# Intensional Query Answering: An Application of Partial Evaluation

Giuseppe De Giacomo
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italia
e-mail: degiacom@assi.ing.uniroma1.it

**Abstract**

We consider intensional answers to be logical formulas expressing sufficient conditions for objects to belong to the usual answer to a query addressed to a knowledge base. We show that in the SLDNF-resolution framework, complete and procedurally complete sets of intensional answers can be generated by using partial evaluation. Specific treatments of recursion and negation are also presented.

## 1 Introduction

Intensional answers are responses that provide an abstract description of the conventional answer to a query addressed to a knowledge base. They are expected to "provide compact and intuitive characterizations of sets of facts, making explicit *why* a specific set of facts answers the query instead of just *which* facts belong to the answer" ([PR89]). Various research studies have investigated this kind of answers (e.g., [CD86], [CD88], [Corella84], [Demolombe92], [DFI91], [Imielinski87], [Motro89], [Motro91], [MY90], [PR89], [PRZ91], [SM88], etc.). Following [CD86], [PR89], [PRZ91], we consider intensional answers to be logical formulas expressing *sufficient conditions* for objects to belong to the conventional answer to a query.

We assume knowledge bases to be, essentially, programs whose proof procedure is the SLDNF-resolution, as in [Lloyd87]. Partial evaluation (PE) for programs in the SLDNF-resolution framework is defined in [LS91]. Although it is usually considered an optimization technique, we use it for quite a different aim in this paper.

We show that given a program $P$ and a query $Q(X)$, a new program $P' = P \cup \{q(X) \leftarrow Q(X)\}$ (where $q$ is a predicate symbol not occuring in $P$) can be defined such that for every PE of $q(X)$ in $P'$ there corresponds a *complete set of intensional answers* to $Q(X)$ in $P$. Furthermore, each set $S_{IA}$ of intensional answers computed in this way is procedurally equivalent to the original query $Q(X)$, i.e., the conventional answers that can be computed from $S_{IA}$ in $P$ are exactly those that can be computed from $Q(X)$ in $P$.

Having pointed out this correspondence we have a tool to produce intensional answers for a very general class of queries and programs, i.e., for every

query in every program intended to run under SLDNF-resolution. Therefore, in principle, we can deal with function symbols, recursion and negation, something usually ruled out by other approaches to intensional query answering.

Specifically, we suggest a simple but quite effective way to return intensional answers when recursion is involved. Notice first of all that by PE we can obtain recursion-free intensional answers for a query involving recursive predicate symbols. On the other hand, if we cannot remove a recursive predicate symbol $p$ from an intensional answer, then we return, together with the intensional answer, an *auxiliary definition* for $p$. This is a specialized definition that is general enough to cover the meaning of $p$ in the context of the intensional answers in which it appears. Note that, if a recursive predicate symbol $p'$ other than $p$ shows up in the auxiliary definition for $p$, then we return an auxiliary definition for $p'$ as well.

The pair $< S_{IA}, AD >$, where $S_{IA}$ is a set of intensional answers to a query $Q(X)$, and $AD$ is a set of auxiliary definitions for $S_{IA}$, can be interpreted as the implicit representation of the infinite set of all the intensional answers to $Q(X)$, which can be inferred from $S_{IA}$, using the axioms corresponding to $AD$.

With regard to negation, we remind the reader that if a negative literal is found at a certain point of the PE process, then either it is completely evaluated, or the atom in the negative literal is partially evaluated and the definition obtained is added to the PE to be returned (e.g. see [BL90]).

We could follow a similar approach in the generation of intensional answers, returning auxiliary definitions for the atoms in the negative literals that cannot be evaluated. Yet, this would be quite unsatisfactory, because we would lose the "interactions" between the positive part and the negative part of an intensional answer. To avoid this problem, we make some additional logical transformations. Roughly, we consider the completions of such auxiliary definitions, negate both sides of the equivalences, perform some logical manipulations on the right sides, and replace the negative literals in the intensional answers by the proper instances of the corresponding right parts of the equivalences obtained.

The rest of the paper is organized as follows. After recalling some preliminary notions in the next section, the basic results are presented in Section 3. Our treatments of recursion and negation are described in Sections 4 and 5, respectively. Conclusions and further work end the paper. Due to lack of space only sketches of the proofs are reported. The full proofs appear in [DeGiacomo92].

# 2 Preliminaries

In this section we introduce some basic definitions, the knowledge bases considered, intensional answers and partial evaluation.

We assume that the reader is familiar with the standard theoretical results of logic programming (cf. [Lloyd87]).[1]

[1]We mainly use the same notation as [Lloyd87] except that we denote sequences of terms by a single capital letter. Few other differences are pointed out when encountered.

As usual, a *program* is a finite set of statements of the form $A \leftarrow W$, where $A$ is an atom and $W$ is a first order formula. All the statements of a program $P$, which have the same predicate symbol $p$ in their head, form the *definition* of $p$ in $P$. Statements whose bodies are conjunctions of literals are called *program clauses* or just *clauses*. A program whose statements are program clauses is called a *normal program*.

The *completion* of a program $P$, denoted as $comp(P)$, is the collection of *completed definitions* of the predicate symbols in $P$ together with Clark's equality theory.

**Definition** Let $P$ and $P'$ be two programs, $G$ and $G'$ two goals with the same free variables. We say that $P \cup \{G\}$ and $P' \cup \{G'\}$ are *procedurally equivalent* if the following holds:

1. $P \cup \{G\}$ has an SLDNF-refutation with computed answer $\theta$ iff $P' \cup \{G'\}$ does.
2. $P \cup \{G\}$ has a finitely failed SLDNF-tree iff $P' \cup \{G'\}$ does.

□

When we talk about SLDNF-resolution for non-normal programs we refer to the corresponding normal forms obtained by applying Lloyd & Topor's transformations (cf. [LT84], [Lloyd87]).

**Definition** Let $S$ be a set of predicate symbol definitions. We call $comp'(S)$ the set of the corresponding completed definitions together with Clark's equality theory. □

Notice that, given a program $P$, $comp'(P)$ is the subset of $comp(P)$ formed by the completed definitions of the predicate symbols explicitly defined in $P$ (i.e., the predicate symbols appearing in the head of a statement of $P$). To further clarify the concept let us see an example.

