Computing Answers to Logic Programs with Weak Model Elimination

Marco A. Casanova, Ramiro A. de T. Guerreiro, Andrea Silva

Rio Scientific Center - IBM Brazil
P.O. Box 4624
22.071, Rio de Janeiro, RJ - Brazil

Abstract

This work addresses the foundations of logic programming systems based on the weak model elimination method. An adaptation of the method is shown to be sound and complete with respect to computing answers when the logic programs are arbitrary sets of clauses. A second variation of the method is also shown to be complete with respect to computing only definite answers.

1. Introduction

Model elimination [8,9,10] offers an interesting theoretical basis for logic programming systems. In particular, Prolog systems use a variation of model elimination, called SLD-resolution, that has been widely investigated (see [7]). However, SLD-resolution accepts only a restricted class of clauses, forcing in a certain sense the adoption of negation by finite failure [3] to extend the expressive power of logic programs and queries.

This paper investigates the foundations of logic programming systems based on another variation, called weak model elimination [9]. The main results establish that an adaptation of the method is sound and complete with respect to computing answers when the logic programs and queries are expressed by sets of generic clauses. The question of computing just definite answers [12] is also settled.

Weak model elimination (WME) has several attractive characteristics. It accepts sets of generic clauses as input, that is, sets of clauses with an arbitrary number of positive or negative literals. Moreover, WME is input linear, does not use factoring and, yet, is refutationally complete.

However, WME is a monotonic formal system, where negation has the classical meaning. A companion paper [6] then extends WME with defaults to capture non-monotonic reasoning.

A logic programming system based on the results reported here is also described in [13]. An early implementation of the original weak model elimination method is described in [4] and the question of implementing an inference engine based on weak model elimination and using the technology of Prolog processors is
discussed in [2,14].

The variation of \textit{WME} described in this paper compares with the basic method reported in [9] as follows. First, it adopts a different literal selection strategy to become closer to SLD-resolution with the selection function that chooses the leftmost literal, which is the basis of standard Prolog. Second and more important, the variation incorporates answer literals [5] to compute answers. However, because the inference rules of \textit{WME} work with ordered sequences of literals, answer literals could not simply be added to the clauses, but rather \textit{WME} had to be adapted to work with pairs of the form \((C,B)\) where \(C\) is a clause and \(B\) is a set of answer literals. This choice turned out to be comfortable both for the development of the metatheory and for the practical examples. Answer literals are also used in [11] to obtain results about the resolution method.

The Soundness and Completeness Theorems reported here generalize the comparable results for computing answers by SLD-resolution, contained in [7], but the details are much more complex, especially the MGU and the Lifting lemmas. This follows because the notion of SLD-refutation is far simpler than that of WME-refutation since SLD-Resolution works only with definite programs and with queries consisting of just one conjunction of literals. Moreover, the answer computed by a SLD-refutation is the composition of the substitutions applied over the variables of the first clause of the refutation, which is always the clause representing the query. Neither of these facts hold in \textit{WME}, as we shall see.

Finally, the variation that computes only definite answers is based on a new result about refutations in \textit{WME}.

The organization of this paper is as follows. Section 2 introduces the notions of program, query and answer. Section 3 reviews the weak model elimination method and extends it to compute answers. Section 4 contains the proof of the basic results. Section 5 describes the specific results for definite answers. Finally, section 6 contains the conclusions.

2. Basic Definitions and Motivation

2.1 Logic Preliminaries

All definitions in what follows are relative to a fixed first-order alphabet \(\mathcal{A}\). By convention, letters from the end of the alphabet will denote variables and letters from the beginning of the alphabet or strings of letters will denote constants.

A \textit{literal} is an atomic formula or a negated atomic formula. Two literals are \textit{complementary} if and only if they are of the form \(P\) and \(\neg P\), for some atomic formula \(P\). We will use \(\neg L\) to denote the literal which is complementary to \(L\) and \(|L|\) to indicate the atomic formula \(P\), if \(L\) is the literal \(P\) or the literal \(\neg P\).

A \textit{clause} is a sequence of literals. Each clause \(C\) represents, by convention, the universal closure of the disjunction of its literals, in the sense that any structure for the alphabet \(\mathcal{A}\) satisfies \(C\) if and only if it satisfies the formula that \(C\) represents. A set of clauses \(S\) is a \textit{clausal representation} for a formula \(F\) iff \(F\) is satisfiable iff \(S\) is satisfiable. We will adopt the symbol \(\Box\) to denote the empty
clause, as usual, and $CL(F)$ to denote a clausal representation for the formula $F$.

We assume some familiarity with the notion of substitution and composition of substitutions. We will use the symbol $\varepsilon$ to denote the identity (or empty) substitution. Given a substitution $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$, the domain of $\theta$ is the set $\{x_1, \ldots, x_n\}$ and the range of $\theta$ is the set $\{t_1, \ldots, t_n\}$. We also say that a substitution is over the variables in its domain. A substitution $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ is a renaming iff $t_1, \ldots, t_n$ are all distinct variables and different from $x_1, \ldots, x_n$. If $\theta = \{x_1/y_1, \ldots, x_n/y_n\}$ is a renaming, then we will use $\theta^{-1}$ to denote the substitution $\{y_1/x_1, \ldots, y_n/x_n\}$. We will call $\theta^{-1}$ the inverse of $\theta$, although it is not the compositional inverse of $\theta$.

Let $\theta$ be a substitution, $A$ be a clause (or a literal) and $S$ be a set of clauses (or a list of literals). We will use $A^\theta$ to denote the clause (or literal) resulting from applying $\theta$ to $A$ and $S^\theta$ to indicate the set of clauses (or the list of literals) resulting from applying $\theta$ to each clause (or literal) in $S$. A clause $D$ is an instance of a clause $C$ iff there is a substitution $\theta$ such that $D = C^\theta$ and a clause $D$ is a variant of a clause $C$ iff there is a renaming $\theta$ such that $D = C^\theta$.

Let $L_1$ and $L_2$ be literals. We say that a substitution $\theta$ is a unifier of $L_1$ and $L_2$ iff $L_1^\theta = L_2^\theta$. We say that a unifier $\theta$ is regular iff the domain of $\theta$ is a subset of the set of variables that occur in $L_1$ or $L_2$ and we say that $\theta$ is normal iff $\theta$ is regular and all variables in $L_1^\theta$ and $L_2^\theta$ already occur in $L_1$ or $L_2$. Finally, we say that a unifier $\theta$ of $L_1$ and $L_2$ is a most general unifier (m.g.u) iff, for any unifier $\phi$ of $L_1$ and $L_2$, there is a substitution $\gamma$ such that $\phi = \theta^\gamma$.

Two literals $L_1$ and $L_2$ are unifiable iff there is a unifier for $L_1$ and $L_2$. In fact, if $L_1$ and $L_2$ are unifiable then there is a normal m.g.u. for them. Also, the standard m.g.u. algorithms compute normal m.g.u.'s. Hence, in the rest of this paper, we will use the term unifier to mean regular unifier and the term m.g.u. to mean normal m.g.u.

Finally, if $F$ is a formula, we will use $\forall F$ to denote the universal closure of $F$, $\exists F$ to denote the existential closure of $F$ and $F^\theta$ to denote the formula resulting from applying $\theta$ to $F$. Also, if $Q$ is a conjunction of literals $L_1 \land \ldots \land L_m$, we will use $\neg Q$ to denote the clause $\neg L_1 \ldots \land \neg L_m$ (which is the clausal representation of $\neg \exists \theta Q$). Conversely, if $C$ is a clause of the form $L_1 \ldots L_m$, we will use $\neg C$ to denote the conjunction $\neg L_1 \land \ldots \land \neg L_m$.

2.2 Programs, Queries and Answers

A program is a finite set of clauses and a query is a disjunction of conjunctions of literals, that is, a quantifier-free formula in disjunctive normal form. A query is definite iff it is a single conjunction of literals, otherwise it is indefinite.

We require that queries be formulas in a restricted syntax simply to trivialize the problem of obtaining clausal representations of the negation of their existential closure and, more importantly, to avoid changing the original alphabet through the introduction of Skolem functions. Indeed, if $Q$ is a query of the form
Q_1 \vee \ldots \vee Q_n$, then the clausal representation of the negation of its existencial closure is the set $CL(\neg \exists Q) = \{ \neg Q_1, \ldots, \neg Q_n \}$.

An answer to a query $Q$ over a program $P$ is either False or a disjunction of instances of conjunctions in $Q$ over the alphabet of $P$ and $Q$, that is, a disjunction of conjunctions obtained from those in $Q$ by substituting variables by terms of the alphabet used to write $P$ and $Q$. An answer is definite iff it consists of a single conjunction, otherwise it is indefinite [12].

