Intensional answers are conditions that tuples of values must satisfy to belong to the usual extensional answer of a query addressed to a deductive database. This paper shows how intensional answers can be generated as logical consequences of the query and of deduction rules, and how integrity constraints can filter out inadequate answers and produce simpler and more informative answers. It also presents an efficient organization for the combination of answers and constraints.

1 Introduction

A deductive database comprises an extensional part and an intensional part. The extensional part contains explicit tuples of base relations while the intensional part, which generalizes the schema of traditional databases, comprises deduction rules and integrity constraints. Deduction rules, which generalize the views of traditional databases, specify sets of tuples for virtual relations in terms of extensional relations and of other virtual relations. Integrity constraints state conditions that the extension of virtual and base relations must satisfy.

Rules, constraints, and queries are written in a database language, typically based on a usual first-order language with a predicate associated with each relation (base or virtual) of the database. A query to the database is a formula of the database language with free variables which are called query variables. In this paper, queries are restricted to conjunctions of the form:

\[ Q(\vec{x}) = \exists \vec{y} (Q_1(\vec{x},\vec{y}) \land \ldots \land Q_n(\vec{x},\vec{y})) \]

where \( \vec{x} \) denotes the query variables and where each \( Q_i(\vec{x},\vec{y}) \) \( (1 \leq i \leq n) \) is a positive literal. The usual extensional answer to a query is the set of tuples \( \vec{a} \) of the extensional database such that \( Q(\vec{a}) \) can be shown to be true when the deduction rules are taken into account to define the virtual predicates in \( Q \).

An intensional answer is a formula of the database language that states a condition to be satisfied by tuples of the extensional database in order to be part of the extensional answer. Intensional answers are obtained by replacing, in the query or in already obtained intensional answers, a virtual predicate by its definition in the deduction rules.

In [10], we present a precise definition of intensional answers and argue that, instead of trying to generate a complex intensional answer that would be equivalent to the query, a better strategy is to construct a collection of simpler partial intensional answers, each of which states a sufficient condition for a set of facts to belong to the extensional answer. The completeness of a set of answers is also discussed in [10].

An important aspect of intensional answers is that they are constructed without accessing the extensional database. Since the latter is typically much larger than the intensional database, they can be constructed faster and more cheaply than the extensional answer. Also, it is usually the case that the extensional database changes more often through updates of individual facts than the intensional database. Therefore, intensional answers provide a more stable answer to a query than the extensional answer. Still, an intensional answer is similar to a query and, if desired, it can be evaluated against the extensional database like a query and return a partial extensional answer.

The essential contribution of this paper is the design of mechanisms for controlling the generation of intensional answers through integrity constraints. The assertions specified by constraints bear on the database but also on queries and intensional answers, since the latter specify sets of values from the database. An intensional answer has an interesting interaction with a constraint only if both have predicates in common.

Example 1. In the presence of the following constraints:

\[ y < 20 \rightarrow emp(y) \land sal(y) \]
\[ \exists y (sal(y) \land y < 20) \rightarrow emp(y) \]

meaning, respectively, "all salaries of employees are lower than 20" and "all employees have salaries", the query ("the employees with a salary lower than 20")

\[ emp(y) \land \exists y (sal(y) \land y < 20) \]

yields the simple intensional answer ("all the employees")

\[ emp(y) \]

If a constraint contains a virtual predicate, then the deduction rules defining the virtual predicate somehow supplement the constraint. Another constraint may be deduced by replacing the virtual predicate in the first constraint by its (partial) definition according to the deduction rule. Both constraints are candidates for interacting with an intensional answer depending on which predicates are present in the answer.

Example 2. If we add the following rule to Example 1:

\[ emp(y) \rightarrow tec(y) \]

("all technicians are employees"), then the following constraint may be derived:

\[ y < 20 \rightarrow tec(y) \land sal(y) \]

meaning that "all salaries of technicians are lower than 20".

Thus, controlling the generation of intensional answers with constraints inherently leads to handling large sets of constraints and, therefore, to the necessity of managing them tightly.
The paper is structured as follows. Section 2 makes a number of definitions and assumptions precise. Section 3 recalls the principle of generating candidate intentional answers from logical consequences of the intentional database and of the query. Section 4 introduces the role of constraints, while Section 5 is devoted to the efficient management of the large set of constraints needed by the method. The combination of answers and constraints is discussed in Section 6. Section 7 briefly discusses strategies for presenting intentional answers to users. Section 8 surveys related work and further developments.

