Obtaining Complete Answers from Incomplete Databases

Alon Y. Levy
AT&T Research
levy@research.att.com

Abstract

We consider the problem of answering queries from databases that may be incomplete. A database is incomplete if some tuples may be missing from some relations, and only a part of each relation is known to be complete. This problem arises in several contexts. For example, systems that provide access to multiple heterogeneous information sources often encounter incomplete sources. The question we address is to determine whether the answer to a specific given query is complete even when the database is incomplete.

We present a novel sound and complete algorithm for the answer-completeness problem by relating it to the problem of independence of queries from updates. We also show an important case of the independence problem (and therefore of the answer-completeness problem) that can be decided in polynomial time, whereas the best known algorithm for this case is exponential. This case involves updates that are described using a conjunction of comparison predicates. We also describe an algorithm that determines whether the answer to the query is complete in the *current* state of the database. Finally, we show that our treatment extends naturally to *partially-incorrect* databases.

Proceedings of the 22nd VLDB Conference
Mumbai(Bombay), India, 1996

1 Introduction

A database is usually assumed to be complete. For example, in a relational database we usually assume that the extension of every relation contains *all* the tuples that need to be in the relation. However, there are situations in which we have access to databases that may be *partial*, i.e., some tuples *may* be missing. If the database is partial, then the meaning of an answer to a given query needs to be reconsidered. For queries that do not contain negation, the answers we obtain are guaranteed to be a subset of the answers that would have been obtained if the database were complete. However, an important question (considered originally in [Mot89, EGW94]) is whether the answer is complete even though the database is incomplete. When queries contain negation, we need to modify our query answering algorithms to guarantee that we obtain only *correct* answers.

We consider the *answer-completeness* problem, i.e., deciding whether an answer to a given query is guaranteed to be complete even in the presence of an incomplete database. We illustrate the problem with several examples.

Example 1.1: Consider an example in which we have access to several online databases with information about movies. Suppose our relation schema contains the following relations:

```
Movie(title, director, year)
Show(title, theater, hour)
Oscar(title, year)
```

The relation `Movie` contains tuples describing the title, director and year of production of movies. The relation `Show` describes the movies playing in New York City, and for each movie it tells us which theaters and at what hours the movie plays. The relation `Oscar` contains a tuple for each movie that won an Oscar award, and the year in which it won the award.

Suppose we know that the relation `Movie` is complete only from the year 1960 and on, and may be missing movies from earlier years. The relations `Show` and **Oscar** are known to be complete. In such a setting, the completeness of the answer depends on the query.

Suppose we are given the following query Q_1 that asks for the pairs of (title, director) for movies currently playing in New York:

```
(Q1):  SELECT m.TITLE, m.DIRECTOR
       FROM Movie m, Show s
       WHERE m.TITLE = s.TITLE.
```

The answer to this query may be incomplete. Intuitively, the answer is incomplete because if we were to *insert* some of the missing tuples to the relation **Movie**, the answer to the query may change.

On the other hand, consider the following query Q_2 that asks for directors whose movies have won Oscars since 1965.[1]

```
(Q2):  SELECT m.DIRECTOR
       FROM Movie m, Oscar o
       WHERE m.TITLE = o.TITLE AND
             m.YEAR = o.YEAR AND
             o.YEAR ≥ 1965.
```

The answer to this query *is* complete, even though the relation **Movie** is not complete. The reason is that only tuples of **Movie** whose third argument is 1965 or more can be joined with tuples from the relation **Oscar** to yield an answer to the query. Therefore, if **Movie** is complete on that part of the relation, the answer to the query will be complete. Furthermore, since the answer to Q_2 is guaranteed to be complete, then we can also guarantee that the answer to the following query that uses the negation of Q_2 is correct. The query Q_3 asks for directors who have won Oscars, but have not won any Oscars since 1965.

```
Q3 :
 SELECT m.DIRECTOR
   FROM Movie m, Oscar o
   WHERE m.TITLE = o.TITLE AND
     NOT EXISTS
       (SELECT * FROM Movie m1, Oscar o1
          WHERE m.DIRECTOR = m1.DIRECTOR AND
            m1.TITLE = o1.TITLE AND
            o1.YEAR ≥ 1965).
```

Finally, consider the query Q_1 again. Even though in general we cannot guarantee that the answer to the query is complete for any database in which the relation **Movie** may be incomplete, we can check whether given the *current* state of the database is complete. To do so, we can compare the projection of the relation **Show** on the attribute **title** and the projection of **Movie** on the attribute title. If the projection of **Movie** *contains* the projection of **Show**, then the answer to Q_1 is guaranteed to be complete. Intuitively, Q_1 is complete in such a case because what really matters is that the relation **Movie** has listings for movies that are currently playing in New York. □

[1] Note that although it is reasonable to assume that a movie wins an Oscar for the year it was produced, we enforce it explicitly in the query.

