in: Proc. Workshop on Non-Standard Queries and
Non-Standard Answers, Toulouse, France
July 1-4, 1991

Constrained Query Answering

– Preliminary Report –

François Bry

ECRC, Arabellastr. 17, D – W 8000 München 81, Germany
bry@ecrc.de

ABSTRACT *Traditional answering methods evaluate queries only against positive and definite knowledge expressed by means of facts and deduction rules. They do not make use of negative, disjunctive or existential information. Negative or indefinite knowledge is however often available in knowledge base systems, either as design requirements, or as observed properties. Such knowledge can serve to rule out unproductive subexpressions during query answering. In this article, we propose an approach for constraining any conventional query answering procedure with general, possibly negative or indefinite formulas, so as to discard impossible cases and to avoid redundant evaluations. This approach does not impose additional conditions on the positive and definite knowledge, nor does it assume any particular semantics for negation. It adopts that of the conventional query answering procedure it constrains. This is achieved by relying on meta-interpretation for specifying the constraining process. The soundness, completeness, and termination of the underlying query answering procedure are not compromised. Constrained query answering can be applied for answering queries more efficiently as well as for generating more informative, intensional answers.*

1. Introduction

Traditional answering methods evaluate queries only against positive and definite knowledge expressed by means of facts and deduction rules. They do not make use of negative, disjunctive or existential information. Negative or indefinite knowledge is however often available in knowledge base systems, either as design requirements – e.g. database integrity constraints or logic program specifications –, or as observed properties – e.g. "memoized" answers or computed generalizations as in, e.g., [Sie88, YS89]. It has often been observed that negative and indefinite knowledge could serve to improve query answering. However, the methods for performing such improvements that have been proposed either impose strong limitations, or require rather complex inference engines – for example theorem provers that are complete for full first-order logic. It is not clear whether their principles can be combined with techniques developed for conventional query answering – e.g. for processing recursive deduction rules.

In this paper, we describe an approach called "constrained query answering" extending conventional backward – or top-down – reasoning procedures with a processing of general, possibly negative or indefinite, knowledge in order to rule out unproductive subexpressions. We restrict ourselves to backward reasoning procedures for two reasons. Firstly, most procedures that have been proposed for processing logic programs or querying deductive databases perform backward reasoning – in the sense which is relevant here, so do the Alexander [Roh86] and Magic Set [Ban86, BR87, Ram88] methods. Secondly, it is not clear whether the paradigm investigated here could be applied with forward reasoning.

The approach described in this article constrains any conventional, backward reasoning query answering procedure – e.g. SLD-resolution [Llo87], OLDT or SLDAL-resolution [TS86, Vie89], the Alexander [Roh86] or Magic Set [Ban86, BR87, Ram88] methods, etc. – by discarding unsatisfiable and redundant subexpressions. This is done by relying on additional knowledge such as integrity constraints or "memoized" answers. If such a knowledge is not available, or if it is not relevant to the query under consideration, the reasoning performed by the underlying answering procedure is not affected.

Constrained query answering does not impose any condition on the definite knowledge used by the underlying procedure – in particular, it does not preclude recursive deduction rules. Neither does it assume any particular semantics for negation, but it adopts the semantics implemented by the underlying answering procedure. This inheritance of features of the underlying answering procedure is achieved by specifying the constraining process as a meta-interpreter for this procedure.

The three features:
(1) no additional conditions imposed on deduction rules,
(2) independence from the formalization of negation as failure,
(3) extension of a conventional query answering procedure by means of meta-interpretation, are, to the best of our knowledge, specific to the method proposed here. They distinguish it from other proposals based on similar ideas, in particular from the approaches described in [HZ80, Kin81a, Kin81b, Cha84, Cha86, SO87, LM88, Cha88, LH88, Mot89, SO89, PR89, Lee91] ([Mot90] gives a comparative introduction to some of the methods described in these articles). Although constraints are general first-order formulas, constrained query answering neither specifies a new "full theorem prover", nor "disguises" a known approach to theorem proving.

Constrained query answering can be applied for answering queries more efficiently as well as for generating more informative, intensional answers. Detecting and ruling out unproductive evaluations can indeed be applied to improve efficiency of query answering. Since the constraining process takes place before the extensional knowledge – i.e. the database of facts – is accessed, considerable gains of efficiency can be expected for database applications that usually contain large numbers of facts. Constrained query answering can also be applied to generate more informative answers. It indeed answers queries without consulting the extensional knowledge, in those cases where the intensional information – i.e. the deduction rules and the constraints – is sufficient. Answers computed only from intensional knowledge such as deduction rules, design requirements, and integrity constraints can be considered more informative than answers depending on casual, extensional data.

This report first defines constrained query answering for Horn knowledge bases, then for knowledge bases with deduction rules containing negative literals. The classical fixpoint of the

immediate consequence operator is used for defining the semantics of Horn knowledge bases and of their constraints. In order to define constrained query answering independently from any actual backward reasoning query answering procedure, we specify it in terms of an abstract model of backward reasoning. This model formalizes the generation of goals during backward reasoning as the fixpoint of a meta-interpreter. This model has been introduced in [Bry90] where it was applied to investigating different paradigms for querying recursive deduction rules.

Rather different semantics have been proposed for non-Horn knowledge bases. Moreover, their definitions refer to a great variety of formalisms. For some of these semantics no backward reasoning query answering procedures have been proposed. For these reasons, it is desirable to keep the definition of constrained query answering as independent as possible from any particular semantics of negation. We show that this is possible, provided one restricts himself to semantics fulfilling a certain condition conveying the notion of "supported deduction". We argue that this weak condition reflects the intuitive meaning of negation in knowledge bases and logic programs. We show how the abstract model of backward reasoning extends to non-Horn knowledge bases under any "support-based" semantics. Relying on this extended model, we show that constrained query answering extends as well to non-Horn knowledge bases under such a "support-based" semantics.

This article consists of seven sections, the first of which is this introduction. Section 2 recalls some background notions and introduces a few notations. We give in Section 3 the abstract formalization of backward reasoning. Relying on this formalization, we show in Section 4 how to rule out inconsistent subexpressions. In Section 5, we define a similar approach to avoiding redundant evaluations. Section 6 is devoted to specifying constrained query answering to non-Horn knowledge bases. In a last section, we briefly discuss relationships with related issues.

2. Background

We define a *knowledge base* KB as a triple of finite sets, KB = (F, DR, C), where F is the set of *facts,* DR the set of *deduction rules,* and C the set of *constraints.* Elements of F, DR, and C are assumed to be expressed in a same first-order language $\mathcal{L}_{KB}$. The variables of $\mathcal{L}_{KB}$ will be denoted by lower case italics characters.

Informally, F and DR specify definite and positive knowledge on which conventional query answering methods rely; C is a set of properties known to be implied by F $\cup$ DR. Typically, elements of C are integrity constraints, program specifications, or observed properties such as "memoized" answers or computed generalizations. Since F $\cup$ DR implies C, completeness of query answering can be achieved by evaluating queries against F $\cup$ DR only as done by conventional answering procedures. Thus, C express redundant knowledge. Formally, facts, deduction rules, and constraints are defined as follows.