**Example** Consider the following program $P = \{p(x) \leftarrow r(x) \wedge s(x), r(a) \leftarrow\}$, $comp(P)$ is $\{\forall x(p(x) \leftrightarrow r(x) \wedge s(x)), \forall x(r(x) \leftrightarrow (x = a)), \forall x(\sim s(x))\}$ while $comp'(P)$ is $\{\forall x(p(x) \leftrightarrow r(x) \wedge s(x)), \forall x(r(x) \leftrightarrow (x = a))\}$. □

We consider a knowledge base $KB$ essentially constituted by a program divided in two strata $IDB$ and $EDB$.

- $IDB$ is a program such that the predicate symbols defined therein may depend upon predicate symbols defined in $EDB$. We call such a program the *intensional program* of the knowledge base $KB$.
- $EDB$ is a program such that *no* predicate symbol defined therein depends upon predicate symbols defined in $IDB$. We call such a program the *extensional program* of the knowledge base $KB$.

Typically, $IDB$ and $EDB$ are intended to model the intensional knowledge and the extensional knowledge of $KB$ respectively.[2]

We say that an intensional program $IDB$ is a *normal intensional program* if it is a normal program. In the same way, we say that an extensional program $EDB$ is a *normal extensional program* if it is a normal program.

A *query* to a knowledge base can be any first order formula[3].

We now turn our attention to intensional answers. We adopt the same definitions as in [CD86], [PR89], [PRZ91], etc, adapting them to the SLDNF-resolution framework. Let $IDB$ be the intensional program of a knowledge base $KB$, and $Q(X)$ a query whose free variables are $X$. Since in the present paper we do not use integrity constraints to generate intensional answers, we are actually considering a special kind of intensional answers defined as follows.

**Definition** A first order formula $A_i(X)$, whose free variables are $X$, is an *intensional answer* for $Q(X)$ (wrt $KB$) if

$$comp'(IDB) \models \forall X(A_i(X) \rightarrow Q(X)).$$

□

Obviously not all the intensional answers are interesting, e.g. we can drop intensional answers which are variants of the query, those inconsistent wrt $comp'(IDB)$, and those subsumed by other ones.

**Definition** A set $S_{IA}$ of intensional answers for $Q(X)$ (wrt $KB$) is *complete* if

$$comp'(IDB) \models \forall X((\bigvee_{A_i \in S_{IA}} A_i(X)) \leftrightarrow Q(X)).$$

□

Since, SLDNF-resolution is sound but not complete in general, it makes sense to introduce the notion of a set of intensional answers, *complete from the procedural point of view.*

**Definition** A set $S_{IA}$ of intensional answers for $Q(X)$ (wrt $KB$) is *procedurally complete* if for every possible extensional program $EDB$ of $KB$,

$$IDB \cup EDB \cup \{\leftarrow \bigvee_{A_i \in S_{IA}} A_i(X)\} \text{ and } IDB \cup EDB \cup \{\leftarrow Q(X)\}$$

are procedurally equivalent. □

[2]Note that, there are no restrictions on the form of the statements neither in $IDB$ nor in $EDB$.

[3]In [Lloyd87] a query is a goal. Let $\leftarrow W$ be a goal, we call "query" the first order formula $W$.

We finish the preliminary section introducing *partial evaluation (PE)*[4]. The formal notion and result described here are from [LS91]. We refer to normal programs and normal goals only. It is convenient to use slightly more general definitions of SLDNF-derivation and SLDNF-tree than those given in [Lloyd87]. In [Lloyd87], an SLDNF-derivation is either infinite, successful or failed. We also allow it to be *incomplete*, in the sense that at any step we are allowed simply not to select any literal and terminate the derivation. Likewise, in an SLDNF-tree we may neglect to unfold a goal.

**Definition** A *resultant* is a first order formula of the form $Q_1 \leftarrow Q_2$, where $Q_i$, $(i = 1, 2)$, is either absent or a conjunction of literals. Any variables in $Q_1$ or $Q_2$ are assumed to be universally quantified in front of the resultant. □

**Definition** Let $P$ be a normal program, $G$ a normal goal $\leftarrow Q$, and $G_0 = G, G_1, \ldots, G_n$ an SLDNF-derivation $P \cup \{G\}$, where the sequence of substitutions is $\theta_1, \ldots, \theta_n$ and $G_n$ is $\leftarrow Q_n$. Let $\theta$ be the restriction of $\theta_1 \ldots \theta_n$ to the variables in $G$. Then we say the derivation has *length* $n$ with *computed answer* $\theta$ and *resultant* $Q\theta \leftarrow Q_n$. [5] □

**Definition** Let $P$ be a normal program, $A$ an atom, and $T$ a (not necessarily complete) SLDNF-tree for $P \cup \{\leftarrow A\}$. Let $G_1, \ldots, G_r$ be a set of (non-root) goals in $T$ such that each non-failed branch of $T$ contains exactly one of them. Let $R_i$ $(i = 1, \ldots, r)$ be the resultant of the derivation from $\leftarrow A$ down to $G_i$ associated with the branch leading to $G_i$.

- The set of resultants $\pi = \{R_1, \ldots, R_r\}$ is a *PE of* $A$ *in* $P$. These resultants have the following form $R_i = A\theta_i \leftarrow Q_i$ $(i = 1, \ldots, r)$, where we have assumed $G_i = \leftarrow Q_i$.
- Let $\mathbf{A} = \{A_1, \ldots, A_s\}$ be a finite set of atoms, and $\pi_i$ $(i = 1, \ldots, s)$ a PE of $A_i$ in $P$. Then $\Pi = \pi_1 \cup \ldots \cup \pi s$ is a *PE of* $\mathbf{A}$ *in* $P$.
- Let $P'$ be the normal program resulting from $P$ when the definitions therein of the predicate symbols in $\mathbf{A}$ are replaced by a PE of $\mathbf{A}$ in $P$. Then $P'$ is a *PE of* $P$ *wrt* $\mathbf{A}$.

□

The next two theorems are the main results on partial evaluation.

**Definition** Let $S$ be a set of first order formulas and $\mathbf{A}$ a finite set of atoms. We say $S$ is $\mathbf{A}$-*closed* if each atom in $S$ containing a predicate symbol occurring in $\mathbf{A}$ is an instance of an atom in $\mathbf{A}$. □

[4] Recently, in the context of logic programming, it has been proposed to replace the name *partial evaluation* with the name *partial deduction*, leaving the original name to denote the optimization oriented use of such a machinery. In this paper we stick to the name *partial evaluation* in conformity with [LS91] and [BL90] whose results are extensively used.

