathbf{P}'_1, \ldots, \mathbf{P}'_{\mathbf{n}-1}$. $\mathbf{I}(\mathbf{n}-1)$ gives us that there exists a substitution $\delta'_{\mathbf{n}-1}$ such that

$$\mathbf{P}_{\mathbf{n}-1}\delta_{\mathbf{n}-1} = \mathbf{P}'_{\mathbf{n}-1} \quad \text{and} \tag{2}$$

$$\mathbf{C}\rho_{\mathbf{i}}\sigma_1 \cdots \sigma_{\mathbf{n}-1}\delta_{\mathbf{n}-1} = \mathbf{C}_{\mathbf{v}}\gamma \tag{3}$$

under the provisos stated above in the definition of $\mathbf{I}(\mathbf{n})$.

In order to prove $\mathbf{I}(\mathbf{n})$ we make a case analysis wrt. the inference rule applied to $\mathbf{P}'_{\mathbf{n}-1}$:

$\boxed{\text{Reduction step}}$: $\mathbf{P}'_{\mathbf{n}-1}$ contains a path $\mathbf{p}$ of the form $\mathbf{p}' = \langle \ldots, \mathbf{A}, \ldots, \overline{\mathbf{A}} \rangle$ to be deleted, i.e. $\mathbf{P}'_{\mathbf{n}} = \mathbf{P}'_{\mathbf{n}-1} - \{\mathbf{p}'\}$. From the given invariant (2) we learn that $\mathbf{P}_{\mathbf{n}-1}$ in particular contains a path $\mathbf{p}$ with $\mathbf{p}\delta_{\mathbf{n}-1} = \mathbf{p}'$. The path $\mathbf{p}$ is of the form $\mathbf{p} = \langle \ldots, \mathbf{K}, \ldots, \overline{\mathbf{L}} \rangle$, where $\mathbf{K}\delta_{\mathbf{n}-1} = \mathbf{A} = \mathbf{L}\delta_{\mathbf{n}-1}$. In other words, $\delta_{\mathbf{n}-1}$ is a unifier for $\mathbf{K}$ and $\mathbf{L}$. Since $\mathbf{S}_{\mathbf{v}}$ is assumed to be a set of variants completely variable disjoint from $\mathbf{S}$ we can also assume that γ neither acts on the variables from $\mathbf{S}$ nor on the variables occurring in the variants taken from $\mathbf{S}$ to build the derivation $\mathbf{P}_0, \ldots, \mathbf{P}_{\mathbf{n}-1}$. But then $\mathbf{P}_{\mathbf{n}-1}\gamma = \mathbf{P}_{\mathbf{n}-1}$. Application of $\delta_{\mathbf{n}-1}$ yields

$$\mathbf{P}_{\mathbf{n}-1}\gamma\delta_{\mathbf{n}-1} = \mathbf{P}_{\mathbf{n}-1}\delta_{\mathbf{n}-1} \overset{(2)}{=} \mathbf{P}'_{\mathbf{n}-1} = \mathbf{P}'_{\mathbf{n}-1}\gamma\delta_{\mathbf{n}-1} \tag{4}$$

The last identity is trivial ($\mathbf{P}'_{\mathbf{n}-1}$ is ground) and is needed below.

More specifically the following holds

$$\mathbf{K}\gamma\delta_{\mathbf{n}-1} = \mathbf{K}\delta_{\mathbf{n}-1} = \mathbf{A} = \mathbf{L}\delta_{\mathbf{n}-1} = \mathbf{L}\gamma\delta_{\mathbf{n}-1} \tag{5}$$

Next we turn to the clauses used in previous extension steps (cf. given invariant 2). Again, since γ does not act on the variants taken from $\mathbf{S}$ it also holds $\mathbf{C}\rho_{\mathbf{i}}\sigma_1 \cdots \sigma_{\mathbf{n}-1}\gamma\delta_{\mathbf{n}-1} = \mathbf{C}\rho_{\mathbf{i}}\sigma_1 \cdots \sigma_{\mathbf{n}-1}\delta_{\mathbf{n}-1}$. Thus we conclude

$$\mathbf{C}\rho_{\mathbf{i}}\sigma_1 \cdots \sigma_{\mathbf{n}-1}\gamma\delta_{\mathbf{n}-1} = \mathbf{C}\rho_{\mathbf{i}}\sigma_1 \cdots \sigma_{\mathbf{n}-1}\delta_{\mathbf{n}-1} \overset{(3)}{=} \mathbf{C}_{\mathbf{v}}\gamma = \mathbf{C}_{\mathbf{v}}\gamma\delta_{\mathbf{n}-1} \tag{6}$$

The last identity holds trivially because $\mathbf{C}_{\mathbf{v}}\gamma$ is ground.

The equations (4,5,6) tell us that $\gamma\delta_{\mathbf{n}-1}$ is a simultaneous unifier for the respective leftmost and rightmost terms in these equations. Hence there also exists a MGU $\sigma_{\mathbf{n}}$ and a substitution $\delta_{\mathbf{n}}$ such that

$$\sigma_{\mathbf{n}}\delta_{\mathbf{n}} = \gamma\delta_{\mathbf{n}-1} \tag{7}$$

By construction, $\sigma_{\mathbf{n}}$ is a MGU for $\mathbf{K}$ and $\mathbf{L}$. Hence we can apply a reduction step to $\mathbf{p} \in \mathbf{P}_{\mathbf{n}-1}$ with MGU $\sigma_{\mathbf{n}}$ to obtain $\mathbf{P}_{\mathbf{n}} = \mathbf{P}_{\mathbf{n}-1}\sigma_{\mathbf{n}} - \{\mathbf{p}\sigma_{\mathbf{n}}\}$.

Altogether we conclude

$$\begin{aligned}\mathbf{P_n}\delta_\mathbf{n} &= (\mathbf{P_{n-1}}\sigma_\mathbf{n} - \{\mathbf{p}\sigma_\mathbf{n}\})\delta_\mathbf{n} \overset{(7,4)}{=} \mathbf{P'_{n-1}}\gamma\delta_\mathbf{n-1} - \{\mathbf{p'}\gamma\delta_\mathbf{n-1}\} \\ &\overset{(*)}{=} \mathbf{P'_{n-1}} - \{\mathbf{p'}\} = \mathbf{P'_n}\end{aligned}$$

The step $(*)$ is justified by the fact that $\mathbf{P'_{n-1}}$ is ground. Note that this chain just proves the invariant 1.

It remains to prove invariant 2:

$$\mathbf{C}\rho_\mathbf{i}\sigma_1 \cdots \sigma_\mathbf{n-1}\sigma_\mathbf{n}\delta_\mathbf{n} \overset{(7)}{=} \mathbf{C}\rho_\mathbf{i}\sigma_1 \cdots \sigma_\mathbf{n-1}\gamma\delta_\mathbf{n-1} \overset{(6)}{=} \mathbf{C_v}\gamma$$

Since this was to be demonstrated the proof for this case is now done.

Extension step: $\mathbf{P'_{n-1}}$ contains a path $\mathbf{p}$ of the form $\mathbf{p'} = \langle \ldots, \mathbf{A}\rangle$ to be extended with a clause $\overline{\mathbf{A}} \vee \mathbf{R} \in \mathbf{S_v}\gamma$, i.e.

$$\mathbf{P'_n} = \mathbf{P'_{n-1}} - \{\mathbf{p'}\} \cup \{\mathbf{p'} \circ \langle \mathbf{B}\rangle \mid \mathbf{B} \in \mathbf{R}\}$$

From the given invariant (2) we learn that $\mathbf{P_{n-1}}$ in particular contains a path $\mathbf{p}$ with $\mathbf{p}\delta_\mathbf{n-1} = \mathbf{p'}$. The path $\mathbf{p}$ is of the form $\mathbf{p} = \langle \ldots, \mathbf{L}\rangle$, and thus, in particular $\mathbf{A} = \mathbf{L}\delta_\mathbf{n-1}$.

Let $\overline{\mathbf{K_v}} \vee \mathbf{Q_v} \in \mathbf{S_v}$ be the lifted version of $\overline{\mathbf{A}} \vee \mathbf{R}$, i.e.

$$(\overline{\mathbf{K_v}} \vee \mathbf{Q_v})\gamma = \overline{\mathbf{A}} \vee \mathbf{R} \tag{8}$$

Recall from the assumption stated at the beginning, that $\overline{\mathbf{K_v}} \vee \mathbf{Q_v}$ is a new variant by means of some renaming substitution τ, i.e. $\overline{\mathbf{K_v}} \vee \mathbf{Q_v} = (\overline{\mathbf{K}} \vee \mathbf{Q})\tau$ for some $\overline{\mathbf{K}} \vee \mathbf{Q} \in \mathbf{S}$. Let $(\overline{\mathbf{K}} \vee \mathbf{Q})\rho_\mathbf{n}$ be a new respective variant. We will show how to carry out an extension step with that variant.

From the last equation and (8) it follows

$$\begin{aligned}\overline{\mathbf{A}} \vee \mathbf{R} &= (\overline{\mathbf{K_v}} \vee \mathbf{Q_v})\gamma = (\overline{\mathbf{K}} \vee \mathbf{Q})\tau\gamma = ((\overline{\mathbf{K}} \vee \mathbf{Q})\rho_\mathbf{n})\rho^{-1}_\mathbf{n-1}\tau\gamma \\ &= ((\overline{\mathbf{K}} \vee \mathbf{Q})\rho_\mathbf{n})\rho^{-1}_\mathbf{n-1}\tau\gamma\delta_\mathbf{n-1}\end{aligned} \tag{9}$$

The last identity is justified by the fact that $\delta_\mathbf{n-1}$ is applied to a ground clause, and hence does not alter the clause.