Facts are atoms and deduction rules are expressions of the form:
$H \leftarrow L_1 \wedge \ldots \wedge L_n$
where $n \geq 1$, H is an atom (called the *head* of the rule), and the L_is are literals (called *body literals*). Such a rule denotes the formula:
$\forall x_1 \ldots \forall x_k \quad (L_1 \wedge \ldots \wedge L_n \Rightarrow H)$
where the x_js are the variables occurring in H or in the L_is. If the body literals L_is are all positive, then the rule is called a *Horn rule.* A knowledge base KB = (F, DR, C) is a *Horn*

knowledge base if all rules in DR are Horn rules. Constraints are closed first-order formulas.

For simplifying the description of the method, but without loss of generality, we assume that constraints are expressions of the form:

$A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \vee \ldots \vee B_m$

where $n \geq 0$, $m \geq 0$, the A_is and B_js are atoms. Such an implication denotes the formula:

$\forall x_1 \ldots \forall x_k \quad (A_1 \wedge \ldots \wedge A_n \Rightarrow B_1 \vee \ldots \vee B_m)$

where the x_js are the variables occurring in the A_is or in the B_js. For $n = 0$ it denotes:

$\forall x_1 \ldots \forall x_k \quad (\text{true} \Rightarrow B_1 \vee \ldots \vee B_m)$

Similarly, if $m = 0$, it denotes:

$\forall x_1 \ldots \forall x_k \quad (A_1 \wedge \ldots \wedge A_n \Rightarrow \text{false})$

Such a constraint is often called a *denial.* The A_is (B_js, resp.) are called *body literals* (*head literals,* resp.). This implicative form is almost identical to the clausal form. It is derived from general first-order logic formulas by representing existentially quantified variables by Skolem functions, and by computing prenex disjunctive normal forms. Note that, due to the implicative form, negation does not occur explicitly in constraints.

The semantics of a Horn knowledge base is defined in terms of the *immediate consequences* derivable from its facts and deduction rules. A ground atom A is an *immediate consequence* of $F \cup DR$ if there exists:

- $R = H \leftarrow L_1 \wedge \ldots \wedge L_n \in DR$
- a substitution σ of constants for variables in R

such that:

- $H\sigma = A$
- $L_i\sigma \in F$, for all $i = 1, \ldots, n$.

The *immediate consequence operator* of a Horn knowledge base $KB = (F, DR, C)$ is the function associating with each subset D of the Herbrand base of $\mathcal{L}_{KB}$ – i.e. the set of all ground atoms expressible in $\mathcal{L}_{KB}$ – the set $T_{DR}(D)$ of immediate consequences of $D \cup DR$. The fixpoint $T_{DR} \uparrow^{\omega}(D)$ is the set of all facts derivable from $D \cup DR$ by repeated applications of T_{DR}. Formally:

$$T_{DR} \uparrow^{\omega}(D) = \bigcup_{n \in N} T_{DR} \uparrow^{n}(D)$$

where:

$$T_{DR} \uparrow^{0}(D) = D$$
$$T_{DR} \uparrow^{n+1}(D) = T_{DR}(T_{DR} \uparrow^{n}(D)) \cup T_{DR} \uparrow^{n}(D)$$

The fixpoint $T_{DR} \uparrow^{\omega}(F)$ is finite if $F \cup DR$ contains no function symbols or if DR is not recursive.

The fixpoint $T_{DR} \uparrow^{\omega}(F)$ specifies the (unique) model $\mathcal{M}_{KB}$ of a Horn knowledge base $KB = (F, DR, C)$ as follows: a ground atom is true in $\mathcal{M}_{KB}$ if it belongs to $T_{DR} \uparrow^{\omega}(F)$, it is false otherwise. A general formula G is true or satisfied in KB – denoted $KB \vdash G$ – if $\mathcal{M}_{KB} \models G$, i.e. if $\mathcal{M}_{KB}$ is a model of G. Note that the truth of formulas in a knowledge base does not depend on its constraints.

The semantics of constraints in a Horn knowledge base $KB = (F, DR, C)$ is formally defined by the condition:

$\forall G \in C \quad KB \vdash G$

It is worth emphasizing this definition of constraints, for two reasons. Firstly, it defines the semantics of constraints in terms of the semantics of facts and rules, i.e. in practice, by relying

on the available query answering procedure. Secondly,the concept of constraint considered here departs from that of "constraint programming languages" [Lel87], in particular from "constraint logic programming" [JL87, Van89] and from the "constraint query languages" investigated in [Kan90]. In these languages, constraints do not necessarily express redundant information.

We conclude this section with some remarks on the computation of the fixpoint $T_{DR} \uparrow^{\omega}(F)$ of a Horn knowledge base KB = (F, DR, C).

The definition $T_{DR} \uparrow^{\omega}(F) = \bigcup_{n \in N} T_{DR} \uparrow^{n}(F)$ suggests to compute the fixpoint $T_{DR} \uparrow^{\omega}(F)$ by computing the sets $T_{DR} \uparrow^{n}(F)$ for increasing values of n. Termination – in those cases where the fixpoint $T_{DR} \uparrow^{\omega}(F)$ is finite – is ensured by avoiding redundant computations. The computation can be interrupted as soon as no new consequences are generated. Formally:
$T_{DR} \uparrow^{\omega}(F) = T_{DR} \uparrow^{k}(F) \Longleftrightarrow T_{DR} \uparrow^{k+1}(F) \subseteq T_{DR} \uparrow^{k}(F)$

This approach to computing the fixpoint of a Horn knowledge base is often called the naïve method. Although sufficient to ensure termination, the redundancy condition
$T_{DR} \uparrow^{k+1}(F) \subseteq T_{DR} \uparrow^{k}(F)$
can be strengthened so as to generate at step n+1 only atoms with some premises that where generated at step n. This refined procedure is often named the semi-naïve method.

In defining constrained query answering, fixpoint computations are described that require more sophisticated redundancy conditions than that of the naïve and semi-naïve methods. Constrained query answering can be viewed as strengthening the redundancy conditions of fixpoint computations using the knowledge expressed by the constraint.

3. A Formalization of Backward Reasoning

In order to answer a query Q posed to a Horn knowledge base KB = (F, DR, C), one can compute the fixpoint $T_{DR} \uparrow^{\omega}(F)$ and then evaluate Q against this fixpoint, using the query evaluator of a relational, non-deductive database system. This approach has some drawbacks. In particular, it requires to materialize the whole of $T_{DR} \uparrow^{\omega}(F)$ although most queries only refer to parts of this fixpoint. Techniques have been developed that aim at restricting the computation of derived facts to data relevant to the considered queries. Backward reasoning is the paradigm upon which most of these techniques rely.