As stated, deciding whether the answer we obtain for a query is complete is important in order to know whether we are missing some answers (and therefore may have to search for them elsewhere), and in order to answer queries involving negation in a sound manner. The main motivation for our work stems from the context of a mediator-based systems that provide access to multiple distributed information sources (e.g., TSIMMIS [CGMH+94], SIMS [ACHK94], the Internet Softbot [EW94] and the Information Manifold [LRO96a, LRO96b]). In practice, many of the sources these systems access contain only partial information. For instance, the system may have access to a university repository that contains publications authored by faculty and students of that university, but does not necessarily contain *all* of them. On the other hand, the system may have access to the database of the library of congress that has *all* the books published in the U.S.A in the past few decades. In such a setting, given a query, it is important to know which sources (or combination of sources) provide all the answers to the query. If we cannot obtain all the answers, we need to query multiple sources, thereby considerably affecting the performance of the system. Experimental results reported by Etzioni et al. [EGW94] showed that identifying answer-completeness of queries enables pruning many redundant accesses to information sources, and therefore to significant speedups in query processing. Finally, the answer-completeness problem is also important in other contexts. For example, during a long transaction, a database may be incomplete, and in other cases, parts of a database may be temporarily inaccessible.

We make the following contributions in this paper.

- We show that the answer-completeness problem can be completely characterized as a problem of deciding whether a query is independent of an insertion update. As a result, we obtain a better understanding of the problem, and, in particular, we obtain novel sound and complete algorithms for deciding answer-completeness. These results apply to a wider range of cases considered in [EGW94, Mot89] for this problem, and are the first ones that are guaranteed to always detect when an a query is answer-complete.

- We show an important case of the problem of determining independence of queries from updates that can be decided in polynomial time, whereas the best previously known algorithms for this case are exponential. This is the case in which the updated tuples are described using constraints with built-in comparison predicates ($\leq, <, =$), and the all comparisons in the update specification and in the query contain a constant. This result provides a polynomial-time algorithm for detecting answer-completeness, but it is also of independent interest in the context of determining query-independence.
- We describe an algorithm that determines whether in the *current* state of the database the answer to a given query is complete. Completeness is determined by issuing additional queries to the database.
- We show that our treatment of the problem of incomplete databases extends naturally to the case of databases that may be incorrect (i.e., contain a superset of the tuples that should be in the database), and the problem of determining whether the answer to a given query is correct.

Section 2 defines the answer-completeness problem. Section 3 shows the relationship between the answer-completeness problem and the query independence problem, and Section 4 presents the polynomial-time algorithm for deciding query independence. Section 5 describes the algorithm for determining answer-completeness in a particular database state, and Section 6 extends our treatment to partially incorrect databases. Section 7 discusses related work, and Section 8 concludes.

2 Problem definition

In our discussion we consider queries over relational databases that involve select, project, join and union and that use the built-in comparison predicates $\leq$, $<$, $=$ and $\neq$. We assume set semantics for queries (and not multisets). In our analysis it is more convenient to use the notation of *conjunctive queries* [Ull89]. For example, the query

```
(Q2):  SELECT m.DIRECTOR
       FROM Movie m, Oscar o
       WHERE m.TITLE = o.TITLE AND
             o.YEAR ≥ 1965
```

is given as a conjunctive query of the form:

$$q_2(Director) :- Movie(Title, Director, Year)\ \&\ Oscar(Title, Year_1)\ \&\ Year_1 \geq 1965.$$

$q_2(Director)$ is called the ***head*** of the query, and its argument $Director$ is its ***distinguished variable.*** The distinguished variables of the query correspond to attributes appearing in the SELECT clause. $Movie(Title, Dir, Year)$, $Year_1 \geq 1965$ and $Show(Title, Theater, Hour)$ are ***atoms*** in the ***body*** of the query. Note that equality predicates in the WHERE clause are represented by equating variables in different atoms of a conjunctive query. The atom $Year_1 \geq 1965$ is said to be an atom of a comparison (or built-in) predicate ($\geq$ in this case). A union of conjunctive queries is a set of conjunctive queries that have the same arity in the head. Unless otherwise specified, we assume that a query is a union of conjunctive queries.

Given a database instance D and a query Q, we denote by $Q(D)$ the result of evaluating Q over D. We say that a query Q is ***satisfiable*** if there is some database instance D such that $Q(D)$ is a non empty set of tuples. Two queries Q_1 and Q_2 are said to be ***equivalent*** if, for any database instance D, $Q_1(D) = Q_2(D)$. The ***length*** of a conjunctive query is the number of non built-in atoms in its body. A conjunctive query Q is said to be ***minimal*** if we cannot remove any of the non-comparison atoms from its body and still obtain a query equivalent to Q.

