Journal of Intelligent Information Systems, 7, 205–233 (1996)

Intensional Query Answering by Partial Evaluation

GIUSEPPE DE GIACOMO degiacomo@dis.uniroma1.it

Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy

Abstract. Intensional query answering aims at providing a response to a query addressed to a knowledge base by making use of the intensional knowledge as opposed to extensional. Such a response is an abstract description of the conventional answer that can be of interest in many situations, for example it may increase the cooperativeness of the system, or it may replace the conventional answer in case access to the extensional part of the knowledge base is costly as for Mobile Systems. In this paper we present a general framework to generate intensional answers in knowledge bases adhering to the logic programming paradigm. Such a framework is based on a program transformation technique, namely Partial Evaluation, and allows for generating complete and procedurally complete (wrt SLDNF-resolution) sets of intensional answers, treating both recursion and negation conveniently.

Keywords: Knowledge bases, intensional query answering, logic programs, partial evaluation

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 it explicit *why* a specific set of facts answers the query instead of just *which* facts belong to the answer" (Pirotte and Roelantes, 1989). Formally, intensional answers are logical formulas expressing *sufficient conditions* for objects in order for them to belong to the conventional answer. Various research studies have investigated this kind of answers, e.g. (Corella, 1984, Cholvy and Demolombe, 1986, Imielinski, 1987, Cuppens and Demolombe, 1988, Shum and Muntz, 1988, Motro, 1989, Pirotte and Roelantes, 1989, Motro and Yuan, 1990, Song, Kim, and Geutner, 1990, Chu, Chen, and Lee, 1991, Chu, Lee, and Chen, 1991, Motro, 1991, Pirotte, Roelantes, and Zimanyi, 1991, Demolombe, 1992, Fonkam, 1992, De Giacomo, 1993, Kim and Kim, 1993, Motro, 1993, Demolombe and Imielinski, 1994). Most of such studies are concerned with increasing the cooperativeness of the system, however it should be mentioned that intensional answers can also be exploited as an alternative way to answer queries when the access to the extensional knowledge (typically a database) is costly, as in the case of Mobile Systems (Imielinski and Badrinath, 1993).

In this paper we assume knowledge bases to be, essentially, logic programs whose proof procedure is the SLDNF-resolution, as in (Lloyd, 1987), and we propose a method for intensional query answering based on a program transformation technique, namely Partial Evaluation (PE). PE for logic programs in the SLDNF-resolution framework is defined

in (Lloyd and Shepherdson, 1991). Although PE is usually considered an optimization technique, it turns out to be also very effective in generating intensional answers.

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 occurring 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 not permitted 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 that often using PE we obtain recursion-free intensional answers for a query involving recursive predicate symbols. In case 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 returned (e.g. see (Benkerimi and Lloyd, 1990)).

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 "interaction" between the positive part and the negative part of an intensional answer. To avoid this problem, we propose making some additional logical transformations. Roughly, given such an auxiliary definition, we consider its completion, negate both sides of the equivalence, perform some logical manipulations on the right side, and replace the corresponding negative literals in the intensional answers by the proper instances of the right part of the equivalence obtained.

The rest of the paper is organized as follows. After recalling some basic notions on logic programming, Section 2 introduces intensional answers. Section 3 introduces partial evaluation. Section 4 presents the basic techniques for intensional query answering by partial evaluation. Our treatment of recursion is described in Section 4, while the one for negation is described in Section 5. Conclusions and further work end the paper.

2. Preliminaries

In this section we introduce some basic definitions, the kind of knowledge bases considered, and intensional answers. We assume the reader's familiarity with the standard theoretical results of logic programming (cf. (Lloyd, 1987)).[1]

Definition. A *statement* is a first order formula of the form

$$A \leftarrow W$$

where A is an atom and W is a first order formula. A is the *head* of the statement and W the *body* of the statement. The body of a statement may be empty, in this case the statement is a *fact*. Statements whose body is a conjunction of literals are called *program clauses* or just *clauses*.

Definition. A *program* is a finite set of statements. A program whose statements are program clauses is called a *normal program*.

Definition. The *definition* of a predicate symbol p in a program P is the set of all statements in P which have p in their head.

Definition. A *goal* is a first order formula of the form

$$\leftarrow W$$

where W is a first order formula. Any free variable in W is assumed universally quantified in front of the goal. A goal $\leftarrow W$ such that W is conjunction of literals is called a *normal goal*.

Definition. Let P be a program. The *dependency graph* of P is a directed graph in which the nodes are the predicate symbols in P and there is a directed arc from p to q if there exists a statement s in P in which p is the predicate symbol in the head of s and q is a predicate symbol occurring in the body of s.

Definition. Let P be a program and W a first order formula. We say that W *depends* upon a predicate symbol p in P if there is a path from a predicate symbol in W to p in the dependency graph for P.

The declarative semantics we adopt in this paper is standard Clark's completion. As usual, we denote as $comp(P)$ the *completion* of a program P, which is the first order theory corresponding to the program P, formed by the collection of *completed definitions* of the predicate symbol in P together with associated Clark's equality theory. SLDNF-resolution is assumed as procedural semantics.

Any program P can be transformed (applying Lloyd-Topor transformations) in a normal program P_N called the *normal form* of P such that $comp(P_N)$ is a conservative extension of $comp(P)$.

To run a goal $\leftarrow W$ in a program P, first a predicate symbol ans, not appearing in P or in W, is defined as

$$ans(X) \leftarrow W$$

where X are the free variables of W, and then the goal $\leftarrow ans(X)$ is run in the normal program P_N^{ans}, that is the normal form of $P \cup \{ans(X) \leftarrow W\}$. Keeping in mind such considerations, when we talk about SLDNF-resolution for non-normal programs we refer to the corresponding normal forms.

Definition. Let P be a program and $\leftarrow W$ a goal. A *correct answer* for $comp(P) \cup \{\leftarrow W\}$ is a substitution θ for the free variables in W such that $comp(P)$ implies the universal closure of $W\theta$:

$$comp(P) \models \forall W\theta.$$

Definition. Let P be a program and G a goal. A *computed answer* θ for $P \cup \{G\}$ is the substitution obtained by restricting the composition $\theta_1 \ldots \theta_n$ to variables of G, where $\theta_1 \ldots \theta_n$ is the sequence of substitutions used in an SLDNF-refutation of $P \cup \{G\}$.

Next definition formalizes the concepts of *procedural equivalence* for programs wrt goals.

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 θ iff $P' \cup \{G'\}$ does.
2. $P \cup \{G\}$ has a finitely failed SLDNF-tree iff $P' \cup \{G'\}$ does.

Intuitively, $P \cup \{G\}$ and $P' \cup \{G'\}$ are procedurally equivalent iff every time an answer (possibly negative) is obtained for one of them, the same answer is obtained for the other one as well.

Now we introduce a definition needed throughout the paper.

Definition. Let S be a set of definitions of predicate symbols. We denote by

$$comp'(S)$$

the set of the corresponding completed definitions together with Clark's equality theory.

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 look at 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 set of predicate definitions that *may depend* upon predicate symbols defined in EDB. We call IDB the *intensional program* of the knowledge base KB.
- EDB is a set of predicate definitions which *do not depend* upon predicate symbols defined in IDB. We call EDB the *extensional program* of the knowledge base KB.