We let False be an answer simply because it will be the most general answer to any query over an inconsistent program.

An answer $A$ to $Q$ over $P$ is correct iff $P$ logically implies $\forall A$. Finally, an answer $A$ to $Q$ over $P$ is more general than an answer $B$ to $Q$ over $P$ iff $A$ logically implies $B$.

For example, the following set of clauses is a program, that we call DIC:

1. program(a,fortran)
2. program(b,pascal)
3. program(c,fortran) program(c,pascal)
4. calls(a,b)
5. calls(b,c)
6. $\neg$ calls$(x,y) \land$ depends$(x,y)$
7. $\neg$ calls$(x,z) \land$ depends$(x,y)$

Thus, clause (3) tells us that the constant $c$ denotes an ordinary program written in fortran or pascal and clauses (6) and (7) indicate that $x$ depends on $y$ if $x$ calls $y$ direct or transitively.

The formula below is a query, that we call DEP:

$$(\text{depends}(a,x) \land \text{program}(x,pascal)) \lor$$
$$(\text{depends}(a,x) \land \text{program}(x,fortran))$$

It asks for a program written in fortran or pascal that the program denoted by the constant $a$ depends on. An answer $A$ to DEP over DIC would be:

$$\text{depends}(a,b) \land \text{program}(b,pascal)$$

Indeed, the conjunction in $A$ is an instance of the first conjunction in DEP. It is in fact a correct answer since DIC logically implies $\forall A$. A second correct answer to DEP over DIC would be:

$$(\text{depends}(a,c) \land \text{program}(c,fortran)) \lor$$
$$(\text{depends}(a,c) \land \text{program}(c,pascal))$$

Therefore, an indefinite query may have both indefinite and definite answers.

As another example, the following formula is also a query (call it LANG):

$$\text{program}(c,x)$$

It asks for a language in which program $c$ is written. It has only one correct answer, which is:
program(c,fortran) v program(c,pascal)

Therefore, a definite query may have just indefinite answers.

We could also find examples where all other cases involving definite or indefinite queries and definite or indefinite answers hold.

The basic goal of this paper is to find a method of computing all correct answers to a query, a task addressed in sections 3 and 4. Section 5 also settles the question of computing only the definite answers to a (definite or indefinite) query.

2.3 Motivation

We briefly discuss in this section three classes of logic programming systems, defined according to the type of logic programs and queries they accept. The discussion will help motivate our interest in logic programming systems based on weak model elimination. Readers not familiar with Prolog are invited to skip this section.

Recall that a **definite program clause** is a clause with exactly one positive literal and a **definite goal clause** is a clause with no positive literals. Usually a definite clause \( L_0 \rightarrow L_1 \ldots \rightarrow L_n \) is denoted by an expression of the form \( L_0 \leftarrow L_1 \ldots, L_n \), where \( L_0 \) is the head of the clause and \( L_1 \ldots, L_n \) is the body of the clause, and a goal clause \( \neg L_1 \ldots \neg L_n \) is denoted by an expression of the form \( \leftarrow L_1 \ldots, L_n \).

We also need two other concepts. We define a **normal program clause** as an expression of the form \( h_0 \leftarrow b_1 \ldots, b_n \), where \( h_0 \), the head of the clause, is a positive literal and \( b_1 \ldots, b_n \), the body of the clause, is a list of positive or negative literals. Finally, we define a **normal goal clause** as an expression of the form \( \leftarrow b_1 \ldots, b_n \) where \( b_1 \ldots, b_n \) is again a list of positive or negative literals.

Now, we say that a logic programming system is **full first-order** if it accepts programs and queries as defined in section 2.2. Note that this paper then discusses a specific class of full first-order systems, those based on **WME**. We say that a system is a **pure** Prolog system if it accepts programs consisting of finite sets of definite program clauses and queries expressed by definite goal clauses. Finally, we say that a system is an **extended** Prolog system if it accepts programs consisting of finite sets of normal program clauses and queries expressed by normal goal clauses and, moreover, it treats negated literals by a special inference rule, called **negation by finite failure** (NFF). Roughly, the NFF rule says to assume that \( p \) is false if one fails finitely to answer YES to the query \( \leftarrow p \) in the presence of the program clauses.

We will argue that in some cases full first-order systems are an interesting alternative to pure or extended Prolog systems. We first emphasize two points about full-first order systems:

- the knowledge representation language of such systems has the same power as full first-order languages and, in particular, it maintains the classical meaning of negation;
- they can be implemented efficiently using the technology developed for Prolog processors, especially those based on weak model elimination, as argued elsewhere [2].
The rest of this section stresses the importance of the first point.

Although pure Prolog systems have Turing machine power, the restriction to
definite clauses makes it awkward to represent certain applications. The answer to
this problem came with extended Prolog systems that treat negated literals by the
NFF rule. We shall now show that this approach may in some cases be a worse
alternative than returning to full first-order systems.

We may point out at least three important and distinct characteristics of the NFF
rule:

• NFF is easy to implement in Prolog systems;
• NFF is a non-monotonic rule justified on the grounds of the so-called
  "Closed World Assumption" (CWA) [12];
• it is very difficult to give a theoretically precise characterization of NFF (that
  is, one for which NFF is sound and complete).

The first characteristic is undoubtedly the greatest argument in favor of the
adoption of the NFF rule.

The second characteristic can be taken either in favor or against the use of NFF,
depending on whether the CWA holds or not for the application in question. If
the CWA does not hold, an extended Prolog system will hardly be adequate and
one should seriously consider a full first-order logic programming system.

The third characteristic can be best examined with the help of a very simple
example. Consider the question of expressing a disjunction \( p \lor q \) as a general
program clause. If we naively take the symbol \( \leftarrow \) to mean (reverse) implication
and \( \lnot \) to mean true negation, then \( p \lor q \) is indeed equivalent to \( p \leftarrow \lnot q \) and to
\( q \leftarrow \lnot p \). But this equivalence is not true because negated literals are treated by
the NFF rule. Indeed, let \( P_1 = \{ p \leftarrow \lnot q \} \) and \( P_2 = \{ q \leftarrow \lnot p \} \) be two programs.
Let \( Q \) be the query \( \leftarrow p \). Then, the answer of \( Q \) to \( P_1 \) will be TRUE and to \( P_2 \)
will be FALSE in an extended Prolog system.

This apparently incorrect behavior will certainly shock naive Prolog users, but it
can be explained by appealing, for example, to Clark's theory of program
completion. Indeed, denote by \( \text{comp}(S) \) the completion of a program \( S \). Then,
\( \text{comp}(P_1) = \{ p \leftarrow \lnot q, \lnot q \} \) and \( \text{comp}(P_2) = \{ q \leftarrow \lnot p, \lnot p \} \). Hence, \( p \) is
indeed a logical consequence of \( \text{comp}(P_1) \) but not of \( \text{comp}(P_2) \), which correctly
explains the previous answers, but by all means justifies the bizarre behaviour of
extended Prolog systems.

From this simple discussion, it becomes clear that, although NFF indeed extends
in some sense the expressiveness of pure Prolog systems, it greatly complicates the
theory to the point of raising serious questions even about soundness.

One last word should be said about disjunctions. In general, from a disjunction
and a definite clause one may infer a clause as defined in section 2.1. Hence, it
requires a full first-order system to process logic programs consisting of finite sets
of arbitrary disjunctions and definite clauses.

To summarize this last part of the discussion, quite expectedly, applications
exhibiting disjunctive information, or some other form of information not easily
captured by definite clauses, may be better modelled using full first-order logic programming systems.

3. Computing Answers with \textit{WME}

3.1 Weak Model Elimination

This subsection introduces the basic weak model elimination method while the next subsection describe the necessary modifications to compute answers.

To achieve completeness, the inference rules of weak model elimination sometimes maintain the resolved literals within the derived clauses and keep the literals (resolved or not) ordered within a clause. To capture this situation, we redefine the concept of clause and consider a fixed first-order alphabet \( A \), augmented with left and right square brackets, \([\ ]\) and \( [\ ]\), that have the status of special punctuation marks.

More precisely, a \textit{resolved literal} (or an \textit{R-literal}) is an expression of the form \([L]\), where \( L \) is a literal. (Hence, an R-literal is not a literal). Two R-literals are \textit{complementary} if and only if they are of the form \([P]\) and \([\neg P]\), for some atomic formula \( P \).

An \textit{element} is a literal or an R-literal.