[5] Note that, if $n = 0$, the resultant is $Q \leftarrow Q$.

**Definition** Let **A** be a finite set of atoms. We say **A** is *independent* if no pair of atoms in **A** have a common instance. □

**Theorem 1 (Lloyd Shepherdson)** *Let $P$ be a normal program, $W$ a closed first order formula,* **A** *a finite set of atoms, and $P'$ a PE of $P$ wrt* **A** *such that $P' \cup \{W\}$ is* **A***-closed. If $W$ is a logical consequence of $comp(P')$, then it is a logical consequence of $comp(P)$, i.e., $comp(P') \models W \Rightarrow comp(P) \models W$.*

**Theorem 2 (Lloyd Shepherdson)** *Let $P$ be a normal program, $G$ a normal goal,* **A** *a finite, independent set of atoms, and $P'$ a PE of $P$ wrt* **A** *such that $P' \cup \{G\}$ is* **A***-closed*[6]*. Then $P \cup \{G\}$ and $P' \cup \{G\}$ are procedurally equivalent.*

Note that the PE of a program wrt a goal is not directly defined. Anyway, there are procedures (e.g. [BL90]) that, given a program $P$ and a goal $G$, compute a set of atom **A** and a PE of the program $P$ wrt **A** such that the original program and the partially evaluated program are procedurally equivalent wrt the goal $G$.

# 3 Intensional query answering by partial evaluation

The intensional program $IDB$ of a knowledge base $KB$ is an "open program", i.e., a program for which some predicate symbol definitions are missing, hence it should be considered more as a collection of predicate symbol definitions than as a running program. It is clear that for $IDB$, the completion $comp(IDB)$ does not make sense, while $comp'(IDB)$ does.

The partial evaluation theorems seen in the previous section are not directly useful in dealing with intensional programs. Here, we give analogous theorems more suitable for such programs. First, we need the next definition reported from [BL90].

**Definition** Let $L$ be a set of predicate symbols. We say that a literal is *$L$-selectable* if its predicate symbol is in $L$. We say that an SLDNF-tree is *$L$-compatible* if the predicate symbol of each selected literal in the tree (including subsidiary refutations and trees) is in $L$. □

Let $IDB$ be a normal intensional program of a knowledge base $KB$, $L_{IDB}$ the set of predicate symbols defined in $IDB$, **A** a finite set of $L_{IDB}$-selectable atoms, and $IDB'$ a PE of $IDB$ wrt **A** obtained from a $L_{IDB}$-compatible SLDNF-tree, such that $IDB'$ is **A**-closed. The following two theorems hold.

**Theorem 3** *Let $W$ be a first order formula which is* **A***-closed. Then*

$$comp'(IDB') \models W \Rightarrow comp'(IDB) \models W.$$

[6] In this theorem, the closedness condition can be replaced by the *coveredness condition* (cf. [LS91]).

**Sketch of the proof** By Theorem 1, for every normal extensional program $EDB$: $comp(IDB' \cup EDB) \models W \Rightarrow comp(IDB \cup EDB) \models W$. Now, the thesis is proved by contradiction, showing that there exists an $EDB$, namely $EDB^* = \{A \leftarrow A$ : the predicate symbol in $A$ is not defined in $IDB$, and an instance of $A$ occurs in the body of a program clause in $IDB\}$, such that Theorem 1 would not hold. □

**Theorem 4** *Let $G$ be a normal goal which is* **A***-closed. If* **A** *is independent, then for every possible normal extensional program $EDB$ of $KB$: $IDB \cup EDB \cup \{G\}$ and $IDB' \cup EDB \cup \{G\}$ are procedurally equivalent.*

**Sketch of the proof** From the definition of PE it is obvious that $IDB' \cup EDB$ is a PE of $IDB \cup EDB$ wrt **A**. By Theorem 2 the thesis follows. □

We are now ready to describe the first results on generating intensional answers by using partial evaluation.

1) Let $\leftarrow W$ be a normal goal. We define a new predicate symbol (i.e., a predicate symbol not appearing in $P$ or $W$), as

$$q(X) \leftarrow W$$

where $X$ are the free variables occurring in $W$, and we add such a new definition to $IDB$, getting

$$IDB^q = IDB \cup \{q(X) \leftarrow W\}.$$

2) Let $L_{IDB^q}$ be the set of the predicate symbols defined in $IDB^q$. We *choose* a PE $\pi$ of $q(X)$ in $IDB^q$ obtained from an $L_{IDB^q}$-compatible SLDNF-tree for $IDB^q \cup \{\leftarrow q(X)\}$. Let $\pi$ be

$$\begin{aligned} q(X)\theta_1 &\leftarrow W_1 \\ &\vdots \\ q(X)\theta_r &\leftarrow W_r \end{aligned}$$

where $\theta_i = \{X_i/T_i\}$, $X_i$ are the variables in $X$ instantiated by $\theta_i$, and $T_i$ are terms.

3) The completed definition for $q$ given by these resultants can be written as follows:

$$\forall X(q(X) \leftrightarrow \exists Y_1((X_1 = T_1) \wedge W_1) \vee \ldots \vee \exists Y_r((X_r = T_r) \wedge W_r)) \qquad (1)$$

where $Y_i$ are the free variables in $(X_i = T_i) \wedge W_i$ other than those in $X$, and $X_i = T_i$ is a loose notation for $(x_{1i} = t_{1i}) \wedge \ldots \wedge (x_{ni} = t_{ni})$ (supposing $X_i$ to be the sequence $x_{1i} \ldots x_{ni}$).

4) The disjuncts in the above formula

$$\begin{aligned} &\exists Y_1((X_1 = T_1) \wedge W_1) \\ &\qquad\vdots \\ &\exists Y_r((X_r = T_r) \wedge W_r) \end{aligned}$$

can be regarded as *intensional answers*. Furthermore the set formed by these intensional answers is *complete* and *procedurally complete*, as the following theorems show.