Notice that EDB cannot contain a predicate symbol defined in IDB, neither in the head nor the body of its statements. The notions of IDB and EDB introduced here are a generalization of the usual notions of intensional database and extensional database in the deductive database context, in which IDB is a set of statements and EDB a set of facts – indeed, wrt the intensional program IDB, the extensional program EDB can be considered as specified by a (possibly infinite) set of facts.

We say that an intensional program IDB is a *normal intensional program* if it is 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[2]. Let $Q(X)$ be a query whose free variables are X. A tuple of ground terms T is an *extensional answer* for $Q(X)$ in a program P iff the substitution $\theta = \{X/T\}$ a correct answer for $comp(P) \cup \{\leftarrow Q(X)\}$.

We now turn our attention to intensional answers. We adopt the same definitions as in (Cholvy and Demolombe, 1986), (Pirotte and Roelantes, 1989), (Pirotte, Roelantes, and Zimanyi, 1991), 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.

Definition. A first order formula $A_i(X)$, whose free variables are X, is an *intensional answer* for $Q(X)$ (wrt IDB) 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 trivial variants of the query, those inconsistent wrt $comp'(IDB)$, and those subsumed by others.

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

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

are procedurally equivalent.

Let us show a simple example of intensional query answering.

Example: Consider the following fragment, concerning scientific publications and the bonus they get, of the intensional program IDB of a research institution knowledge base.

$publication\_bonus(x, 50) \leftarrow$
$\quad conference\_publication(x, y)$
$publication\_bonus(x, 100) \leftarrow$
$\quad conference\_publication(x, y) \wedge major\_conference(y)$
$publication\_bonus(x, 150) \leftarrow$
$\quad journal\_publication(x, y)$

$major\_conference(x) \leftarrow sponsor(x, ACM)$
$major\_conference(x) \leftarrow sponsor(x, IEEE)$
$major\_conference(x) \leftarrow accepted\_rate(x, y) \wedge (y \leq 0.2)$
$\ldots$

Suppose we want the answer to the query "Which are the papers that get a publication-bonus greater or equal to 100?", that is:

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

A (complete and procedurally complete) set of intensional answers can be:

$\{\exists z(conference\_publication(x,z) \wedge sponsor(z, ACM)),$
$\exists z(conference\_publication(x,z) \wedge sponsor(z, IEEE)),$
$\exists z(conference\_publication(x,z) \wedge accepted\_rate(x,z) \wedge (z \leq 0.2)),$
$\exists z(journal\_publication(x,z))\}.$

That is, "Papers published in an ACM conference, papers published in an IEEE conference, papers published in a conference whose accepted rate is less or equal to 0.2, and papers published in a journal." □

Beside IDB and EDB, a knowledge base KB may contain additional components that could be exploited in generating intensional answers. In particular KB may contain a set IC of integrity constraints which are closed first order formulas such that:

$$comp(IDB \cup EDB) \models IC.$$

Note that the integrity constraints IC can be considered as part of the intensional knowledge of KB, thus we may define intensional answers $A_i(X)$, as

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

and complete sets of intensional answers S_{IA}, as

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

However arguments have been exhibited (e.g. see (Pirotte, Roelantes, and Zimanyi, 1991)) showing that integrity constraints are often inadequate for inferring additional intensional answers, and that they are better suited for controlling the inference process. In the present paper we leave open the possibility of exploiting integrity constraints for controlling the generation of intensional answers, although we do not investigate the issue further.

3. Partial evaluation

Partial evaluation was introduced as an optimization technique, first in functional programming, and then in logic programming (cf. (Komorowski, 1981)). It consists in deriving a "custom" version of a program, wrt some known input data. Usually, partial evaluation is used to increase the efficiency of a program, computing, a priori, as much as possible of the program wrt a certain class of input.

In logic programming terms, partial evaluation[3] can be described as follows. Given a program P and a goal G, partial evaluation produces a new program P', which is "customized" for the goal G. Obviously, G should have the same answers wrt P and P'.

Partial evaluation relies on the following transformations of the program:

- *unfolding* (in-line substitution) of procedure calls,
- *specializing* with forward and backward propagation of data structure.

The two transformations above merge together in the logic programming framework. The first transformation corresponds to unfolding of a literal in a derivation, and the second, i.e. the propagation of partially instantiated data structures, is automatically supported by unification during the unfolding process. The basic technique to obtain a partial evaluation P' of a logic program P is to construct "partial" search trees for P and suitably chosen atoms as goals, and then extract P' from the definitions associated with the leaves of these trees.

The formal notions and results described here are from (Lloyd and Shepherdson, 1991). 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 (Lloyd, 1987). In (Lloyd, 1987), 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 at the front of the resultant.

In general a resultant is not a clause because Q_1 stands for a conjunction and not a disjunction of literals.

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 for $P \cup \{G\}$, where the sequence of substitutions is $\theta_1, \ldots, \theta_n$, and G_n is $\leftarrow Q_n$. Let θ 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* θ and *resultant* $Q\theta \leftarrow Q_n$. [4]

Now, we state the definition of *partial evaluation* (PE for short). Note that the definition refers to three kinds of PE: the PE of an atom in a program, of a set of atoms in a program, and of a program wrt a set of atoms.

Definition. Let P be a normal program, A an atom, and $\mathcal{T}$ 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 $\mathcal{T}$ such that each non-failed branch of $\mathcal{T}$ 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:

  $$R_i = A\theta_i \leftarrow Q_i \quad (i = 1, \ldots, r),$$

  where we have assumed $G_i = \leftarrow Q_i$

- Let $\mathbf{A} = \{A_1, \ldots, A_s\}$ be a finite set of atoms, and π_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* $\mathbf{A}$.

Intuitively, to obtain a PE of an atom A in P we consider a (not necessarily complete) SLDNF-tree $\mathcal{T}$ for $P \cup \{\leftarrow A\}$, and choose a *cut* in $\mathcal{T}$. The PE is defined as the resultants of the derivations from the original goal $\leftarrow A$ down to the goals in the cut that do not fail in $\mathcal{T}$.

Note that given a (not necessarily complete) SLDNF-tree $\mathcal{T}$ and a cut C of $\mathcal{T}$, we can always define an incomplete SLDNF-tree $\mathcal{T}'$, such that the PE formed by the resultants of the derivations in $\mathcal{T}'$ from the original goal to the non-failing *leaves*, is identical to the PE formed by the resultants of the derivations in $\mathcal{T}$ from the original goal to the non-failing goals in the cut C.[5]

Thus, without losing generality, we can obtain PE as the resultants of the derivations from the original goal to the non-failing leaves of an incomplete SLDNF-tree. In such a way, the choice of the PE depends entirely on the choice of the SLDNF-tree, which, in turn, depends on the selection rule (computation rule)[6] used to expand goals in the nodes of the SLDNF-tree.

The next theorem is the main result on the declarative semantics. First we report the definition of the *closedness condition* to be used in the theorem.

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 $\mathbf{A}$.

Intuitively, the reason we need this condition is that if we "specialize" the definition of a predicate symbol p wrt an atom A containing p, then we cannot expect to be able to correctly answer calls to p that are not instances of A.