The renaming substitution $\rho_\mathbf{n}$ introduces new variables; hence $\rho^{-1}_\mathbf{n}$ does not affect $\mathbf{P_{n-1}}$. Also, τ does not affect $\mathbf{P_{n-1}}$, because $\mathbf{P_{n-1}}$ is built from new variants (possibly except the query clause which has to be variable disjoint to all other clauses in advance). Together we obtain that $\mathbf{P_{n-1}} = \mathbf{P_{n-1}}\rho^{-1}_\mathbf{n}\tau$. Further, as in the case of reduction step, we can also assume that γ does neither act on the variables from $\mathbf{S}$ nor on the variables occurring in the variants taken from $\mathbf{S}$ to build the derivation $\mathbf{P}_0, \ldots, \mathbf{P_{n-1}}$. But then $\mathbf{P_{n-1}}\gamma = \mathbf{P_{n-1}}$. Putting things together we obtain:

$$\mathbf{P_{n-1}}\rho^{-1}_\mathbf{n}\tau\gamma\delta_\mathbf{n-1} = \mathbf{P_{n-1}}\delta_\mathbf{n-1} \overset{(2)}{=} \mathbf{P'_{n-1}} = \mathbf{P'_{n-1}}\rho^{-1}_\mathbf{n}\tau\gamma\delta_\mathbf{n-1} \tag{10}$$

The last identity is trivial ($\mathbf{P'_{n-1}}$ is ground) and is needed below.

Recall from above that we extended $\mathbf{P'_{n-1}}$ at a path $\mathbf{p'}$ with leaf $\mathbf{A}$ to obtain $\mathbf{P'_n}$. From $\mathbf{A} = \mathbf{L}\delta_{\mathbf{n}-1}$, as given, we can now conclude with (10) even $\mathbf{L}\rho_{\mathbf{n}}^{-1}\tau\gamma\delta_{\mathbf{n}-1} = \mathbf{A}$. But then we have

$$\mathbf{L}\rho_{\mathbf{n}}^{-1}\tau\gamma\delta_{\mathbf{n}-1} = \mathbf{A} \overset{(9)}{=} (\mathbf{K}\rho_{\mathbf{n}})\rho_{\mathbf{n}}^{-1}\tau\gamma\delta_{\mathbf{n}-1} \tag{11}$$

Next we turn to the clauses used in previous extension steps (cf. given invariant 2). By the same line of reasoning as for $\mathbf{P_{n-1}}$ above we can assume that $\rho_{\mathbf{n}}^{-1}\tau$ does not affect a clause $\mathbf{C_v}$ whose ground instance $\mathbf{C_v}\gamma$ is used in the ground derivation. Thus we conclude

$$\mathbf{C_v}\gamma = \mathbf{C_v}\rho_{\mathbf{n}}^{-1}\tau\gamma = \mathbf{C_v}\rho_{\mathbf{n}}^{-1}\tau\gamma\delta_{\mathbf{n}-1} \tag{12}$$

The last identity holds because $\delta_{\mathbf{n}-1}$ is applied to a ground clause.

Since the derivation from $\mathbf{S}$ uses new variants, and the MGUs used there can be supposed to introduce no new variables we have

$$\mathbf{C}\rho_{\mathbf{i}}\sigma_1 \cdots \sigma_{\mathbf{n}-1} = \mathbf{C}\rho_{\mathbf{i}}\sigma_1 \cdots \sigma_{\mathbf{n}-1}\rho_{\mathbf{n}}^{-1}\tau$$

Again, since γ does not act on the variants taken from $\mathbf{S}$ $\mathbf{C}\rho_{\mathbf{i}}\sigma_1 \cdots \sigma_{\mathbf{n}-1}\gamma = \mathbf{C}\rho_{\mathbf{i}}\sigma_1 \cdots \sigma_{\mathbf{n}-1}$ also holds. Using these identities and applying $\delta_{\mathbf{n}-1}$ we conclude

$$\begin{aligned}\mathbf{C}\rho_{\mathbf{i}}\sigma_1 \cdots \sigma_{\mathbf{n}-1}\rho_{\mathbf{n}}^{-1}\tau\gamma\delta_{\mathbf{n}-1} &= \mathbf{C}\rho_{\mathbf{i}}\sigma_1 \cdots \sigma_{\mathbf{n}-1}\delta_{\mathbf{n}-1} \\ &\overset{(3)}{=} \mathbf{C_v}\gamma \overset{(12)}{=} \mathbf{C_v}\rho_{\mathbf{n}}^{-1}\tau\gamma\delta_{\mathbf{n}-1}\end{aligned}$$

The equations (10, 11, 13) tell us that $\rho_{\mathbf{n}}^{-1}\tau\gamma\delta_{\mathbf{n}-1}$ is a simultaneous unifier for the respective leftmost and rightmost terms in these equations. Hence there also exists a MGU $\sigma_{\mathbf{n}}$ and a substitution $\delta_{\mathbf{n}}$ such that

$$\sigma_{\mathbf{n}}\delta_{\mathbf{n}} = \rho_{\mathbf{n}}^{-1}\tau\gamma\delta_{\mathbf{n}-1} \tag{13}$$

By (11) and (13), $\sigma_{\mathbf{n}}$ is a MGU for $\mathbf{L}$ and $\mathbf{K}\rho_{\mathbf{n}}$. Hence we can apply an extension step to $\mathbf{p} \in \mathbf{P_{n-1}}$ with the variant clause $(\overline{\mathbf{K}} \vee \mathbf{Q})\rho_{\mathbf{n}}$ and MGU $\sigma_{\mathbf{n}}$ to obtain

$$\mathbf{P_n} = (\mathbf{P_{n-1}} - \{\mathbf{p}\} \cup \{\mathbf{p} \circ \langle \mathbf{B} \rangle \mid \mathbf{B} \in \mathbf{Q}\rho_{\mathbf{n}}\})\sigma_{\mathbf{n}}$$

Altogether we conclude

$$\begin{aligned}\mathbf{P_n}\delta_{\mathbf{n}} &= (\mathbf{P_{n-1}} - \{\mathbf{p}\} \cup \{\mathbf{p} \circ \langle \mathbf{B} \rangle \mid \mathbf{B} \in \mathbf{Q}\rho_{\mathbf{n}}\})\sigma_{\mathbf{n}}\delta_{\mathbf{n}} \\ &= (\mathbf{P_{n-1}} - \{\mathbf{p}\})\sigma_{\mathbf{n}}\delta_{\mathbf{n}} \cup \{\mathbf{p} \circ \langle \mathbf{B} \rangle \mid \mathbf{B} \in \mathbf{Q}\rho_{\mathbf{n}}\}\sigma_{\mathbf{n}}\delta_{\mathbf{n}} \\ &\overset{(13,10)}{=} \mathbf{P'_{n-1}} - \{\mathbf{p'}\} \cup \{\mathbf{p} \circ \langle \mathbf{B} \rangle \mid \mathbf{B} \in \mathbf{Q}\rho_{\mathbf{n}}\}\sigma_{\mathbf{n}}\delta_{\mathbf{n}} \\ &\overset{(13,10,9)}{=} \mathbf{P'_{n-1}} - \{\mathbf{p'}\} \cup \{\mathbf{p'} \circ \langle \mathbf{B} \rangle \mid \mathbf{B} \in \mathbf{R}\} \\ &= \mathbf{P'_n}\end{aligned}$$

Note that this chain of reasoning just proves the invariant 1.

We still have to prove invariant 2: First, it has to be proved for all clauses used up to, but not including this extension step:

$$\mathbf{C}\rho_{\mathbf{i}}\sigma_1 \cdots \sigma_{\mathbf{n}-1}\sigma_{\mathbf{n}}\delta_{\mathbf{n}} \overset{(13)}{=} \mathbf{C}\rho_{\mathbf{i}}\sigma_1 \cdots \sigma_{\mathbf{n}-1}\rho_{\mathbf{n}}^{-1}\tau\gamma\delta_{\mathbf{n}-1} \overset{(13)}{=} \mathbf{C}_{\mathbf{v}}\gamma$$

Second, invariant 2 has to be proved for the clause used in this extension step. For this note that $\rho_{\mathbf{n}}$ renames $\overline{\mathbf{K}} \vee \mathbf{Q}$ to a new variant. Hence $((\overline{\mathbf{K}} \vee \mathbf{Q})\rho_{\mathbf{n}})\sigma_1 \cdots \sigma_{\mathbf{n}-1} = (\overline{\mathbf{K}} \vee \mathbf{Q})\rho_{\mathbf{n}}$. From this it follows

$$\begin{aligned} & ((\overline{\mathbf{K}} \vee \mathbf{Q})\rho_{\mathbf{n}})\sigma_1 \cdots \sigma_{\mathbf{n}-1}\sigma_{\mathbf{n}}\delta_{\mathbf{n}} \\ = \; & (\overline{\mathbf{K}} \vee \mathbf{Q})\rho_{\mathbf{n}})\sigma_{\mathbf{n}}\delta_{\mathbf{n}} \\ \overset{(13)}{=} \; & ((\overline{\mathbf{K}} \vee \mathbf{Q})\rho_{\mathbf{n}})\rho_{\mathbf{n}}^{-1}\tau\gamma\delta_{\mathbf{n}-1} \\ \overset{(9)}{=} \; & (\overline{\mathbf{K}_{\mathbf{v}}} \vee \mathbf{Q}_{\mathbf{v}})\gamma \end{aligned}$$

Now invariants 1 and 2 have been shown, which concludes the proof of the extension step.

Restart step: Since in a restart step no substitution is applied we can take $\delta_{\mathbf{n}} := \delta_{\mathbf{n}-1}$. Together with the induction hypothesis the result follows trivially. Q.E.D.