2 Definitions and Assumptions

Usual definitions for deductive databases are assumed [6] [7] [11]. Specifically, for this paper, a deductive database consists of:

- a usual first-order logical language, with variables, constants, predicate symbols, quantifiers, logical connectives, and the usual first-order theory with equality and proof system supporting the language;
- a set of relations or ground base predicates, the extensional database or EDB; there are no mixed predicates (i.e., the sets of base and virtual predicate symbols are disjoint);
- evaluable predicates, limited to comparisons (=, <, <=, >, >=), with the usual restriction that every variable appearing in a comparison predicate must also appear in a positive noevaluable predicate of the same query or of the body of the same rule or constraint. This restriction can be waived in most cases as intentional answers have no possibility of making the extension of comparisons explicit;
- a set DR of nonrecursive deduction rules of the form H ⊃ B, where H, the head or consequent, is a positive virtual predicate and B, the body or antecedent, is a conjunction of virtual and/or base predicates and/or comparisons. In this paper, noevaluable predicates in B are supposed to be positive (Horn form). Issues raised by the introduction of negated literals are discussed in [10]. The only terms allowed in rules are constants and variables. Rules are assumed to be range-restricted, that is, every variable occurring in H also occurs in B;
- a set IC of integrity constraints of the form H ⊃ B or ⊃ B or H ⊃ , where H is not required to be virtual. Constraints are also usually required to be range-restricted, although this is not fundamental here, as we neither control updating nor check that constraints are satisfied. The clause form of constraints are allowed to contain Skolem functions, resulting from existential variables occurring in H and not in B;
- a set CR of completion rules of the form Bi U ... U Bk H H H H Bk for virtual predicates, to make explicit that they are completely defined by the set of deduction rules {H = B1, ..., H = Bk}. The form of completion rules is different from that of deduction rules. They are used differently also.

The intentional database IDB is the set DR U IC U CR. In this paper, queries are restricted to be conjunctions of positive literals.

The other notations should be obvious. For rules and constraints, we mostly use a Prolog-like rule notation and sometimes switch to a clause notation when more convenient. For generic examples, we use a vector notation to denote tuples of variables and tuples of constants. For examples in the database language, lower-case letters are used for predicate names and variables, while strings denote constants.

3 Basic Method for Generating Answers

An intentional answer to a query Q(\vec{z}) is a formula A(\vec{z}) with free variables \vec{z}. When evaluated against the EDB, it returns a set of values that satisfies the query. As shown in [10], the basic condition to be satisfied by an intentional answer is:

\[ IDB \vdash A(\vec{z}) \supset Q(\vec{z}). \]

or, more precisely, if existential variables are made explicit:

\[ IDB \vdash (\exists \vec{y} \ A(\vec{x}, \vec{y})) \supset (\exists \vec{y} \ Q(\vec{x}, \vec{y})) \]

Thus, according to (1), answers A are negations of logical consequences of the IDB and the negation of the query Q, with the extra requirement that the query variables \vec{z} must all occur in A (and, in particular, that A is not derived from the IDB alone).

With resolution, intentional answers are obtained as the negation of resolvents R(\vec{x}, \vec{y}) obtained from clauses in IDB U \{¬Q(\vec{x}, \vec{z})\}.

Condition (1) states a necessary condition that any intentional answer must satisfy, but it is not restrictive enough to serve as a constructive characterization of a useful set of answers. Further selection in the set of candidate answers must take place to filter out several kinds of useless or inadequate answers.

First, some candidate answers (and also some queries) have an empty extension when they are evaluated against the EDB (i.e., \{\vec{a} | EDB \cup IDB \vdash A(\vec{b})\} = \emptyset). This property may depend on the particular extension of the EDB. But if an answer is empty regardless of the EDB and if this can be derived from the constraints, this is useful intentional information to make explicit.

Example 3. If, with the constraint and the deduction rule of Examples 1 and 2, we ask the following query ("the employees with a salary higher than 20"):

\[ \text{emp}(x) \land \exists y \ (\text{sal}(x,y) \land y > 20), \]

then the following intentional answer:

\[ \text{tec}(x) \land \exists y \ (\text{sal}(x,y) \land y > 20), \]

is always empty when evaluated against the EDB.

A second class of uninteresting intentional answers are those that can be shown to be less general or less informative than other intentional answers. This happens when an answer is subsumed by another one (e.g., P(\vec{z}) subsumes P(\vec{z}) \land Q(\vec{z})).

Thus, a basic method for generating intentional answers is as follows [3]:

1. generate logical consequences (e.g., resolvents) from the IDB and the negation of the query; their negations A(\vec{z}) are candidate answers;
2. simplify simple syntactic redundancies (e.g., replace P(\vec{z}) \land P(\vec{z}) by P(\vec{z}) and remove answers that are simple syntactic variants of other answers (e.g., differing by variable names or order of literals);
3. filter out empty answers, i.e., answers \(A(\emptyset)\) for which \(IDB \vdash \neg(\exists \emptyset A(\emptyset))\);
4. filter out subsumed answers, i.e., answers \(A_2(\emptyset)\) such that there exists another answer \(A_1(\emptyset)\) with \(\vdash \forall \emptyset (A_1(\emptyset) \supset A_2(\emptyset))\).