A \textit{(first-order) elementary chain} is any sequence of literals and a \textit{(first-order) chain} is any sequence of elements. The symbol \( \square \) will again denote the empty chain, which is elementary by definition. Each chain \( C \) represents, by convention, the universal closure of the disjunction of its literals, in the sense that any structure for the alphabet \( A \) satisfies \( C \) if and only if it satisfies the formula that \( C \) represents. Hence, the R-literals of a chain do not influence its semantics.

It should be clear that elementary chains and clauses are one and the same concept because we defined clauses as sequences of literals (and not sets of literals).

The following definitions introduce the classes of chains admitted as antecedents and consequents of the inference rules. A chain is \textit{preadmissible} if and only if:

1) complementary literals are separated by an R-literal;
2) if a literal is identical to an R-literal, then the literal is to the right of the R-literal;
3) two R-literals are not complementary;
4) two R-literals are not equal.

A chain is \textit{admissible} if and only if it is preadmissible and the leftmost element is a literal.

The next definitions are basic for the inference rules of the method and assume familiarity with the notion of unification. Two literals \( L' \) and \( L'' \) can be \textit{cancelled by a substitution} \( \theta \) if and only \( L'\theta \) and \( L''\theta \) are complementary and \( \theta \) is a most general unifier of \( \{L',L''\} \). In what follows, \( B'B'' \) denotes the concatenation of two chains \( B' \) and \( B'' \).
Let $A'$ be a chain and let $A''$ be an elementary chain. Let $\beta$ be a renaming of some variables of $A''$ such that $A''\beta$ has variables distinct from those of $A'$ and $A''$. Let $L'$ be the leftmost element of $A'$ and suppose that $L'$ is a literal. A chain $A$ is an extension of $A'$ by $A''$ if and only if there exists a literal $L''$ of $A''$ and a substitution $\theta$ such that $L'$ and $L''\beta$ can be cancelled by $\theta$ and $A = B''B'$, where $B''$ is the chain $A''\beta\theta$ with the literal $L''\beta\theta$ removed and $B'$ is the chain $A'\theta$ with the literal $L'\theta$ replaced by $[L'\theta]$.

Let $A'$ be a chain. Let $L'$ be the leftmost element of $A'$ and suppose that $L'$ is a literal. A chain $A$ is a reduction of $A'$ if and only if there exists a R-literal $[M']$ of $A'$ and a substitution $\theta$ such that $L'$ and $M'$ can be cancelled by $\theta$ and $A$ is $A'\theta$ with the literal $L'\theta$ removed.

A chain $A$ is a contraction of a chain $A'$ if and only if $A$ can be obtained by removing all R-literals that are to the left of the leftmost literal. In particular, if $A$ has only R-literals, the contraction of $A$ will be the empty chain.

A chain $A$ is a full extension of $A'$ by $A''$ if and only if $A$ is the contraction of an extension of $A'$ by $A''$. A chain $A$ is a full reduction of a chain $A'$ if and only if $A$ is the contraction of a reduction of $A'$.

The weak model elimination method, $WME$, is defined as follows:

**class of languages**: the sets of first-order chains

**axioms**: none

**inference rules**: full extension and full contraction, defined as follows:

**Full Extension**:
- If $A'$ is an admissible chain
- $A''$ is an elementary chain and
- $A$ is a full extension of $A'$ by $A''$
- which is an admissible chain,
- then derive $A$ from $A'$ and $A''$.

**Full Reduction**:
- If $A'$ is an admissible chain and
- $A$ is a full reduction of $A'$
- which is an admissible chain,
- then derive $A$ from $A'$.

Note that the admissibility test may block the application of a rule if an antecedent or the consequent fails the test.

A $WME$-deduction of a chain $C$ from a set $S$ of elementary chains is any finite sequence of chains $E = (E_1,...,E_n)$ such that $C$ is the last chain of $E$, there is $i \leq n$ such that $E_1,...,E_i$ are chains in $S$ and, for each $j \in [i+1,n]$, $E_j$ is derived from $E_{j-1}$, the parent chain of $E_j$, by full reduction or full extension, in the latter case using as auxiliary chain a chain $E_k$ such that $k \leq i$. The sequence $E_1,...,E_i$ is called the prefix of $E$ and $E_i$ is called the initial chain of $E$.

A $WME$-refutation from a set of elementary chains $S$ is a $WME$-deduction of the empty chain from $S$. 
The \textit{WME} method defined above is slightly different from the original version introduced in [8], but the results therein can be easily adapted to establish that \textit{WME} is refutationally sound and complete.

For technical reasons, in the formal development of section 4 we will use relaxed versions of the inference rules and, consequently, of the notions of deduction and refutation.

We then define \textit{unrestricted full extension} and \textit{unrestricted full reduction} exactly as before, except that the unifier used need not be most general and the renaming substitution $\beta$ used in the extension of a chain $A'$ by an elementary chain $A''$ need not guarantee that $A''\beta$ has variables distinct from those of $A'$ and $A''$. From these concepts, we immediately obtain the notions of \textit{unrestricted WME-deduction} and \textit{unrestricted WME-refutation}.

Likewise, we define \textit{free unrestricted full extension} and \textit{free unrestricted full reduction} by further dropping the admissibility test, which then induces the notions of \textit{free unrestricted WME-deduction} and \textit{free unrestricted WME-refutation}.

Note that every WME-deduction is also an unrestricted WME-deduction. Hence, when necessary to emphasize that an unrestricted WME-deduction $R$ is actually a WME-deduction in the stricter sense, we will call $R$ a \textit{restricted WME-deduction}. Likewise, when necessary to emphasize that an unrestricted WME-deduction $R$ is not a WME-deduction in the stricter sense, we will call $R$ a \textit{non-restricted WME-deduction}.

Finally, we observe that these relaxed versions destroy neither the consistency nor the completeness of \textit{WME}.

\textbf{3.2 Computing Answers with WME}

Given a WME-refutation $R$ from the elementary chains in a program $P$ and in the clausal representation $CL(\neg \exists Q)$ of the negation of the existential closure of a query $Q$, it is possible to show that the substitutions applied to the free variables of chains in $CL(\neg \exists Q)$ during the construction of $R$ induce a correct answer to $Q$ over $P$. However, to recover such substitutions is not exactly simple since $CL(\neg \exists Q)$ may possibly contain more than one chain, that may also be reused in $R$. This section then redefines the notion of chain and the inference rules of \textit{WME} to register such substitutions.

An \textit{activated chain} is a pair of the form $(C,L)$, where $C$ is a chain and $L$ is a list of literals.

In what follows, we will use $\lambda$ to denote the empty list, $<a_1,...,a_n>$ to denote the list $(a_1,(...,a_n,\lambda)...))$ and $A | A'$ to denote the concatenation of list $A$ with list $A'$.

To compute answers to a query $Q$ over a program $P$, we must first prepare the chains in $P$ and in the clausal representation of $\neg \exists Q$ as follows.

The \textit{activation} of $P$ is the set $\text{activate}(P)$ consisting of the activated chains $(C,\lambda)$, where $C \in P$. An \textit{activation} of a query $Q$ of the form $Q_1 \land ... \land Q_n$ is a set of
activated chains \((\sim Q_i \langle r_i(\bar{x}_i) \rangle, i=1,...,n\), where \(\sim Q_i\), by convention, is the chain consisting of the complement of the literals of \(Q_i\), \(\bar{x}_i\) is a list of the variables of \(Q_i\) in any order and \(r_i\) is a predicate symbol, not in the original alphabet, whose arity is equal to the length of \(\bar{x}_i\). The literal \(r_i(\bar{x}_i)\) is the answer literal for \(Q_i\) in this activation of \(Q\).

An activation of a query \(Q\) therefore produces a clausal representation of \(\neg \exists Q\), with each clause annotated with an answer literal whose function will be to record the substitutions applied to the variables of the clause. But, to effect this recording, the inference rules of \textit{WME} had to be modified as we will shortly describe.

Since neither the list of the variables nor the answer literals are fixed a priori, there could be countably many activations for the same query. To avoid these ambiguities, we assumed that the answer literals are always \(r_1(\bar{x}_1), r_2(\bar{x}_2),...\), where the variables in \(\bar{x}_1, \bar{x}_2,...\) are listed lexicographically. Note that the arity of \(r_1, r_2,...\) vary from query to query. We will denote by \textit{activate}(\(Q\)) the activation of a query \(Q\) constructed in this way.