THEOREM 1 (LLOYD-SHEPHERDSON) *Let P be a normal program, W a closed first order formula, $\mathbf{A}$ a finite set of atoms, and P' a PE of P wrt $\mathbf{A}$ such that $P' \cup \{W\}$ is $\mathbf{A}$-closed. If W is a logical consequence of comp(P'), then W is also a logical consequence of comp(P), i.e.*

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

Notice that, the converse of this theorem does not hold.

For the procedural semantics we have a theorem stating the procedural equivalence of the original program and its partial evaluation, under the closedness condition and the additional condition of *independence*.

Definition. Let $\mathbf{A}$ be a finite set of atoms. We say $\mathbf{A}$ is *independent* if no pair of atoms in $\mathbf{A}$ have a common instance.

The meaning of the independence condition can be understood as follows. Let p by the predicate symbol appearing in an atom $A_i \in \mathbf{A}$. A PE of A_i in a program P contributes to the corresponding PE of P wrt $\mathbf{A}$ with a part of the definition of p. The independence condition $\mathbf{A}$ imposes that such contributes be disjoint, that is the heads of the clauses coming from the PE of two different atoms cannot have common instances.

THEOREM 2 (LLOYD-SHEPHERDSON) *Let P be a normal program, G a normal goal, $\mathbf{A}$ a finite, independent set of atoms, and P' a PE of P wrt $\mathbf{A}$ such that $P' \cup \{G\}$ is $\mathbf{A}$-closed. Then $P \cup \{G\}$ and $P' \cup \{G\}$ are procedurally equivalent.*

In the theorem above, the closedness condition can be replaced by the *coveredness condition* given below.

Definition. Let P be a normal program and G a normal goal $\mathbf{A}$ a finite set of atoms, P' a PE of P wrt $\mathbf{A}$, and P^* the subprogram of P' consisting of the definitions of the predicate symbols in P' upon which G depends. We say that $P' \cup \{G\}$ is $\mathbf{A}$-*covered* if $P^* \cup \{G\}$ is $\mathbf{A}$-closed.

In the coveredness condition we only force the definitions of the predicate symbols that are actually used in the derivation from the goal G in the partially evaluated program P', to be as "general" as those in the original program.

We remark that the PE of a program wrt a goal is not directly defined. Anyway, there are procedures (e.g. (Benkerimi and Lloyd, 1990)) that, given a program P and a goal G, compute a set of atom $\mathbf{A}$ and a PE of the program P wrt $\mathbf{A}$ such that the original program and the partially evaluated program are procedurally equivalent wrt the goal G.

4. Intensional answers by partial evaluation

In this section we set up the basic results on generating intensional answers by means of partial evaluation.

Let us stress that the intensional program IDB of a knowledge base KB is an "incomplete 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 evident that for IDB, the completion $comp(IDB)$ does not make sense (all predicate symbols defined in EDB would be set to false), while $comp'(IDB)$ does.

The partial evaluation theorems seen in the previous section are not directly useful in dealing with intensional programs. Here, we state analogous theorems which are suitable for such programs. First, we need the next definition (Benkerimi and Lloyd, 1990).

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, $\mathbf{A}$ 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*

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

Proof: First, IDB' and W being **A**-closed, by Theorem 1, we have that for every normal extensional program EDB, which is obviously **A**-closed,

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

Now, suppose the thesis was not true, that is

$$comp'(IDB') \models W \not\Rightarrow comp'(IDB) \models W.$$

Consider the extensional program EDB^* such that $EDB^* = \{A \leftarrow A$: the predicate symbol in A is not defined in IDB, and an instance of A occurs in the body of a program clause in $IDB\}$. Since $comp(IDB \cup EDB^*)$ is identical to $comp'(IDB)$, and $comp(IDB' \cup EDB^*)$ is identical to $comp'(IDB')$, we would have that

$$comp(IDB' \cup EDB^*) \models W \not\Rightarrow comp(IDB \cup EDB^*) \models W,$$

which is a contradiction. ■

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.*

Proof: From the definition of PE it is obvious that $IDB' \cup EDB$ is a PE of $IDB \cup EDB$ wrt **A**. Since **A** is independent and $IDB' \cup EDB \cup \{G\}$ is **A**-closed, 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 KB or W), as

$$q(X) \leftarrow W$$

where X are the free variables occurring in W, and we add this 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 π of $q(X)$ in IDB^q obtained from an L_{IDB^q}-compatible SLDNF-tree for $IDB^q \cup \{\leftarrow q(X)\}$. Let π be

$$q(X)\theta_1 \leftarrow W_1$$
$$\vdots$$
$$q(X)\theta_r \leftarrow W_r$$

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

3) We rewrite the completed definition for q given by these resultants as follows:

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

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) We are returned the set constituted by the disjuncts in the above formula

$$\exists Y_1((X_1 = T_1) \wedge W_1)$$
$$\vdots$$
$$\exists Y_r((X_r = T_r) \wedge W_r).$$

Each of these formulas can be regarded as *intensional answers*. Furthermore the whole set of these formulas is a *complete* and *procedurally complete* set of intensional answers, as the following theorems show.

THEOREM 5 *The formulas returned by the process above form a complete set of intensional answers for the query W in the program P.*

Proof: Let $IDB^{q'}$ be the PE of IDB^q wrt $\mathbf{A} = \{q(X)\}$ obtained by substituting the original definition for q with the PE π of q in IDB^q at step 2. The atom $q(X)$ is L_{IDB^q}-selectable. $IDB^{q'}$ is $\mathbf{A}$-closed, being $q(X)$ the most general atom whose predicate is q. Hence, by Theorem 3 for every $\mathbf{A}$-closed first order formula W we have

$$comp'(IDB^{q'}) \models W \Rightarrow comp'(IDB^q) \models W.$$

In particular, indicating the disjuncts in the formula at step 3 as E_i, $(i = 1 \ldots r)$ the formula $\forall X(q(X) \leftrightarrow E_1 \vee \ldots \vee E_r)$ is $\mathbf{A}$-closed, and so

$$comp'(IDB^q) \models \forall X(q(X) \leftrightarrow E_1 \vee \ldots \vee E_r).$$

By the axiom $\forall X(q(X) \leftrightarrow W)$ in $comp'(IDB^q)$ we can write

$$comp'(IDB^q) \models \forall X(W \leftrightarrow E_1 \vee \ldots \vee E_r).$$

Now, since the predicate symbol q does not appear in $\forall X(W \leftrightarrow E_1 \vee \ldots \vee E_r)$, we can drop the axiom $\forall(q(X) \leftrightarrow W)$ from $comp(IDB^q)$, getting

$$comp'(IDB) \models \forall X(W \leftrightarrow E_1 \vee \ldots \vee E_r).$$

Obviously the following holds as well

$$comp'(IDB) \models \forall X(W \leftarrow E_i),\ i = 1 \ldots r.$$

Recalling the definitions of intensional answer and complete set of intensional answers the thesis follows. ■

THEOREM 6 *The set of intensional answers returned by the process above is procedurally complete.*

Proof: We want to show that

$$IDB \cup EDB \cup \{\leftarrow W\} \tag{2}$$

and

$$IDB \cup EDB \cup \{\leftarrow \bigvee_{i=1}^{r} \exists Y_i((X_i = T_i) \wedge W_i)\} \tag{3}$$

are procedurally equivalent, for any EDB.