Backward reasoning is formalized in different manners, usually, as input resolution, or as a degenerated form of SL-resolution called SLD-resolution [Llo87]. We give here a different formalization in term of the fixpoint of a set of deduction rules. These deduction rules are quite similar to that of a knowledge base, but their variables range over formulas of the language $\mathcal{L}_{KB}$ of the knowledge base KB under consideration. The variables occurring in the rules of KB in contrast range over constants of $\mathcal{L}_{KB}$. Thus, the deduction rules formalizing backward reasoning specify a meta-interpreter for the rules of the knowledge base under consideration. We call Backward Fixpoint Procedure, short BFP, this meta-interpreter. Upper case characters in italics denote meta-variables.

$$
\begin{aligned}
\text{fact}(Q) &\leftarrow \text{query}(Q) \wedge \text{rule}(Q \leftarrow B) \wedge \text{evaluate}(B) \\
\text{query}(B) &\leftarrow \text{query}(Q) \wedge \text{rule}(Q \leftarrow B) \\
\text{query}(Q_1) &\leftarrow \text{query}(Q_1 \wedge Q_2) \\
\text{query}(Q_2) &\leftarrow \text{query}(Q_1 \wedge Q_2) \wedge \text{evaluate}(Q_1)
\end{aligned}
$$

The predicate "rule" denotes access to the set of deduction rules of the knowledge base under consideration. The predicate "evaluate" answers conjunctive and atomic queries against the set of facts extended with the already generated atomic answers. The first rule describes the generation of answers to posed queries. The three other rules describe how atomic goals are generated by reasoning backward: the first one generates a conjunctive goal from an already posed query which unifies with the head of a rule. The last two rules decompose conjunctive queries into atomic goals.

The Backward Fixpoint Procedure was introduced in [Bry90] for investigating the relationship between various query answering procedures for recursive rules. It is shown there that the rewritings of the Magic Set methods result from the partial evaluation of the Backward Fixpoint Procedure over the object rules, i.e. the rules of the considered knowledge base. For this reason, the Magic Set methods can be seen as backward reasoning query answering procedures.

Proposition 1: [Bry90] Let KB = (F, DR, C) be a Horn knowledge base, Q a set of "query(G)" expressions where the Gs are atoms in the language $\mathcal{L}_{\text{KB}}$ of KB, and A an atom in the language $\mathcal{L}_{\text{KB}}$. If there is an expression "query(G)" in Q such that A and G unify, the following equivalence holds:

$$\text{fact(A)} \in T_{\text{BFP}} \uparrow^{\omega} (\text{F} \cup \text{DR} \cup \text{Q}) \Longleftrightarrow \text{A} \in T_{\text{DR}} \uparrow^{\omega}(\text{F})$$

Though unconventional, the formalization of backward reasoning given above has, in the context of this article, a major advantage: as it is shown in Section 6, it extends to any semantics of negation which satisfies a condition formalizing the intuitive notion of "supported deduction". This allows us to extend constrained query answering to non-Horn knowledge bases.

Termination – in those cases where the queries in Q have finitely many answers – requires to refine the redundancy tests of the naïve and semi-naïve methods. Since the last three meta-rules in BFP might generate facts containing object language variables, syntactically distinct but semantically undistinguishable expressions such as "query(p(x, a))" and "query(p(y, a))" can be generated. It follows that the fixpoint $T_{\text{BFP}} \uparrow^{\omega}(\text{F} \cup \text{DR} \cup \text{Q})$ is subsumed by the set $T_{\text{BFP}} \uparrow^{k}(\text{F} \cup \text{DR} \cup \text{Q})$ for some finite k if all elements of $T_{\text{BFP}} \uparrow^{k+1}(\text{F} \cup \text{DR} \cup \text{Q})$ are instances of facts in $T_{\text{BFP}} \uparrow^{k}(\text{F} \cup \text{DR} \cup \text{Q})$. Moreover, at each step n of the computation, elements of $T_{\text{BFP}} \uparrow^{n}(\text{F} \cup \text{DR} \cup \text{Q})$ that are instances of facts in $T_{\text{BFP}} \uparrow^{n-1}(\text{F} \cup \text{DR} \cup \text{Q})$ can be eliminated before computing the next set of consequences.

The Backward Fixpoint Procedure does not relate a goal – i.e. a meta-language "query" expression – to the goals it is generated from. This relationship is however needed for specifying constrained query answering. We therefore conclude this section with a modified version called BFP* of the Backward Fixpoint Procedure which associates with each goal its "ancestor" and "cousin" goals.

$$
\begin{array}{ll}
\text{query}(Q, \{Q\}) & \leftarrow \text{query}(Q) \\
\text{fact}(Q) & \leftarrow \text{query}(Q, S) \wedge \text{rule}(Q \leftarrow B) \wedge \text{evaluate}(B) \\
\text{query}(B, S_2) & \leftarrow \text{query}(Q, S_1) \wedge \text{rule}(Q \leftarrow B) \wedge \text{merge}(S_1, B, S_2) \\
\text{query}(Q_1, S) & \leftarrow \text{query}(Q_1 \wedge Q_2, S) \\
\text{query}(Q_2, S) & \leftarrow \text{query}(Q_1 \wedge Q_2, S) \wedge \text{evaluate}(Q_1)
\end{array}
$$

The first rule initializes the set of goals related to an initial goal. The second rule describes the generation of answers to queries, as in the previous version of the Backward Fixpoint Procedure. The third rule generates a conjunctive goal from an already generated goal which unifies with the head of a rule. We assume that "merge(S_1, B, S_2)" holds if and only if S_2 denotes the set S_1 augmented with the literals occurring in B. The set S_2 is associated by the third rule with the generated conjunctive query B. The last two rules decompose a conjunctive query into atomic ones, like in the previous version of the procedure. During this decomposition, the set of related goals is transmitted to the generated component goals.

The predicate "merge" can be implemented either procedurally, assuming that the rule language accepts procedural attachements, or by means of additional deduction rules. Since these rules are rather straightforward to specify, we do not give them here.

The following proposition establishes the soundness and completeness of the BFP* procedure.

> **Proposition 2:** Let KB = (F, DR, C) be a Horn knowledge base, Q a set of "query(G)" expressions where the Gs are atoms in the language $\mathcal{L}_{KB}$ of KB, and A a ground atom in the language $\mathcal{L}_{KB}$. If there is an expression "query(G)" in Q such that A and G unify, the following equivalence holds:
>
> $$\text{fact(A)} \in T_{BFP*} \uparrow^{\omega} (F \cup DR \cup Q) \Longleftrightarrow A \in T_{DR} \uparrow^{\omega}(F)$$

Proof: (sketched) It suffices to prove that for all integers n:
(1) For each expression query(Q_1, S) in $T_{BFP*} \uparrow^{\omega}$ (F ∪ DR ∪ Q), there exists an expression "query(Q_2)" in $T_{BFP} \uparrow^{\omega}$ (F ∪ DR ∪ Q) such that Q_1 and Q_2 are variants.
(2) For each expression "query(Q_2)" in $T_{BFP} \uparrow^{\omega}$ (F ∪ DR ∪ Q) there exists an expression "query(Q_1, S)" in $T_{BFP*} \uparrow^{\omega}$ (F ∪ DR ∪ Q) such that Q_1 and Q_2 are variants.
Both properties result from the definition of BFP and BFP*. □

Termination – in case the considered queries have finitely many answers – requires to refine once again the redundancy condition. Eliminating variant "query(Q, S)" expressions while computing the sets $T_{BFP*} \uparrow^{n}$ (F ∪ DR ∪ Q) does not suffice to ensure termination, because larger sets S of related goals are generated for increasing values of n. Termination is achieved by not applying the third rule in BFP*

$$\text{query}(B, S_2) \leftarrow \text{query}(Q, S_1) \wedge \text{rule}(Q \leftarrow B) \wedge \text{merge}(S_1, B, S_2)$$

in case Q is an instance of an element of S_1. This does not compromise completeness for the following reasons. Firstly, by construction of the set of goals S_1, if Q is an instance of an element Q' of S_1, then a goal "query(Q', S')" has already been generated. Secondly, the generation of answers by the second rule of BFP* depends only on the first argument Q of the "query" expression and does not take the set S of goals into account.