**Theorem 5** *The formulas at step 4 of the process above form a complete set of intensional answers for the query $W$ in the program $P$.*

**Sketch of the proof** By Theorem 3 it can be shown that (1) is a logical consequence of $comp'(IDB^q)$. Then, considering the axiom $\forall X(q(X) \leftrightarrow W)$ in $comp'(IDB^q)$, the formula resulting from (1) replacing $q(X)$ with $W$ can be proved to be a logical consequence of $comp'(IDB)$, hence the thesis follows. □

**Theorem 6** *The set of intensional answers obtained by the process above is procedurally complete.*

**Sketch of the proof** Let $IDB^{q'}$ be the PE of $IDB^q$ wrt $\{q(X)\}$. By Theorem 4, for every possible $EDB$ of $KB$, $IDB^{q'} \cup EDB \cup \{\leftarrow q(X)\}$ is procedurally equivalent to $IDB^q \cup EDB \cup \{\leftarrow q(X)\}$, which, in turn, is procedurally equivalent to $IDB \cup EDB \cup \{\leftarrow W\}$. On the other hand, $IDB \cup EDB \cup \{\leftarrow \bigvee_{i=1}^{r} \exists Y_1((X_1 = T_1) \wedge W_1)\}$, once transformed into normal form, and assuming for the predicate symbol "=" the standard procedural meaning "unifiable", can be shown to be procedurally equivalent to $IDB^{q'} \cup EDB \cup \{\leftarrow q(X)\}$, regardless of $EDB$. Hence the thesis follows. □

**Example** Consider the following fragment of the intensional program $IDB$ of a knowledge base.

$$\begin{aligned} &publication\_bonus(x, 50) \leftarrow \\ &\qquad conference\_publication(x, y) \\ &publication\_bonus(x, 100) \leftarrow \\ &\qquad conference\_publication(x, y) \wedge major\_conference(y) \\ &publication\_bonus(x, 150) \leftarrow \\ &\qquad journal\_publication(x, y) \end{aligned}$$

$$\begin{aligned} &major\_conference(x) \leftarrow sponsor(x, ACM) \\ &major\_conference(x) \leftarrow sponsor(x, IEEE) \\ &major\_conference(x) \leftarrow accepted\_rate(x, y) \wedge (y \leq 0.2) \\ &\ldots \end{aligned}$$

Suppose we want the answer to the query

$\leftarrow \exists y(publication\_bonus(x, y) \wedge (y \geq 100))$,

i.e., "Which are the papers that get a publication-bonus greater or equal to 100?".

1) We define a new predicate symbol $q$ as

$q(x) \leftarrow publication\_bonus(x, y) \wedge (y \geq 100)$,

Let $IDB^q$ be $IDB \cup \{q(x) \leftarrow publication\_bonus(x, y) \wedge (y \geq 100)\}$.

2) We choose a PE $\pi$ of $q(x)$ in $IDB^q$ obtained from an $L_{IDB^q}$-compatible SLDNF-tree. Let such a tree be the one in Figure 1, and $\pi$ the PE associated

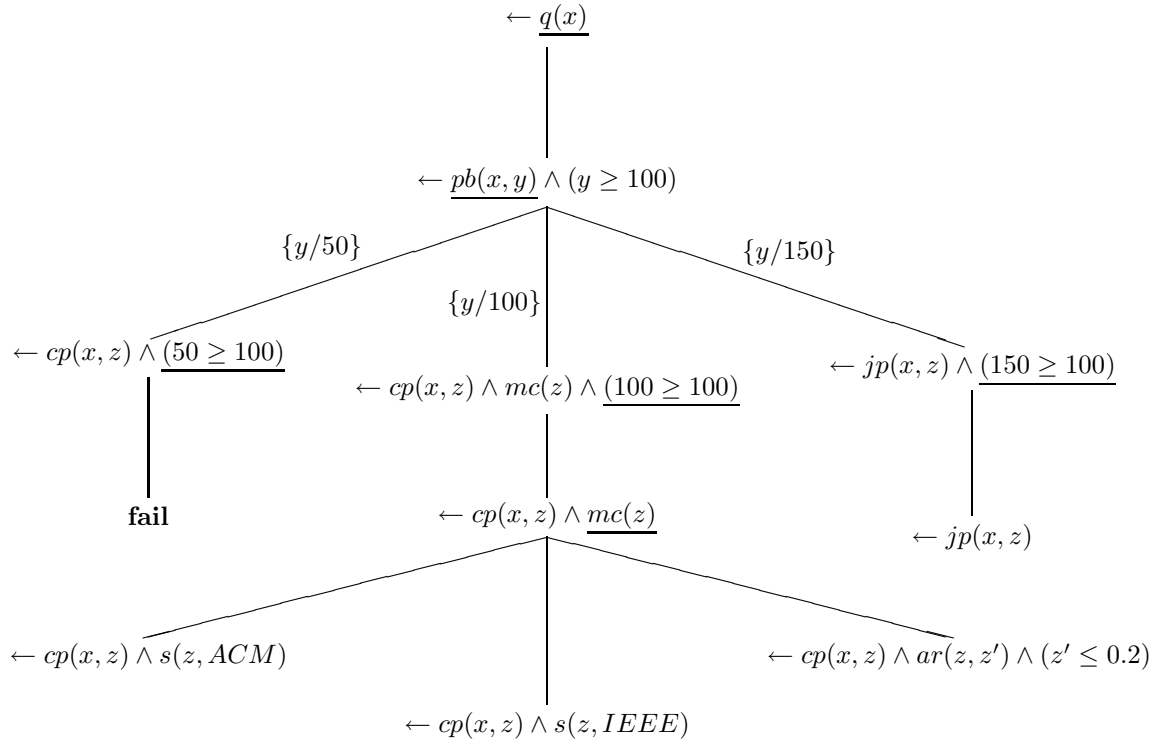


Figure 1: The SLDNF-tree used for the partial evaluation.

with the non-failing leaves of such a tree, i.e.

$q(x) \leftarrow conference\_publication(x, z) \wedge sponsor(z, ACM)$
$q(x) \leftarrow conference\_publication(x, z) \wedge sponsor(z, IEEE)$
$q(x) \leftarrow conference\_publication(x, z) \wedge accepted\_rate(x, z) \wedge (z \leq 0.2)$
$q(x) \leftarrow journal\_publication(x, z)$.

3) The completed definition of $q$ in $IDB^q$ is

$$\begin{aligned}\forall x(q(x) \quad \leftrightarrow \quad & \exists z(conference\_publication(x,z) \wedge sponsor(z,ACM)) \vee \\ & \exists z(conference\_publication(x,z) \wedge sponsor(z,IEEE)) \vee \\ & \exists z(conference\_publication(x,z) \wedge accepted\_rate(x,z) \wedge (z \leq 0.2)) \vee \\ & \exists z(journal\_publication(x,z))).\end{aligned}$$

4) The disjuncts in the right-hand part of such a formula form a complete and procedurally complete set of intensional answers, that can be read as "Papers published in an ACM conference, papers published in an IEEE conference, papers published in a conference whose accepted rate is less than or equal to 0.2, and papers published in a journal." □

The process above is *not completely specified* since we are free to choose any PE $\pi$ of $q$ in $IDB^q$ at step 2.