THEOREM 2.3

(Answer completeness of restart ME) *If $\mathbf{Q}\theta_1 \vee \ldots \vee \mathbf{Q}\theta_{\mathbf{l}}$ is a correct answer for a program $\mathbf{P}$, then there exists a strict restart ME refutation from $\mathbf{S}$ with computed answer $\mathbf{Q}\sigma_1 \vee \ldots \vee \mathbf{Q}\sigma_{\mathbf{m}}$ such that $\mathbf{Q}\sigma_1 \vee \ldots \vee \mathbf{Q}\sigma_{\mathbf{m}}$ entails $\mathbf{Q}\theta_1 \vee \ldots \vee \mathbf{Q}\theta_{\mathbf{l}}$, i.e.*

$$\exists\delta\, \forall \mathbf{i} \in \{1, \ldots, \mathbf{m}\}\, \exists \mathbf{j} \in \{1, \ldots, \mathbf{l}\}\; \mathbf{Q}\sigma_{\mathbf{i}}\delta = \mathbf{Q}\theta_{\mathbf{j}}.$$

Informally, the theorem states that for every given correct answer we can find a computed answer which can be instantiated by means of a *single* substitution δ to a subclause of the given answer (and hence implies it). Unfortunately we can *not* obtain a result stating that the computed answer contains less (or equal number of) literals than the given answer.

This behaviour sometimes results in confusing answers. For instance, let the program be

$$\begin{aligned} \mathcal{P}\colon\; & \mathbf{P}(\mathbf{X}) \leftarrow \mathbf{Q}(\mathbf{X}), \mathbf{Q}(\mathbf{X}) \\ & \mathbf{Q}(\mathbf{A}), \mathbf{Q}(\mathbf{B}) \leftarrow \end{aligned}$$

The refutation in Figure 2 computes the answer $\mathbf{P}(\mathbf{a}) \vee \mathbf{P}(\mathbf{b}) \vee \mathbf{P}(\mathbf{b})$. Although $\mathbf{P}(\mathbf{a}) \vee \mathbf{P}(\mathbf{b})$ is a correct answer, restart ME will not compute it. The reason for this is that two identical instances $\leftarrow \mathbf{P}(\mathbf{b})$ of the query have to be used. In Section 3 below we will describe a calculus variant which is more optimal wrt. the length of the disjuncts. – The proof follows now.

PROOF. Now for the proof of Theorem 2.3, proper:

Given the correct answer $\mathbf{Q}\theta_1 \vee \ldots \vee \mathbf{Q}\theta_\mathbf{l}$ we know by definition $\mathbf{P} \models \forall(\mathbf{Q}\theta_1 \vee \ldots \vee \mathbf{Q}\theta_\mathbf{l})$. Hence we conclude that $\mathbf{P} \cup \{\neg\forall(\mathbf{Q}\theta_1 \vee \ldots \vee \mathbf{Q}\theta_\mathbf{l})\}$ is unsatisfiable. By transforming this into CNF we get the unsatisfiable set of clauses $\mathbf{S'} = \mathbf{P} \cup \{\neg\mathbf{Q}\theta_1\tau_1, \ldots, \neg\mathbf{Q}\theta_\mathbf{l}\tau_\mathbf{l}\}$ where each $\tau_\mathbf{i}(1 \leq \mathbf{i} \leq \mathbf{l})$ substitutes new Skolem constants for the free variables of $\mathbf{Q}\theta_\mathbf{i}$.

With the abbreviation $\theta'_\mathbf{i} = \theta_\mathbf{i}\tau_\mathbf{i}$ for all $1 \leq \mathbf{i} \leq \mathbf{l}$, we get an unsatisfiable set of clauses

$$\mathbf{S'} = \mathbf{P} \cup \{\neg\mathbf{Q}\theta'_1, \ldots, \neg\mathbf{Q}\theta'_\mathbf{l}\}$$

By the Herbrand-Löwenheim-Skolem Theorem there exists an unsatisfiable ground clause set

$$\mathbf{S''} = \mathbf{P'} \cup \{\neg\mathbf{Q}\theta'_1, \ldots, \neg\mathbf{Q}\theta'_\mathbf{l}\}$$

where $\mathbf{P'}$ is a finite set of ground instances of clauses from $\mathbf{P}$. From $\mathbf{S''}$ we select a *minimal* unsatisfiable subset

$$\mathbf{S'''} = \mathbf{P''} \cup \{\neg\mathbf{Q}\theta'_1, \ldots, \neg\mathbf{Q}\theta_\mathbf{r}\}$$

where $\mathbf{P''} \subseteq \mathbf{P'}$, and (without loss of generality) $\mathbf{r} \leq \mathbf{l}$. From ground completeness of restart ME (Theorem 1.3) we learn that there exists a restart ME refutation of the goal normal form of $\mathbf{S'''}$, i.e. there exists a restart ME refutation $\mathbf{D'}$ of

$$\mathbf{S''''} = \mathbf{P''_{goal}} \cup \{\mathbf{goal} \leftarrow \mathbf{Q}\theta'_1, \ldots, \mathbf{goal} \leftarrow \mathbf{Q}\theta'_\mathbf{r}\} \cup \{\leftarrow \mathbf{goal}\}.$$

Here, a clause set $\mathbf{P_{goal}}$ is obtained from a clause set $\mathbf{P}$ by replacing every purely negative clause $\neg\mathbf{B}_1 \vee \cdots \vee \neg\mathbf{B_n}$ by $\mathbf{goal} \vee \neg\mathbf{B}_1 \vee \cdots \vee \neg\mathbf{B_n}$.

The minimality condition ensures that each of clauses $\{\mathbf{goal} \leftarrow \mathbf{Q}\theta'_1, \ldots, \mathbf{goal} \leftarrow \mathbf{Q}\theta'_\mathbf{r}\}$ in $\mathbf{S''''}$ is used at least once for an extension step. Let $\mathbf{m} \geq \mathbf{r}$ be the total number of extension steps carried out with clauses from that set.

$\mathbf{D'}$ is a refutation, i.e. a derivation of the empty path set $\mathbf{P'_n} = \{\}$. By the lifting theorem there exists a restart ME derivation $\mathbf{D}$ of some more general path set $\mathbf{P_n}$ from $\mathbf{P_{goal}} \cup \{\mathbf{goal} \leftarrow \mathbf{Q}\} \cup \{\leftarrow \mathbf{goal}\}$. Since only the empty path set is more general than itself we conclude that $\mathbf{P_n}$ is also a refutation. This proves refutational completeness.

Next we turn to answer completeness, proper. The second part of the lifting theorem gives us that for any extension step in $\mathbf{D'}$ with clause $\mathbf{goal} \leftarrow \mathbf{Q}\theta'_\mathbf{f(i)}$ (where $\mathbf{f}$ is a surjection from $\{1, \ldots, \mathbf{m}\}$ onto $\{1, \ldots, \mathbf{r}\}$) there is exactly one extension step in $\mathbf{D}$ with the clause $\{\mathbf{goal} \leftarrow \mathbf{Q}\rho_\mathbf{i}\}$ ($\mathbf{i} \in \{1, \ldots, \mathbf{m}\}$), where $\rho_\mathbf{i}$ is the renaming substitution of that step. Furthermore (by the lifting theorem) there is a substitution δ' such that

$$\mathbf{goal} \leftarrow \mathbf{Q}\rho_\mathbf{i}\sigma\delta' \quad = \quad \mathbf{goal} \leftarrow \mathbf{Q}\theta'_\mathbf{f(i)} \tag{14}$$

This, however, is equivalent to the condition that every element in the disjunction

$$\mathbf{Ans} = \mathbf{Q}\sigma_1 \vee \ldots \vee \mathbf{Q}\sigma_\mathbf{r} \vee \mathbf{Q}\sigma_{\mathbf{r}+1} \vee \ldots \vee \mathbf{Q}\sigma_\mathbf{m}$$

where $\sigma_{\mathbf{i}} = \rho_{\mathbf{i}}\sigma|_{\mathbf{dom}(\rho_{\mathbf{i}})}$ (for $\mathbf{i} = 1 \ldots \mathbf{m}$), is mapped by application of δ' into some element of $\mathbf{Q}\theta'_1 \vee \cdots \mathbf{Q}\theta'_{\mathbf{r}}$. Note here that $\mathbf{Ans}$ is nothing but the computed answer substitution.