First notice that (3) has to be transformed into normal form. Applying Lloyd-Topor transformations we get

$$IDB \cup EDB \cup \{ans(X) \leftarrow (X_i = T_i) \wedge W_i,\ \ i = 1 \ldots r\} \cup \{\leftarrow ans(X)\} \tag{4}$$

Assuming for the predicate symbol "=" the standard procedural meaning "unifiable", then (4) is procedurally equivalent to

$$IDB \cup EDB \cup \{ans(X)\theta_i \leftarrow W_i,\ \ i = 1 \ldots r\} \cup \{\leftarrow ans(X)\} \tag{5}$$

where $\theta_i = \{X_i/T_i\}$.

On the other hand (2) is procedurally equivalent to

$$IDB^q \cup EDB \cup \{\leftarrow q(X)\} \tag{6}$$

because every SLDNF-derivation for $\leftarrow q(X)$ in $IDB^q \cup EDB$ has $\leftarrow W$ as node at depth 1, and $q(X) \leftarrow W$ is not used again in the derivation.

Let $IDB^{q'}$ be the PE of IDB^q wrt $\mathbf{A} = \{q(X)\}$ which is obtained by substituting the original definition for q with the PE π of q in IDB^q at step 2. $\mathbf{A}$ is independent and $q(X)$ is L_{IDB^q}-selectable. $IDB^{q'}$ is $\mathbf{A}$-closed, being $q(X)$ the most general atom whose predicate is q. Hence, by Theorem 4, (6) and

$$IDB^{q'} \cup EDB \cup \{\leftarrow q(X)\} \tag{7}$$

are procedurally equivalent.

Noting that (7) and (5) differ only for the names of the predicate symbols q and ans, combining the procedural equivalence above, the thesis follows. ■

Let us illustrate the method just presented with an example.

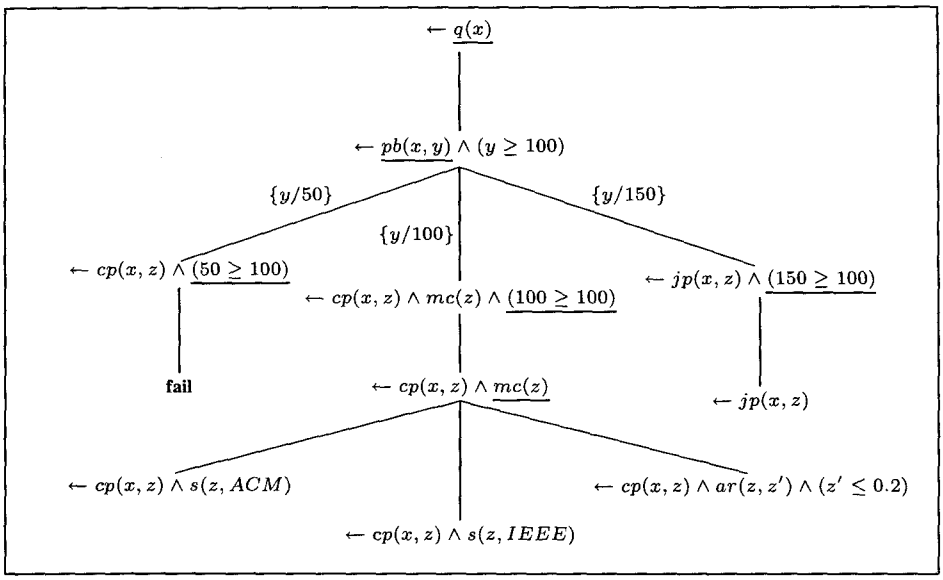

Figure 1. The SLDNF-tree used for the partial evaluation.

Example: Consider again the intensional program IDB on scientific publications and the query "Which are the papers that get a publication-bonus greater or equal to 100?", of the previous example. We proceed as follows:

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 π of $q(x)$ in IDB^q obtained from an L_{IDB^q}-compatible SLDNF-tree. Let such a tree be the one in Figure 4, and π the PE associated 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) We rewrite the completed definition of q in IDB^q:

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

4) We return the disjuncts in the right hand part the above formula. These form a complete and procedurally complete set of intensional answers, precisely the one previously seen. □

We conclude the section making some remarks of the method presented. Observe that, the process above is *parametric* wrt the choice of the PE π of q in IDB^q at step 2.

The *quality* of the intensional answers returned strongly depends on the choice of π, which in turn essentially depends on the selection rule for the related SLDNF-tree. While we do not directly address such an issue in this paper, finding criteria from which to devise a "good" selection rule is one of the most crucial to doing intensional query answering in practice. Suggestions for possible options can be found in (Pirotte, Roelantes, and Zimanyi, 1991), however more work has to be done in defining quality measures for intensional answers.

The *termination* of the above process depends again on the selection rule to be used in the generation of the PE π. 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. Van Harmelen, 1989, Martens, De Schreye, and Bruynooghe, 1992, Martens, De Schreye, and Horváth, 1994, Bol, 1993)).

5. Recursion

The basic method presented in the previous section allows one 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 might 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 would 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:

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

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

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 a type 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 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". □

Given 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, that for some reason are marked *unknown*[8], occurring in the answer. Notice that, if other predicate symbols marked unknown appear in such definitions, then their definitions are to be included in RD as well.[9] To formalize the set RD we now introduce the notion of "set of auxiliary definitions".

The general idea is to return an answer made up of two components: a set of intensional answers, and a set of special definitions for the elements of a given set of predicate symbols L. The latter component is called *set of auxiliary definitions*.

Intuitively a set of auxiliary definitions supplies the meaning of each atom occurring in the composite answer returned, whose predicate symbol is in L (i.e. wrt the atoms occurring in the composite answer whose predicate is in L, the auxiliary definitions retain the same meaning as the corresponding definitions in the intensional program).

Typically, the set L will be the set of unknown recursive predicate symbols occurring in the composite answer. However this is not the only interesting case, sometimes, for example, we may choose a different L to return a shorter composite answer instead of an

excessively large set of intensional answers. Another choice of L is shown in the next section, when the treatment of negation is presented.

We now formally introduce the notion of set of auxiliary definitions. As usual, 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, and S_{IA} a set of intensional answers $A_i(X)$ $(i = 1, \ldots, n)$ for $Q(X)$.

Definition. Let L be a subset of L_{IDB}, $\mathbf{A}_L$ a set of atoms, one for each predicate symbol in L, and AD a set of statements such that all predicate symbols in L occur in the head of at least one of its statements.

We say AD is a *set of auxiliary definitions* (wrt $\mathbf{A}_L$) for predicate symbols in L if:

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

where $comp'(AD)_{\mathbf{A}_L}$ denotes 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 $\mathbf{A}_L$.