2.1 Partial databases

A database is said to be *partial* when the tuples in each relation are only a subset of the tuples that *should* be in the relation. Formally, such a situation can be modeled as having two sets of relations, the ***virtual*** relations and the ***available*** relations. The virtual relations are $\bar{R} = R_1, \ldots, R_n$, while the available relations are $\bar{R}' = R'_1, \ldots, R'_n$. For every $i \in \{1 \ldots n\}$, the extension of the available relation R'_i contains a ***subset*** of the tuples in the extension of the virtual relation R_i. The user poses queries in terms of the virtual relations, but the system has access only to the extensions of the available relations. Therefore, given a query Q over the virtual relations, the query processor actually evaluates the query Q' obtained by replacing every occurrence of R_i by R'_i, for i, $1 \leq i \leq n$.

The question we address is whether the answer we obtain for a query is ***complete***, that is, whether the answer to Q' contains all the tuples we would have obtained by evaluating Q over the virtual relations. Clearly, if all we know is that $R'_i \subset R_i$ for every i, $1 \leq i \leq n$, then whenever the query Q is satisfiable the answer to Q' may be incomplete. However, it is often the case that we know that R'_i is ***partially complete***, i.e., that some parts of it are identical to R_i.

Example 2.1: Continuing with Example 1.1, the relation Movie may be known to be complete for tuples for which $Year \geq 1965$. As another example, the relation **Movie** may be complete for titles of movies being shown in New York. □

Formally, we specify local completeness of a relation R' in the real database by a *constraint* on the tuples of R that are *guaranteed* to be in R'.

Definition 2.1 (Constraint): Let R be a relation of arity n, and $X_1, \ldots, X_n$ be variables standing for its attributes. A constraint C on the relation R is a conjunction of atoms that includes constants, variables from $X_1, \ldots, X_n$ and other variables. The relations used in C can either be either database relations or comparison predicates, but not R itself. A tuple $(a_1, \ldots, a_n)$ satisfies C w.r.t. a database instance D if the conjunction resulting from substituting a_i for X_i in C is satisfied in D. We denote the complement of C by $\neg C$. □

Note that a tuple $(a_1, \ldots, a_n)$ does not have to be in the extension of the relation R in a database instance D in order to satisfy a constraint C on R. In our discussion on local-completeness statements we consider only constraints that involve the virtual relations and comparison predicates.

Definition 2.2 (Local Completeness): Let C be a constraint on the relation R. A database instance D that includes the relations R and R' is said to satisfy the local-completeness statement $LC(R', R, C)$ if R' contains all the tuples of R that satisfy C, i.e., if the results of following two queries are identical over D:

$$q_1(X_1, \ldots, X_n) : -R(X_1, \ldots, X_n) \,\&\, C,$$
$$q_2(X_1, \ldots, X_n) : -R'(X_1, \ldots, X_n) \,\&\, C.$$

□

Example 2.2: The two local completeness statements in Example 2.1 can be stated as follows. The fact that the relation `Movie'` contains all movies after 1965 is represented by

$LC(\texttt{Movie}', \texttt{Movie}, Year \geq 1965)$.

We can represent that the relation `Movie'` has all the movies that are currently playing in New York by

$LC(\texttt{Movie}', \texttt{Movie}, \texttt{Show}(\texttt{Title}, \texttt{Theater}, \texttt{Hour}))$.

Finally, the statement

$$LC(\texttt{Movie}', \texttt{Movie}, \texttt{Show}(\texttt{Title}, \texttt{Theater1}, \texttt{Hour1}) \& \texttt{Show}(\texttt{Title}, \texttt{Theater2}, \texttt{Hour2}) \& \texttt{Theater1} \neq \texttt{Theater2}).$$

says that the relation `Movie'` is complete w.r.t. movies that are playing in at least two theaters in NY. □

We can now define the answer-completeness problem formally. The problem has two variants. In the first, we consider whether the answer is complete w.r.t. *any* database that satisfies the local-completeness statements, and in the second we consider only a single database instance.

Definition 2.3 (Answer-completeness): Let Γ be a set of local completeness statements of the form $LC(R', R, C)$, where $R \in \bar{R}$ (the virtual relations) and $R' \in \bar{R}'$ (the available relations). Let Q be a query over the virtual relations $\bar{R}$, and let Q' be the result of replacing every occurrence of R_i by R'_i in Q, for i, $1 \leq i \leq n$.
The query Q is said to be answer-complete w.r.t. Γ if for any database instance D for the relations $\bar{R}$ and $\bar{R}'$ such that D satisfies Γ, then $Q(D) = Q'(D)$. □

Instance answer-completeness considers whether the answer to a query is complete w.r.t. a specific database instance for the available relations.