Recall that $\mathbf{Q}\theta'_{\mathbf{k}} = \mathbf{Q}\theta_{\mathbf{k}}\tau_{\mathbf{k}}$ (for $\mathbf{k} = 1, \ldots, \mathbf{r}$), and hence with (14) we have

$$\mathbf{Q}\sigma_{\mathbf{i}}\delta' = \mathbf{Q}\theta_{\mathbf{f(i)}}\tau_{\mathbf{f(i)}} \tag{15}$$

However, in order to prove the theorem we have to find a substitution δ such that $\mathbf{Q}\sigma_{\mathbf{i}}\delta = \mathbf{Q}\theta_{\mathbf{f(i)}}$. In order to define δ recall that $\tau_{\mathbf{k}}$ is a Skolemizing substitution and hence can be written as

$$\tau_{\mathbf{k}} = \{\mathbf{x_o} \leftarrow \mathbf{a_o} \mid \mathbf{o} \in \mathbf{O}\}$$

for some finite index set $\mathbf{O}$ and new constants $\mathbf{a_o}$. In this case we can treat in the refutation $\mathbf{D}$ the $\mathbf{a_o}$s as new variables and define the substitutions

$$\tau_{\mathbf{k}}^{-1} = \{\mathbf{a_o} \leftarrow \mathbf{x_o} \mid \mathbf{x_o} \leftarrow \mathbf{a_o} \in \tau_{\mathbf{k}}\}$$

Every $\tau_{\mathbf{k}}$ introduces new Skolem constants. Hence the domains of the $\tau_{\mathbf{k}}^{-1}$s are pairwise disjoint. But then with defining $\tau^{-1} := \tau_1^{-1} \cdots \tau_{\mathbf{r}}^{-1}$ we get

$$\tau^{-1}|_{\mathbf{dom}(\tau_{\mathbf{k}}^{-1})} = \tau_{\mathbf{k}}^{-1} \tag{16}$$

Next define

$$\delta := \delta'\tau^{-1}$$

This is the desired substitution since

$$\mathbf{Q}\sigma_{\mathbf{i}}\delta = \mathbf{Q}\sigma_{\mathbf{i}}\delta'\tau^{-1} \overset{(15)}{=} (\mathbf{Q}\theta_{\mathbf{f(i)}}\tau_{\mathbf{f(i)}})\tau^{-1} \overset{(16)}{=} (\mathbf{Q}\theta_{\mathbf{f(i)}}\tau_{\mathbf{f(i)}})\tau_{\mathbf{f(i)}}^{-1} = \mathbf{Q}\theta_{\mathbf{f(i)}}$$

Finally, with the fact that $\mathbf{f}$ is a suitable surjection the theorem is proved. Q.E.D.

3 Definite Answers and Regularity

From theorem proving with ME we know that the regularity check is an important means for improving efficiency. Regularity for ordinary ME means 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).

Unfortunately, regularity is *not* compatible to restart ME. In this section we will present a variant of restart ME, the *ancestry restart* variant, which allows for extended regularity checks. This variant is motivated by Loveland's UnH-Prolog [Loveland and Reed, 1992].

As an interesting side effect it turns out that this variant offers considerable benefits with respect to logic programming: occasionally one is interested in the question

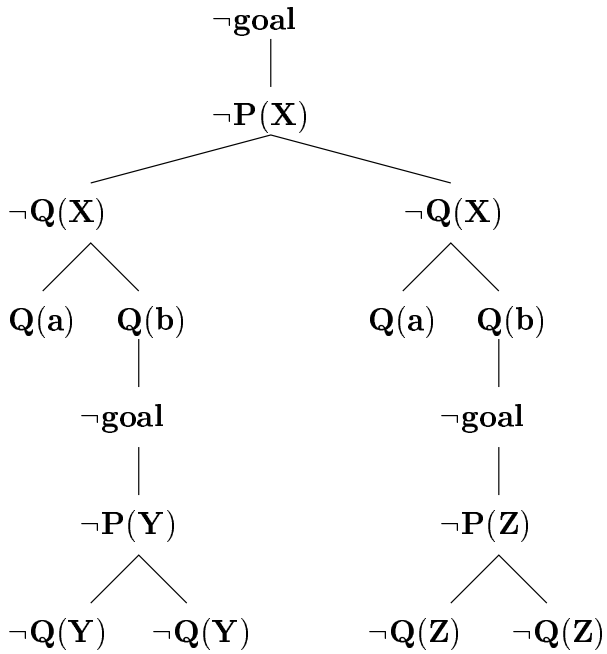

Figure 2: A refutation of $\mathcal{P} \cup \{\leftarrow \mathbf{goal}, \mathbf{goal} \leftarrow \mathbf{P}(\mathbf{X})\}$ depicted as a tree; the computed answer is $\mathbf{P}(\mathbf{a}) \vee \mathbf{P}(\mathbf{b}) \vee \mathbf{P}(\mathbf{b})$ where the substitution $\{\mathbf{X} \leftarrow \mathbf{a}, \mathbf{Y} \leftarrow \mathbf{b}, \mathbf{Z} \leftarrow \mathbf{b}\}$ is applied.

whether a given program with query admits a *definite* answer, i.e. an answer which is a single conjunction of atoms, but not a disjunction. Of course, in general, a non-definite program does not always admit a definite answer, but some programs do. It is the latter class of problems we are interested in now.

Consider the program $\mathcal{P}_{\mathbf{Def}} = \{\mathbf{P}(\mathbf{X}, \mathbf{a}) \vee \mathbf{P}(\mathbf{b}, \mathbf{Y}) \leftarrow\}$ and the query $\leftarrow \mathbf{P}(\mathbf{X}, \mathbf{Y})$. Note that, among others, $\mathbf{P}(\mathbf{b}, \mathbf{a})$ and $\mathbf{P}(\mathbf{X}, \mathbf{a}) \vee \mathbf{P}(\mathbf{b}, \mathbf{Y})$ are correct answers. Now, by Theorem 2.3 strict restart ME is answer complete; hence we can find a refutation yielding a computed answer which entails $\mathbf{P}(\mathbf{b}, \mathbf{a})$ (Figure 3, left side). As noted after the statement of Theorem 2.3 restart ME will not always compute *minimal* (wrt. length) answers. It is easily verified that in this example the shortest computed answer is $\mathbf{P}(\mathbf{X}, \mathbf{a}) \vee \mathbf{P}(\mathbf{b}, \mathbf{Y})$, which can be factored to $\mathbf{P}(\mathbf{b}, \mathbf{a})$. This is due to repeated use of the query in the refutation.[1]

The key idea to the direct computation of definite answers is to restrict the use of the query to one single application in the refutation, namely at its top. Then, by definition, definite answers are obtained. However, such a restriction is incomplete. But if restart

[1] However, in this example we could find a *non*-strict restart ME using the query only once. Hence, one might object that this example is vacuous. This, however, misses the point because there exists a more complicated example where this argumentation would not work.

ME is modified in such a way that *every* negative literal along a branch, not only the topmost literal, may be used for the restart step then completeness is recovered. This follows from a more general result which states that we can restrict our calculus to *globally regular* refutations (i.e. no literal except the literal used for the restart occurs more than once along a branch). Let us now introduce all this more formally.

DEFINITION 3.1
(Ancestry Restart Model Elimination) The calculus *ancestry restart ME* is the same as *strict restart ME* (Definition 1.2), except that the inference rule *restart* is modified by replacing the condition 3. by the new condition 3'.:

3'. $\mathbf{L}$ is a negative literal occurring in $\mathbf{p}$. In this context $\mathbf{L}$ is also called the *restart literal*.

The modified rule is called *ancestry restart*. (END DEFINITION)

The term "ancestry" in the definition is explained by the use of ancestor literals for restart steps. Note that any reduction from a *positive* leaf literal to a negative ancestor literal can be simulated in ancestry restart ME by a restart step followed by a strict reduction step. Thus, non-strictness is "built into " ancestry restart ME.

Note that the ancestry restart rule includes the restart rule since the first literal can be used for the restart as well.

Figure 3 (right side) shows an example ancestry restart ME refutation of $\mathcal{P}_{\mathbf{Def}}$ with query $\leftarrow \mathbf{P}(\mathbf{X}, \mathbf{Y})$. Note that no new copy of the query clause is used, but instead the present instance is copied by the ancestry restart rule, and then the resulting path is closed by a reduction step. This example also demonstrates that — unlike in strict restart ME — it makes sense to apply a reduction step to a restart literal. This motivated us to change the definitions in [Baumgartner and Furbach, 1994a] in now letting the restart step be an explicit inference rule.

Clearly, in terms of a proof procedure the ancestry restart rule induces a larger local search space than the restart rule. On the other hand, refutations may become much shorter. In order to see this think of a derivation containing a path $\neg\mathbf{B}_1 \cdots \neg\mathbf{B}_\mathbf{n}\mathbf{A}$. It might be necessary in restart ME to repeat all the $\neg\mathbf{B}_\mathbf{i}$'s in order to find a refutation. This derivation of $\neg\mathbf{B}_1 \cdots \neg\mathbf{B}_\mathbf{n}\mathbf{A}\neg\mathbf{B}_1 \cdots \neg\mathbf{B}_\mathbf{n}$ can be abbreviated in ancestry restart ME to $\neg\mathbf{B}_1 \cdots \neg\mathbf{B}_\mathbf{n}\mathbf{A}\neg\mathbf{B}_\mathbf{n}$ by guessing the right $\neg\mathbf{B}_\mathbf{n}$ for the restart. Indeed, this is the rationale for our proof procedure to search the restart literals from the leaf towards the top. As a further benefit of this search order note that a definite answer will be enumerated *before* a non-definite answer, provided it allows for a shorter proof.