Notice that, each predicate symbol p in L is contained in no more that a single atom in $\mathbf{A}_L$, hence we have exactly one logical equivalence in $comp'(AD)_{\mathbf{A}_L}$ involving p. This equivalence can be thought of as the logical definition of p in the context of S_{IA} and AD.[10]

Notice also that given a set $L \subseteq L_{IDB}$ of predicate symbols, a set AD of auxiliary definitions for predicate symbols in L, always exists. In fact, the IDB definitions of these predicate symbols form one such a set. But generally the definitions in AD are specialized versions of those in IDB.

Finally, let us remark that we *do not require* the definitions in AD to be used, instead of the corresponding definitions in IDB, to evaluate the intensional answers in S_{IA}, preserving the *correct answers*, or at least the *computed answers*, of the original query. Indeed such a property would be 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 throughout the 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 choosing the generality of the auxiliary definition for p' we should consider the occurrences of p' arising from the evaluation of these atoms as well.

Observe that, the intensional answers in S_{IA} have the same status as queries, while the set AD of auxiliary definitions is an (incomplete) 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 $comp'(AD)_{\mathbf{A}_L}$.

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

$$comp'(AD)_{\mathbf{A}_L} \models \forall(\chi_{ij}(X) \rightarrow A_i(X)). \tag{8}$$

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

$$comp'(IDB) \models comp'(AD)_{\mathbf{A}_L}. \tag{9}$$

From (8) and (9) we get

$$comp'(IDB) \models \forall(\chi_{ij}(X) \rightarrow A_i(X)). \tag{10}$$

Now, for $A_i(X)$ we have

$$comp'(IDB) \models \forall(A_i(X) \rightarrow Q(X)). \tag{11}$$

Hence, from (10) and (11)

$$comp'(IDB) \models \forall(\chi_{ij}(X) \rightarrow Q(X)), \tag{12}$$

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

Let us turn to the problem of how to compute a set of auxiliary definitions. Assuming IDB to be normal, we get the following result.

THEOREM 7 *Let L be a subset of L_{IDB}, $\mathbf{A}_L$ a set of atoms, one for each predicate symbol in L, and S_{IA} a set of intensional answers for a query W. Then, any PE Π 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 (wrt $\mathbf{A}_L$) for the predicate symbols in L.*

Proof: We have to show that:

1. $comp'(IDB) \models comp'(\Pi)_{\mathbf{A}_L}$,
2. $S_{IA} \cup comp'(\Pi)_{\mathbf{A}_L}$ is $\mathbf{A}_L$-closed.

The second condition is an immediate consequence of the assumption that $S_{IA} \cup \Pi$ is $\mathbf{A}_L$-closed.

Turning to the first condition, we have that $comp'(\Pi) \models comp'(\Pi)_{\mathbf{A}_L}$, and hence by Theorem 3 $comp'(IDB) \models comp'(\Pi)_{\mathbf{A}_L}$. ■

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

THEOREM 8 *Let L be a subset of L_{IDB}, and $\mathbf{A}_L$ a set of atoms, one for each predicate symbol in L. Let also S_{IA} be a complete and procedurally complete set of intensional answers for the query W, and AD a set of auxiliary definitions (wrt $\mathbf{A}_L$) for the predicate*

symbols in L, which is a PE of $\mathbf{A}_L$ *obtained from an* L_{IDB}*-compatible SLDNF-tree, and such that* $S_{IA} \cup AD$ *is* $\mathbf{A}_L$*-closed. Then every set* S'_{IA} *of intensional answers for Q derived by SLDNF-resolution from* S_{IA} *using statements in AD, is complete and procedurally complete.*

Proof: By the Sub-Derivation Lemma and Lemma 4.12 in (Lloyd and Shepherdson, 1991), it follows that each SLDNF-derivation from the original query, using resultants in AD can be expanded in a *corresponding* SLDNF-derivation that uses only statements of the original intensional program IDB. This in turn implies that any SLDNF-tree built using resultants in AD can be expanded into an SLDNF-tree built using only program clauses in IDB.

Now, suppose S_{IA} to be the set $\{A_i(X),\ \ i = 1, \ldots, n\}$ where X are the free variables of the query W. We introduce a new predicate symbol ans defined as $\{ans(X) \leftarrow A_i(X)\ \ i = 1, \ldots, n\}$ and add such a definition for ans to IDB obtaining IDB^{ans}.

From what has been said above, every S'_{IA} can be computed as a PE of $ans(X)$ in IDB^{ans}. By Theorem 5 and Theorem 6, S'_{IA} is a complete and procedurally complete set of intensional answers for $ans(X)$, and hence for Q. ■

Now that we have characterized the notion of a set of auxiliary definitions, we can turn back to dealing with recursive predicate symbols. 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.

Note that, by Theorem 8, 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.

In Figure 2 we show a procedure, based on partial evaluation, to compute S_{IA} and RD, which is adapted from the procedure in (Benkerimi and Lloyd, 1990). The underlying idea is to build "run-time" the set of atoms $\mathbf{A}_R$ that is partially evaluated, while computing S_{IA} and RD. The procedure makes use of *most specific generalization*, briefly *msg* (cf. (Plotkin, 1969, Reynolds, 1969)), formally defined as follows.

Definition. Let S be a set of atoms with the same predicate symbol. Then an atom A is a *most specific generalization* (*msg*) of S if:

- for every atom B in S, B is an instance of A, and
- if all the atoms in S are instances of an atom C, then A is an instance of C as well.

6. Negation

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

1. A negative literal can be selected only if it is *ground.*

Procedure *IQA1*
Input: A normal query W, and a normal intensional program IDB.
Output: A set of intensional answers S_{IA}, a set $\mathbf{A}_R$ of atoms, and a set RD of auxiliary definitions (wrt $\mathbf{A}_R$) for the recursive predicate symbols in S_{IA} and RD marked unknown.

1. Define a new predicate symbol q as $q(X) \leftarrow W$, where X are the free variables in W. Let $IDB^q = IDB \cup \{q(X) \leftarrow W\}$.
2. Let $\mathbf{A} = \{q(X)\}$, $\mathbf{A}_{aux} = \{\}$, $\mathbf{A}_{old} = \{\}$, $\mathbf{D} = \{\}$.
3. **Repeat**
   (A) Select an atom A in $\mathbf{A}$ that has not been selected before (modulo variant).
   (B) Choose a PE of A in IDB^q obtained by an L_{IDB^q}-selectable SLDNF-tree.
   (C) Put (or replace if it is already present) in $\mathbf{D}$ the definition D for the predicate symbol in A.
   (D) $\mathbf{A}_{aux} := \mathbf{A} \cup \{p(T) \mid p(T)$ is in a literal of the body of a statement in D, and p is recursive and marked unknown $\}$.
   (E) $\mathbf{A}_{old} := \mathbf{A}$.
   (F) $\mathbf{A} :=$ { the *msgs* of $\mathbf{A}_{aux}$}.
4. **Until** $\mathbf{A} = \mathbf{A}_{old}$.
5. Extract from the definition of q the set of intensional answers S_{IA}.
6. Extract $\mathbf{A}_R$ dropping the atom containing q from $\mathbf{A}$.
7. Extract RD dropping the definition for q from $\mathbf{D}$.

End Procedure.

Figure 2. Procedure to compute S_{IA} and RD

2. If a ground negative literal is selected then it is either completely evaluated (if possible), or not evaluated at all.