The *quality* of the intensional answers returned strongly depends on such a choice of $\pi$, which in turn substantially depends on the selection rule for the related SLDNF-tree. While we do not directly address such an issue in this paper, the problem of finding a "good" selection rule is one of the most crucial to effectively do intensional query answering by means of partial evaluation.

The *termination* of the above process depends again on the selection rule to be used in the generation of the PE $\pi$. Such a selection rule should build finite (incomplete) SLDNF-trees. Conditions on the selection rules, dealing with the termination of the partial evaluation, can be found in the related literature (e.g. [vanHarmelen89]).

# 4 Dealing with recursion

The basic method presented in the previous section allows one, in principle, to return intensional answers for every query in every logic program. In particular, it does not rule out recursion. Obviously, such intensional answers should be expressed in a language that is known by the user.[7] If recursive predicate symbols (i.e., predicate symbols which appear in a loop in the dependency graph of a program) are allowed to appear in the intensional program of a knowledge base, then it could be impossible to obtain a complete set of intensional answers in which no occurrences of recursive predicate symbols, that are not known by the user, appear. In this case, no satisfying set of intensional answers could be returned.

The next example shows the problem arising when recursion cannot be eliminated, and hints on how it can be tackled.

**Example** Consider the following fragment of the intensional program of a knowledge base:

[7] We assume that the user knows a set of predicate symbols which includes those defined in the extensional program of the knowledge base, and all constants and function symbols.

$collateral\_line\_relative(x, y) \leftarrow ancestor(x, z) \wedge ancestor(y, z)$

$ancestor(x, y) \leftarrow parent(x, y)$
$ancestor(x, y) \leftarrow parent(x, z) \wedge ancestor(z, y)$
...

and suppose we want intensional answers for the query:

$\leftarrow collateral\_line\_relative(x, y)$

Possible complete sets of intensional answers are

$\{\exists z(ancestor(x, z) \wedge ancestor(y, z))\}$

or

$\{\exists z(parent(x, z) \wedge ancestor(y, z)),$
$\exists z \exists z'(parent(x, z') \wedge ancestor(z', z) \wedge ancestor(y, z))\}$

or, also

$\{\exists z(parent(x, z) \wedge parent(y, z)),$
$\exists z \exists z'(parent(x, z') \wedge ancestor(z', z) \wedge parent(y, z)),$
$\exists z \exists z'(parent(x, z) \wedge parent(y, z') \wedge ancestor(z', z)),$
$\exists z \exists z' \exists z''(parent(x, z') \wedge ancestor(z', z) \wedge parent(y, z'') \wedge ancestor(z'', z))\}$

etc.

As we can see, we cannot eliminate the predicate symbol *ancestor* in the set of intensional answers returned. Now, if the meaning of *ancestor* is known by the user, then the most intuitive set of answers is probably the first one, being the simplest. But, if the meaning of *ancestor* is not known (e.g., the user may not be clear on whether or not his wife's grandfather is his ancestor), none of the above sets is satisfying, because *ancestor* appears in each of them. We need some kind of definition giving the meaning of *ancestor* in the context of the set of intensional answers returned.

For instance we may return:

$\{\exists z(ancestor(x, z) \wedge ancestor(y, z))\}$

$ancestor(x, y) \leftarrow parent(x, y)$
$ancestor(x, y) \leftarrow parent(x, z) \wedge ancestor(z, y).$

In this way, asking "which are the collateral-line relatives?" we get an answer such as " the individuals that have a common ancestor, *where* an ancestor is a parent or a parent of an ancestor". □

In view of the observations in the above example, we propose to answer a query by a set $S_{IA}$ of intensional answers and a set $RD$ of definitions for the

recursive predicate symbols which are somehow marked *unknown*[8], occuring in the answer. Notice that, if other predicate symbols marked unknown appear in such definitions, then their definitions are included in $RD$ as well.[9] To formalize the set $RD$ we now introduce the notion of a *set of auxiliary definitions*.

Let $IDB$ be the intensional program of a knowledge base $KB$, $L_{IDB}$ the set of predicate symbols defined in $IDB$, $Q(X)$ a query whose free variables are $X$, $S_{IA}$ a set of intensional answers $A_i(X)$ $(i = 1, \ldots, n)$ for $Q(X)$, $L$ a subset of $L_{IDB}$, and $AD$ a set of definitions for the predicate symbols in $L$.

Then, let $\mathbf{A}_L$ be a set of atoms, one for each predicate symbol in $L$[10], such that $S_{IA}$ is $\mathbf{A}_L$-closed, and let $comp'(AD)_{inst}$ be the instance of $comp'(AD)$ such that the atoms on the left-hand sides of the completed definitions therein coincide (modulo variants) with the corresponding atoms in $\mathbf{A}_L$.

**Definition** We say $AD$ is a *set of auxiliary definitions*[11] for $S_{IA}$ wrt $L$ if:

1. $comp'(IDB) \models comp'(AD)_{inst}$, and
2. $comp'(AD)_{inst}$ is $\mathbf{A}_L$-closed.

□

Notice that a set $AD$ of auxiliary definitions always exists. In fact, the $IDB$ definitions of the predicate symbols in $L$ form such a set. But the definitions in $AD$ are not necessarily those in $IDB$. Intuitively, they can be a "specialized" version of those which are general enough to cover the meaning of each atom occurring in the answer returned (i.e., wrt the atoms in the answer, the definitions in $AD$ retain the same meaning as those in the intensional program).

We could also require the auxiliary definitions in $AD$ to be used, instead of the corresponding definitions in $IDB$, to evaluate the intensional answers in $S_{IA}$ without losing *correct answers*, or at least *computed answers*. Such a property is quite "severe", since, to enforce it, we should return auxiliary definitions that are not only general enough to cover the meaning of the predicate symbols in $L$, in the context of $S_{IA}$ and $AD$, but also to cover their meaning through

[8] We may consider a predicate symbol to be marked unknown either generally (e.g., because its meaning is not known by the user) or more specifically, wrt the formulas in which it appears.

[9] In very unfortunate cases, the set of definitions $RD$ may almost coincide with the whole intensional program.