Now we proceed to an appropriate completeness result wrt. definite answers. As mentioned above, this result shall be a consequence of a more general result concerning a regularity restriction. Let us define this notion precisely:

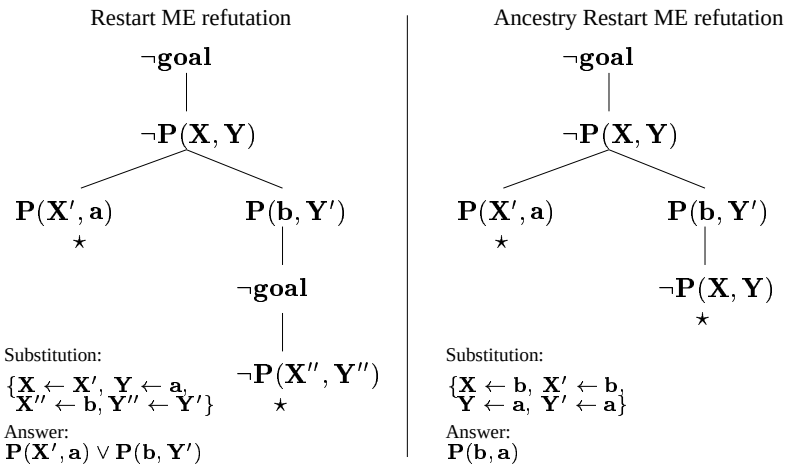

Figure 3: A restart ME (left side) and an Ancestry restart ME (right side) refutation of $\{\mathbf{P}(\mathbf{X},\mathbf{a}) \vee \mathbf{P}(\mathbf{b},\mathbf{Y}) \leftarrow,\ \leftarrow \mathbf{goal},\ \mathbf{goal} \leftarrow \mathbf{P}(\mathbf{X},\mathbf{Y})\}$ depicted as trees

DEFINITION 3.2
(Regularity) Let $\mathbf{p}$ be path written as follows (the $\mathbf{A}$s and $\mathbf{B}$s are atoms):

$$\mathbf{p} = \neg\mathbf{B}_1^1 \cdots \neg\mathbf{B}_{\mathbf{k}_1}^1 \mathbf{A}^1 \neg\mathbf{B}_1^2 \cdots \neg\mathbf{B}_{\mathbf{k}_2}^2 \mathbf{A}^2 \cdots \mathbf{A}^{\mathbf{n}-1} \neg\mathbf{B}_1^{\mathbf{n}} \cdots \neg\mathbf{B}_{\mathbf{k}_\mathbf{n}}^{\mathbf{n}}$$

Then $\mathbf{p}$ is called *blockwise regular* iff

1. $\mathbf{A^i} \neq \mathbf{A^j}$ for $1 \leq \mathbf{i},\mathbf{j} \leq \mathbf{n}-1$, $\mathbf{i} \neq \mathbf{j}$ *(Regularity wrt. positive literals)* and
2. $\mathbf{B_i^l} \neq \mathbf{B_j^l}$ for $1 \leq \mathbf{l} \leq \mathbf{n}$, $1 \leq \mathbf{i},\mathbf{j} \leq \mathbf{k_l}$, $\mathbf{i} \neq \mathbf{j}$ *(Regularity inside blocks).*

If additionally it holds that

3. $\mathbf{B_i^l} \neq \mathbf{B_j^m}$ for $1 \leq \mathbf{l} < \mathbf{m} \leq \mathbf{n}$, $1 \leq \mathbf{i} \leq \mathbf{k_l}$, $2 \leq \mathbf{j} \leq \mathbf{k_m}$ *(Global negative regularity)*

then $\mathbf{p}$ is called *globally regular*. A path set is called *(blockwise, globally) regular* iff every path in it is (blockwise, globally) regular. Similarly, a derivation is called *(blockwise, globally) regular* iff every of its path sets is (blockwise, globally) regular. (END DEFINITION)

Condition 1 states that all positive literals along a path are pairwise different, and condition 2 states that negative literals inside blocks are pairwise different, where by a block we mean a smallest subpath delimited by positive literals or the ends of the path. Condition 3 means that a negative literal may be equal to one of its ancestors only if it follows a positive literal, i.e if it is used as a restart literal. Thus we have a global regularity condition, except for restart literals. In all example refutations given so far,

all branches are blockwise regular. However, the refutation in Figure 1 (right side) is not globally regular, as can be seen by the two occurrences of $\neg \mathbf{Q}$ in the rightmost path. From this example we learn that restart ME is incompatible with the global regularity restriction. However we have:

THEOREM 3.3
(Completeness of Ancestry Restart Model Elimination) *Let* $\mathbf{f}$ *be a head selection function and* $\mathbf{S}$ *be an unsatisfiable clause set in* $\mathbf{goal}$*-normal form. Then there exists a globally regular ancestry restart ME refutation of* $\mathbf{S}$ *starting with* $\leftarrow \mathbf{goal}$ *and selection function* $\mathbf{f}$*.*

PROOF. The proof builds on the above answer completeness Theorem 2.3, which in turn relies on the ground completeness proof in [Baumgartner and Furbach, 1994a]. Although not stated here explicitly, that proof in [Baumgartner and Furbach, 1994a] shows the ground completeness for the blockwise regular strict restart ME with selection function. We can thus rely on this result, and show how to transform a given blockwise regular refutation in restart ME on the ground level to a globally regular refutation in ancestry restart ME (also on the ground level).

This suffices to prove the theorem on the first-order level. Reason: Suppose some unsatisfiable clause set in goal normal form is given. From the cited results we know that there exists a blockwise regular refutation, say $\mathbf{D}$, in restart ME which is lifted from a refutation $\mathbf{D^g}$ on the ground level. By the supposed transformation then there exists a globally regular refutation $\mathbf{D^g_{Anc}}$ in ancestry restart ME. It is straightforward to adapt the lifting theorem (Theorem 2.2) to take into account the ancestry restart step. Now suppose, to the contrary of the theorem, that the lifted version $\mathbf{D_{Anc}}$ is not globally regular. Then some path $\mathbf{p}$ along the refutation $\mathbf{D_{Anc}}$ is not globally regular. This means that one of the inequality constraints stated in the definition of globally regular is violated. Thus $\mathbf{p}$ contains two occurrences of a literal $\mathbf{B}$ which violate one of these constraints. However, with the path $\mathbf{p}$ being a lifted version of a path $\mathbf{p^g}$ in $\mathbf{D^g_{Anc}}$ the inequality constraint must be violated in $\mathbf{p^g}$ as well. This plainly contradicts the assumption that $\mathbf{D^g_{Anc}}$ is globally regular. Hence $\mathbf{D_{Anc}}$ is also globally regular.

It remains to prove the desired transformation. For this let $\mathbf{D^g}$ again be the given blockwise regular refutation in restart ME from above. By the completeness result in [Baumgartner and Furbach, 1994a] we can further assume that $\mathbf{D^g}$ is a strict refutation (no reduction steps from positive literals). Since by definition an ancestry restart step includes the possibility of a restart step, $\mathbf{D^g}$ is also a blockwise regular ancestry restart ME refutation. Let $\mathbf{n}$ be the number of violations of the global negative regularity constraint (Condition 3 in Definition 3.2). Either $\mathbf{n} = 0$ and we are done, or else $\mathbf{D^g}$ can be transformed to contain strictly less than $\mathbf{n}$ such violations, without sacrificing the blockwise regularity constraints. Repeated application will eventually result in the desired refutation.

The transformation step can most easily be expressed using the tree view of ME.

Figure 4 (left) shows the general situation: assume $\mathbf{D^g}$ contains a path

$$\mathbf{p} = \neg\mathbf{goal} \cdots \mathbf{A^k} \neg\mathbf{goal} \cdots \neg\mathbf{B} \cdots \mathbf{A^l} \neg\mathbf{goal} \cdots \neg\mathbf{B}$$

where $\mathbf{A^l}$ is the bottommost (rightmost) positive literal dominating $\neg\mathbf{B}$. Further, as indicated by the two occurrences of $\neg\mathbf{B}$ in $\mathbf{p}$, the global negative regularity constraint is violated for $\mathbf{p}$. Let $\mathbf{D'}$ be the tableau corresponding to the refutation of the path $\mathbf{p}$.

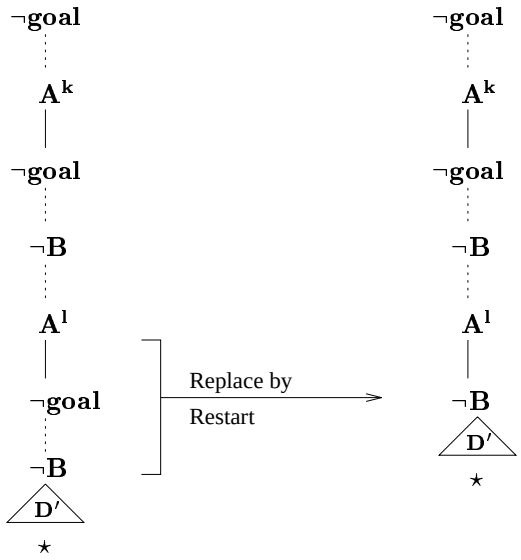

Figure 4: Transformation step removing a violation of a global negative regularity constraint.

Now delete the subtree between $\mathbf{A^l}$ and $\neg\mathbf{B}$ below (bounds excluded from deletion). We arrive at a new tableau with path

$$\mathbf{p'} = \neg\mathbf{goal} \cdots \mathbf{A^k} \neg\mathbf{goal} \cdots \neg\mathbf{B} \cdots \mathbf{A^l} \neg\mathbf{B}$$

Next append $\mathbf{D'}$ to $\mathbf{p'}$. Since *only negative* literals have been deleted, and we assume that $\mathbf{D^g}$ is a *strict* restart ME refutation, in $\mathbf{D'}$ no reduction steps from positive literals to negative literals have occurred. Thus $\mathbf{D'}$ is also a closed tableau below $\mathbf{p'}$ (right side in Figure 4).

It is clear from the definition that this new tableau can be constructed by means of an ancestry restart step, using $\neg\mathbf{B}$ as restart literal. Furthermore, (at least) one violation of the global negative regularity constraint has been removed, and since we only have deleted some literals from $\mathbf{p}$ the blockwise regularity constraints will not be violated. Thus we have found a suitable transformation step. Q.E.D.

We can use this result to obtain the desired completeness result for definite answers.

THEOREM 3.4

(Answer completeness of ancestry restart ME) *Ancestry restart ME is answer complete in the sense of Theorem 2.3. In particular, if* $\mathbf{Q}\theta$ *is a correct definite answer for a program* $\mathbf{P}$*, then there exists an ancestry restart ME refutation from* $\mathbf{P}$ *with computed answer* $\mathbf{Q}\sigma$ *such that* $\mathbf{Q}\sigma\delta = \mathbf{Q}\theta$*, for some substitution* δ*. Furthermore, the input clause* $\mathbf{goal} \leftarrow \mathbf{Q}$ *is used exactly once, namely at the first extension step of* $\leftarrow \mathbf{goal}$*.*

This last theorem enables us to enumerate definite answers *only*, by simply restricting the use of $\mathbf{goal} \leftarrow \mathbf{Q}$ to one extension step at the beginning. So we have the desirable properties of loop checking by regularity and the computation of definite answers.