In the literature on partial evaluation (e.g. (Benkerimi and Lloyd, 1990)) the negation is, usually, dealt with as follows: if a negative literal $\sim p(T)$, where T are terms, cannot be evaluated during the PE of a certain atom A in a program P, then $p(T)$ is separately partially evaluated, returning a PE of $\{A, p(T)\}$ in P instead of just a PE of A in P – the set of atoms $\mathbf{A}$, to be partially evaluated, is incrementally computed, starting from the atom A, and adding in $\mathbf{A}$ the atoms in the negative literal that cannot be evaluated. Notice that, this way to proceed strongly resembles how SLDNF-resolution works.

Similarly to what is shown in the previous section, given a query W, we can generate a composite answer, 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 occurring 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 procedure in Figure 2, can be modified to compute S_{IA}, RD and ND. The resulting procedure is shown in Figure 3.

Procedure *IQA2*
Input: A normal query W, and a normal intensional program IDB.
Output: A set S_{IA}, two sets of atoms $\mathbf{A_R}$ and $\mathbf{A_N}$, and two sets RD and ND.

1. Define a new predicate symbol q as $q(X) \leftarrow W$, where X are the free variables in W. Let $IDB^q = IDB \cup \{q(X) \leftarrow W\}$.
2. Let $\mathbf{A} = \{q(X)\}$, $\mathbf{A}_{aux} = \{\}$, $\mathbf{A}_{old} = \{\}$, $\mathbf{D} = \{\}$.
3. **Repeat**
   (A) Select an atom A in $\mathbf{A}$ that has not been selected before (modulo variant).
   (B) Choose a PE of A in IDB^q obtained by an L_{IDB^q}-selectable SLDNF-tree.
   (C) Put (or replace if it is already present) in $\mathbf{D}$ the definition D for the predicate symbol in A.
   (D) $\mathbf{A}_{aux} := \mathbf{A} \cup \{p(T) \mid p(T)$ is in a literal of the body of a statement in D, and p is marked unknown and either recursive, or non-recursive but appearing in a negative literal $\}$.
   (E) $\mathbf{A}_{old} := \mathbf{A}$.
   (F) $\mathbf{A} := \{$ the *msg*s of $\mathbf{A}_{aux}\}$.
4. **Until** $\mathbf{A} = \mathbf{A}_{old}$.
5. Extract from the definition of q in $\mathbf{D}$ the set of intensional answers S_{IA}.
6. Extract form $\mathbf{A}$ the two sets $\mathbf{A_R}$ and $\mathbf{A_N}$, where $\mathbf{A_R}$ is formed by the atoms of $\mathbf{A}$ in which the recursive predicate symbols occur, and $\mathbf{A_N}$ is obtained from $\mathbf{A} - \mathbf{A_R}$, dropping the atom containing q.
7. Extract form $\mathbf{D}$ the two sets RD and ND, where RD is formed by the definitions in $\mathbf{D}$ of recursive predicate symbols, and ND is obtained from $\mathbf{D} - RD$, dropping the definition of q.

End Procedure.

Figure 3. Procedure to compute S_{IA}, RD, and ND

Within such an approach, however, in the formulas of S_{IA}, RD and ND, the "interaction" (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 should try to recover any such interaction in order to make the answer more 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 $comp'(ND)_{\mathbf{A_N}}$ of the form:

$$\forall X(p(T(X)) \leftrightarrow \exists Y(F(X,Y))), \tag{13}$$

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:

$$\forall X(\sim p(T(X)) \leftrightarrow \sim \exists Y(F(X,Y))). \tag{14}$$

The literals of $S_{IA} \cup RD \cup ND$ in which p occurs must, by definition, be instances of $p(T(X))$, so we may replace them with the proper instances of the right hand side of (13) or (14). 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.

Such a treatment is tightly related to *constructive negation* (Chan, 1988, Chan, 1989, Przymusinski, 1989, Drabent, 1995)[11], and has been used to handle negation during the partial evaluation process in (Chan and Wallace, 1989). 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 (Lloyd and Shepherdson, 1991). Moreover, our aim is to expand the negative literals in such a way as not to lose computed answers.

In Figure 4, we present a procedure for such an expansion.

Let us make some remarks on such a procedure. First, formulas obtained by the procedure are logically equivalent to the original ones. Second such formulas may contain existential quantifiers (possibly negated), but no universal quantifiers. Third, the procedure always terminates, since the definitions in ND are non-recursive. Fourth, after the procedure has terminated, ND is no longer needed and can be eliminated. Furthermore, the next theorem states that a kind of procedural containment holds.

THEOREM 9 *Let S_{IA}, RD, and ND be obtained by partial evaluation as shown above. Let S'_{IA} and RD' result from transforming these by the expansion procedure in Figure . Finally, let S''_{IA} be any set of intensional answers derived by any number of SLDNF-resolution steps from S'_{IA} using statements in RD'. Then for every extensional program EDB, we have:*

1. *If $IDB \cup EDB \cup \{\leftarrow \bigvee_{A_i \in S_{IA}} A_i\}$ has an SLDNF-refutation with computed answer θ, then so does $IDB \cup EDB \cup \{\leftarrow \bigvee_{A''_i \in S''_{IA}} A''_i\}$.*
2. *If $IDB \cup EDB \cup \{\leftarrow \bigvee_{A_i \in S_{IA}} A_i\}$ has a finitely failed SLDNF-tree, then so does $IDB \cup EDB \cup \{\leftarrow \bigvee_{A''_i \in S''_{IA}} A''_i\}$.*

Proof: First of all notice that, by Theorem 8, we can unfold the goal $\leftarrow \bigvee_{A_i \in S_{IA}} A_i$ using the definitions in ND, preserving the procedural equivalence. Now, by induction on the number of nested expansions, it is easy to prove that Phase 1 corresponds to such an unfolding.

Base case. If no expansions are performed, then it corresponds to performing no unfolding.

Procedure *NegExp*
Input: S_{IA}, RD, ND.
Output: S'_{IA} and RD' with no references to predicates defined in ND.

For every formula in S_{IA} and RD, apply the two-phase procedure below.

- Phase 1. Recursively apply the following sequence of transformations until it is no longer possible.
  1. Replace atoms in the positive and negative literals of the formula, with the right hand sides of the corresponding instances of the completed definitions in $comp'(ND)$.
  2. Simplify the equalities in the formula as follows:
     (A) 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 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. [11], also Lemma 15.2 in [27]).
     (B) 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. Push the existential quantifiers as far right as possible, eliminating the redundant ones (those that quantify variables which do not occur free in the formula).
- Phase 2. Move negation all the way inward (eliminating double negations), stopping in front of the existential quantifiers.

End Procedure.

Figure 4. A procedure for expanding negative literals

Inductive case. Assume the correspondence to be defined for n nested expansions, we now define it for $n + 1$ expansions. Consider the goal

$$\leftarrow \ldots A_G \ldots$$

and the completed definition $A \leftrightarrow \bigvee_i \exists(X_i = T_i \wedge B_i)$ of the predicate symbol in A_G. We first replace the atom A_G by the right hand side of the instance of the above completed definition corresponding to A_G getting:

$$\leftarrow \ldots \bigvee_i \exists((X_i = T_i)\theta \wedge B_i\theta) \ldots .$$

where $mgu(A_G, A) = \theta$ (notice that $A\theta$=A_G).