[10] We assume that for each predicate symbol $p$ in $L$ there corresponds just one atom, and hence we have one logical equivalence involving $p$ in $comp'(AD)_{inst}$ which may be thought of as the logical definition of $p$ in the context of $S_{IA}$ and $AD$. We could also assume that an independent set of atoms corresponds to $p$. This would entail that in $comp'(AD)_{inst}$ there would be a distinct logical equivalence involving $p$ for each such atom, therefore the idea of a single logical definition of $p$ in the context of $S_{IA}$ and $AD$ should be replaced by the idea of a logical definition of $p$ in the context of a single intensional answer of $S_{IA}$ or statement of $AD$ in which it appears. In this paper we stick to the first assumption; nevertheless the results shown here are straightforward extended to the case where the second assumption is adopted.

[11] Single auxiliary definitions are defined just as elements of $AD$.

the whole evaluation of each intensional answer in $S_{IA}$. Indeed, if a predicate symbol $p \notin L$, which depends on predicate symbol $p' \in L$, appears in some atoms of $S_{IA} \cup AD$, then in chosing the generality of the auxiliary definition for $p'$ we should consider the occurrences of $p'$ arising from the evaluation of these atoms as well.

On the other hand, the formalization of the notion of set of auxiliary definitions above is sufficient to give a nice characterization of the pair $< S_{IA}, AD >$, as shown below.

The intensional answers in $S_{IA}$ have the same status as queries, while the set $AD$ of auxiliary definitions is an (open) program. How does the pair $< S_{IA}, AD >$ relate to the original notion of intensional answers?

*The pair $< S_{IA}, AD >$ can be interpreted as the implicit representation of the infinite set of all the intensional answers for $Q(X)$ which can be inferred from the intensional answers in $S_{IA}$ using the axioms of $comp'(AD)_{inst}$.*

Indeed, the pair $< S_{IA}, AD >$ may be thought of as representing the infinite set of all the formulas $\chi_{ij}(X)$ $(i = 1, \ldots, n;\ \ j = 1, 2, \ldots)$ such that

$$comp'(AD)_{inst} \models \forall(\chi_{ij}(X) \rightarrow A_i(X)). \tag{2}$$

Note that $\chi_{ij}(X)$ $(j = 1, 2 \ldots)$ are intensional answers to $A_i(X)$ wrt the intensional program $AD$.

By definition of a set of auxiliary definitions, the following holds

$$comp'(IDB) \models comp'(AD)_{inst}. \tag{3}$$

From (2) and (3) we get

$$comp'(IDB) \models \forall(\chi_{ij}(X) \rightarrow A_i(X)). \tag{4}$$

Now, for $A_i(X)$ we have

$$comp'(IDB) \models \forall(A_i(X) \rightarrow Q(X)). \tag{5}$$

Hence, from (4) and (5)

$$comp'(IDB) \models \forall(\chi_{ij}(X) \rightarrow Q(X)), \tag{6}$$

that is, $\chi_{ij}(X)$ $(i = 1, \ldots, n;\ \ j = 1, 2, \ldots)$ are intensional answers to $Q(X)$ wrt $KB$.

Turning to the problem of how to compute a set of auxiliary definitions, assuming $IDB$ to be normal, it can be shown that a PE $\Pi$ of $\mathbf{A}_L$ in $IDB$, obtained from an $L_{IDB}$-compatible SLDNF-tree, and such that $S_{IA} \cup \Pi$ is $\mathbf{A}_L$-closed, is a set of auxiliary definitions for $S_{IA}$ wrt $L$.

When $AD$ is computed by PE, unfolding the intensional answers in $S_{IA}$ using program clauses in $AD$ leads to new sets of intensional answers $S'_{IA}$ which preserve the completeness and the procedural completeness, as the following theorem shows.

**Theorem 7** *Let $S_{IA}$ be a complete and procedurally complete set of intensional answers, and AD a set of auxiliary definitions for $S_{IA}$ wrt L obtained as a PE of $\mathbf{A}_L$. Then every set $S'_{IA}$ of intensional answers derived by SLDNF-resolution from $S_{IA}$ using program clauses in AD, is complete and procedurally complete.*

**Sketch of the proof** By the sub-derivation lemma in [LS91], and lemma 4.12 in [LS91], it follows that an SLDNF-tree built using resultants in $AD$ can be expanded into an SLDNF-tree built using only program clauses in $IDB$. Now, consider the query given by the disjunction of the intensional answers in $S_{IA}$, and let $ans(X)$ be the query introduced by its transformation into normal form. It can be shown that every $S'_{IA}$ can be computed as a PE of $ans(X)$ in the transformed intensional program. Hence, by Theorem 5 and Theorem 6, the thesis follows. □

Now that we have characterized the notion of a set of auxiliary definitions, we can employ it to clarify the idea presented at the beginning of the section.

We answer a query with a set $S_{IA}$ of intensional answers and a set $RD$ of auxiliary definitions for the recursive predicate symbols marked unknown appearing in $S_{IA}$ or in $RD$ itself.

Notice that, by Theorem 7, if an auxiliary definition $D \in RD$ of some predicate symbol $p$ is not recursive in reality, then (assuming, for now, that $p$ does not occur in a negative literal) we may unfold the corresponding positive literals in $S_{IA}$ and $RD$, and drop $D$ from $RD$.

An algorithm to compute $S_{IA}$ and $RD$, based on partial evaluation, can be adapted from the one in [BL90]. The underlying idea is to build the set of atoms that is partially evaluated "run-time" while computing $S_{IA}$ and $RD$.

# 5 More about negation

The notion of PE is directly derived from the notion of SLDNF-tree. Therefore, negation during the PE process is treated in a somewhat limited way. In fact

1. A negative literal can be selected only if it is *ground*.
2. If a ground negative literal is selected, then it is either completely evaluated (if possible), or not evaluated at all.

Similarly to what proposed in the literature on partial evaluation (e.g. [BL90]), we can generate an answer to a query $W$, constituted by a set $S_{IA}$ of intensional answers, and a set of auxiliary definitions for predicate symbols marked unknown occurring in the answer, partitioned into two subsets $RD$ and $ND$. $RD$ concerns recursive predicate symbols occuring in either positive or negative literals of the answer, whereas $ND$ concerns those non-recursive occurring in negative literals. Supposing $IDB$ and $W$ to be normal, partial evaluation can be used to generate such an answer. Actually the algorithm,

mentioned in the previous section, can easily be modified to compute $S_{IA}$, $RD$ and $ND$.