PROOF. The answer completeness follows directly from the fact that the ancestry variant still permits restart steps with the goal, i.e. it allows for additional derivations. The proof of the last part is given by a careful analysis of the proof of Theorem 2.3. Recall from that proof that $\mathbf{Q}\theta$ is a correct answer for $\mathbf{P}$ implies that $\mathbf{S}' = \mathbf{P} \cup \{\neg\mathbf{Q}\theta\tau\}$ is unsatisfiable, where τ is some substitution introducing new Skolem constants for the variables of $\mathbf{Q}\theta$. Then we considered a set $\mathbf{S}''''$ of ground instances of the goal normal form of $\mathbf{S}'$. It is important to recognise that the goal normal form $\mathbf{S}''''$ contains exactly one instance of the query, namely $\mathbf{goal} \leftarrow \mathbf{Q}\theta\tau$. Since we assume that $\mathbf{P}$ is a *program*, and hence consistent, it follows from the proof of Theorem 1.3 that there exists a strict restart ME refutation of $\mathbf{S}''''$ where the first extension step is done with the clause $\mathbf{goal} \leftarrow \mathbf{Q}\theta\tau$. In other words, every path in the refutation is of the form

$$\neg\mathbf{goal}\neg\mathbf{Q_i}\theta\tau\cdots$$

where $\mathbf{Q_i}$ is some literal in $\mathbf{Q}$ (recall that $\mathbf{Q}$ is an abbreviation for $\mathbf{Q}_1 \wedge \cdots \wedge \mathbf{Q_n}$). Now suppose that $\mathbf{goal} \leftarrow \mathbf{Q}\theta\tau$ is used once again in the refutation. For syntactical reasons this can only happen if $\mathbf{goal} \leftarrow \mathbf{Q}\theta\tau$ is extended to a path resulting from a restart step. This path takes the form

$$\neg\mathbf{goal}\neg\mathbf{Q_i}\theta\tau\cdots\mathbf{A}\neg\mathbf{goal}$$

where $\mathbf{A}$ is some positive literal. Extension with $\mathbf{goal} \leftarrow \mathbf{Q}\theta\tau$ results (among possible others) in a path

$$\neg\mathbf{goal}\neg\mathbf{Q_i}\theta\tau\cdots\mathbf{A}\neg\mathbf{goal}\neg\mathbf{Q_i}\theta\tau$$

But now note that this extension step leads to a violation of the global negative regularity restriction and thus will be eliminated by the transformation given in the proof of Theorem 3.3. To be more precise, the path

$$\neg\mathbf{goal}\neg\mathbf{Q_i}\theta\tau\cdots\mathbf{A}\neg\mathbf{Q_i}\theta\tau$$

will come up instead. Since every extension with $\mathbf{goal} \leftarrow \mathbf{Q}\theta\tau$ will be eliminated in this way, the query clause $\mathbf{goal} \leftarrow \mathbf{Q}\theta\tau$ is used precisely once in this transformed refutation, namely for the first extension step. The lifting argument for this refutation then is the same as in Theorem 2.3. Thus we arrive at a refutation with computed answer $\mathbf{Q}\sigma$ which is more general than $\mathbf{Q}\theta$, which was to show. Q.E.D.

4 Implementation

All variants and refinements of ME discussed so far, i.e. the restart, strict and ancestry variants (possibly with selection function), loop checking by regularity and factorisation, are implemented in the PROTEIN system [Baumgartner and Furbach, 1994b]. It is a first order theorem prover based on the Prolog technology theorem proving (PTTP) technique, implemented in ECLiPSe-Prolog [ECRC, 1995].

Since ME is a goal-oriented, linear and answer complete calculus, it is well suited as an interpreter for disjunctive logic programming. PROTEIN facilitates computing disjunctive and definite answers. In its newest release their is also a flag which allows us to look for definite answers only.

5 Comparative Theorem Prover Study

In the sequel, we want to relate our experiences in computing answers by using theorem provers. First of all, we had to overcome some technical problems because theorem provers usually do not supply answers apart from ”yes” or (possibly) ”no”. We will illustrate our experiences with a puzzle example which allows for indefinite and definite answers.

5.1 Knights and Knaves

The example follows problem #36 in [Smullyan, 1978]. A similar example is studied in [Ohlbach, 1985]. The natural language description of the problem is stated below.

1. On an island, there live exactly two types of people: knights and knaves.
2. Knights always tell the truth and knaves always lie.
3. I landed on the island, met two inhabitants, asked one of them: ”Is one of you a knight?” and he answered me.
4. What can be said about the types of the asked and the other person depending on the answer I get? –
5. We assume, that either a proposition or its negation is true.
6. If the disjunction of two propositions is true then at least one of them must be true.

The last two pieces of information 5 and 6 explicitly state some knowledge about inferencing. We need them in order to be able to cope with the information in 2 because our description language is first order. – The reader may think about the problem solution for a while, before he looks at the solution which is given afterwards. For that, we have to make a case distinction:

(a) Suppose, the asked person answers with yes. Then he may be a knight, because then of course it is true that one of them is a knight. In this case, the other person can be a knight or a knave. – We cannot even exclude that the asked person is a knave. Then it must be true that none of them is a knight, hence the other person is also a knave in this case.

(b) Let us now assume, the asked person answers with no. Then this person must be a knave, since a knight cannot answer with no honestly. It follows, that the other person is a knight, because one of them must be a knight. So, in this case we get a definite answer.

In our formalization of the problem below, the formulae in 1 and 2 express the corresponding pieces of information from above. Depending on the case considered, we choose one of the formulae (a) or (b) in 3. We view the fact that a person denies a question as that he says that the thing in question is not true using the binary predicate $\mathbf{says}$ (instead of a ternary predicate). Formula 4 can be considered as the query. We have to express the pieces of information 5 and 6 explicitly by introducing the unary predicate $\mathbf{true}$. The transformation of the formulae below into clausal form is straightforward and therefore omitted here. It consists of 11 clauses. – The symbol $\dot{\vee}$ denotes *exclusive or*.

1. $\mathbf{true}(\mathbf{isa}(\mathbf{Q},\mathbf{knight}))\dot{\vee}\mathbf{true}(\mathbf{isa}(\mathbf{Q},\mathbf{knave}))$

2. $\mathbf{says}(\mathbf{P},\mathbf{S}) \rightarrow (\mathbf{true}(\mathbf{S}) \leftrightarrow \mathbf{true}(\mathbf{isa}(\mathbf{P},\mathbf{knight})))$

3. (a) $\mathbf{says}(\mathbf{asked},\bullet)$ ("yes")
   (b) $\mathbf{says}(\mathbf{asked},\mathbf{not}(\bullet))$ ("no")
   where $\bullet = \mathbf{or}(\mathbf{isa}(\mathbf{asked},\mathbf{knight}),\mathbf{isa}(\mathbf{other},\mathbf{knight}))$

4. $\neg\mathbf{true}(\mathbf{isa}(\mathbf{asked},\mathbf{X})) \vee \neg\mathbf{true}(\mathbf{isa}(\mathbf{other},\mathbf{Y}))$

5. $\mathbf{true}(\mathbf{not}(\mathbf{C}))\dot{\vee}\mathbf{true}(\mathbf{C})$

6. $\mathbf{true}(\mathbf{or}(\mathbf{A},\mathbf{B})) \leftrightarrow (\mathbf{true}(\mathbf{A}) \vee \mathbf{true}(\mathbf{B}))$

We can prove the query in many different ways. As a consequence we get many trivial and hence useless answers. The (most) trivial one – a four part disjunction – can be obtained in both cases. We only need formula 1 and the query in order to infer it. But it only says that each of both persons are either knights or knaves. In case (a) (if the asked person says yes) we can get an indefinite answer consisting of only three disjuncts. In the other case (b) there exists a definite answer. It follows a list of these possible answers where $\mathbf{X}/\mathbf{Y}$ is an abbreviation of $\mathbf{true}(\mathbf{isa}(\mathbf{asked},\mathbf{X})) \wedge \mathbf{true}(\mathbf{isa}(\mathbf{other},\mathbf{Y}))$.