This basic method can still be considerably improved as shown in the next section.

4 Improving the Generation of Answers with Constraints

4.1 The Role of Constraints

Major improvements are obtained by having constraints and deduction rules play very different roles in the generation process.

Constraints specify assertions that virtual, base, and evaluable predicates must satisfy. These assertions also apply to queries and intensional answers, since they specify sets of tuples of values from the database. Constraints can improve the generation and the adequacy of intensional answers in several ways. We will see that they can help in:

- identifying empty answers and queries. This can be done earlier and more efficiently than in the basic method of the previous section. In particular, no new answer should be generated from an empty answer;
- improving the quality of answers, by not generating some inadequate answers. This also improves efficiency by shortening the deduction process;
- transforming answers with constraints. In some cases, this amounts to a simplification as was illustrated by Example 1. In other cases, as we will see in Section 6, constraints supply information that makes answers more precise.

The first major improvement to the basic method is obtained by generating candidate answers from the query with deduction rules only and by checking them for emptiness with integrity constraints. Thus, steps (1) and (3) are modified as follows:

1. generate logical consequences from the deduction rules \(DR\) and the negation of the query; their negations \(A(\emptyset)\) are candidate answers;
3. filter out empty answers, i.e., those for which \(IC \vdash \neg(\exists \emptyset A(\emptyset))\).

The basic method checks anew, for each candidate answer generated, whether its extension is empty according to the constraints and whether it is not subsumed by other simpler candidate answers. This may become very expensive with a large \(IDB\). Indeed, a lot of work is duplicated. When, an intensional answer is constructed from another answer and from a deduction rule, only a part of the new intensional answer is different from its parent answer. Thus, another major improvement is achieved by an incremental approach, where each deduction step is accompanied by a part of the check for subsumption and for nonemptiness according to constraints. As will be seen, much of the work can be performed statically before starting the answer generation process.

4.2 "Missing" Constraints

As shown in Example 2, new, more specific constraints may be derived from deduction rules and initial constraints \(IC\) in the \(IDB\). These new constraints are needed to detect emptiness of some answers (or queries).

Example 4. The derived constraint of Example 2 is needed to prove that the following answer (or query) is empty:

\[\text{tec}(z) \land \text{sal}(x,y) \land y > 20\]

("the technicians whose salary is higher than 20"). This could not be proved from the initial constraint of Example 1. •

In more general terms, the detection of empty intensional answers by constraints is expressed by saying that if \(A(\emptyset)\) is nonempty, then \(\exists \emptyset A(\emptyset)\) must be consistent with the constraints \(IC\) or, equivalently, that if \(\exists \emptyset A(\emptyset)\) is inconsistent with the constraints, then \(A(\emptyset)\) is empty. Derived constraints are necessary because the converse of this property is not necessarily true; namely, if \(A(\emptyset)\) is consistent with the initial constraints \(IC\), it cannot be concluded in general that it is nonempty or, equivalently, if \(A(\emptyset)\) is indeed empty according to the \(IDB\), then it is not necessarily inconsistent with the initial constraints alone.

There is another case where an answer cannot be directly proved inconsistent with the initial constraints, although its extension is in fact always empty. In a sense, it is the opposite of the situation above where a specialization of a constraint was needed to show inconsistency. A generalization of constraints may also be needed, as shown by the following example.

Example 5. Let the emp predicate be defined by the rules:

\[\text{emp}(z) \leftarrow \text{tec}(z)\]
\[\text{emp}(z) \leftarrow \text{admin}(z)\]

with the constraints:

\[y < 20 \leftarrow \text{tec}(z) \land \text{sal}(x,y)\] (2)
\[y < 20 \leftarrow \text{admin}(z) \land \text{sal}(x,y)\] (3)

The answer

\[Q(z) \equiv \text{emp}(z) \land \text{sal}(z,y) \land y > 20\]

cannot be proved inconsistent with the constraints, although it is indeed empty if "technicians" and "administratives" are the only "employees". To be able to conclude that "all employees have a salary lower than 20", that is:

\[y < 20 \leftarrow \text{emp}(z) \land \text{sal}(z,y)\] (4)

the following completion rule for predicate emp is necessary:

\[\text{admin}(z) \lor \text{tec}(z) \rightarrow \neg \text{emp}(z)\]. •

This situation suggests a better database design, where explicit constraints would have been formulated at a higher level of generality, (4) in Example 5 instead of (2) and (3), but database design is not the point of the discussion.