4. Ruling out Inconsistencies

In this section, we show how to reason on the constraints of a knowledge base and on the goals generated by the BFP* procedure for ruling out inconsistencies. We first informally suggest the approach on an example. Then, we formalize it in the general case and we establish its correctness. Finally, we specify an extension to the BFP* procedure which discards inconsistent goals.

Consider the following knowledge base KB = (F, DR, C), the set Q of queries, and the facts and meta-facts generated by the BFP* procedure. Below, S^n denotes the set of newly generated facts at step n, i.e. the difference set

$T_{BFP*} \uparrow^n(F \cup DR \cup Q) \setminus T_{BFP*} \uparrow^{n-1}(F \cup DR \cup Q)$.

F: b

DR: $a \leftarrow b$ $\quad q \leftarrow r \wedge s$
$p \leftarrow q \wedge u$ $\quad q \leftarrow t$
$r \leftarrow a$ $\quad t \leftarrow b$

C: $s \wedge u \rightarrow$ false
$p \wedge t \rightarrow$ false
$p \wedge q \rightarrow v$

Q: query(p)

S^1: query(p, {p})
S^2: query($q \wedge u$, {p, q, u})
S^3: query(q, {p, q, u})
S^4: query($r \wedge s$, {p, q, u, r, s})
query(t, {p, q, u, t}
S^5: query(r, {p, q, u, r, s})
query(b, {p, q, t, b})
S^6: t
S^7: q

By definition, constraints express properties that are satisfied by the facts and deduction rules. Thus, the constraint

$s \wedge u \rightarrow$ false

states that "s" and "u" are not both true in KB. As a consequence, there is no need to further expand partial proofs relying on these two facts. In particular, the goal

query($r \wedge s$, {p, q, u, r, s})

generated at step 4 is useless in the sense that it must fail. It cannot yield a proof of "$r \wedge s$" since its associated set of related goals contains both "s" and "u". Similarly, the constraint

$p \wedge t \rightarrow$ false

gives rise to discarding the second goal

query(t, {p, q, t})

of S^4. Thus, by reasoning on the constraints, one can interrupt the BFP* procedure already at step 4.

One should notice that the constraint
$p \wedge q \rightarrow v$
does not allows us to rule out queries referring to a set of goals such as {p, q, u} (steps 2 and 3). Although this set does not satisfy the constraint $p \wedge q \rightarrow v$, neither does it falsify this constraint: the set of goals {p, q, u} represents an incomplete proof tree, indeed. Intuitively, the above-mentioned constraint would be definitely violated only if the set would be extended with $\neg v$. Then, there would be no ways to extend furthermore a set containing p, q, and $\neg v$ into a set satisfying the constraint $p \wedge q \rightarrow v$.

In presence of non-ground rules and constraints, one should pay a special attention to the meaning of the sets of goals generated by the BFP* procedure. Consider for example the following set S of goals and the constraint E_1:
$S = \{p(a, x), q(x, y), r(y, d)\}$
$E_1 = p(a, b) \wedge q(b, c) \rightarrow$ false
S contains the goals of a partly expanded proof tree, which can be completed in many ways. In particular, further expansions of this proof tree may well bind x and y to other values than b, and c, respectively. Thus, the constraint E_1 is not sufficient for ruling out the "query" expression containing the set of goals S. On the contrary, a more general constraint such as
$E_2 = p(x, y) \wedge q(y, z) \rightarrow$ false
would be violated by S.

The following proposition formalizes and generalizes these remarks. It relies on the notion of existential closure of a set of atoms which we first define.

Definition 1: The *existential closure* of a finite set $S = \{L_1, \ldots, L_n\}$ of literals is the formula $EC(S) = \exists x_1 \ldots \exists x_k \ (L_1 \wedge \ldots \wedge L_n)$ where the x_is are the variables occurring in the L_js.

Proposition 3: Let KB = (F, DR, C) be a Horn knowledge base, Q a set of "query(G)" expressions where the Gs are atoms in the language $\mathcal{L}_{KB}$ of KB, n an integer, H an atom, and S a set of atoms such that $query(H, S) \in T_{BFP*} \uparrow^n(F \cup DR \cup Q)$. If $C \cup \{EC(S)\}$ is inconsistent, then for all atoms A in the language $\mathcal{L}_{KB}$ which unify with a goal in Q, we have:

$$fact(A) \in T_{BFP*} \uparrow^\omega (T_{BFP*} \uparrow^n(F \cup DR \cup Q) \setminus \{query(G, S)\})$$
$$\Longleftrightarrow$$
$$A \in T_{DR} \uparrow^\omega(F)$$

Proof: (sketched) By induction on k such that
$T_{BFP*} \uparrow^\omega(F \cup DR \cup Q) = T_{BFP*} \uparrow^k(F \cup DR \cup Q)$
one establishes that
$fact(A) \in T_{BFP*} \uparrow^n(F \cup DR \cup Q) \Leftrightarrow fact(A) \in T_{BFP*} \uparrow^\omega (T_{BFP*} \uparrow^n(F \cup DR \cup Q) \setminus \{query(G, S)\})$. The result follows by Proposition 2. □

By Proposition 3, the expressions "query(G, S)" such that the existential closure of S violates the constraints can be discarded while answering queries with the BFP* procedure without compromising completeness of the query answering procedure.

The formalism of deduction rules gives rise to specifying intuitively and elegantly when constraints and sets of goals are inconsistent.

$$\begin{array}{ll} \text{inconsistent}(S) & \leftarrow \text{constraint}(C) \wedge \text{violates}(S, C) \\ & \\ \text{violates}(S, \text{false}) & \\ \text{violates}(S, B \rightarrow \text{false}) & \leftarrow \text{satisfies}(S, B) \\ & \\ \text{satisfies}(S, B) & \leftarrow A \in S \wedge \text{instance}(B, A) \\ \text{satisfies}(S, B_1 \wedge B_2) & \leftarrow \text{relevant-conjunction}(B_1 \wedge B_2) \\ & \qquad \wedge \text{satisfies}(S, B_1) \wedge \text{satisfies}(S, B_2) \end{array}$$

The meta-predicate "constraint" denotes access to the set of constraints. We assume that, given two atoms A and B, "instance(B, A)" holds if there exists a substitution σ for variables in B – but not in A – such that $B\sigma = A$. The predicate "relevant-conjunction" aims at restricting the computation of satisfied conjunctive formulas to bodies of rules. This predicate can be specified as follows: The body of a deduction rule of the considered knowledge base is a relevant conjunction. A subformula of a relevant conjunction is also a relevant conjunction. This predicate can also be specified by means of meta-rules. We do not give such rules here, because reasoning backward on the rules for "satisfies" make the "relevant-conjunction" filter useless.