An activated chain \((A,L)\) is a full activated reduction of an activated chain \((A',L')\) if and only if \(A\) is a full reduction of \(A'\) with m.g.u. \(\emptyset\) and \(L = L' \emptyset\). An activated chain \((A,L)\) is a full activated extension of \((A',L')\) by an elementary activated chain \((A'',L'')\) if and only if \(A\) is a full extension of \(A'\) by \(A''\), with m.g.u. \(\emptyset\) and renaming \(\beta\) of \(A''\), and \(L = L' \emptyset \parallel L'' \beta \emptyset\).

The notions of \textit{activated \textit{WME}-deduction} and \textit{activated \textit{WME}-refutation} follow directly from those of \textit{WME}-deduction and \textit{WME}-refutation when we replace 'chain' by 'activated chain', 'full reduction' by 'full activated reduction' and 'full extension' by 'full activated extension'. Likewise, the notions of \textit{activated unrestricted \textit{WME}-deduction} and \textit{activated unrestricted \textit{WME}-refutation} follow directly from those of unrestricted \textit{WME}-deduction and unrestricted \textit{WME}-refutation.

An answer \(A\) to \(Q\) over \(P\) is \textit{\textit{WME}-computed} if and only if there is an activated \textit{WME}-refutation \(R\) from \textit{\textit{activate}(P)} \cup \textit{\textit{activate}(Q)} such that either \(R\) terminates in \((\Box,\lambda),\) in which case \(A\) must be equal to \textit{\textit{False}}, or \(R\) terminates in \((\Box,\lambda),\) with \(L \neq \lambda,\) and \(A\) is a disjunction of all conjunctions \(B\) such that there is \((\sim Q_i \langle r_i(\bar{x}_i) \rangle \langle\textit{\textit{activate}(Q)}\rangle \land r_i(t) \in L\) such that \(B\) is equal to \(Q_i(\bar{x}_i/t)\).

Note that this definition actually decodes the information expressed by \((\Box,\lambda)\) into an answer to \(Q\).

Likewise, we define the notion of an answer \textit{unrestrictedly \textit{WME}-computed} when the refutation is unrestricted. However, a logic programming system need never consider such notion since it will matter only to the intermediate lemmas leading to the Completeness Theorem of Section 4.

3.3 An Example

Consider again the program \textit{DIC} and the query \textit{DEP} introduced in section 2.2. An activated \textit{WME}-refutation from the set of chains in the activation of \textit{DIC} and
DEP is:

1. (program(a,forran), λ) . activation of DIC
2. (program(b,pascal), λ) . activation of DIC
3. (program(c,fortran) program(c,pascal), λ) . activation of DIC
4. (calls(a,b), λ) . activation of DIC
5. (calls(b,c), λ) . activation of DIC
6. (¬ calls(x,y) depends(x,y), λ) . activation of DIC
7. (¬ calls(x,z) ¬depends(x,y) depends(x,y), λ) . activation of DIC
8. (¬ depends(a,u) ¬program(u,fortran), <r₁(u)>). activation of DEP
9. (¬ depends(a,v) ¬program(v,pascal), <r₂(v)>). activation of DEP
10. (¬ calls(a,z) ¬depends(z,y) [¬ depends(a,y)]

    ¬program(y,pascal), <r₂(y)>). full ext. of 9 by 7
11. (¬ depends(b,y) [¬ depends(a,y)]

    ¬program(y,pascal), <r₂(y)>). full ext. of 10 by 4
12. (¬ calls(b,y) [¬ depends(b,y)] [¬ depends(a,y)]

    ¬program(y,pascal), <r₂(y)>). full ext. of 11 by 6
13. (¬ program(c,pascal), <r₂(c)>). full ext. of 12 by 5
14. (program(c,fortran)

    [¬ program(c,pascal)], <r₂(c)>). full ext. of 13 by 3
15. (¬ depends(a,c) [program(c,fortran)]

    [¬ program(c,pascal)], <r₂(c),r₁(c)>). full ext. of 14 by 8
16. (¬ calls(a,z) ¬depends(z,c) [¬ depends(a,c)] [program(c,fortran)]

    [¬ program(c,pascal)], <r₂(c),r₁(c)>). full ext. of 15 by 7
17. (¬ depends(b,c) [¬ depends(a,c)] [program(c,fortran)]

    [¬ program(c,pascal)], <r₂(c),r₁(c)>). full ext. of 16 by 4
18. (¬ call((b,c) [¬ depends(b,c)] [¬ depends(a,c)] [program(c,fortran)]

    [¬ program(c,pascal)], <r₂(c),r₁(c)>). full ext. of 17 by 6
19. (□, <r₂(c),r₁(c)>). full ext. of 18 by 5

Hence, the formula

\[
depends(a,c) \land program(c,fortran) \lor \\
\land program(a,fortran)
\]

is a WME-computed answer to DEP over DIC since \( r₁(c) \) and \( r₂(c) \) in (12) indicates that the variable \( u \) of the chain in (8) was substituted by \( c \) as well as the variable \( v \) of the chain in (9).

4. Main Results

The WME method, modified as described in section 3.2, is sound and complete for computing answers in the sense that, given any program \( P \) and any query \( Q \), every WME-computed answer to \( Q \) over \( P \) is correct and, given any correct answer to \( Q \) over \( P \), there is a WME-computed answer which is more general. This section contains a proof of the main theorems. (For the complete proofs see [1]).

We begin with an auxiliary lemma that relates computed answers with the
substitutions performed in the refutation. If $\bar{x}$ is a tuple of variables of the form $(x_1, \ldots, x_n)$ and $\bar{\theta}$ is a substitution, we will use $\bar{x}(\bar{\theta})$ to denote the tuple of terms $(x_1\bar{\theta}, \ldots, x_n\bar{\theta})$.

**Definition 1:**

Let $P$ be a program, $Q$ be a query and $R$ be a (restricted or unrestricted) activated WME-deduction from $activate(P) \cup activate(Q)$ with length $n$. The *answer index set*, $s(R, Q)$, for $R$ and $Q$ is the subset of $[0, n]$ such that:

(i) $0 \in s(R, Q)$ iff the initial chain of $R$ belongs to $activate(Q)$;

(ii) $p \in s(R, Q)$, with $p \neq 0$, iff the $p^{th}$ derived chain of $R$ was obtained by full (restricted or unrestricted) extension with an activated chain in $activate(Q)$ as auxiliary chain.

**Lemma 1:** (Lemma of Computed Answer)

Let $P$ be a program and $Q$ be a query of the form $Q_1 \vee \ldots \vee Q_n$.

(a) **False** is a computed answer to $Q$ over $P$ iff $P$ is unsatisfiable.

(b) $A_1 \vee \ldots \vee A_k$ is the answer to $Q$ over $P$ computed by a (restricted or unrestricted) activated WME-refutation $R$ from $activate(P) \cup activate(Q)$ with $n$ derived chains and sequence of unifiers $\bar{\theta}_1, \ldots, \bar{\theta}_n$ iff there is a function $F$ from $s(R, Q)$ into $[1, k]$ and a function $G$ from $s(R, Q)$ into $[1, h]$ such that:

(i) $F$ is bijective;

(ii) if $0 \in s(R, Q)$ then $(\sim Q_{G(0)}, \langle r_{G(0)}(\bar{x}_{G(0)}) \rangle)$ is the initial chain of $R$ and $A_{F(0)}$ is equal to $Q_{G(0)}(\bar{\theta}_1, \ldots, \bar{\theta}_n)$;

(iii) For each $p \in s(R, Q)$, with $p \neq 0$, there is a renaming $\delta$ of the variables of $Q_{G(p)}$ such that the $p^{th}$ derived chain of $R$ was obtained by full (restricted or unrestricted) extension, with $(\sim Q_{G(p)}, \langle r_{G(p)}(\bar{x}_{G(p)}) \rangle)$ as auxiliary chain and $\delta$ as renaming of $\sim Q_{G(p)}$, and $A_{F(p)}$ is equal to $Q_{G(p)}(\delta \bar{\theta}_1, \ldots, \delta \bar{\theta}_n)$.

**Theorem 1:** (Soundness Theorem)

Let $P$ be a program and $Q$ be a query. If $A$ is a WME-computed answer to $Q$ over $P$ then $A$ is a correct answer to $Q$ over $P$.

**Proof**

Let $P$ be a program, $Q$ be a query and $A$ be a WME-computed answer to $Q$ over $P$. Suppose that $Q$ is of the form $Q_1 \vee \ldots \vee Q_m$. Then, by definition, there is an activated WME-refutation $R$ from $activate(P) \cup activate(Q)$ such that $A$ is the answer to $Q$ over $P$ computed by $R$.