Definition 2.4 (Instance answer-completeness): Let Γ be a set of local completeness statements of the form $LC(R', R, C)$, where $R \in \bar{R}$ (the virtual relations) and $R' \in \bar{R}'$ (the available relations). Let Q be a query over the virtual relations $\bar{R}$, and let Q' be the result of replacing every occurrence of R_i by R'_i in Q, for i, $1 \leq i \leq n$.
The query Q is said to be answer-complete w.r.t. Γ and the database instance D if for any database D' such that the extensions of $\bar{R}'$ are identical in D and D', and such that D' satisfies Γ, then $Q(D') = Q'(D')$. □

Example 2.3: Consider Example 1.1 and two sets of completeness information:

$\Gamma_1 : LC(\texttt{Movie}', \texttt{Movie}, Year \geq 1965)$,
$\Gamma_2 : LC(\texttt{Movie}', \texttt{Movie}, \texttt{Show}(\texttt{Title}, \texttt{Theater}, \texttt{Hour}))$

The query Q_1, asking for pairs of (title, director) of movies playing in New York, is not complete w.r.t. Γ_1 because it may miss pairs in which the movie was produced before 1965. However, the answer to Q_1 is complete w.r.t. Γ_2. The query Q_2 that asks for directors whose movies have won Oscars since 1965 is complete w.r.t. Γ_1 but not w.r.t. Γ_2.
For a specific database instance D in which the relation `Movie'` contains all movies whose titles appear in the relation `Show`, the answer to Q_1 is complete w.r.t. Γ_1 and D. □

2.2 Independence of queries from updates

The insight underlying our solution to the answer-completeness problem is based on showing that it is closely related to the problem of detecting independence of queries from updates [BCL89, Elk90, LS93]. The problem of independence of queries from updates is to determine whether the answer to a query Q changes as a result of an insertion to the database or as a result of a deletion from the database. Formally, we specify a possible update by specifying a relation R that is updated and a *constraint* C that describes set of tuples of the relation R *may* be inserted or deleted. Formally, the independence problem is defined as follows.

Definition 2.5 (Independence): Let R be a relation and C be a constraint on the arguments of R. The query Q is independent of the insertion update (R, C), denoted by $In^+(Q, (R, C))$, if for any database instance D and for any database instance D' that results from D by adding to R some tuples that satisfy C, $Q(D) = Q(D')$.

The query Q is independent of the deletion update (R, C), denoted by $In^-(Q, (R, C))$, if for any database instance D and for any database instance D' that results from D by deleting from R some tuples that satisfy C, $Q(D) = Q(D')$. □

Example 2.4: Consider the query Q_3, asking for the movies playing in New York that have received an Oscar in the past twenty years:

$$q_3(Title, Theater) :- Show(Title, Theater, Hour) \,\&\, Oscar(Title, Year) \,\&\, Year \geq 1976.$$

The query q_3 is independent of deleting tuples from the relation `Oscar` for which the year is less than 1970. That is, $In^-(q_3, (\texttt{Oscar}, Year \leq 1970))$ holds. On the other hand, the query is not independent of adding tuples to the relation `Show` for which the show time is after 8pm. That is, $In^+(q_3, (\texttt{Show}, Hour \geq 8pm))$ does not hold. □

3 Deciding answer-completeness of a query

Our solution to the answer-completeness problem is based on showing that the problem can be equivalently translated to a problem of detecting independence of a query from an insertion update. We establish the connection between these two problems in this section. We first illustrate the connection with an example.

Example 3.1: Recall the query Q_1 that asks for the pairs of (title, director) for movies currently playing in New York, and suppose that our local completeness information states that the relation `Movie'` is complete for movies produced after 1965, i.e., the relation may be missing tuples of movies produced before 1965. The query Q_1 is not answer-complete. To see why, suppose we insert tuples into the relation `Movie'` whose year is before 1965. In this case, the answer to the query Q_1 can change. That is, Q_1 is not independent of inserting tuples into `Movie'` whose year is before 1965.

On the other hand, suppose we consider the query Q_2 that asks for directors whose movies have won Oscars since 1965. The answer to Q_2 *is* guaranteed to be complete, because even if we update the database with movies produced before 1965, that would not change the answer to the query. □

The following theorem formalizes the connection between independence and answer-completeness.

Theorem 3.1: *Let Q be a union of conjunctive queries over the virtual relations $\bar{R}$ and comparison predicates, and let Γ be a set of local completeness statements of the form $LC(R'_j, R_j, C_j)$, where $R'_j \in \bar{R}'$ and $R_j \in \bar{R}$. The query Q is answer-complete w.r.t. Γ if and only if $In^+(Q, (R_j, \neg C_j))$ holds for every statement in Γ.* □

Proof: For the first direction, suppose $In^+(Q, (R_j, \neg C_j))$ holds for every statement in Γ, and let D be a database instance that satisfies Γ. We need to show that $Q(D) = Q'(D)$. Let $D_0, \ldots, D_n$ be the databases such that:

- The extensions of $\bar{R}'$ are the same as in D in all databases, and
- In $D_0, \ldots, D_{i-1}$ the extension of R_i is the extension of R'_i in D, and in $D_i, \ldots, D_n$ it is the extension of R_i in D.