1. $\mathbf{knave}/\mathbf{knave} \vee \mathbf{knave}/\mathbf{knight} \vee \mathbf{knight}/\mathbf{knave} \vee \mathbf{knight}/\mathbf{knight}$ (trivial)

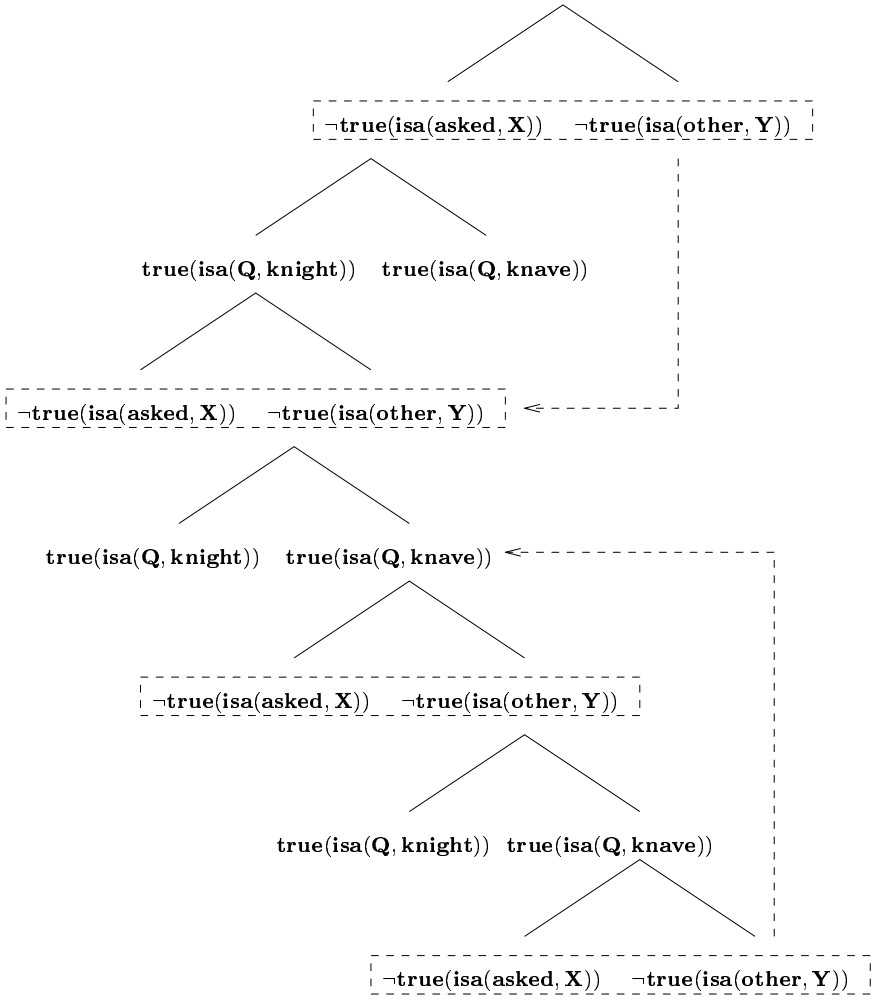

Figure 5: Derivation of the trivial answer

2. **knave/knave ∨ knight/knave ∨ knight/knight** (indefinite)

3. **knave/knight** (definite)

The Figures 5, 6 and 7 show tableaux for the derivations of the trivial, indefinite and definite answer respectively. All occurrences of the query are emphasised with dashed boxes. Substitutions are not annotated in order to keep the presentation clearer. Dashed upward links denote reduction steps. Dashed downward links abbreviate the presentation. They indicate proof parts that can be used repeatedly which can also also be explained by factorisation.

Before turning to our experiments we want to mention some interesting facts. Firstly, answer completeness requires that we are able to compute the indefinite and definite answer in the respective cases. Secondly, to derive these answers we need a

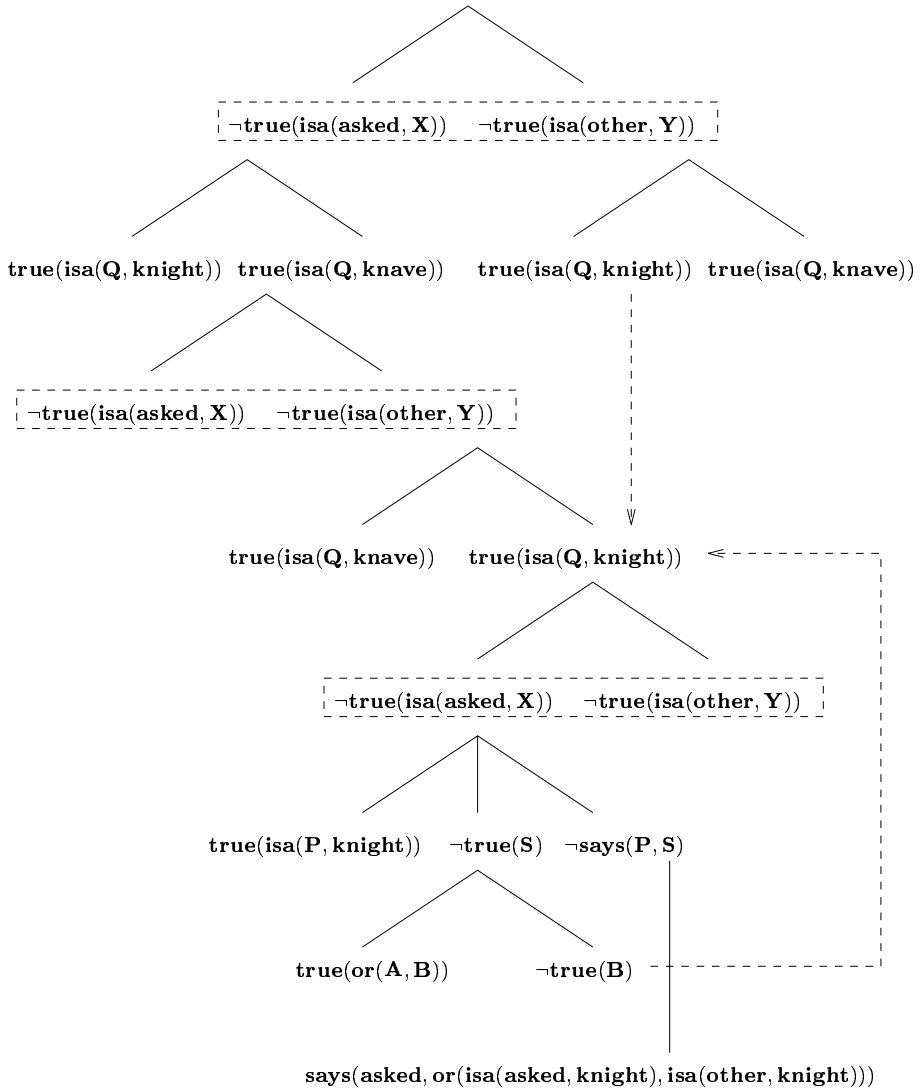

Figure 6: Derivation of the indefinite answer

clause set which is not minimally unsatisfiable; notice that the clauses of 1 and 4 together are (minimally) unsatisfiable yielding the trivial answer. Thirdly, 9 extension steps are needed to derive the indefinite or the definite answer respectively, while only 7 extension steps are needed to derive the trivial answer (in both cases). – These remarks indicate that it can be more difficult to find the more precise answers.

5.2 Experimental Results

We tried to get the answers from above automatically by using the theorem proving systems OTTER [McCune, 1994] which is a resolution-style theorem proving program coded in C for first order logic (with equality), SETHEO [Letz *et al.*, 1992] which is

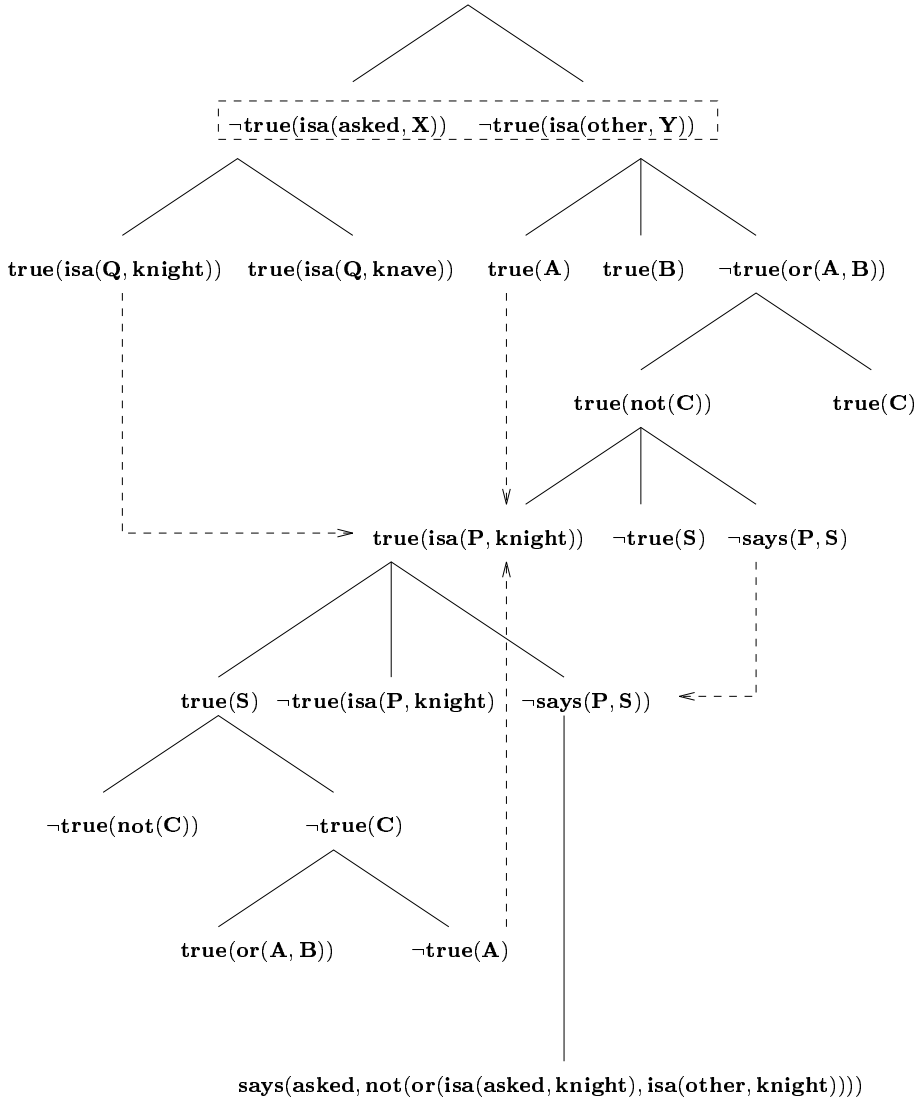

Figure 7: Derivation of the definite answer

a top-down prover for first order predicate logic based on the calculus of the so-called connection tableaux which generalises weak ME. SETHEO uses a WAM-compiler similar to Prolog, and the resulting code is interpreted by a C program. Of course, we also used our PROTEIN prover ([Baumgartner and Furbach, 1994b], Section 4).