Suppose that $A$ is **False**. Then, by the Lemma of Computed Answer, $P$ is unsatisfiable, which implies that **False** is a correct answer to $Q$ over $P$.

Suppose now that $A$ is of the form $A_1 \vee \ldots \vee A_k$. Suppose that the initial chain of $R$ is $(R_0, L_0)$, the derived chains in $R$ are $(R_1, L_1), \ldots, (R_n, L_n)$ and the sequence of
m.g.u's used in \( R \) is \( \theta_1, \ldots, \theta_n \).

Let \( C = \{ \sim A_1 \varphi, \ldots, \sim A_k \varphi \} \) be a clausal representation of \( \neg \forall A \), where \( \varphi \) is a substitution of the variables occurring in \( A \) by distinct constants that do not occur in \( P \) or \( Q \). We shall show that there is a free unrestricted WME-refutation from \( P' \cup C \), where \( P' \) is the set of chains or instances of chains in \( P \). Hence, we have that \( P \cup C \) is unsatisfiable, by an easy adaptation of the Soundness Theorem for the basic weak model elimination method [9], which, by definition of \( C \), is equivalent to saying that \( P \) logically implies \( \forall A \). Therefore, we establish that \( A \) is a correct answer to \( Q \) over \( P \).

Indeed, construct a sequence of chains \( R' = (S_0, \ldots, S_p, R'_{0}, \ldots, R'_{n}) \) such that:

- for each \( i \in \{0, n\} \), \( R'_{i} = R_{0}\theta_{i+1}\ldots\theta_{n}\varphi \);
- for each \( i \in \{1, n\} \), if \( (R_{i}, L_{i}) \) was obtained in \( R \) using full extension with auxiliary chain \( (C, M) \), then there must be \( j \in \{0, p\} \) such that \( S_{j} = C\delta_{0}\ldots\theta_{n}\varphi \), where \( C \) was the auxiliary chain and \( \delta \) the renaming of \( C \) used in the full extension. These are the only chains in \( S_0, \ldots, S_p \).

We shall now prove that \( R' \) is a free unrestricted WME-refutation from \( P' \cup C \).

We first prove that the prefix of \( R' \), which is \( S_0, \ldots, S_p, R'_{0} \), is a sequence of chains in \( P' \cup C \). Indeed, if \( (R_{0}, L_{0}) \in \text{activate}(P) \) then \( R'_{0} \in P' \). Otherwise, \( (R_{0}, L_{0}) \in \text{activate}(Q) \) and, by the Lemma of Computed Answer, we have that \( \sim R_{0}\theta_{1}\ldots\theta_{n} \) is a conjunction of \( A \). Hence, \( R'_{0} = R_{0}\theta_{1}\ldots\theta_{n}\varphi \in C \), by definition of \( C \). Likewise, by construction of \( S_{j} \), we can show that \( S_{j} \in P' \cup C \), for each \( j \in \{0, p\} \).

We now prove, by induction on \( i \in \{1, n\} \), that \( R'_{i} \) can be obtained from \( R'_{i-1} \) by free unrestricted full extension or free unrestricted full reduction.

**basis:** Suppose \( i = 1 \). Then, \( R_1 \) must necessarily be obtained by full extension from \( R_0 \) in \( R \). Let \( C \) be the auxiliary chain, \( \delta \) the renaming of \( C \) and \( \theta_1 \) the m.g.u. used in the extension. Let \( L \) be the literal selected from \( C \), let \( B \) be the chain \( C\delta \) without the literal \( L\delta \), let \( M \) be the first literal of \( R_0 \) and let \( T_0 \) be the chain \( R_0 \) with \( M \) transformed into a resolved literal. Then, by definition of full extension, we have that:

\[
R_1 = \text{contraction of } BT_0\theta_1
\]

Since \( \theta_1 \) unifies \( \{ |L\delta|, |M| \} \), the empty substitution \( \varepsilon \) is a unifier of \( \{ |L\delta\theta_1\ldots\theta_n\varphi|, |M\theta_1\ldots\theta_n\varphi| \} \). Hence, we can apply the free unrestricted extension rule to \( R'_{0} = R_{0}\theta_{1}\ldots\theta_{n}\varphi \) and \( C\delta\theta_1\ldots\theta_n\varphi \), with \( L\delta\theta_1\ldots\theta_n\varphi \) as selected literal, \( \varepsilon \) as the unifier and no renaming of the variables of \( C\delta\theta_1\ldots\theta_n\varphi \). The result will be \( B\theta_1\theta_2\ldots\theta_n\varphi \in T_0 \), \( \theta_1\theta_2\ldots\theta_n\varphi \varepsilon = BT_0\theta_1\theta_2\ldots\theta_n\varphi \), whose contraction, by (1), is exactly \( R_1 = R_{1}\theta_2\ldots\theta_n\varphi \).

**induction:** Let \( i > 1 \). Suppose that \( R'_{i} \) can be obtained from \( R'_{i-1} \) by free unrestricted full extension or free unrestricted full reduction, for each \( j \in \{1, i\} \). We shall prove that \( R'_{i} \) can also be so obtained.
case 1: \((R_i, L_i)\) was obtained by full extension in \(R\). This case follows exactly as in the basis step.

case 2: \((R_i, L_i)\) was obtained by full reduction in \(R\). Let \([L]\) be the resolved literal of \(R_{i-1}\) that was selected, \(M\) be the first literal of \(R_{i-1}\), \(\theta_i\) the m.g.u. used and \(B\) the chain \(R_{i-1}\) without \(M\). By definition of full reduction, we then have that:

\[
(2) \quad R_i = \text{the contraction of } B\theta_i
\]

Since \(\theta_i\) unifies \([L_i, M]\), the identity substitution \(\varepsilon\) unifies \([L_i, \theta_i, \ldots, \theta_n, \varphi]\), \([M, \theta_i, \ldots, \theta_n, \varphi]\). Hence, we can apply the free unrestricted reduction rule to \(R'_{i-1} = R_{i-1}\theta_i, \ldots, \theta_n, \varphi\), with \([L_i, \theta_i, \ldots, \theta_n, \varphi]\) as selected resolved literal and \(\varepsilon\) as the unifier, and the result will be \(B\theta_i, \ldots, \theta_n, \varphi\), whose contraction, by (2), is exactly \(R'_{i} = R_i\theta_{i+1}, \ldots, \theta_n, \varphi\).

This concludes the induction and, hence, the proof.

The next lemmas are technical. The first one indicates how to move from an unrestricted WMF-refutation to an WME-refutation (recall that, in an unrestricted WMF-refutation, the unifiers used do not have to be most general). Define the \textit{unrestriction degree} of a WME-refutation \(R\) as \((n-k)\) iff \(R\) has \(n\) derived chains and \(k \in [1, n]\) is the largest integer such that all derived chains in \(R\) up to, and including, the \(k\)'th derived chain are obtained by restricted derivation.

**Lemma 2:** (MGU Lemma)

Let \(S\) be a set of elementary chains. Let \(R\) be an unrestricted WME-refutation from variants of chains in \(S\) and suppose that:

- \(R_0\) is the initial chain of \(R\);
- the derived chains of \(R\) are \(R_1, \ldots, R_n\), where \(R_{i_1}, \ldots, R_{i_p}\) are the chains obtained by extension and \(r < s\) implies \(i_r < i_s\);
- the unifiers used in \(R\) are \(\theta_1, \ldots, \theta_n\);
- for each \(j \in [1, p]\), \(R_j\) was obtained by full (restricted or unrestricted) extension using the empty substitution, \(\varepsilon\), as renaming substitution and a variant \(A_j\) of a chain in \(S\) as auxiliary chain, where \(A_j\) has variables distinct from those of \(R_0\) and \(A_i\) for each \(q \in [j, j-1]\);
- the unrestriction degree of \(R\) is \(n-k+1\).