In general, consider the answers obtained from a given intensional answer \(A \equiv A_1 \land \ldots \land A_n\) by applying all the deduction rules defining a virtual predicate \(A_i\) of \(A\). If all these descendant answers are empty according to the constraints, it can be concluded that the parent \(A\) is also empty. The "missing" constraints can be deduced from existing constraints, provided a completion rule is added for each virtual predicate whose definition involves a predicate that appears in the body of a constraint.
4.3 Preprocessing Constraints

For obvious reasons of efficiency, the derivation of the new constraints is best done statically, before starting the answer generation process. The enlarged set of constraints, denoted $IC^+$, can be preserved as long as the IDB is not changed. It is built as follows from the initial set $IC$:

- **process constraints so that there exists a formulation of constraints at all levels of generality, namely**
  - generate specialized constraints at all intermediate levels by applying deduction rules to replace virtual predicates in the body of constraints;
  - generalize constraints as far as possible, with the help of completion rules for virtual predicates;
- **combine constraints together in all possible ways (e.g., by resolution on non-evaluable predicates) to make deducible constraints explicit**;
- **simplify constraints for combinations of comparison predicates (see Section 6)**;
- **remove any constraint that is subsumed by another constraint or that is a tautology**.

Then the following property holds.

**Property.** $A(E)$ is an empty intensional answer iff a contradiction can be deduced from $\{E A(E) \cup IC^+\}$.

In general, $IC^+$ is a large set but, with the mechanism of reduced constraints that will now be described, it can be indexed in such a way that only a small portion of it ever has to be searched when matching intensional answers with constraints.

5 Reduced Constraints

5.1 Relevant Constraints

In order to enhance the efficiency of answer computation, a (normally small) subset of relevant constraints is attached to each deduction rule and to each intensional answer. When a deduction rule and an answer are combined to produce a new answer, the constraints relevant to the new answer are computed from the constraints attached to the parent answer and those of the deduction rule.

A constraint is relevant to an intensional answer if it expresses some property of a literal of the intensional answer. More precisely, a constraint $C$ of the form $\forall x \exists y (R(x, y))$ is relevant to an answer $A$ if the answer $A$ is obtained from $A$ by replacing the literals in the body of $C$ by those in $A$. In that case, if $\theta$ is the unifying substitution, $C$ expresses that the property $(\forall x \exists y (R(x, y)))$ is true whenever $\theta$ is true.

**Example 6.** Given an answer

$$A(z) \equiv \text{tec}(z)$$

and a constraint

$$C(z, v) \equiv (v < 20 \iff \text{tec}(z) \land \text{sal}(z, v)),$$

the literal $\text{tec}(z)$ in $C(z, v)$ subsumes $\text{tec}(z)$ in $A(z)$ with substitution $\{z/2\}$. Hence, the constraint expresses that, "when talking about technicians, salaries are lower than 20".

Note that subsumption in the above definition cannot be replaced by unification, as this would treat as relevant, in general, a constraint that expresses a property of an instance of a literal of the answer, and not of the literal itself.

The specific property about the answer expressed by a relevant constraint is obtained by replacing the constraint with the answer without instantiating variables of the answer. This resolved, called reduced constraint of the answer, gives a simplified version of the constraint that holds when the answer is assumed. This is expressed by the following relationship:

$$C(x, y) \land A(x, y) \supseteq C(y, z).$$

In Example 6, the reduced constraint of $C(z, v)$ for $A(z)$ is $R_{\text{tec}}(z, v) \equiv (v < 20 \iff \text{sal}(z, v))$.

General resolution is a complex and expensive process. Due to the simple form of the intensional answer (a conjunction of positive literals), resolution of an answer and a constraint amounts to suppressing literals of the answer in the body of an instance of the constraint. Therefore, reduced constraints can be constructed efficiently.

Similarly, relevant constraints or their reduced versions can be attached to individual literals and to the body of deduction rules. The set of reduced constraints of a formula $F$ is noted $IC(F)$. The next section shows how to construct the reduced constraints of an intensional answer from the reduced constraints of individual literals, of deduction rules, and of previously constructed answers.

5.2 Construction of Reduced Constraints

As shown in the previous section, computing the reduced constraints of an intensional answer involves the identification of the constraints in $IC^+$ that are relevant to the answer and a reduction of relevant constraints pinpointing the actual assertions that supplement the answer.

The selection of relevant constraints can be achieved by indexing $IC^+$ for each deduction rule and each answer. Suppressing literals in relevant constraints to construct the reduced constraints is not essential in this indexing process, but, as will be seen, this helps visualize the incremental construction and simplification of reduced constraints as well as the transformation of answers by constraints.