Then we simplify the equalities as in the step 2 arriving at

$$\leftarrow \ldots \bigvee_i \exists(Z_i = R_i \wedge B_i\theta\theta_i) \ldots$$

Where the equalities $Z_i = R_i$ denote the result of the simplification, and $\theta_i = \{y/x \mid \exists y(x = y \wedge \ldots)\}$. Now we push the existential quantifiers as far right as possible, and

we put the formula in normal form. Notice that last step has no influence on the resulting normal form. We get:

$$IDB \cup EDB\{A_{new} \leftarrow Z_i = R_i \wedge B_i\theta\theta_i \mid i = 1,\ldots\} \cup \{\leftarrow \ldots A_{new} \ldots\}$$

Consider the SLDNF-derivation from $\leftarrow \ldots A_{new} \ldots$ which at the first step unfolds A_{new} by using the statement $A_{new} \leftarrow Z_i = R_i \wedge B_i\theta\theta_i$ and then resolves all the equalities in $(Z_i = R_i)$. The following is the final goal of this derivation:

$$\leftarrow \ldots B_i\theta\theta_i\sigma_i \ldots$$

where $\theta\theta_i\sigma_i = mgu(A_G, A\{X_i/T_i\})$.[12]

Turning to Phase 2, we prove that it preserves the computed answers and the finite failures of the original intensional answers. We proceed by induction on the number of times a negation is pushed inward.

Base case. If no negations are pushed inward, then the result holds trivially.

Inductive case. Given that the result holds pushing negations inward n times, we prove that it holds if negations are pushed inward $n+1$ times. We use the following notation: for any formula W, we denote by W' the formula obtained from W applying step 3.

Let us assume that in a derivation from the original goal, we select a formula

$$F = \sim(\bigvee_i \bigwedge_{j_i} B_{ij_i}),$$

(to be precise we select the negative literal that arises transforming F in normal form). Note that in F no free variables can occur, otherwise the derivation would flounder.[13]

Suppose F fails. It means that, for some k, $\bigwedge_{j_k} B_{kj_i}$ succeeds, that is all B_{kj_k} (for all j_k) succeed. Consider the transformed subgoal

$$F' = \bigwedge_i \bigvee_{j_i} \sim B'_{ij_i}.$$

In each of its conjuncts (once transformed in normal form) there is exactly one B'_{kj_k} (for some j_k) that occurs in it. By inductive hypothesis, such a B'_{kj_k} succeeds, so $\sim B'_{kj_k}$ fails. It follows that the whole $(\bigwedge_i \bigvee_{j_i} \sim B'_{ij_i})$ fails.

Suppose F succeeds. This means that, for every k, $\bigwedge_{j_k} B_{kj_k}$ fails, that is, for every k, there is a B_{kj_k} (for some j_k) that fails. Let us denote this B_{kj_k} by β_k. It follows that the transformed subgoal F' succeeds too, indeed there is a conjunction C in F' formed exactly by all $\sim\beta'_k$ (for all k). Since by inductive hypothesis all β'_k fail, C succeeds. ■

If the intensional program of the knowledge base is not a normal program, then by using Lloyd-Topor transformations to apply partial evaluation, we introduce new predicate symbols[14] that are obviously *unknown* (i.e. they are meaningless to the user). By the procedure presented here, such predicate symbols can always be replaced by a meaningful formula.

Let us illustrate the treatment of negation with an example.

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):

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

and the query "Who are the drinkers that are neither 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.

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

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.

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

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)

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

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

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

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

7. 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 techniques developed here have two main features wrt those presented in the literature, e.g. (Cholvy and Demolombe, 1986, Pirotte and Roelantes, 1989, Pirotte, Roelantes, and Zimanyi, 1991), which are based on theorem proving.

First, we have a substantial increase of efficiency. Indeed, PE and the procedures proposed here, are much more efficient than general theorem proving. Of course, there is a price, it is possible that some interesting intensional answers cannot be captured by PE, although we believe that a considerable number of them can in fact be captured.

Second, in a logic programming setting where we use as a reasoning procedure SLDNF-resolution which is not complete wrt the declarative semantics, we become interested in making sure that the intensional answers characterize the extensional answers not only theoretically (from a declarative point of view) but also in practice (from a procedural point of view). In such conditions PE, and more generally program transformation techniques, which have been developed with such a duality in mind, give us better adapted tools for generating effective intensional answers, than general theorem proving.

Further extensions of the present work are possible along several directions. We outline some of them below.

The PE process tends to destroy the structure of the program to which it is applied. 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, an issue to investigate further is the use of additional components, usually considered for modeling structural aspects of a knowledge base (e.g. taxonomies or integrity constraints[15]), to improve the quality of intensional answers.

We remark that the techniques developed in this paper do not refer to any particular PE. So, in general, as the research on strategies and criteria to define "intuitive" intensional answers progresses, the new results can be reflected in the particular choices of PE.

This 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" (Motro and Yuan, 1990) and, adding folding techniques, "Intelligent query answering" (Imielinski, 1987).

Finally we want to mention the possibility of adopting different declarative and procedural semantics. For instance, sometimes we may want to interpret negation as *classical negation* in contrast to negation as failure. The work (Inoue, 1991, Inoue, 1992) can be used as a starting point for investigation in this direction. Similarly we may want to adopt the well-founded semantics as declarative semantics. PE in this setting as been studied in (Aravindan and Dung, 1994).

Acknowledgments

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

Notes

1. We mainly use the same notation as (Lloyd, 1987) except that we denote sequences of terms by a single capital letter. The few other differences are pointed out as encountered.
2. In (Lloyd, 1987) a query is a goal. Let $\leftarrow W$ be such a query, we call "query" the first order formula W.
3. Recently, in the context of logic programming, the name *partial deduction* has been proposed to replace the name *partial evaluation*, leaving the original name to denote the optimization oriented use of such a machinery. In this paper we continue to use to the name *partial evaluation* in conformity with (Lloyd and Shepherdson, 1991) and (Benkerimi and Lloyd, 1990) whose results are extensively used.
4. Note, if $n = 0$, the resultant is $Q \leftarrow Q$.
5. $\mathcal{T}'$ is formed by the complete failed branches of $\mathcal{T}$, and by the branches of $\mathcal{T}$ corresponding to the derivations from the original goal to the goals in the cut that do not fail.
6. We consider the selection rule not as a function of the current goal alone (cf. (Lloyd, 1987)) but as a function of the whole history of the derivation from the root goal to the current goal (cf. (Shepherdson, 1984)).
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.
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 could have also assumed that an independent set of atoms corresponds to p. This would entail that in $comp'(AD)_{\mathbf{A}_L}$ 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 can be immediately extended to the case where the second assumption is adopted.
11. Incidentally we observe that, intensional answers are syntactically similar to "equational formulas" of (Przymusinski, 1989, in which not only equalities but also extensional and possibly intensional predicates are allowed.
12. Note that this is identical to the goal obtained resolving A_G with the statement $A\{X_i/T_i\} \leftarrow B_i$ in ND.
13. Obviously there may be existentially quantified variables, but recall that we do not push negation inside existential quantifications.
14. New predicate symbols are introduced to eliminate the negated existentially quantified (universally quantified) formulas.
15. Integrity constraints can also be used by selection rules to prune away inconsistent goals.