Then, there is a (restricted) WME-refutation \(R'\) from the same variants of chains in \(S\), with length \(n\), derived chains \(R'_{i_1}, \ldots, R'_{i_p}\) and m.g.u.'s \(\theta'_{i_1}, \ldots, \theta'_{i_p}\), such that:

- \(R'\) is equal to \(R\) up to, and including, the \((k-1)\)'th derived chain;
- \(\theta'_{i_j} = \theta_j\) for each \(i \in [1, k-1]\);
- the chains obtained by extension are \(R'_{i_1}, \ldots, R'_{i_p}\), using \(A_{i_1}, \ldots, A_{i_p}\) as auxiliary chains, respectively, and \(\varepsilon\) as renaming substitution;
• there is a substitution γ such that:
\[ \theta_k \ldots \theta_n = \theta'_k \ldots \theta'_n \gamma \]
\[ A_{ij} \theta_{ij} \ldots \theta_n = A_{ij} \theta'_{ij} \ldots \theta'_n \gamma, \text{ for each } j \in [1,p] \]

Lemma 3: (Lifting Lemma)

Let S be a set of elementary chains. Let R be a WME-refutation from instances of chains in S. Suppose that:

• R, the initial chain of R, is an instance of the form \( A_0 \beta_0 \), where \( A_0 \) is a chain in S and \( \beta_0 \) is a substitution over some variables in \( A_0 \);
• the derived chains of R are \( R_1, \ldots, R_n \), where \( R_i = R_i, \ldots, R_{ip} \) are obtained by extension and \( r < s \) implies \( i_r < i_s \);
• the unifiers used in R are \( \theta_1, \ldots, \theta_n \);
• for each \( j \in [1,p] \), \( \theta_{ij} \) was obtained by full (restricted or unrestricted) extension using the empty substitution, \( \varepsilon \), as renaming substitution and a chain of the form \( A_{ij} \beta_{ij} \) as auxiliary chain, where \( A_{ij} \) is a variant of a chain in S and \( \beta_{ij} \) is a substitution over some variables in \( A_{ij} \);
• for every \( j \in [1,j-1] \), \( A_{ij} \) and \( A_{ij} \beta_{ij} \) have variables distinct from \( A_{i+1}q \), \( A_{i+1} \theta_{ij} \), \( A_0 \) and \( A_0 \beta_0 \).

Then, there is a WME-refutation \( R' \) from the same variants of chains in S, with the same length as R, initial chain \( A_0 \), derived chains \( R'_1, \ldots, R'_n \) and m.g.u.'s \( \theta'_1, \ldots, \theta'_n \), such that:

• the chains obtained by extension are \( R'_1, \ldots, R'_p \), using \( A_{i+1}, \ldots, A_{ip} \) as auxiliary chains, respectively, and \( \varepsilon \) as renaming substitution;
• there is a substitution γ such that:
\[ A_0 \beta_0 \beta_i \theta_{ij} \ldots \theta_n = A_0 \theta'_1 \ldots \theta'_n \gamma \]
\[ A_{ij} \beta_{ij} \theta_{ij} \ldots \theta_{ij+1} \theta_{ij+1} \ldots \theta_n = A_{ij} \theta'_{ij} \ldots \theta'_{ij+1} \ldots \theta'_n \gamma, \]
for all \( j \in [1,p] \).

Lemma 4: (Identity Lemma)

Let P be a program and Q be a query of the form \( Q_1 \lor \ldots \lor Q_n \). Suppose that P is satisfiable and that P logically implies \( \forall Q \). Then, there is a WME-refutation R from \( activate(P) \cup activate(Q) \), possibly non-restricted, with length n, that computes an answer to Q over P which is of the form \( Q_1 \lor \ldots \lor Q_p \), where \( \langle \sim Q_i, \langle r, i \rangle \rangle \), for \( j \in [1,p] \), are exactly the activated chains in \( activate(Q) \) used in R. Moreover, all unifiers used in R are m.g.u.'s and all renamings used in extensions that have auxiliary chains in \( activate(Q) \) are the empty substitution \( \varepsilon \).
Theorem 2: (Completeness Theorem)

Let \( P \) be a program, \( Q \) be a query and \( A \) be a correct answer to \( Q \) over \( P \). Then, there is a WME-computed answer \( B \) to \( Q \) over \( P \) such that \( B \) is more general than \( A \).

Proof

Let \( P \) be a program, \( Q \) be a query and \( A \) be a correct answer to \( Q \) over \( P \). Suppose that \( Q \) is of the form \( Q_1 \lor \ldots \lor Q_n \).

Case 1: Suppose that \( P \) is unsatisfiable.

Then, by the Lemma of Computed Answer, False is a computed answer to \( Q \) over \( P \), which is the most general answer to \( Q \) in this case.

Case 2: Suppose now that \( P \) is satisfiable.

Part I: Use of the definition of answer.

Since \( A \) is a correct answer and \( P \) is satisfiable, \( A \) cannot be False. So, suppose that \( A \) is of the form \( A_1 \lor \ldots \lor A_m \). Hence, by definition of answer, for each \( u \in [1,m] \), there is a conjunction \( Q_{i_u} \) of \( Q \) and a substitution \( \beta_{i_u} \) over the variables of \( Q_{i_u} \) such that:

\[
A_u = Q_{i_u} \beta_{i_u}
\]

Part II: Use of the Identity Lemma

By definition of correct answer, \( P \) logically implies \( \forall A \). Now, note that \( A \) is also a query. Then, since \( P \) is satisfiable, by the Identity Lemma, there is a WMP-refutation \( R \) from activate(\( P \)) \cup activate(\( A \)), possibly non-restricted, whose computed answer, \( A^* \), has a specific format we explicitate in this part of the proof.

Ignoring for the moment the answer literals, suppose about \( R \) that:

S1. \( R_1, \ldots, R_n \) are the derived chains;
S2. \( 0_1, \ldots, 0_n \) are the m.g.u.'s used;
S3. \( R_{k_1}, \ldots, R_{k_t} \) are the chains obtained by full extension having as auxiliary chains \( E_{1, \ldots, E_t} \) and as renaming substitutions \( \rho_1, \ldots, \rho_t \);
S4. among these chains, \( R_{k_1}, \ldots, R_{k_p} \) have auxiliary chains of the form \( \sim A_{i_1}, \ldots, \sim A_{i_p} \), that is, originating from conjunctions of \( A \) (note that \( E_{s_r} = \sim A_{j_r} \), for all \( r \in [1,p] \));
S5. the initial chain of \( R \) is a chain \( R_0 \) of the form \( \sim A_{j_0} \), where \( A_{j_0} \) is a conjunction of \( A \).

The last assumption is included to cover the most complex case only.

We can now explicitate the exact format of the answer \( A^* \) computed by \( R \).
Indeed, by the previous assumptions and the Identity Lemma, we can assume that $A^\pi$ is of the form:

(2) $A^\pi = A^\pi v_1 \ldots v A^\pi p + 1$ with $A^\pi_r = A^\pi_{r-1}$, for each $r \in [1, p + 1]$

Moreover, the Identity Lemma also tells us that:

(3) $\rho_{s_r} = \varepsilon$, for each $r \in [1, p]$

Part III: Use of the Lemma of Computed Answer

Let $F$ and $G$ be the functions constructed in the Lemma of Computed Answer for $R$, the query $A$ and the answer $A^\pi$. Recall that $F$ maps each index of a derived chain of $R$ obtained by extension using as auxiliary chain an activated chain from $activate(A)$ into the index of a disjunct of the answer $A^\pi$ and $G$ maps each such index into the index of the disjunct of the query $A$ that generated the auxiliary chain. More precisely, using the notation of the Lemma of Computed Answer, we have:

(4) $s(R, A) = \{0, k_{s_1}, \ldots, k_{s_p}\}$

(5a) $F: s(R, A) \rightarrow [1, p + 1]$, with

(5b) $F(0) = 1$

(5c) $F(k_{s_r}) = r + 1$, for each $r \in [1, p]$, by (2) and S5

(5d) $G: s(R, A) \rightarrow [1, m]$, with

(6a) $G(0) = 1$

(6b) $G(k_{s_r}) = 1$, for each $r \in [1, p]$, by (2) and S4

By the Lemma of Computed Answer, we then have that:

(7) $A^\pi F(0) = A^\pi G(0) \theta_1 \ldots \theta_n$

and

(8) $A^\pi F(k_{s_r}) = A^\pi G(k_{s_r}) \rho_{s_r} \theta_1 \ldots \theta_n$, for each $r \in [1, p]$

Hence, we have that (by (2), (5b), (7) and (6b)):

(9) $A_{i_0} = A^\pi 1 = A^\pi F(0) = A^\pi G(0) \theta_1 \ldots \theta_n = A_{i_0} \theta_1 \ldots \theta_n$

and, for each $r \in [1, p]$, that (by (2), (5c), (8), (6c) and (3)):

(10) $A_{i_r} = A^\pi r + 1 = A^\pi F(k_{s_r}) = A^\pi G(k_{s_r}) \rho_{s_r} \theta_1 \ldots \theta_n = A_{i_r} \theta_{k_{s_r}} \ldots \theta_n$

Part IV: Use of the Lifting Lemma

We will apply the Lifting Lemma to a refutation obtained from $R$, again ignoring answer literals.