We used the clause ordering given by the problem description, but our experiments show that the (run time) results depend on the ordering.

OTTER has some problems with computing answers because it enumerates resolvents but not all (refutational) proofs. Especially during the subsumption test, it did not take the answer literals into account which are provided for computing answers. That is the reason why OTTER with (forward and backward) subsumption is *not* answer com-

Prover	Answer	Time (s)	Settings
OTTER	trivial	2.1	plain hyper-resolution
	indefinite	0.3	hyper-resolution + factor.
	definite	∞	several trials
SETHEO	trivial	0.5	with constraints
	indefinite	1.0	with constraints
	definite	0.6	with constraints
PROTEIN	trivial	0.5	any setting
	indefinite	∞	plain ME
		41.4	restart + sel. function
	definite	2022.8	plain ME
		38.4	ancestry restart
SATCHMO	trivial	∞	any setting
	indefinite	∞	any setting
	definite	∞	any setting

Table 1: Timings I: Knights and Knaves #36

plete. An example which illustrates this is case (a) where the search stops after finding 15 times only the trivial answer with binary resolution. However, we find a proof by using hyper-resolution with factorisation immediately within 0.4s. – There is a solution to the problem with subsumption; it can be shown that we only have to take the answer literals into account during the subsumption steps. Unfortunately, it is not (yet) possible to test OTTER in this setting and find out whether this improves the behaviour, because it is not built in.

We generate answers with SETHEO by using global variables. The answers are kept in a list. By this and other technical tricks, we find the indefinite answer within 1.0s and the definite answer within 0.6s. That is quite good and may be explained by the subgoal reordering heuristics built into SETHEO, which are not (yet) incorporated into our system. But in addition, SETHEO also has subsumption constraints which are used in the default setting. It is not quite clear, whether these constraints destroy answer completeness in SETHEO.

Table 1 shows the timings for OTTER and SETHEO. All timings are measured on a Sparc 10. The symbol ∞ denotes the fact that no proof was found within 1 hour; that is true for OTTER applied to case (b) of our example. It also shows the timings for PROTEIN and SATCHMO. Since SATCHMO differs from the provers mentioned so far because it can be used for model generation, we will discuss SATCHMO later in Section 5.4.

PROTEIN *is* answer complete; that has been stated in this paper. It finds the indefinite and definite answer for the respective case. Table 1 also shows some timings for finding these answers with PROTEIN. We tried both, plain and restart ME. In the case

Prover	Problem	Time (s)	Settings
OTTER	#27	∞	binary resolution
	#35	281.8	binary resolution
	#39	0.7	plain hyper-resolution
	#42	∞	binary resolution
SETHEO	#27	4.5	with constraints
	#35	1.2	with constraints
	#39	296.8	with constraints
	#42	14.4	with constraints
PROTEIN	#27	∞	any setting
	#35	153.1	plain restart
	#39	906.5	ancestry restart
	#42	∞	any setting
SATCHMO	#27	0.2	any setting
	#35	0.2	any setting
	#39	0.4	any setting
	#42	0.2	any setting

Table 2: Timings II: Other Puzzles

of the restart variant we also tried its refinements: with or without ancestry restart or selection function (no contrapositives). We tried to compute the desired answers with settings where all solutions are computed in case (a) (indefinite answer). For the case (b) (definite answer) we used the setting where only definite answers are searched for. By this, we get a significant speed up of the search. – As one can see, using restart helps for this problem, since plain ME does not find the desired answers quickly, although it does so for trivial answers. But it is not quite clear which flags should be used in addition.

5.3 Other Examples

We investigated more puzzle examples from [Smullyan, 1978] as formulated in the TPTP problem library [Sutcliffe *et al.*, 1994] that allow for definite answers. Table 2 shows the best run times we achieved for the considered examples. SETHEO behaves quite well on these examples because we only searched for definite answers; there are some technical problems to compute disjunctive answers. Again, OTTER has some problems and runs out of set of support with hyper-resolution while binary resolution does not find the desired answers concerning the examples #27 and #42. Look also at Table 3 where some timings for the so-called Blind Hand problem are shown. The version number corresponds to that in the TPTP problem library.

All our experiments corroborate the following facts: resolution has difficulties in solving puzzles because of the problem with subsumption; model elimination is better

Prover	Version	Time (s)	Settings
OTTER	1	7.13	plain hyper-resolution
	2-1	7.47	plain hyper-resolution
	2-2	7.21	plain hyper-resolution
SETHEO	1	1.37	default setting
	2-1	0.08	default setting
	2-2	0.08	default setting
PROTEIN	1	1.12	ancestry + sel. function
	2-1	0.13	plain ME
		0.60	ancestry restart
	2-2	0.10	plain ME
		0.47	ancestry restart
SATCHMO	1	∞	any setting
	2-1	∞	any setting
	2-2	∞	any setting

Table 3: Timings III: Blind Hand Problem

suited although it could not solve all puzzles that we tested. Further investigations are necessary. Last but not least, we want to point out that both, OTTER and SETHEO do not support a procedural reading of program clauses – they all need contrapositives – but PROTEIN does; and that is useful if we want to use logic as a real programming language.

5.4 Model Elimination versus Model Generation

It seems also that a model generation approach is very adequate for these kind of problems because they often allow for finite models. In this special case we can derive the answers from the models by the following non-deterministic procedure: Extract from each of the (finitely many) models one instance of the query, and combine them into one disjunctive answer. The proof that justifies this procedure is trivial for the finite case and hence left out here.

With SATCHMO [Manthey and Bry, 1988], we get all answers quite fast by using it for model generation since all considered puzzles allow for finite models – except #36. That is clear because in our formulation only infinite models exist: As soon as $\mathbf{true}(\mathbf{S})$ is valid for some expression $\mathbf{S}$, it must hold $\mathbf{true}(\mathbf{not}(\mathbf{not}(\mathbf{S})))$ too; that means, we can derive facts with any number of double negations. This increases the search space to such a degree that the refutationally complete level-saturation version of SATCHMO is not able to find the desired answers.

Therefore, we investigated two other formalisations of this problem (due to François Bry) which have finite models and are a bit longer than ours. Both versions

permit the coding of the problem into one single clause set. Concerning version #1, SATCHMO finds the desired (disjunctive) answer within 0.05s. But PROTEIN also finds the answer within this time in proof depth 6. Version #2 is a bit tricky since it uses a the SATCHMO rule $(\mathbf{false} \leftarrow \mathbf{S}) \leftarrow \mathbf{true}(\mathbf{not}(\mathbf{S}))$ which mixes object- and meta-level. On the one hand, this is an advantage of SATCHMO; but on the other hand, it is not quite clear whether completeness can be assured. Ordinary first order theorem provers need three additional lemmata in order to be able to find out the solution.

In conclusion, model generation seems to be promising for examples which allow for finite domains whereas model elimination approaches are more robust for a wider range of applications.

6 Conclusion

To conclude, it seems to be very promising to use ME as a base calculus for computing answers in disjunctive logic programming. In this paper, we introduce (among others) the ancestry restart variant which is quite well suited for this purpose. We also give some practical evidence. Nevertheless, further investigation is necessary in order to find out yet more efficient calculi and to incorporate nonmonotonic extensions.

Acknowledgements

We would like to thank François Bry, Jürgen Dix, Bertram Fronhöfer, Reinhold Letz and William W. McCune for helpful discussions, and Olaf Menkens and Dorothea Schäfer for their implementational work.

References

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

[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.

[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).

[Baumgartner, 1994] P. Baumgartner. Refinements of Theory Model Elimination and a Variant without Contrapositives. In A.G. Cohn, editor, *11th European Conference on Artificial Intelligence, ECAI 94*. Wiley, 1994. (Long version in: Research Report 8/93, University of Koblenz, Institute for Computer Science, Koblenz, Germany).

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

[Demolombe and Imielski, 1994] R. Demolombe and T. Imielski. *Nonstandard Queries and Nonstamdard Answers*. Oxford University Press, 1994.

[ECRC, 1995] ECRC GmbH, München. *ECLiPSe 3.5: User Manual – Extensions User Manual*, February 1995.

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

[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.

[Lobo *et al.*, 1992] Jorge Lobo, Jack Minker, and Arcot Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, Cambridge, MA, London, England, 1992.

[Loveland and Reed, 1992] D. Loveland and D. Reed. Near-Horn Prolog and the Ancestry Family of Procedures. Technical Report CS-1992-20, Department of Computer Science, Duke University, Durham, North Carolina, December 1992.

[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] Rainer Manthey and François Bry. SATCHMO: a theorem prover implemented in Prolog. In Ewing Lusk and Ross Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction, Argonne, Illinois, USA, May 1988*, pages 415–434. Springer, Berlin, Heidelberg, New York, 1988. LNCS 310.

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

[Ohlbach, 1985] Hans Jürgen Ohlbach. Predicate logic hacker tricks. *Journal of Automated Reasoning*, 1:435–440, 1985.

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

[Smullyan, 1978] Raymond M. Smullyan. *What is the name of this book? The riddle of Dracula and other logical puzzles*. Prentice-Hall, Englewood Cliffs, NJ, 1978.

[Sutcliffe *et al.*, 1994] G. Sutcliffe, C. Suttner, and T. Yemenis. The TPTP problem library. In A. Bundy, editor, *Proc. CADE-12*, volume 814 of *LNAI*. Springer, 1994.