Recall that Q' is the query in which every occurrence of R_i in Q is replaced by R'_i. Note that $Q(D_0) = Q'(D)$ and $Q(D_n) = Q(D)$. We prove by induction on i that $Q(D_i) = Q(D_{i-1})$. The claim $Q(D) = Q'(D)$ follows.

Consider the case $i = 1$. Because D satisfies Γ, the database D_1 is obtained from D_0 by adding tuples that satisfy $\neg C_1$ to the extension of R_1. However, since $In^+(Q, (R_1, \neg C_1))$ holds for *any* database, it holds in particular for D_0, i.e., $Q(D_1) = Q(D_0)$. The proof of the inductive step is similar.

For the other direction, suppose that one of the independence assumptions does not hold. Suppose $In^+(Q, (R_1, \neg C_1))$ does not hold. In that case, there must be a database instance E and a set of tuples S that satisfy $\neg C_1$, such that $Q(E) \neq Q(E')$, where E' is the result of adding the tuples S to the extension of R_1. Let D be the database in which the extension of R'_i is the extension of R_i in E, and the extensions of $\bar{R}$ are identical to those in E'. Note that D satisfies Γ, however, $Q(D) \neq Q'(D)$, and therefore, the answer to Q is not complete w.r.t. Γ. □

Using Theorem 3.1 we can apply algorithms for detecting independence (e.g., [BCL89, Elk90, LS93]) to the problem of deciding answer-completeness. Levy and Sagiv [LS93] describe an algorithm for detecting independence based on checking *equivalence* between two queries. Figure 1 describes an algorithm that adopts the method in [LS93] to decide answer-completeness based on equivalence checking.

```
procedure decide-completeness($Q$, $\Gamma$)
/*  $Q$ is a query over the relations $R_1, \dots, R_n$;
    $\Gamma$ is a set of local completeness statements:
        $LC(R'_i, R_i, C_i)$ for $i$, $1 \leq i \leq n$.
    The procedure returns yes if and only if $Q$
        is answer-complete w.r.t. $\Gamma$. */

    Let $E_1, \dots, E_n$ be new relation symbols.
    Define the views $V_1, \dots, V_n$ as follows:
        ($\bar{X}_i$ are the arguments of $R_i$)
        $V_i(\bar{X}_i) : -E_i(\bar{X}_i) \& \neg C_i$.
        $V_i(\bar{X}_i) : -R_i(\bar{X}_i)$.
    Let $Q_1$ be the query in which every occurrence of $R_i$
        is replaced by $V_i$, for $i$, $1 \leq i \leq n$.
return yes if and only if $Q$ is equivalent to $Q_1$.
end.
```

Figure 1: An algorithm for detecting answer-completeness of a query.

The problem of checking query equivalence is well studied in the literature, and therefore algorithm **decide-completeness** can use a host of known results to decide completeness. For example, algorithms for equivalence of queries containing unions and negations are given in [SY81, LS93]. When queries are recursive, the equivalence problem is known to be undecidable [Shm93]. However, algorithms for restricted cases are given in [CV92, CV94, Sag88, LS93]. Finally, if the database relations are known to satisfy integrity constraints (e.g., functional dependencies, tuple generating dependencies), the algorithms in [CM77, ASU79b, ASU79a, JK83] can be used for deciding equivalence. We obtain the following decidability results for the answer-completeness problem.

Theorem 3.2: *Let Q be a union of conjunctive queries over the relations $R_1, \dots, R_n$ and comparison predicates, and let Γ be a set of local completeness statements of the form $LC(R'_j, R_j, C_j)$, where $R'_j \in \bar{R}'$ and $R_j \in \bar{R}$. The answer-completeness problem is decidable in the following cases:*

1. *if each of the C_j's contains only arguments of R_j or constants, or*
2. *if the head of Q contains all the variables of the body of Q, and neither the C_j's or Q use the comparison predicates.* □

When the constraints involve comparison predicates the problem of deciding answer-completeness for the first case of Theorem 3.2 is Π_2^p. In fact, the lower bound on the problem of equivalence [vdM92] implies that this is also the lower bound on the answer-completeness problem. The proof of Theorem 3.2 follows from Theorem 3.1 and from decidability results for query containment [SY81, Klu88, LS93]. In the next section we consider the very common case in which the C_j's include variable-interval built-in atoms, and show that deciding independence can be done in polynomial-time, and therefore so can answer-completeness.

4 Deciding independence efficiently

In this section we identify an important practical case in which independence of queries from updates can be detected more efficiently than in previous algorithms. Consequently, deciding answer-completeness of a query can also be done more efficiently. Intuitively, in this case detecting independence is equivalent to deciding when the updated tuples and the query are *mutually unsatisfiable*, whereas in general, detecting independence requires that we detect that the updated tuples are *redundant* w.r.t. the query. The following example, adopted from [LS93], explains the difference between satisfiability and redundancy.