Let I denote the above-defined set of meta-rules for "inconsistent". By the following proposition, I is a sound and complete specification.

Proposition 4: Let KB = (F, DR, C) be a knowledge base and S a set of atoms in the language $\mathcal{L}_{\text{KB}}$ of KB. C $\cup$ {EC(S)} is inconsistent if and only if

$$\text{inconsistent(S)} \in \text{T}_{\text{I}} \uparrow^{\omega}(\{\text{S}\})$$

Proof: (sketched) Proposition 4 is implied by the soundness and completeness of binary resolution. The rationale of requiring "instance" not to bind variables occurring in its first argument is that variables in sets of goals S must be interpreted as existentially quantified. The conventional framework of binary resolution assumes these variables being represented by Skolem functions. □

Refining the redundancy condition of the naïve and semi-naïve method so as to discard "query(Q, S)" expressions such that "inconsistent(S)" holds gives rise to avoid unproductive computation without compromising the completeness of the BFP* procedure.

5. Detecting Redundancies

Some of the goals generated during backward reasoning are redundant. This is the case in particular when several deduction rules repeatedly specify some conditions. The additional knowledge provided by constraints can also be used for eliminating redundant goals.

We first consider intrinsic redundancies, i.e. redundancies that are independent from the constraints. We outline on an example how such redundancies can be detected and eliminated. We formalize the approach and specify it as an extension to the BFP* procedure. Then, we extend the approach to constraint dependent redundancies.

Consider the following knowledge base KB = (F, DR, $\emptyset$), the set Q of queries, and the sets S^n of facts generated at steps n of the BFP* procedure.

F: r(a) t(a)
t(b)

DR: $p(x) \leftarrow q(x) \wedge t(x)$
$q(x) \leftarrow t(x) \wedge r(x)$

Q: query(p(x))

S^1:	query(p(x),	{p(x)})
S^2:	query(q(x) $\wedge$ t(x),	{p(x), q(x), t(x)})
S^3:	query(q(x),	{p(x), q(x), t(x)})
	query(t(x) $\wedge$ r(x),	{p(x), q(x), t(x), r(x)})
S^4:	query(t(x),	{p(x), q(x), t(x), r(x)})

etc.

The goal "t(x)" is encountered at step 2 and once again at step 3, each occurrence resulting from a different deduction rule. There is of course no need to evaluate it twice. The conjunctive goal t(x) $\wedge$ r(x) generated from the expression "query(q(x), {p(x), q(x), t(x)})" can therefore be simplified into r(x).

It is worth noting that a strict instance of an already encountered goal in general cannot be discarded. Consider for example the same initial goal "p(x)" and assume that the predicates p and q are defined as follows:
$p(x) \leftarrow q(x) \wedge t(x)$
$q(x) \leftarrow t(a) \wedge r(x)$
It is possible that t(x) only holds for other substitutions than x = a. Therefore, discarding the goal "t(a)" could result in incorrect derivations. Only goals that are syntactically identical to previously generated goals can be discarded.

Multiple processing of redundant goals is avoided by modifying the third rule of the BFP* procedure into:

$$\text{query}(B, S_2) \leftarrow \text{query}(Q, S_1) \wedge \text{rule}(Q \leftarrow B) \wedge \text{simplify}(B, S_1, B_1) \wedge \text{merge}(S_1, B, S_2)$$

where the predicate "simplify(B, S_1, B_1)" holds if B_1 is the expression resulting from removing from B all literals belonging to the set S_1. (In case all literals in B are in S_1, B_1 is equal to "'true'": no "query" expression needs to be generated).

Let us denote BFP^+ the BFP* procedure modified as it is specified above. The following proposition establishes the soundness and completeness of the BFP^+ procedure.

Proposition 5: Let KB = (F, DR, C) be a Horn knowledge base, Q a set of "query(G)" expressions where the Gs are atoms in the language $\mathcal{L}_{KB}$ of KB, and A a ground atom in the language $\mathcal{L}_{KB}$. If there is an expression "query(G)" in Q such that A and G unify, the following equivalence holds:

$$\text{fact(A)} \in T_{BFP+} \uparrow^{\omega} (F \cup DR \cup Q) \Longleftrightarrow A \in T_{DR} \uparrow^{\omega}(F)$$

Proof: (sketched) Relying on Proposition 2, it suffices to prove that:
$\text{fact(A)} \in T_{BFP+} \uparrow^{\omega} (F \cup DR \cup Q) \Longleftrightarrow \text{fact(A)} \in T_{BFP*} \uparrow^{\omega} (F \cup DR \cup Q)$
This is done by induction on k such that
$T_{BFP*} \uparrow^{\omega} (F \cup DR \cup Q) = T_{BFP*} \uparrow^{k} (F \cup DR \cup Q)$. □

Constraints can sometimes be used for detecting redundancies. This is quite obvious for constraints with atomic heads. Such constraints could be used like deduction rules by the underlying, conventional query answering procedure. This is however often inefficient, because this results in re-evaluating these constraints instead of exploiting the additional knowledge they provide. Consider the following knowledge base KB = (F, DR, C) and the set Q of queries.

F:	q(a)	r(a)	t(a)	u(a)	v(a)
			t(c)	u(d)	v(e)

DR: $p(x) \leftarrow q(x) \wedge r(x)$
$s(x) \leftarrow u(x) \wedge v(x)$

C: $s(x) \wedge t(x) \rightarrow q(x)$

Q: query(p(x))

Processing the constraint like a deduction rule corresponds to computing the fixpoint
$T_{BFP*} \uparrow^{\omega}(F \cup DR \cup C \cup Q)$
instead of
$T_{BFP*} \uparrow^{\omega}(F \cup DR \cup Q)$
Both computations are listed below (for simplifying, the second argument of "query" expressions is omitted):

	$T_{BFP*} \uparrow^{\omega}(F \cup DR \cup C \cup Q)$	$T_{BFP*} \uparrow^{\omega}(F \cup DR \cup Q)$
S^1:	query(p(x))	query(p(x))
S^2:	query(q(x) $\wedge$ r(x))	query(q(x) $\wedge$ r(x))
S^3:	p(a)	p(a)
	query(q(x))	query(q(x))
	query(r(a))	query(r(a))
S^4:	query(s(x) $\wedge$ t(x))	
S^5:	query(s(x))	
S^6:	query(u(x) $\wedge$ v(x))	
S^5:	query(u(x))	
	query(v(x))	

This example clearly shows that generating goals by reasoning backward on the constraint is undesirable because useless "query" expressions are generated. Processing constraints with atomic heads like rules has another drawback. This can generate answers containing Skolem constants. Since Skolem functions usually have no meaning for the end user, this is in general considered undesirable. Constraints with atomic heads can nevertheless be used for speeding up the generation of answers. In presence of a constraint like

$s(x) \wedge t(x) \rightarrow q(x)$

the goal "q(x)" can be discarded as soon as both goals "s(x)" and "t(x)" have already been encountered.