References

C. Aravindan and P. M. Dung. Partial Deduction of Logic Programs wrt Well-Founded Semantics. *New Generation Computing*, 13(1):45–71, 1994.

K. Benkerimi and P. Hill. Supporting transformation for partial evaluation of logic programs. *J. of Logic and Computation*, 3(5):469–486, 1993.

K. Benkerimi and J. W. Lloyd. A partial evaluation procedure for logic programs. In *Proc. of North American Conf. on Logic Programming*, pages 343–358. MIT Press, 1990.

R. Bol. Loop checking in partial deduction. *J. of Logic Programming* 16(1-2):25–45, 1993.

D. Chan. Constructive negation based on the completed database. In *Proc. of 5th International Conference and Symposium on Logic Programming*, pages 111–125. MIT Press, 1988.

D. Chan. An extension of constructive negation and its application in coroutining. In *Proc. of North American Conf. on Logic Programming*, pages 477–493. MIT Press, 1989.

D. Chan and M. Wallace. A treatment of negation during partial evaluation. In *Meta-Programming in Logic Programming (Proc. META'88)*, pages 299–317. MIT Press, 1989.

L. Cholvy and R. Demolombe. Querying a rule base. In *Proc. 1st Int. Conf. on Expert Database Systems*, pages 365–371, 1986.

W. W. Chu, Q. Chen, and R.-C. Lee. A pattern-based approach for deriving approximate and intensional answers. In *Proc. of the 1st Int. Work. on Interoperability in Multidatabase Systems*, pages 262–265. IEEE Comput. Soc. Press, 1991.

W. W. Chu, R.-C. Lee, and Q. Chen. Using type inference and induced rules to provide intensional answers. In *Proc. of IEEE Int. Conf. on Data Engineering*, 1991.

K. L. Clark. Negation as failure. In *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

F. Corella. Semantic retrieval and levels of abstraction. In *Proc. 1st Int. Workshop on Expert Database Systems*, pages 397–420, 1984.

F. Cuppens and R. Demolombe. Cooperative answering: a methodology to provide intelligent access to databases. In *Proc. 2nd Int. Conf. on Expert Database Systems*, pages 333–353, 1988.

G. De Giacomo. Intensional query answering: an application of partial evaluation. In *Logic Program Synthesis and Transformation (Proc. of LOPSTR'92)*, pages 132–150. Springer-Verlag, 1993.

R. Demolombe. A strategy for the computation of conditional answers. In *Proc. of the 10th Europ. Conf. on Artificial Intelligence*, 1992.

R. Demolombe and T. Imielinski, editors. *Nonstandard Queries and Nonstandard Answers*. Studies in Logic and Computation, Oxford Science Publications, 1994.

D. A. De Waal. The power of partial evaluation. In *Logic Program Synthesis and Transformation (Proc. of LOPSTR'93)*, pages 159–161. Springer-Verlag, 1994.

W. Drabent. What's failure? An approach to constructive negation. *Acta Informatica*, 5(1):27–60, 1995.

M. M. Fonkam. Employing integrity constraints for query modification and intensional answer generation in multi-database systems. In *Advances in Database Systems (Proc. of the 10th British Nat. Conf. on Databases)*, pages 244–260. Springer-Verlag, 1992.

T. Imielinski. Intelligent query answering in rule based systems. *J. of Logic Programming*, 4(3):229–257, 1987.

T. Imielinski and B. R. Badrinath. Data management for mobile computing. *SIGMOD RECORD*, 22(1):34–39, 1993.

K. Inoue. Extending logic programs with default assumptions. In *Proc. of the 8th Int. Conf. on Logic Programming*, pages 490–504, 1991.

K. Inoue. Linear resolution for consequence finding. *Artificial Intelligence*, 56:301–353, 1992.

Y. H. Kim and H.-Y. Kim. Applying intensional query processing techniques to object-oriented database systems. In *Proc. of the 3rd Int. Sym. on Database Systems for Advanced Applications*, pages 405–412. World Scientific, 1993.

H. J. Komorowski. A specification of an abstract prolog machine and its application to partial evaluation. Technical Report LSST 69, Linköping University, 1981.

H. J. Komorowski. An introduction to partial deduction. In *Proc. of the 3rd Int. Work. of Meta-Programming in Logic (META'92)*, pages 49–69. Springer-Verlag, 1992.

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

J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *J. of Logic Programming*, 11(3&4): 217–242, 1991.

J. W. Lloyd and R. W. Topor. Making prolog more expressive. *J. of Logic Programming*, 1(3):225–240, 1984.

B. Martens, D. De Schreye, and M. Bruynooghe. Sound and complete partial deduction with unfolding based on well-founded measures. In *Proc. of the International Conference on Fifth Generation Computer Systems*, pages 473–480, 1992.

B. Martens, D. De Schreye, and T. Horváth. Sound and complete partial deduction with unfolding based on well-founded measures. *Theoretical Computer Science*, 122(1-2):97–117, 1994.

A. Motro. Using integrity constraints to provide intensional answers to relational queries. In *Proc. 15th Int. Conf. on Very Large Data Bases*, pages 237–246, 1989.

A. Motro. Intensional answers to database queries. Technical report, Department of Information and Software Systems Engineering, George Mason University, Fairfax, Virginia, 1991.

A. Motro. Responding with knowledge. In *Advances in Databases and Artificial Intelligence, Vol. 1: The Landscape of Intelligence in Database and Information Systems*. JAI Press, 1993.

A. Motro and Q. Yuan. Querying database knowledge. In *Proc. of ACM SIGMOD-90*, pages 173–183, 1990.

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*, pages 652–659, 1989.

A. Pirotte, D. Roelantes, and E. Zimanyi. Controlled generation of intensional answers. *IEEE Trans. on Knowledge and Data Engineering*, 3(2):221–236, 1991.

G. D. Plotkin. A note on inductive generalization. In *Machine Intelligence*, Vol. 5. University Press, 1969.

T. C. Przymusinski. On constructive negation in logic programming. In *Proc. of North American Conf. on Logic Programming*, pages 1–19 (addendum). MIT Press, 1989.

J. C. Reynolds. Transformational systems and algebraic structure of atomic formulas. In *Machine Intelligence*, Vol. 5. Edinburgh University Press, 1969.

J. C. Shepherdson. Negation as failure: a comparison of Clark's completed data base and Reiter's closed world assumption. *J. of Logic Programming*, 1(1):51–81, 1984.

C. Shum and R. Muntz. Implicit representation for extensional answers. In *Proc. 2nd Int. Conf. on Expert Database Systems*, pages 257–273, 1988.

I.-Y. Song, H.-Y. Kim, and P. Geutner. Intensional query processing: a three-step approach. In *Proc. of the Int. Conf. on Database and Expert Systems Applications*, pages 542–549. Springer-Verlag, 1990.

F. Van Harmelen. The limitations of partial evaluation. In *Logic-Based Knowledge Representation*, pages 87–111. MIT Press, 1989.