First observe that, by S3, $R$ can be viewed as a refutation from the set $E = \{R_0\} \cup \{E_u \rho_u / u \in [1, t]\}$, with $\varepsilon$ used as renaming substitution in all applications of the full extension rule. Thus, it is easy to satisfy one of the requirements of the Lifting Lemma. However, $E$ does not satisfy the last condition of the Lifting Lemma. To avoid this problem, we have to transform $R$ as follows.

For each $r \in [1, p]$ and each $u \in [1, t]$, let $\delta_r$ and $\varphi_u$ be renamings of all variables of
$Q_{ij_r}$ and $E_u \rho_u$, respectively, such that the variables in all $Q_{ij_r} \delta_r$ and $E_u \rho_u \varphi_u$ are all distinct and do not occur in any chain used in $R$ or in $Q_{ij_r}$ or in $E_u \rho_u$ or in $Q_{ij_0}$. Let $\delta^{-1}_r$ be the inverse of $\delta_r$ and $\varphi^{-1}_s$ be the inverse of $\varphi_s$.

Define, for each $r \in [1, p]$:

11a) $\lambda_{sr}$ is the substitution $\delta^{-1}_r \circ \beta_{ij_r} \circ \varphi_{sr}$

restricted to the variables in the domain of $\delta^{-1}_r$

11b) $D_s = \sim Q_{ij_r} \delta_r$

and, for each $u \in [1, t]$ such that $u \not\in \{s_1, ..., s_p\}$:

12a) $\lambda_u = \varepsilon$

12b) $D_u = E_u \rho_u \varphi_u$

Also define:

13a) $\lambda_0 = \beta_{i0}$

13b) $D_0 = \sim Q_{ij_0}$

We show that $R$ can be transformed into a refutation $R^*$ from

$\{D_0 \lambda_0, D_1 \lambda_1, ..., D_t \lambda_t\}$

such that:

- $R_1, ..., R_n$ are the derived chains;
- the unifiers are $\theta_v$, for each $v \in [1, n]$ and $v \not\in \{k_1, ..., k_t\}$, and $\varphi^{-1}_u \theta_{k_u}$, for each $u \in [1, t]$;
- $R_{k_1}, ..., R_{k_t}$ are the chains obtained by full extension having as auxiliary chains
  $D_1 \lambda_1, ..., D_t \lambda_t$ and as renaming substitution $\varepsilon$;
- the initial chain is $D_0 \lambda_0$.

Indeed, the initial chain of $R$ can be rewritten as:

14) $\sim A_{i0} = \sim Q_{ij_0} \beta_{i0} = D_0 \lambda_0$

by SS, (1), (13a) and (13b)

Now, for each $r \in [1, p]$, we have that:

15a) $Q_{ij_r} \delta_r \lambda_{sr} = Q_{ij_r} \delta_r (\delta^{-1}_r \circ \beta_{ij_r} \circ \varphi_{sr}) = Q_{ij_r} \beta_{ij_r} \varphi_{sr}$

This follows because, if $x/t$ is a simple substitution in $\delta^{-1}_r \circ \beta_{ij_r} \circ \varphi_{sr}$, but not in $\lambda_{sr}$, then $x$ does not occur in $Q_{ij_r} \delta_r$. Indeed, by definition of $\lambda_{sr}$ and the composition of substitutions, there are two cases to consider: $x/t$ must be a simple substitution of $\beta_{ij_r}$ or a simple substitution of $\varphi_{sr}$. In the first case, $x$ must occur in $Q_{ij_r}$, by definition of $\beta_{ij_r}$, and, in the second case, $x$ must occur in $E_{sr} \rho_{sr}$, by definition of $\varphi_{sr}$. However, in both cases, $x$ does not occur in $Q_{ij_r} \delta_r$, by the requirement that all variables of $Q_{ij_r} \delta_r$ be distinct from the variables of $Q_{ij_r}$.
and the variables of $E_s, p_s$.

Using (15a), we may then obtain, for each $r \in [1,p]$ (by (11a), (15a), (1), (3), S3 and S4):

(15b) $D_s \lambda_s = \sim Q_{ir} \delta_r \lambda_s = \sim Q_{ir} \beta_{ir} \varphi_{sr} = \sim A_{ij} \varphi_s = \sim A_{ij} p_s \varphi_{sr}$

$= E_s p_s, \varphi_{sr}$

and, for each $u \in [1,t]$ such that $u \notin \{s_1, \ldots, s_r\}$:

(16) $D_u \lambda_u = E_u p_u \varphi_u$ by (12a) and (12b)

Furthermore, by the choice of $\varphi_u$, its inverse $\varphi^{-1}_u$ does not affect any variable occurring in $R_w$, for any $w \in [0,n]$. Therefore, we have that:

(17) $R_0 \varphi^{-1}_1 = R_0$

(18) $R_{k_u} \varphi^{-1}_u = R_{k_u-1}$, for each $u \in [1,t]$

Then, by (15b), (16), (17) and (18), and by S2 and S3, for each $u \in [1,t]$, $R_{k_u}$ can still be obtained by extending $R_{k_u-1}$ with $D_u \lambda_u$ as the auxiliary chain, $\varphi^{-1}_u \theta_{k_u}$ as unifier and $\varepsilon$ as renaming substitution.

Note now that, for every $u \in [1,t]$, for every $q \in [1,u-1]$, $D_u$ and $D_u \lambda_u$ have variables distinct from those of $D_q, D_q \lambda_q, D_0$ and $D_0 \lambda_0$.

Therefore, for $R'$ the conditions of the Lifting Lemma hold, recalling that $R'$ is from the set $\{D_0 \lambda_0, D_1 \lambda_1, \ldots, D_t \lambda_t\}$.

Hence, there is a WME-refutation $R'$ from $\{D_0, D_1, \ldots, D_t\}$ such that the initial chain is $D_0 = \sim Q_{ij}$, and if the derived chains are $R'_1, \ldots, R'_n$ and the m.g.u.'s are $\theta'_1, \ldots, \theta'_n$, then $R'_1, \ldots, R'_n$ are the chains obtained by extension and the auxiliary chains are $D_1, \ldots, D_t$. Moreover, recalling that $D_s = \sim Q_{ij} \delta_r$, for each $r \in [1,p]$, there is a substitution $\gamma$ such that

(19) $Q_{ij} \beta_{ij} \lambda_{ij} \varphi^{i}_1 \theta_{ij} \lambda_{ij} \varphi^{i}_1 \theta_{ij} \ldots \theta_{ij} \gamma = Q_{ij} \theta'_{ij} \ldots \theta'_{ij} \gamma$ by (13a)

(20) $Q_{ij} \delta_r \lambda_s \varphi_s \theta_{sr} \lambda_s \varphi_s \theta_{sr} \ldots \theta_{sr} \gamma = Q_{ij} \delta_r \theta'_{sr} \ldots \theta'_{sr} \gamma$, for $r \in [1,p]$

Part V: Conclusion

Define $B$ as the disjunct $B_0 \varphi \ldots \varphi \varphi_p$ such that:

(21) $B_0 = Q_{ij} \theta'_{ij} \ldots \theta'_{ij}$

(22) $B_r = Q_{ij} \delta_r \theta'_{sr} \ldots \theta'_{sr}$, for each $r \in [1,p]$

Then, by the Lemma of Computed Answer, $B$ is the answer computed by $R'$, when activated to compute answers to $Q$ over $P$. Moreover, note that $R'$ is from the set $\{D_0, D_1, \ldots, D_t\}$, which is a set of chains that are variants of the chains in $P$ or in $\{\sim Q_1, \ldots, \sim Q_t\}$. Hence, when activated, $R'$ can be viewed as an activated WME-refutation from $\text{activate}(P) \cup \text{activate}(Q)$. 
We shall now show that $A_{ir} = B_r \gamma$, for each $r \in [0,p]$.

To simplify the notation, for each $r \in [1,p]$, let:

$$\overline{\delta}[r] = \lambda_{s_r} \phi^{-1} s_r \delta_{k_{s_r}} ... \lambda_q \phi^{-1} q \theta_{k_q} ... \theta_n$$

(23) $$\overline{\delta}[r] = \lambda_{s_r} \phi^{-1} s_r \delta_{k_{s_r}} ... \theta_n$$

(24) $$\overline{\delta}[r] = \lambda_{s_r} \phi^{-1} s_r \delta_{k_{s_r}} ... \theta_n$$

That is, $\overline{\delta}[r]$ includes $\lambda_q$ and $\phi^{-1} q$, for all $q \in [s_r,t]$, while $\overline{\delta}[r]$ includes only $\lambda_{s_r}$ and $\phi^{-1} s_r$.