Such redundant goals can be discarded by refining the above mentioned meta-predicate "simplify": "simplify(B, S_1, B_1)" should hold if B_1 is the expression resulting from removing from B all literals belonging to the set S_1 as well as all literals that are implied from S_1 by the constraints. Implied literals L can be specified by the following rules. (The meta-predicate "satisfies" has been defined in Section 4.)

$$\text{implies}(S, L) \leftarrow \text{constraint}(B \rightarrow L) \wedge \text{satisfies}(S, B)$$

Let BFP^{++} denote the resulting extension of the BFP^{+} procedure. Reasoning similarly as for proving Proposition 5, one can establish the soundness and completeness of the BFP^{++} procedure:

Proposition 6: Let KB = (F, DR, C) be a Horn knowledge base, Q a set of "query(G)" expressions where the Gs are atoms in the language $\mathcal{L}_{KB}$ of KB, and A a ground atom in the language $\mathcal{L}_{KB}$. If there is an expression "query(G)" in Q such that A and G unify, the following equivalence holds:

$$\text{fact(A)} \in T_{BFP^{++}} \uparrow^{\omega} (F \cup DR \cup Q) \Longleftrightarrow A \in T_{DR} \uparrow^{\omega}(F)$$

Constraints with disjunctive heads can also be used for answering disjunctive queries. In this article, we do not investigate this issue which involves classical theorem proving techniques. Constraint reasoning techniques do not seem to provide new insight on the subject.

6. Extension to Non-Horn Knowledge Bases

The semantics of knowledge bases has been extended in various manners to non-Horn knowledge bases – among other in [Apt88, GL88, Van88, Bry89]. The various proposals are quite different from each other and they rely on a great variety of formalisms. Moreover, for some of the proposed semantics, no backward reasoning query answering procedures have been proposed. Therefore, extending constrained query answering to non-Horn knowledge bases seems to require considering the various semantics the one after the other. In this section, we show that such a laborious approach can be avoided. We prove that the principles of constrained query answering extend to any semantics of non-Horn knowledge bases which conveys the notion of "supported deduction". We argue that this condition reflects the intuitive meaning of negation in knowledge bases and logic programs.

We first define a two-valued model $\mathcal{M}^{\mathcal{S}}_{KB}$ of a non-Horn knowledge base KB relatively to any semantics $\mathcal{S}$. This definition gives rise to defining the semantics of constraints in a uniform manner, whatever semantics is considered. Then, we extend the BFP and BFP* procedures to processing negation. We show that these extensions are correct for semantics reflects the notion of "supported deduction". Finally, we generalize to non-Horn knowledge bases the propositions establishing the correctness of constrained query answering.

A semantics for non-Horn knowledge bases defines "true" and "false" literals, like in the Horn case. This definition is achieved in different manners, depending on the semantics. This is sometimes explicit as in [Van88], by assigning one of the three truth-values "true", "false", and "unknown" to the atoms of the Herbrand base. This is in other cases done implicitly, through the definition of several, alternative models as in [GL88]: the atoms satisfied (falsified, resp.) in all models can be viewed as assigned the truth-value "true" ("false", resp.); atoms satisfied in some models, falsified in others, can be viewed as assigned the truth-value "unknown".

Given a non-Horn knowledge base KB = (F, DR, C) and a semantics $\mathcal{S}$ of non-Horn knowledge bases, we shall denote by $\mathcal{M}^{\mathcal{S}}_{KB}$ the set consisting of:

- The ground atoms A that are true in F ∪ DR according to $\mathcal{S}$;
- The negative literals ¬A for all ground atoms A that are false in F ∪ DR according to $\mathcal{S}$.

In contrast to the model $\mathcal{M}_H$ of a Horn knowledge base H, the model $\mathcal{M}^{\mathcal{S}}_{KB}$ of a non-Horn knowledge base KB is not necessarily complete: $\mathcal{M}^{\mathcal{S}}_{KB}$ might well contain neither A nor ¬ A for some ground atom A of the Herbrand base of KB. If the semantics $\mathcal{S}$ is specified in terms of distinct models, $\mathcal{M}^{\mathcal{S}}_{KB}$ can be seen as the agreement between these models. If $\mathcal{S}$ is defined in a three-valued logic, $\mathcal{M}^{\mathcal{S}}_{KB}$ is defined from the atoms assigned one of the truth values "true" and "false".

We say that a formula G is true or satisfied in KB relatively to a semantics $\mathcal{S}$ - noted $KB \vdash_{\mathcal{S}} G$ - if $\mathcal{M}^{\mathcal{S}}_{KB}$ is a model of G. Intuitively, if $\mathcal{S}$ is a three-valued semantics, $KB \vdash_{\mathcal{S}} G$ means that G is true in KB however the "unknown" information is interpreted. For a multiple model semantics $\mathcal{S}$, $KB \vdash_{\mathcal{S}} G$ holds if G is true in all $\mathcal{S}$-models of F ∪ DR.

The semantics of constraints is defined for a non-Horn knowledge base KB = (F, DR, C) relatively to a semantics $\mathcal{S}$ as in the Horn case by the condition:
$\forall G \in C \;\; KB \vdash_{\mathcal{S}} G$

The following definition formalizes the notion of "supported deduction".

Definition 2: Let KB = (F, DR, C) be a knowledge base and $\mathcal{M}^{\mathcal{S}}_{KB}$ its model relatively to a semantics $\mathcal{S}$. *Proof trees in* $\mathcal{M}^{\mathcal{S}}_{KB}$ are recursively defined as follows:
T is a proof tree of a ground literal L in $\mathcal{M}^{\mathcal{S}}_{KB}$ if $T = L \leftarrow T_1, \ldots T_n$ and if one of the following conditions holds:

- L is a negative literal, $n = 1$, $L \in \mathcal{M}^{\mathcal{S}}_{KB}$, and $T_1 = true$.
- L is a positive literal, $L \in F$, $n = 1$, and $T_1 = true$.
- There is a rule $H \leftarrow U_1 \wedge \ldots \wedge U_n$ in DR and a substitution σ such that $L = H\sigma$ and each T_i is a proof tree of $U_i\sigma$ in $\mathcal{M}^{\mathcal{S}}_{KB}$.

$\mathcal{S}$ is a *support-based semantics* if for all knowledge bases KB and all literals $L \in \mathcal{M}^{\mathcal{S}}_{KB}$ there are proof trees of L in $\mathcal{M}^{\mathcal{S}}_{KB}$.