Computing from scratch the set of reduced constraints every time a candidate answer is generated would be very inefficient. An incremental construction of reduced constraints is possible as follows. First, the reduced constraints associated with a query $Q$ are computed from the reduced constraints of the individual literals of $Q$. If an instance of a given literal occurs in an answer, then the corresponding instances of the reduced constraints of the literal are attached to the answer. Second, when an answer is combined with a deduction rule to produce a new answer, the reduced constraints of the new answer are constructed systematically from the reduced constraints of the old one and from those of the deduction rule.

Furthermore, the computation can be split into a static phase and a dynamic phase. The static phase computes once and for all from the IDB the reduced constraints of individual literals that can appear in queries and the reduced constraints of deduction rules. In the dynamic phase, the reduced constraints of a query (the query itself is the first answer) are computed from the reduced constraints of its literals. Then, the reduced constraints of every new answer are computed as the answer is generated. These computations are now described in detail.
5.2.1 Expansion of Constraints for Evaluable Predicates

A special treatment of constants occurring in constraints increases the efficiency of a systematic computation and use of reduced constraints.

**Example 7.** Consider the constraint ("all employees working in the personnel department are administrative")

\[ C \equiv (\text{admin}(z) \leftarrow \text{worksin}(z, \text{Pers})). \]

The literal \( \text{worksin}(z, y) \) has no reduced constraint with \( C \), while \( \text{admin}(z) \) is a reduced constraint of \( C \) for \( \text{worksin}(z, \text{Pers}) \).

This raises two kinds of problems with a query that contains the literal \( \text{worksin}(z, y) \). First, as discussed in Section 7, the strategy for generating answers may wish, when faced with an answer containing the literal \( \text{worksin}(z, y) \), to specialize the answer to the employees of the personnel department. Then, the constraint becomes relevant for the literal \( \text{worksin}(z, y) \). Second, variable \( y \) may become instantiated to \( \text{Pers} \) later in the answer generation process, and then again, the constraint becomes relevant to the answer. If no change is made to the construction of reduced constraints, constraint \( C \) must be re-examined every time a specialization of an answer is considered and every temporary variable \( y \) is instantiated.

A more efficient solution consists in extracting the reduced constraint \( \text{admin}(z) \leftarrow y = \text{Pers} \) for \( \text{worksin}(z, y) \). It contains all the information from the constraint that will ever be usable by the literal or its instances and thus re-examining the constraint later will never be necessary. For such a reduced constraint to be obtainable, constraint \( C \) is rewritten in an expanded form as follows:

\[ \text{admin}(z) \leftarrow \text{worksin}(z, y) \land y = \text{Pers}. \]

Let us call \( IC^+ \) the set of constraints thus obtained by rewriting the constraints of \( IC^* \) so that no constant appears in nonevaluable predicates of the body of a constraint and no variable appears more than once in nonevaluable predicates of the body of a constraint. The transformation of constraints from \( IC^* \) to \( IC^+ \) and its reverse raise no special problems and their description is omitted for brevity. The reverse transformation has to be preceded by a simplification of the constraints (see Section 5.2.5). The expanded form is useful only for the construction of reduced constraints. Before combining them with intensional answers (see Section 6), constraints are reexpressed as far as possible in unexpanded form.

Constraints in \( IC^+ \) form enjoy the following property (which is not true in general for the unexpanded constraints of \( IC \) or \( IC^* \)).

**Property.** If \( L \) is a literal or a conjunction of literals, then the reduced constraints of an instance of \( L \) are the corresponding instances of the reduced constraints of \( L \). Formally:

\[ IC^+_L = (IC^+_C)^\sigma \]

where \( IC^+_C \) denotes the set of reduced constraints of \( L \) and \( \sigma \) is a substitution.

5.2.2 Reduced Constraints of Individual Literals

Given a positive nonevaluable literal \( L \) and a constraint \( C \) in clause form, \( L \) has a reduced constraint with \( C \) if \( \neg L \) unifies with a literal of \( C \) without instantiating the variables of \( L \). Reduced constraints are obtained by removing one or more unifying literals in \( C \) and performing the unifying substitution in the remainder of \( C \). In other words, a set of reduced constraints is obtained as follows:

- instantiate \( L \) to a ground literal \( L\theta \);
- resolve \( L\theta \) with \( C \) in all possible ways;
- apply the inverse substitution \( \theta^{-1} \) to the resolvent;
- remove subsumed reduced constraints;
- iterate for each of the reduced constraints thus obtained.

If \( \neg L \) does not unify with any literal of \( C \), then \( L \) has no reduced constraint with \( C \). The set of reduced constraints of evaluables literals (i.e., comparisons) is considered to be empty.

5.2.3 Reduced Constraints of Answers

The reduced constraints \( IC_A \) associated with an intensional answer \( A \) can be obtained from the reduced constraints associated with the individual literals in the answer or, if the answer was obtained from another answer and a deduction rule, from the reduced constraints of the parent answer and those of the deduction rule.