Now observe that the variables in the domain of $\lambda_q$ and $\phi^{-1} q$, for each $q \in (s_r,t)$, occur neither in $e$, for any simple substitution $x/e$ in $\theta_i$, for any $j \in [k_{s_r},k_q)$, nor in $A_{ir} = Q_{ij} \delta_i \lambda_{s_r} \phi^{-1} s_r$. From these observations one can prove that:

(25) $$Q_{ij} \delta_i \theta_i[r] = Q_{ij} \delta_i \overline{\delta}[r], \text{ for each } r \in [1,p]$$

For each $r \in [1,p]$, we may then prove that:

(26) $$B_r \gamma = Q_{ij} \delta_i \overline{\delta}_i[r] \phi^{-1} k_{s_r} ... \phi^{-1} n \gamma$$

by (22)

$$= Q_{ij} \delta_i \overline{\delta}_i[r] \phi^{-1} n$$

by (20) and (23)

$$= Q_{ij} \delta_i \overline{\delta}_i[r]$$

by (25)

$$= Q_{ij} \beta_i \theta_i \phi^{-1} k_{s_r} ... \theta_n$$

by (24) and (15a)

$$= A_{ir}$$

by (1) and (10)

Likewise, for $r = 0$, we may prove that:

(27) $$B_0 \gamma = Q_{ij} \theta_1 ... \theta_n \gamma$$

by (21)

$$= Q_{ij} \lambda_1 \phi^{-1} \theta_1 ... \lambda_q \phi^{-1} q \theta_q ... \theta_n$$

by (19)

$$= Q_{ij} \beta_0 \theta_1 ... \theta_n$$

by a step similar to (25)

$$= A_{i0}$$

by (1) and (9)

Therefore, we obtain a WME-refutation, $R'$, such that, when activated for $P$ and $Q$, computes $B = B_0 v ... v B_p$, which is such that there is a substitution $\gamma$ for which $A_{ir} = B_r \gamma$, for each $r \in [0,p]$. But this implies that $B$ is more general than $A$, which concludes the proof.

\[\square\]

5. Computing Definite Answers

This section describes another variation of weak model elimination that computes only definite answers.

Let $S$ be a set of activated elementary chains and $T$ be a subset of $S$. We say that an activated WME-refutation $R$ from $S$ has initial support from $T$ iff the initial activated chain of $R$ is in $T$ and no activated chain in $T$ is ever used as an
auxiliary chain in derivations in $R$.

Let $Q$ be a query to a program $P$. An answer $A$ to $Q$ over $P$ is WME-computed with initial support from $Q$ iff there is an activated WME-refutation $R$ from $\text{activate}(P) \cup \text{activate}(Q)$ with initial support from $\text{activate}(Q)$.

From the above definitions and the Soundness Theorem, we have that:

**Theorem 3:** (Soundness Theorem for Definite Answers)

Let $P$ be a program and $Q$ be a query. If $A$ is a WME-computed answer to $Q$ over $P$ with initial support from $Q$, then $A$ is a definite correct answer to $Q$ over $P$.

We shall now prove the corresponding version of the Completeness Theorem. The first lemma, which does not appear in the literature, uses the notion of admissible chains to obtain a result about WME-refutations.

**Lemma 5:**

Let $S$ be a set of elementary chains and $C$ be a ground elementary chain. Suppose that $S$ is satisfiable. Then, $S \cup \{C\}$ is unsatisfiable iff there is a WME-refutation from $S \cup \{C\}$ such that $C$ is the initial chain and $C$ is never used as auxiliary chain.

**Proof**

Let $S$ be a set of elementary chains and $C$ be a ground elementary chain. Suppose that $S$ is satisfiable and that $C$ is of the form $L_1...L_k$.

By the Soundness Theorem for $WME$, if there is a WME-refutation from $S \cup \{C\}$ satisfying the conditions of the lemma, then $S \cup \{C\}$ is unsatisfiable.

Conversely, suppose that $S \cup \{C\}$ is unsatisfiable. By the assumption on $S$ and $C$ and Theorem 3.6.3 in [10], there is a WME-refutation $R$ from $S \cup \{C\}$ whose initial chain is $C$. Suppose that the derived chains of $R$ are $R_1,...,R_n$ and that the m.g.u.'s are $\theta_1,...,\theta_n$.

First note that $C$ cannot be a tautology since otherwise $S$ being satisfiable would imply that $S \cup \{C\}$ is also satisfiable, which contradicts our assumptions. We shall use this fact, together with the fact that all derived chains are admissible, to show that $C$ is never used as auxiliary chain in $R$.

Suppose by contradiction that there is $k \in [1,n]$ such that $R_k$ was obtained by extending $R_{k-1}$ by $C$. Let $L_j$ be the literal selected from $C$.

We first prove that:

1. $R_{k-1} = M_1...M_n[L_j]L_{j+1}...L_k$

   for some elements $M_1,...,M_n$ where $M_1$ is not a literal descending from the initial chain $C$. 

Indeed, first observe that, since the initial chain is \( C \), which is ground, by definition of the inference rules of \( WME \), the rightmost elements of \( R_{k-1} \) must be the rightmost literals of \( C \) or \( R \)-literals descending from them. But \( R_{k-1} \) cannot be of the form \( L_{j+1} \ldots L_k \), for some \( j \geq 0 \), since, by the construction of \( R_k \), \( C \) would contain two complementary literals, \( L_i \) and \( L_{i+1} \), and hence would be a tautology. Then, at least the first element of \( R_{k-1} \) must not be a literal of \( C \), which implies that \( R_{k-1} \) satisfies (1).

Then, by (1), assumptions about \( R_k \) and since all literals descending from \( C \) are ground, we have:

\[
R_k = L_1 \ldots L_{i-1} L_i \ldots L_k [M_1 \phi] M_2 \phi \ldots M_m \phi [L_j] L_{j+1} \ldots L_k
\]

where \( \phi \) is a m.g.u. of \( \{L_i, M_1\} \).

Now, as \( L_i \) is ground, \( \phi \) is a m.g.u. of \( \{L_i, M_1\} \) and \( L_i \) and \( M_1 \) have opposite signs, \( M_1 \phi \) and \( L_i \) are complementary literals.

Thus, if \( i = j \) then \( R_k \) has two complementary \( R \)-literals, \( [M_1 \phi] \) and \( [L_j] = [L_i] \), and if \( i \neq j \) then \( R_k \) has a literal, \( L_j \), identical and to the left of a \( R \)-literal, \( [L_i] \). Therefore, in both cases, \( R_k \) is not admissible, which implies that \( R \) is not a valid \( WME \)-refutation. Contradiction.

The second lemma specializes the Identity Lemma stated in section 4.

**Lemma 6:** (Identity Lemma for Definite Answers)

Let \( P \) be a program and \( Q \) be a query of the form \( Q_1 \lor \ldots \lor Q_m \). Suppose that \( P \) is satisfiable and that \( P \) logically implies \( \forall Q_i \), for some \( i \in [1, m] \). Then, there is an activated \( WME \)-refutation \( R \) from \( activate(P) \cup activate(Q) \) such that:

(i) the initial chain is \( (\sim Q_i, <r_i(\overline{x_i})>) \in activate(Q) \);

(ii) no activated chain in \( activate(Q) \), including \( (\sim Q_i, <r_i(\overline{x_i})>) \), is used as an auxiliary chain in \( R \);

(iii) the answer computed by \( R \) is \( Q_i \).

Note that the above lemma states the existence of a restricted \( WME \)-refutation, rather than an unrestricted \( WME \)-refutation, as in the original Identity Lemma. This is possible because the \( WME \)-refutation uses exactly one chain in \( activate(Q) \), which is in fact its initial chain.

We may finally state the Completeness Theorem, whose proof follows as in Theorem 2, using Lemma 5 and Lemma 6.

**Theorem 4:** (Completeness Theorem for Definite Answers)

Let \( P \) be a program, \( Q \) be a query and \( A \) be a definite correct answer to \( Q \) over \( P \). Then, there is a definite answer \( B \) to \( Q \) over \( P \) such that \( B \) is \( WME \)-computed with initial support from \( Q \) and \( B \) is more general than \( A \).
6. Conclusions

Weak model elimination offers an interesting alternative for the development of logic programming systems, since it works with classes of programs and queries which are more general than those commonly considered. This paper established the theoretical foundations of such systems, proving soundness and completeness results for computed answers. It also described the modifications that are necessary to compute only definite answers.

As mentioned in the Introduction, a companion paper [6] extends the results reported here with defaults to capture non-monotonic reasoning. A logic programming systems based on these results is also described in [13].

References