The problem with such an approach is that, in the formulas of $S_{IA}$, $RD$ and $ND$, the "interactions" (i.e., possible simplifications) between the part of information in the positive literals and the one in the negative literals is lost, because the latter is embedded in separate definitions. We need to recover such interactions if the answer is to be effective.

Now, for each predicate symbol $p$ in a negative literal there is an auxiliary definition in $ND$ to which corresponds a logical equivalence in $comp'(ND)_{inst}$ of the form:

$$\forall X(p(T(X)) \leftrightarrow \exists Y(F(X,Y))), \tag{7}$$

where $T(X)$ denotes a tuple of terms, $X$ the variables therein, and $Y$ the variables, other than those in $X$, which are free in $F$. We may negate both sides of such an equivalence getting:

$$\forall X(\sim p(T(X)) \leftrightarrow\sim \exists Y(F(X,Y))). \tag{8}$$

The literals of $S_{IA} \cup RD \cup ND$ in which $p$ occurs, must be instances of $p(T(X))$, so we may replace them with the proper instances of the right-hand side of (7) or (8). Obviously, when such an expansion of a negative literal is applied, the formulas obtained are logically equivalent to the original ones, but they may not be procedurally equivalent, hence while no correct answers are lost or gained, the same is not true for the computed answers, in general.

The idea of negating both sides of the completed definitions and replaceing the negative literals by the right-hand side of the equivalences obtained is related to *constructive negation* ([Chan88], [Chan89], [Przymusinski89]), and has been used to treat negation during the partial evaluation process in [CW89]. Here we want to apply such a treatment of negation off-line wrt the partial evaluation process, so as to retain the notions and the results in [LS91]. Moreover, our aim is to expand the negative literals in such a way as not to lose computed answers. We now present a method for such an expansion.

For every formula in $S_{IA}$ and $RD$ we recursively apply expansion steps, defined by the following sequence of transformations, until no more expansion steps are possible.

1. We substitute atoms in the positive and negative literals of the formula, with the right-hand sides of the corresponding instances of the completed definitions in $comp'(ND)_{inst}$.

2. Equalities in the formula are treated as follows.

   (a) We substitute equalities whose terms unify by the equality corresponding to their *mgu* (if the mgu is the empty substitution then the equality is eliminated), and we eliminate the conjunctions in which there is an equality whose terms do not unify.

The result of such a transformation is logically equivalent to the original formula, by Clark's Lemma (cf. [Clark78], also Lemma 15.2 in [Lloyd87]).

(b) We eliminate the equalities in which one of the terms is an existentially quantified variable, by means of the following logical equivalence: $\exists y((x = y) \wedge B) \leftrightarrow B\{y/x\}$.

3. We push the (existential) quantifiers to the right as much as possible, eliminating the redundant ones.

4. We move negation all the way inward, stopping in front of the existential quantifiers, by means of the usual logical equivalences.

A few things must be pointed out. First, a formula resulting from the above process is logically equivalent to the original one. Second, such a process always terminates, since the definitions in $ND$ are non-recursive. Third, at the end of such a process, $ND$ is not needed any more and can be eliminated. Furthermore, although we do not yet have the complete proof, it seems that a kind of procedural containment holds, that is, if $G$ is a goal, and $G'$ the goal resulting from processing $G$ as above, then for every extensional program $EDB$

1. If $IDB \cup EDB \cup \{G\}$ has an SLDNF-refutation with computed answer $\theta$, then so does $IDB \cup EDB \cup \{G'\}$.

2. If $IDB \cup EDB \cup \{G\}$ has a finitely failed SLDNF-tree, then so does $IDB \cup EDB \cup \{G'\}$.

Notice that, if the intensional program of the knowledge base is not a normal program, then by normalizing it using Lloyd & Topor's transformations to apply partial evaluation, we introduce new predicate symbols[12] that are obviously *unknown* (i.e., they are meaningless to the user). By the method sketched here, such predicate symbols can always be replaced by a meaningful formula.

**Example** Consider the following intensional program $IDB$:

$should\_visit(x, y) \leftarrow serves(y, z) \wedge likes(x, z)$

$happy(x) \leftarrow frequents(x, y) \wedge should\_visit(x, y)$

$very\_happy(x) \leftarrow \forall y(frequents(x, y) \rightarrow should\_visit(x, y))$

$unhappy(x) \leftarrow \forall y(frequents(x, y) \rightarrow\sim should\_visit(x, y))$,

the following extensional program $EDB$ (schema):

$frequents(DRINKER, PUB)$
$serves(PUB, BEER)$
$likes(DRINKER, BEER)$,

[12]New predicate symbols are introduced to eliminate the negated existentially quantified (universally quantified) formulas.

and the query "Who are the drinkers that neither are unhappy nor very happy ?", that is:

$\leftarrow\sim unhappy(x)\wedge\sim very\_happy(x).$

First notice that the last two statements must be transformed into normal form.

$very\_happy(x) \leftarrow\sim np1(x)$

$np1(x) \leftarrow frequents(x,y)\wedge\sim should\_visit(x,y)$

$unhappy(x) \leftarrow\sim np2(x)$

$np2(x) \leftarrow frequents(x,y)\wedge should\_visit(x,y).$

The only possible set of intensional answers computed by the basic method is the one constituted by the query itself. To it we may add the following set $ND$ of auxiliary definitions.

$very\_happy(x) \leftarrow\sim np1(x)$
$np1(x) \leftarrow frequents(x,y)\wedge\sim should\_visit(x,y)$
$unhappy(x) \leftarrow\sim np2(x)$
$np2(x) \leftarrow frequents(x,y)\wedge serves(y,z)\wedge likes(x,z).$

Now we proceed to the expansions. We expand (in parallel, for sake of brevity) both $\sim unhappy(x)$ and $\sim very\_happy(x)$:

$\sim unhappy(x)\wedge\sim very\_happy(x)$ (original goal)

$np2(x)\wedge np1(x)$ (first expansion step)

$\exists y(frequents(x,y)\wedge should\_visit(x,y))\wedge$
$\quad\exists y(frequents(x,y)\wedge\sim\exists z(serves(y,z)\wedge likes(x,z)))$ (second expansion step)

$\exists y(frequents(x,y)\wedge\exists z(serves(y,z)\wedge likes(x,z)))\wedge$
$\quad\exists y(frequents(x,y)\wedge\sim\exists z(serves(y,z)\wedge likes(x,z)))$ (third expansion step)

The last formula is a nice intensional answer, i.e., "The drinkers who visit at least both a pub where a beer they like is served, and a pub where no beer they like is served." □

# 6 Conclusions

In this paper we have presented a set of tools, based on PE, to generate intensional answers in the SLDNF-resolution framework, allowing function symbols, recursion, and negation.