From the reduced constraints of individual literals Let \( A = L_1 \land \ldots \land L_m \), then \( IC_A \) can be constructed as follows:

- include in \( IC_A \) the reduced constraints \( IC_{L_i} \) of each \( L_i \);
- if a reduced constraint \( C \) resolves with a ground instance \( L_i\theta \) of an \( L_i \) giving a resolvent \( R \), then add \( R\theta^{-1} \) to \( IC_A \) (\( R\theta^{-1} \) is obtained by suppressing \( \neg L_i\theta \) from \( C\theta \) and performing the inverse \( \theta^{-1} \) of the unifying substitution in the resulting clause);
- remove subsumed constraints from \( IC_A \).

From the reduced constraints of its parents If \( A \) is obtained from another answer \( A_1 = L_1 \land \ldots \land L_n \) and a deduction rule \( H \leftarrow B_1 \land \ldots \land B_k \) such that \( H \) unifies with a ground instance of \( L_i \), then the reduced constraints of \( A \) are the reduced constraints of \( A_1 \) together with the reduced constraints of the deduction rule, that is, of \( B_1 \land \ldots \land B_k \) (see below).

The set thus obtained is then augmented by attempting to resolve each reduced constraint of the deduction rule with a literal of \( A_1 \) as for the construction from individual literals.

5.2.4 Reduced Constraints of Deduction Rules

For each deduction rule \( D \) of the form \( H \leftarrow B_1 \land \ldots \land B_n \), the set of reduced constraints of \( D \) is the set of reduced constraints of the body \( B_1 \land \ldots \land B_n \), since the virtual predicate \( H \) is resolved away when the rule is applied. It is computed from the reduced constraints of the individual literals \( B_1, \ldots, B_n \) like the reduced constraints of candidate answers.

5.2.5 Simplification of Reduced Constraints

The expansion of constraints has introduced extra explicit comparison predicates which may get instantiated when reduced constraints are computed. For example, the arguments of a comparison may become constants, in which case the comparison can always be evaluated. Therefore, every newly constructed reduced constraint must be tested for a number of possible simplifications (see [10] for details).
6 Combination of Answers and Reduced Constraints

When generating intensional answers, each time a new answer and its associated reduced constraints are obtained, the presence of a contradiction in the set of reduced constraints is tested and a number of transformations are attempted on the answer. This section describes the possible combinations of answers and constraints.

6.1 Checking for Emptiness

An intensional answer is empty if the empty clause is one of its reduced constraints, since, in that case, the answer and the original constraints imply a contradiction.

Contradictions can also be deduced from comparison predicates with numerical arguments, as in the following example.

Example 8. With the constraint
\[ y < 20 \rightarrow \text{emp}(x) \land \text{sal}(x, y), \]
the intensional answer
\[ \text{emp}(x) \land \text{sal}(x, y) \land y > 30 \]
is empty although this can be detected only by taking into account the meaning of comparison predicates. In effect, the reduced constraint of the query is:
\[ y < 20 \]
and, by the first transformation schema below (Section 6.2), the following is obtained as an answer:
\[ \text{emp}(x) \land \text{sal}(x, y) \land y > 30 \land y < 20. \]

Starting from [12] (see also [9]), a general method was designed for combining arbitrary comparisons and testing their inconsistency [10].

6.2 Transforming Answers

6.2.1 General Transformation Schemas

The transformations that we have considered for intensional answers fall under three general schemas. If \( A(\xi) \) is an intensional answer and \( C \) a reduced constraint of \( A \) in clause form, then, provided that \( A_1(\xi) \) actually contains all the query variables of \( A \) and that no Skolem functions appears in it, each of the following three schemas produces another valid intensional answer \( A_1(\xi) \):

1. \[ A_1(\xi) = A(\xi) \land C. \]
2. \[ A_1(\xi) = A(\xi) \land C_1, \] for any \( C_1 \) such that \( C_1 \Rightarrow C \).
3. \[ A_1(\xi) = \text{the negation of any resolvent of } \neg A \land C. \]

The logical relationship between \( A \) and \( A_1 \) is:
\[ IDB \vdash A(\xi) \equiv A_1(\xi) \]
for the first schema, while, for the latter two, it is:
\[ IDB \vdash A_1(\xi) \supset A(\xi). \]

Several transformations can be applied successively to an answer. Obviously, not every application of the general schemas produces an interesting answer. In their general form, the first (\#1) schemas produce answers whose form is no longer a conjunction of literals.

Exactly which cases should be selected depends on user requirements (as discussed in Section 7). It may or may not be acceptable, for example, to output answers more complex than a conjunction of literals. The following sections present a number of special cases of transformation worth examining in more detail.