Example 4.1: Consider a database containing the relation $inCar(Person, Car, Age)$. A tuple (P, C, A) is in the relation $inCar$ if the person P, whose age is A is in car C. The view $canDrive(Person, Car, Age)$ is defined as follows:

$$canDrive(P, C, A) \;\; :- \;\; inCar(P, C, A),\; driver(P),\; inCar(P_1, C, A_1),\; A_1 \geq 18.$$

That is, a person P is allowed to drive a car C if P has a driver's license and there is someone in the car above the age of 18. Suppose our query is to find all the adult drivers:

$$adultDriver(P) \;\; :- \;\; canDrive(P, C, A),\; A \geq 18.$$

Consider a deletion update that removes from the relation $inCar$ some tuples (P, C, A) for which $\neg Driver(P)$ and $A < 18$. Clearly, such tuples cannot be part of a derivation of an answer to the query $adultDriver$ because $adultDriver$ uses only tuples of $inCar$ for which either P is a driver or the age is at least 18. In this case we say that the tuples involved in the update are mutually unsatisfiable with the query.

Clearly, if the updated tuples are mutually unsatisfiable with the query, the query is independent of the update.

Consider a deletion update that removes from the relation *inCar* tuples involving only non-drivers, i.e., tuples (P, C, A) for which $\neg Driver(P)$. Tuples of this set *can* be used in a derivation of the query *adultDriver*. For example, if the database contains the tuples $inCar(Alice, C, 19)$, $driver(Alice)$ and $inCar(Bob, C, 20)$, the tuple of Bob can be used to derive that Alice is an adult driver. However, the tuple of Bob is *redundant*, because Alice (being older than 18) is allowed to drive the car C even if she is alone in the car, and therefore is an answer to *adultDriver*. Therefore, removing such tuples will not change the result of the query. □

Detecting redundancy is a more expensive procedure than detecting satisfiability. For example, for conjunctive queries with comparison predicates, the time complexity of detecting redundancy is π_2^p [Klu88, vdM92], while checking satisfiability can be done in polynomial time [Ull89].

The case we consider involves *variable-interval* queries and updates. In particular, a conjunctive query Q is a variable-interval query if all the built-in atoms in Q have one constant (i.e., there are no comparisons between pairs of variables). A *variable-interval update* is an update in which the updated tuples are specified by a conjunction of built-in atoms where each atom contains one constant. It should be noted that variable-interval queries are a more general class than semiinterval queries considered in [Klu88]. Figure 2 describes an algorithm for detecting independence of variable-interval queries from variable-interval updates. The following theorem shows that the algorithm *completely* decides independence in this case in polynomial time.

Theorem 4.1: *Let Q be a conjunctive variable-interval query over the relations $E_1, \ldots, E_n$ and the comparison predicates $<, \leq$. Suppose U is an update (either deletion or insertion) to the relation E_1, in which tuples satisfying the constraint C are added to E_1, where C is a constraint on E_1 involving only the comparison predicates $\leq, <$, and each conjunct in C has one constant.*

If Q is a minimal query then procedure **detect-independence** *will return* independent *if and only if Q is independent of U. The running time of procedure* **detect-independence** *is polynomial in the size of Q and C.* □

Proof Sketch: In [LS93] it is shown that Q is independent of deleting (E_1, C) if and only if it is independent of adding (E_1, C), since the update is oblivious. We show that if Q is independent of the deletion update (E_1, C), and Q has a conjunct $E_1(\bar{Y})$ that is consistent with $C(\bar{Y})$, then the conjunct must be redundant, and therefore the query is not minimal.

```
procedure detect-independence($Q$, $E$, $C$)
/*  $Q$ is conjunctive query, and
    $E$ is one of the relations mentioned in $Q$;
    $C(X_1, \ldots, X_m)$ is a conjunctive constraint on the
    arguments of $E$ that uses only comparison atoms,
    each with one constant. */

for every occurrence $E(Y_1, \ldots, Y_m)$ of the relation $E$
    in the body of $Q$ do:
        Let $\psi$ be the mapping $X_j \rightarrow Y_j$ for $j$, $1 \leq j \leq m$.
        Let $Q'$ be the conjunctive query in which $\psi(C)$
            is added to the subgoals of $Q$.
        if $Q'$ is satisfiable then return not independent.
return independent.
end.
```

Figure 2: An algorithm for detecting independence of a query from an insertion update, for updates specified by built-in predicates.

In order to be satisfiable, the comparison atoms of the conjunctive query Q force the value of every variable X to be in an *interval* I_X. The interval may be open or closed and may have $\infty, -\infty$ as endpoints.

For any given mapping ψ of the variables of Q to constants in their intervals, we can create a database D_ψ that contains exactly the tuples $\psi(g)$ for every subgoal of $g \in Q$. From D_ψ we can derive the answer $\psi(\bar{X})$, where $\bar{X}$ are the head variables of Q. In order to show that the conjunct $E_1(\bar{Y})$ is redundant, we need to show that for every ψ, we can derive the answer $\psi(\bar{X})$ from D_ψ *without* the tuple $\psi(E_1(\bar{Y}))$.