Some semantics are not support-based. This is in particular the case if non-constructive

inference principles such as the excluded middle are considered, or if implications are not constructively interpreted.

A correct specification of the goal generation during backward reasoning requires to extend the Backward Fixpoint Procedure with the following meta-rule:

$$\text{query}(Q) \leftarrow \text{query}(\neg Q)$$

This rule ensures that the answers to Q, if there are some, will be generated, thus preventing wrong evaluations of $\neg Q$. Let BFP$^{\neg}$ denote the BFP procedure extended with this rule. The BFP* procedure requires a similar extension. Note that sets of goals occurring as second argument of a "query" expression must be correspondingly be modified:

$$\text{query}(Q, S_2) \leftarrow \text{query}(\neg Q, S_1) \wedge S_2 = (S \setminus \{\neg Q\}) \cup \{Q\}$$

Let BFP$^{\neg *}$ denote the BPF* procedure extended with this rule. The following generalization of Proposition 1 establishes the correctness of the BFP$^{\neg}$ and BFP$^{\neg *}$ procedures, provided the considered semantics is support-based. Given a semantics $\mathcal{S}$, a knowledge base KB = (F, DR, C) and a set Q of "query(G)" expressions where the Gs are literals in the language $\mathcal{L}_{\text{KB}}$ of KB, $\mathcal{M}^{\mathcal{S}}_{\text{BFP}^{\neg}}$(KB, Q) ($\mathcal{M}^{\mathcal{S}}_{\text{BFP}^{*\neg}}$(KB, Q), resp.) denotes the $\mathcal{S}$-model of the BFP$^{\neg}$ (BFP$^{*\neg}$, resp.) relatively to the knowledge base KB and the set Q of queries.

Proposition 7: Let KB = (F, DR, C) be a (possibly non-Horn) knowledge base, Q a set of "query(G)" expressions where the Gs are literals in the language $\mathcal{L}_{\text{KB}}$ of KB, and L a literal in the language $\mathcal{L}_{\text{KB}}$. If S is support-based and if there is an expression "query(G)" in Q such that L and G unify, the following equivalences hold:

$$\text{fact(L)} \in \mathcal{M}^{\mathcal{S}}_{\text{BFP}^{\neg}}(\text{KB, Q}) \Longleftrightarrow \text{fact(L)} \in \mathcal{M}^{\mathcal{S}}_{\text{BFP}^{*\neg}}(\text{KB, Q}) \Longleftrightarrow \text{L} \in \mathcal{M}^{\mathcal{S}}_{\text{KB}}$$

Proof: (sketched) By induction on the maximum depth n of the proof trees of the literal L, one first establishes the result for positive literals L. The result for negative literals then follows because of the second condition in the definition of support-based semantics. □

Reasoning similarly, one establishes the following result which generalizes Proposition 3. This result can be applied to ruling out inconsistent goals during answering queries posed to non-Horn knowledge bases.

Proposition 8: Let KB = (F, DR, C) be a (possibly non-Horn) knowledge base, Q a set of "query(G)" expressions where the Gs are literals in the language $\mathcal{L}_{\text{KB}}$ of KB, S a set of literals in $\mathcal{L}_{\text{KB}}$, and H a literal in $\mathcal{L}_{\text{KB}}$ such that:

$$\text{query(H, S)} \in \mathcal{M}^{\mathcal{S}}_{\text{BFP}^{*\neg}}(\text{KB, Q})$$

C ∪ {EC(S)} is inconsistent if and only if no proof trees of a literal in $\mathcal{M}^{\mathcal{S}}_{\text{KB}}$ contain instances of S.

The set I of meta-rules for detecting inconsistent goals given in Section 4 needs to be extended, in order to take negatives goals into account. More precisely, the definition of the predicate "violates" has to be extended with the following rules:

$$\begin{aligned}
\text{violates}(S,\ B \rightarrow H) \quad &\leftarrow \text{constraint}(B \rightarrow H) \\
&\quad \wedge\ \text{satisfies}(S,\ B) \wedge \text{violates}(S,\ H) \\
\text{violates}(S,\ H_1 \vee H_2) \quad &\leftarrow \text{relevant-disjunction}(H_1 \vee H_2) \\
&\quad \wedge\ \text{violates}(S,\ H_1) \wedge \text{violates}(S,\ H_2)
\end{aligned}$$

The predicate "relevant-disjunction" restricts the derivation of disjunctive formulas to conclusions of constraints. It is defined as follows: The conclusion of a constraint is a relevant disjunction. A subformula of a relevant disjunction is also a relevant disjunction. Let $\mathrm{I}^{\neg}$ denote the extension of I with the above-defined rules.

Proposition 9: Under the assumptions of Proposition 8, C ∪ {ES(S)} is inconsistent if and only if inconsistent(S) $\in \mathcal{M}^{\mathcal{S}}_{\mathrm{I}^{\neg} \cup \{\mathrm{S}\}}$

Proof: (sketched) Similarly as the proof of Proposition 4. □

The following result is a counterpart to Proposition 5. It gives rise to avoid multiple evaluations of redundant goals during a same proof.

Proposition 10: Under the assumptions of Proposition 8, if $x_1, \ldots, x_n$ are the variables occurring in H, $\forall x_1 \ldots \forall x_n$ (EC(S) ⇒ H) if and only if no proof trees of literals in $\mathcal{M}^{\mathcal{S}}_{\mathrm{KB}}$ contain instances of S and of the complement of H.

The predicate "implies" of Section 5 must be extended with the following rule, in order to correctly detect redundant goals:

$$\text{implies}(S,\ B \rightarrow H,\ A) \leftarrow \text{element-rest}(A,\ R,\ H) \wedge \text{satisfies}(S,\ B) \wedge \text{violates}(S,\ R)$$

The expression "element-rest(A, R, H)" holds if A is one of the disjunct of the disjunction H, and if R is the disjunction resulting form H once A is removed. (R reduces to "true" if H is an atom.) Reasoning like for proving Proposition 5 and 6, one establishes the soundness and correctness of the extended definition of "implies".

7. Concluding Remarks

Constrained query answering does not compromise the completeness of the underlying conventional answering procedure. Its constraining process is complete in the sense that an expression which is contradicted or subsumed by a constraint is not evaluated by the traditional answering procedure – provided this procedure is complete and terminates.

Although it performs deduction on general formulas – the constraints –, constrained query answering is neither a (yet another) "full theorem prover", nor a known theorem prover in

disguise. Constrained query answering relies on redundant knowledge – the constraints – for discarding unproductive evaluations. Classical theorem prover, in contrast, in general increase the number of tentative proofs when redundant information is added. Constrained query answering is however closely related with subsumption techniques that are implemented in some automated reasoning systems.

Constrained query answering extends the so-called "semantic query optimization" methods of relational databases, e.g. [Aho79, HZ80, Kin81a, Kin81b, SO87, SO89], because it handles deduction rules and it is complete. The approach outlined here differs from [LM88]: the meta-program given there transforms the deduction rules, while constrained query answering refines the reasoning performed on unmodified rules. The approaches described in [Cha84, Cha86, Cha88, PR89, Mot89, Lee91] also transform the rules through quite complex reasonings. It is not clear whether a conventional query answering procedure would suffice to implement them. Moreover, rule transformation methods do not seem to be applicable in presence of recursive deduction rules without compromising completeness of the constraining process [LH88, MY90].