6.2.2 Removal of Literals

Applying Schema 3 is especially interesting when \( C \) consists of a single literal, because the transformation then simplifies an answer of the form \( A = B \land C' \) by suppressing the literal \( C' \). Then, \( IDB \vdash B \supset A \).

Equivalence (i.e., \( IDB \vdash B \equiv A \)) holds at the condition that no variable of \( A \) gets instantiated in \( B \). In other words, \( C \) must subsume \( C' \) as far as variables of \( B \) are concerned. For variables occurring in \( C' \) and not in \( B \) (and thus distinct from query variables), it is sufficient to require that they unify with the corresponding terms in \( C \). In other words, since these variables are existentially quantified, the parent terms may be less general in the constraint than in the answer, while the subsumption condition requires the opposite of the other terms. This is useful in getting rid of Skolem functions in some constraints, like:

\[ \forall z \exists y \text{emp}(z, y) \leftarrow \text{emp}(x) \]
that is, in clause form
\[ \text{sal}(x, f(x)) \leftarrow \text{emp}(x) \]
("every employee has a salary"). Thus, the condition that \( C \) and \( C' \) must satisfy is \( C \sigma \theta = C' \sigma \theta \) where \( \theta \) is a unifying substitution for variables \( \forall \) (not occurring in \( B \)), and \( \sigma \) is the subsuming substitution for the other variables.

6.2.3 Instantiating Answers

An obviously interesting case of application of Schema 1 is when the reduced constraint \( C \) reduces to the form \( y = t \), where \( y \) is a variable of \( A \) and \( t \) is another variable of \( A \) or a constant. If \( y \) is an existential variable of \( A \) (i.e., not a query variable), then the instance \( A(y/t) \) is an equivalent answer. If \( y \) is a query variable, then \( y = t \) must stay as a new literal in the answer.

6.2.4 Combining Comparisons

Applying Schema 1 is also interesting when the reduced constraint \( C \) exclusively consists of comparisons.

Example 9. Consider the constraint
\[ y < 20 \rightarrow \text{emp}(x) \land \text{sal}(x, y) \]
and the answer
\[ \text{emp}(x) \land \text{sal}(x, y) \land y < 30. \]
The constraint implies that the restriction \( y < 30 \) in the answer is superfluous and an equivalent answer is
\[ \text{emp}(x) \land \text{sal}(x, y) \]
("the employees who have a salary") and even
\[ \text{emp}(x) \]
("all the employees"), obtained by removing the literal \( \text{sal}(x, y) \), if another constraint states that "all employees have a salary". In effect, a reduced constraint of the first answer is \( y < 20 \), which implies the comparison \( y < 30 \) in the first answer.
As already mentioned, a general algorithm for combining and simplifying comparisons was designed [10] to take care of those transformations.

6.2.5 Specializing Answers

A remarkable application of Schema 2 concerns the frequent case where a reduced constraint has the form \( C_1 \rightarrow C_2 \).

Since \((C_1 \land C_2) \Rightarrow (C_1 \lor C_2)\), \( A = A \land C_1 \land C_2 \) is another valid answer such that \( IDB \vdash A \lor A \). Thus, \( A \) is a special case of \( A \), it is what \( A \) becomes when the additional hypothesis \( C_1 \) is made.

Of course, this transformation is very often applicable and it is a matter of strategy to select suitable cases. An example where it can introduce a simplification is as follows.

Example 10. For the query

\[
\text{emp}(x) \land \text{sal}(x, y) \land y < 50
\]

the constraint
\[
v < 15 \land \text{emp}(x) \land \text{sal}(x, y) \land \text{worksin}(x, \text{Food})
\]

has the reduced constraint
\[
y < 15 \land \text{worksin}(x, \text{Food})
\]

By the transformation, after simplifying the comparisons, and supposing that "all employees have a salary", the following answer is obtained:

\[
\text{emp}(x) \land \text{worksin}(x, \text{Food}).
\]

7 Strategies for Presenting Intensional Answers

Beyond the mechanics of answer generation, the interest of the approach depends on an adequate strategy for selecting which answers to return to a user submitting a query. In general, this requires taking into account the goals of the user and the degree of his/her knowledge about the information content of the database. An example of parameter of an intelligent system for intensional answers is how much of the information expressed by constraints should be supplied as justifications of answers. In that respect, the generation of intensional answers raises complex issues of suitability similar to those faced by the design of truly intelligent dialogue management systems and expert systems (see e.g., [11]). The following only gives a few ideas. This whole question is elaborated upon in [10].

A first idea of strategy consists in generating answers in successive layers of increasing level of detail and in presenting them to the user until he/she is satisfied that enough information has been provided. The user can accept or reject individual answers, and the strategy pursues the generation of answers only from the rejected answers, as answers derived from an already accepted answer merely specialize information already acknowledged by the user.