There are two cases. If $\psi(\bar{Y})$ satisfies C, then we can remove $E(\psi(\bar{Y}))$ from D_ψ without affecting the answer to Q because the query is independent of deleting (E_1, C).

In the second case, $\psi(\bar{Y})$ does not satisfy C. Since the comparison atoms in Q do not compare *among* variables, it is possible to modify D_ψ to a database D'_ψ such that one of the variables in $\bar{Y}$ is mapped to a new constant, and the resulting database contains a tuple that may be removed by the update. Furthermore, the mapping of the head variables of Q in D'_ψ is the same as it is in D_ψ, and the mapping of D'_ψ is consistent with the comparison atoms in Q. In D'_ψ, there must be a proper subset of the tuples that suffices to derive the answer. In the proof we show that the corresponding subset in D_ψ also suffices to derive the answer. □

5 Answer-completeness for a database instance

In Section 3 we considered the problem of deciding answer-completeness of a query Q by examining *only* Q and the local-completeness information Γ. When our algorithm returned that the query is answer-complete, then that will be true no matter of the specific database instance as long as it satisfies Γ. However, if the algorithm returned that the query is not answer-complete, then there still may be database instances in which the answer is complete.

In this section we describe an algorithm that decides whether the answer to the query is complete in the *current* database state. The algorithm is based on submitting a couple of additional queries whose answers will show whether the answer to Q is guaranteed to be complete. We first illustrate the algorithm with an example.

Example 5.1: Consider again the query Q_1 asking for the pairs of (title,director) of movies currently playing in New York.

$$q(Title, Director) : -Movie(Title, Director, Year) \,\&\, Show(Title, Theater, Time)$$

Suppose the **Show** relation is known to be complete, but the **Movie** relation is not (i.e., **Movie'** $\subset$ **Movie**), and assume that the functional dependency $Movie : Title \rightarrow \{Director, Year\}$ holds, i.e., the title of the movie uniquely determines its director and year of production. To check whether the answer we get in the current state of the database is complete we can issue the following two queries:

$q_1(Title) : -Show(Title, Theater, Time)$
$q_2(Title) : -Movie(Title, Director, Year)$

If the answer to q_2 is a superset of the answer to q_1, then we can conclude that the answer to the query Q_1 is complete. The reason is that although the relation **Movie'** may not be complete, it contains all the movies *currently* playing in NY, and therefore all the tuples that would be needed in the complete answer. □

Intuitively, the algorithm is based on finding a subset of the subgoals of the query that provide a *superset* of the complete answer to the query. We then check whether the conjunctive query formed by the rest of the subgoals is complete on that superset. Before we describe the algorithm, we need one additional definition.

Given a set of variables in the query, they may functionally determine the values of other variables.

Definition 5.1 (Functional determination): Let Q be a conjunctive query, and assume that all equalities between variables have already been propagated in Q (i.e., Q does not imply additional equalities between variables or between a variable and a constant). Let $\bar{X}$ be a subset of the variables that appear in the body of Q. The variable Y in Q is said to be functionally determined by $\bar{X}$ if

1. $Y \in \bar{X}$, or
2. There exist variables $Y_1, \ldots, Y_l$ that are functionally determined by $\bar{X}$, and an atom of the relation R in the body of Q in which the variables $Y_1, \ldots, Y_l, Y$ appear in the positions $i_1, \ldots, i_{l+1}$ respectively, and the functional dependency $R : \{x_1, \ldots, x_l\} \rightarrow x_{l+1}$ holds.

□

In Example 5.1, the variables $Title, Theater$ and $Time$ functionally determine all the other variables in the query.

For simplicity of exposition we describe our algorithm for conjunctive queries. Suppose we are given a set of local-completeness statements Γ, a database instance D and a query Q of the form

$$Q\text{: } q(\bar{X}) : -p_1(\bar{X}_1) \,\&\, \ldots \,\&\, p_n(\bar{X}_n) \,\&\, C_q$$

where C_q is the conjunction of comparison atoms in the query. We denote by S_1 a maximal subset of $p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)$ such that:

- The variables in S_1 functionally determine all the distinguished variables $\bar{X}$ of Q, and
- The following query is answer-complete w.r.t. Γ:

  $Q' : q'(\bar{X}') : -S_1 \,\&\, C_{S_1}$

  where $\bar{X}'$ are the subset of $\bar{X}$ that appear in S_1, and C_{S_1} is the projection of C_q on the variables in S_1.[2]

If no subset S_1 exists, then the algorithm returns *unknown*. Otherwise, we denote by S_2 the set of non comparison atoms of Q that are not in S_1, and by C_{S_2} the projection of C_q on the variables in S_1. The set $\bar{Y}$ is the set of variables that appear in both S_1 and S_2, and $\bar{Y}'$ is a minimal subset of $\bar{Y}$ that functionally determines all the other variables in $\bar{Y}$. We define the following two queries:

Q_1: $q_1(\bar{Y}') : -S_1 \,\&\, C_{S_1}$.
Q_2: $q_2(\bar{Y}') : -S_2 \,\&\, C_{S_2}$.