The results stated on the application of PE techniques to the generation of intensional answers and auxiliary definitions do not refer to any particular

PE. It is engaging to investigate ways to choose PE, specific to intensional answering, such as heuristics that make the resulting intensional answers more "intuitive", or selection rules that use integrity constraints to prune away inconsistent goals.

Regardless to the PE chosen, the PE process tends to destroy the structure of the program to which it is applied. Now there are no reasons to preserve the structure of the original program. In fact, such a structure is normally hidden from the user, and is too general, in the sense that it does not reflect the particular query asked. Nevertheless, if the structure of the user's knowledge is at hand, it could be used to re-express the intensional answers in a language that is more familiar to the user. Hence, another issue to explore is the use of additional components, usually considered for modelling structural aspects of a knowledge base (e.g., taxonomies and integrity constraints), to improve the quality of the intensional answers.

Finally, our work may be considered a first step toward a program transformation approach to intensional answering, and it could be naturally extended using other program transformation techniques. Moreover such an approach can be applied to other kinds of non-conventional query answering. For instance, PE can be used for both "Knowledge query answering" [MY90] and, adding folding techniques, "Intelligent query answering" [Imielinski87].

### Acknowledgements

I am grateful to J. W. Lloyd who supervised me during the early stages of this research, and to M. Lenzerini who gave me precious advice and supported me throughout the work.

## References

[BL90] K. Benkerimi and J. W. Lloyd. A Partial Evaluation Procedure for Logic Programs. In *Proc. of North American Conf. on Logic Programming*, S. K. Derbray and M. Hermenegildo eds., pp.343-358, Austin, MIT Press, 1990.

[Clark78] K. L. Clark. Negation as Failure. In *Logic and Data Bases*, H. Gallaire and J. Minker eds., pp.293-322, Plenum Press, 1978.

[CD86] L. Cholvy. and R. Demolombe. Querying a Rule Base. In *Proc. 1st Int Conf. on Expert Database Systems*, pp.365-371, Charlesoton, South Carolina, April 1986.

[CD88] F. Cuppens and R. Demolombe. Cooperative Answering: A Methodology to Provide Intelligent Access to Databases. In *Proc. 2nd Int. Conf. on Expert Database Systems*, pp.333-353, Tysons Corner, Virginia, April 1988.

[Chan88] D. Chan. Constructive Negation Based on the Completed Database. In *Proc.of 5th International Conference and Symposium on Logic Programming*, R. A. Kowalski and K. A. Bowen eds., pp.111-125, MIT Press, 1988.

[Chan89] D. Chan. An Extension of Constructive Negation and its Application in Coroutining. In *Proc. of North American Conf. on Logic Programming*, E. Lusk and R. Overback eds., pp.477-493, MIT Press, 1989.

[Corella84] F. Corella. Semantic Retrieval and Levels of Abstraction. In *Proc. 1st Int. Workshop on Expert Database Systems*, pp.397-420, Kiawah Island, South Carolina, October 1984.

[CW89] D. Chan and M. Wallance. A Treatment of Negation During Partial Evaluation. In *Meta-Programming in Logic Programming*, H. D. Abramson and M. H. Rogers eds., pp.299-317, MIT Press, 1989. (*Proc. Meta88*).

[Demolombe92] R. Demolombe. A Strategy for the Computation of Conditional Answers. In *Proc. ECAI'92*, to appear.

[DFI91] R. Demolombe, L. Farinas del Cerro, T. Imielinski (eds.). *Proc. Workshop on Nonstandard Queries and Answers*, Toulouse, France, July, 1991.

[DeGiacomo92] G. De Giacomo. Intensional Query Answering by Partial Evaluation. *Technical Report*, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza". In preparation.

[Imielinski87] T. Imielinski. Intelligent Query Answering in Rule Based Systems. In *The Journal of Logic Programming*, 4(3):229-257, September 1987.

[Lloyd87] J. W. Lloyd. *Foundations of Logic Programming* (2nd edition). Springer-Verlag, 1987.

[LS91] J. W. Lloyd and J. C. Shepherdson. Partial Evaluation in Logic Programming. In *The Journal of Logic Programming*, 11(3&4):217-242, October/November 1991.

[LT84] J. W. Lloyd and R. W. Topor. Making Prolog More Expressive. *The Journal of Logic Programming*, 1(3):225-240, 1984.

[Motro89] A. Motro. Using Integrity Constraints to Provide Intensional Answers to Relational Queries. In *Proc. 15th Int. Conf on Very Large Data Bases*, pp.237-246, Amsterdam, August 1989.

[Motro91] A. Motro. Intensional Answers to Database Queries. *Technical Report*, Department of Information and Software Systems Engineering, George Mason University, Fairfax, Virginia, 1991.

[MY90] A. Motro and Q. Yuan. Querying Database Knowledge. In *Proc. of ACM SIGMOD-90*, pp.173-183, 1990.

[PR89] A. Pirotte and D. Roelantes. Constraints for Improving the Generation of Intensional Answers in a Deductive Database. In *Proc. 5th Int. Conf. on Data Engineering*, pp.652-659, Los Angeles, California, February 1989.

[PRZ91] A. Pirotte, D. Roelantes, E Zimanyi. Controlled Generation of Intensional Answers. In *IEEE Trans. on Knowledge and Data Engineering*, Vol 3, No.2, pp.221-236, June 1991.

[Przymusinski89] T. C. Przymusinski. On Constructive Negation in Logic Programming. In *Proc. of North American Conf. on Logic Programming*, E. Lusk and R. Overback eds., pp.1-19 (addendum), MIT Press, 1989.

[SM88] C. Shum and R. Muntz. Implicit Representation for Extensional Answers. In *Proc. 2nd Int. Conf. on Expert Database Systems*, pp.257-273, Tysons Corner, Virginia, April 1988.

[vanHarmelen89] F. van Harmelen. The Limitations of Partial Evaluation. In *Logic-Based Knowledge Representation*, P. Jackson, H. Reichgelt, F. van Harmelen eds., pp.87-111, MIT Press, 1989.