It is interesting to compare constrained query answering with programming languages based on the paradigm of constraint [Lel87], in particular with constrained logic programming [JL87, Van89, Bür90] which inspired us. Two major differences should be noticed:

- The constraints in constrained query answering are general formulas.
- they are assumed to be implied by the facts and deduction rules.

This distinguishes our proposal from constraint logic programming, from the query languages discussed in [Kan90], as well as from the notion of exception proposed in [KS90].

It would be interesting to investigate whether the second of the above-mentioned assumptions can be relaxed. If this is the case, the relationship between the resulting approach to query answering should be compared with query answering methods for disjunctive databases and logic programs [Min90].

Acknowledgement

This work was supported in part by the ESPRIT Basic Research Action *Compulog* No. 3012. The author is grateful to Giuseppe Serrecchia, Jesper L. Träff, and Suryanarayana M. Sripada for comments and discussions.

References

[Aho79] A.V. Aho *et al.* Equivalences among Relational Expressions. *SIAM Jour. of Comput.*, 8, 1979.

[Apt88] K. R. Apt *et al. Foundations of Deductive Databases and Logic Programming*, chapter Towards a Theory of Declarative Knowledge. Morgan Kaufmann, Los Altos, Calif., 1988.

[Ban86] F. Bancilhon *et al.* Magic Sets and other Strange Ways to Implement Logic Programs. In *Proc. 5th ACM Symp. Principles of Database Systems*, 1986.

[BR87] C. Beeri and R. Ramakrishnan. On the Power of Magic. In *Proc. 6th ACM Symp. Principles of Database Systems*, 1987.

[Bry89] F. Bry. Logic programming as Constructivism: A Formalization and its Application to Databases. In *Proc. 6th ACM Symp. Principles of Database Systems*, 1989.

[Bry90] F. Bry. Query Evaluation in Recursive Databases: Bottom-up and Top-down Reconciled. *Data & Knowledge Engineering*, 5, 1990.

[Bür90] H. J. Bürkert. A Resolution Principle for Clauses with Constraints. In *Proc. 10th Int. Conf. Automated Deduction (CADE)*. Springer-Verlag, LNCS 449, 1990.

[Cha84] U.S. Chakravarthy *et al.* Semantic Query Optimization in Expert Systems and Database Systems. In *Proc. Int. Workshop Expert Database Systems*, 1984.

[Cha86] U.S. Chakravarthy *et al.* Semantic Query Optimization: Additional Constraints and Control Strategies. In *Proc. 1st Int. Conf. Expert Database Systems*, 1986.

[Cha88] U.S. Chakravarthy *et al.* Foundations of Semantic Query Optimization for Deductive Databases. In *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann, 1988.

[GL88] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. 5th Int. Conf. and Symp. Logic Programming*, 1988.

[HZ80] M. Hammer and S.B. Zdonik. Knowledge-based Query Processing. In *Proc. 6th Int. Conf. Very Large Data Bases*, 1980.

[JL87] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *Proc. 14th Annual ACM Symp. Principles of Programming Languages*, 1987.

[Kan90] P.C. Kanellakis *et al.* Constraint Query Languages. In *Proc. 9th ACM Symp. Principles of Database Systems*, 1990.

[Kin81a] J.J. King. Query Optimization by Semantic Reasoning. Technical Report STAN-CS-81-857, Stanford Univ., Dept. of Computing, 1981.

[Kin81b] J.J. King. QUIST: A System for Semantic Query Optimization in Relational Databases. In *Proc. 7th Int. Conf. Very Large Data Bases*, 1981.

[KS90] R. Kowalski and F. Sadri. Logic Programs with Exceptions. In *Proc. 7th Int. Conf. Logic Programming*, 1990.

[Lee91] S. Lee *et al.* Semantic Query Reformulation in Deductive Databases. In *Proc. 7th IEEE Int. Conf. Data Engineering*, 1991. To appear.

[Lel87] W. Leler. *Constraint Programming Languages.* Addison Wesley, 1987.

[LH88] S. Lee and J. Han. Semantic Query Optimization in Recursive Databases. In *Proc. 4th IEEE Int. Conf. Data Engineering*, 1988.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987. 2nd edition.

[LM88] J. Lobo and J. Minker. A Metaprogramming Approach to Semantically Optimize Queries in Deductive Databases. In *Proc. 2nd Int. Conf. Expert Database Systems*, 1988.

[Min90] Minker, J. A Fixpoint Semantics for Disjunctive Logic Programs. *Jour. of Logic Programming*, 9, 1990.

[Mot89] A. Motro. Using Integrity Constraints to Provide Intensional Responses to Database Queries. In *Proc. 15th Int. Conf. Very Large Data Bases*, 1989.

[Mot90] A. Motro. Intensional Answers to Database Queries. Unpublished manuscript, 1990.

[MY90] A. Motro and Q. Yuan. Querying Database Knowledge. In *Proc. 15th Int. Conf. Management of Data*, 1990.

[PR89] A. Pirotte and D. Roelants. Constraints for Improving the Generation of Intensional Answers in a Deductive Database. In *Proc. 5th IEEE Int. Conf. Data Engineering*, 1989.

[Ram88] Ramakrishnan, R. Magic Templates: a Spellbinding Approach to Logic Programs. In *Proc. Int. Conf. and Symposium on Logic Programming*, 1988.

[Roh86] J. Rohmer *et al.* A Technique for the Processing of Recursive Axioms in Deductive Databases. *New Generation Computing*, 4(3), 1986.

[Sie88] M.D. Siegel. Automatic Rule Derivation for Semantic Query Optimization. In *Proc. 2nd Int. Conf. Expert Database Systems*, 1988.

[SO87] S.T. Shenoy and Z.M. Ozsoyoglu. A System for Semantic Query optimization. In *Proc. ACM-SIGMOD Int. Conf. Management of Data*, 1987.

[SO89] S.T. Shenoy and Z.T. Ozsoyoglu. Design and Implementation of a Semantic Query Optimizer. *IEEE Trans. on Knowledge and Data Engineering*, 1(3), 1989.

[TS86] H. Tamaki and T. Sato. OLD Resolution with Tabulation. In *Proc. 3rd Int. Conf. Logic Programming*, 1986.

[Van88] A. Van Gelder *et al.* The Well-Founded Semantics for General Logic Programs. In *Proc. 7^{th} ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS)*, Austin, Texas, March 1988.

[Van89] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[Vie89] L. Vieille. Recursive Query Processing: The Power of Logic. *Theoret. Comp. Sc.*, 69(1), 1989.

[YS89] C.T. Yu and W. Sun. Automatic Knowledge Acquisition and Maintenance for Semantic Query Optimization. *IEEE Trans. on Knowledge and Data Engineering*, 1(3), 1989.