An alternative strategy consists in requesting from the user a language for expressing intensional answers. The simplest specification is a list of predicates (base and virtual) from the deductive database. One selection criterion is to output a candidate answer when all its predicates belong to the user language or are base predicates. An alternative criterion is to freeze in the intermediate answers any predicate that belongs to the user language or are base predicates. An assumption that leads to output a candidate answer is that, with the idea that, for the quality of the information supplied (and also for the cost of generating it), more general answers are preferable to less general ones. Those two selection criteria do not in general output the same set of answers.

Exactly which transformations should be performed on answers is also a matter of suitability (see Example 10). For example, in large databases, very specific constraints may exist next to very general ones. Indeed, elementary facts can be promoted as constraints just because they are permanent. The transformation schemas presented in Section 6.2 apply to any constraint and they may produce specialized instances of answers that are very precise but not necessarily clearly useful.

8 Conclusions

8.1 Related Work

The idea of intensional answer naturally follows from a proof-theoretic view of a deductive database as a set of logical formulas [8] [11].

Still, relatively little work was specifically devoted to intensional answers in the database field.

Our work started from ideas introduced in [3] and [4]. An important simplification and a better control was introduced in our work, not surprisingly, by distinguishing between deduction rules and constraints. Our analysis quickly led us to realize that further control can and should be exercised on the generation process. If a general inference mechanism like resolution starts generating logical consequences of a set of formulas, it is inevitable that formulas too complex and without a satisfactory intuitive relationship with the original query are obtained. For example, the problems that we identified with handling negative literals [10] are completely ignored by resolution.

The so-called semantic optimization of database queries uses integrity constraints to optimize the evaluation of extensional answers. The constraint residues of [2] are similar to our reduced constraints. However, the goals of the approaches are different and our management of constraints inherently has to be considerably more complex. In effect, for semantic query optimization, it is sufficient to "compile" the constraints, that is, to apply deduction rules to reexpress constraints entirely in terms of base and evaluable predicates and to similarly "compile" deduction rules, that is, to reexpress them as definitions of virtual predicates in terms of base predicates. Instead, for intensional answers, constraints must be expressed and preserved at all levels of generality.

The basic idea of [5] is similar to our approach, but the management of constraints is considerably simplified by formulating them in terms of base and evaluable predicates. That paper contains an interesting discussion of some strategies for producing cooperative answers. Another recent work [13] describes the production of intensional answers in a different and more restricted setting. An intensional answer is produced by analyzing the extensional answer according to a hierarchical taxonomy of concepts supposed to exist for the objects of the extensional database.

8.2 Overview and Further Work

In summary, our work investigates the combination of intensional answers and constraints, both to detect the presence of empty answers and to transform answers. The mechanism of reduced constraints was devised to efficiently manage the large set of derived constraints that are inherently relevant to the control of intensional answers. A general generation method was designed and a prototype implemented.
in Prolog. The prototype works in two stages. A first (static) stage constructs reduced constraints of literals and of deduction rules for a given intensional database. A second (dynamic) stage constructs, for a given query, intensional answers and their reduced constraints, it checks whether answers are empty and implements some of the transformations described in Section 6.2. The prototype was designed so as to permit easy modification of the selection strategy for outputting answers.

We have studied the generalization of the method when negated literals are present in queries, rules, and constraints [10]. Albeit more complex and more expensive, the techniques for computing and using reduced constraints generalize to the presence of negated literals. A negated literal in a query (or intensional answer) resolves away with a positive unifying literal in the body of a deduction rule or of a completion rule. For simple cases, both mechanisms have a clear interpretation. However, as complexity increases, the first mechanism quickly leads to unintuitive answers, while the second produces fairiy complex answers, for example with universal quantifiers. We have considered several solutions out of those problems, but our intensional answers should most often not be resolved away. In other words, we suspect that, in most realistic situations, a user mentioning a negated predicate in a query accepts that predicate in the intensional answers.

Another extension is the introduction of some recursive rules by including transitive closure as a construct in the language for expressing intensional answers. Still another issue is how to allow functions. Our current method permits a limited use of functional terms in constraints, namely the presence of Skolem functions from existential variables in positive literals of constraints. They are useful in simplifying some answers, but their use is restricted so that they do not find their way into intensional answers.

ACKNOWLEDGEMENTS
This work was partially supported by the Commission of the European Communities, under Project ESTEAM-316 of the ESPRIT Program. We are grateful to L. Cholvy, R. Demolombe, and V. Royer, our colleagues of ONERA-CERT (Toulouse) in that project, for initially suggesting the topic of intensional answers and discussing it with us. We have also benefited from stimulating discussions with E. Zimanyi of the University of Brussels.

References

659