If the answer to Q_2 from D is a superset of the answer to Q_1 from D, then the algorithm returns *complete*, and otherwise it returns *unknown*. □

[2]Note that the projection of C_q on S_1 may actually be a disjunction, in which case Q' is a union of conjunctive queries.

Example 5.2: Continuing with example 5.1, the set S_1 includes the subgoal

$Show(Title, Theater, Hour)$

since *Title* determines *Director* and *Year*, and the relation **Show** is assumed to be complete. The set of variables $\bar{Y}$ would include only *Title*, and therefore the queries would be

$q_1(Title) : -Show(Title, Theater, Time).$
$q_2(Title) : -Movie(Title, Director, Year).$ □

The correctness of the algorithm is established by the following theorem.

Theorem 5.1: *Let Q be a conjunctive query over the relations $R_1, \ldots, R_n$ and comparison predicates, and let Γ be a set of local completeness statements of the form $LC(R'_j, R_j, C_j)$, where $R'_j \in \bar{R}'$ and $R_j \in \bar{R}$. Let Q_1 and Q_2 be the queries defined by our algorithm. For a given database D, if $Q_2(D) \supseteq Q_1(D)$ then Q is answer-complete w.r.t. Γ and D.* □

Proof Sketch: By the construction of the set S_1, the answer to Q' contains a superset of the projection of the complete answer of Q on the variables $\bar{X}'$. This follows because Q' is answer-complete w.r.t. Γ, and contains a subset of the subgoals of Q. If the answer to Q_2 contains the answer to Q_1 over D, then insertions to incomplete parts of S_2 cannot change the join of S_1 and S_2. Therefore, since the variables in S_1 functionally determine all the other variables in the body of Q, the answer to Q is guaranteed to be complete. □

6 Answer-correctness

An additional advantage of our treatment of the answer-completeness problem is that there is a very analogous treatment for the case of *incorrect* information in the database. In this case, the tuples of a relation R'_i are a superset of the tuples of R_i. In the same way we defined that a relation is locally complete, we can define a relation to be locally correct:

Definition 6.1 (Local Correctness): Let C be a constraint on the relation R. A database instance D that includes the relations R and R' is said to satisfy the local-correctness statement $LCor(R', R, C)$ if R' does not contain tuples satisfying C that are not in R, i.e., if the results of following two queries are identical over D:

$q_1(X_1, \ldots, X_n) : -R(X_1, \ldots, X_n) \,\&\, C,$
$q_2(X_1, \ldots, X_n) : -R'(X_1, \ldots, X_n) \,\&\, C.$

□

The question we are interested in now is whether the answer to a given query Q is correct from the partially correct database:

Definition 6.2 (Answer-correctness): Let Γ be a set of local-correctness statements of the form $LCor(R', R, C)$, where $R \in \bar{R}$ (the virtual relations) and $R' \in \bar{R}'$ (the available relations). Let Q be a query over the virtual relations $\bar{R}$, and let Q' be the result of replacing every occurrence of R_i by R'_i in Q, for i, $1 \leq i \leq n$.

The query Q is said to be answer-correct w.r.t. Γ if for any database instance D for the relations $\bar{R}$ and $\bar{R}'$ such that D satisfies Γ, then $Q(D) = Q'(D)$. □

Theorem 3.1 showed that the answer-completeness problem can be equivalently formulated as the problem independence of a query from an *insertion* update. The following theorem shows that the answer-correctness problem can be equivalently formulated as the problem independence of a query from an *deletion* update.

Theorem 6.1: *Let Q be a union of conjunctive queries over the relations $R_1, \ldots, R_n$ and comparison predicates, and let Γ be a set of local completeness statements of the form $LCor(R'_j, R_j, C_j)$, where $R'_j \in \bar{R}'$ and $R_j \in \bar{R}$. The query Q is answer-correct w.r.t. Γ if and only if $In^-(Q, (R_j, \neg C_j))$ holds for every statement in Γ.* □

Given Theorem 6.1 we can use algorithms for detecting independence also for deciding answer-correctness. Theorem 6.1 has an additional interesting consequence. As shown by Elkan [Elk90], independence from a deletion is a sufficient condition for independence from an update, i.e.,

$$In^-(Q, (R, C))) \Rightarrow In^+(Q, (R, C))).$$

Therefore, if a query is deemed to be answer-correct, it is also answer-complete. More importantly, this implies that if we have a database that is partially incomplete *and* partially incorrect, then determining answer-correctness is enough for detecting that the answer is both correct and complete.

7 Related work

Motro [Mot89] considers the problem of determining answer-correctness (which he calls *validity*) and answer-completeness in the presence of incorrect or incomplete databases. His approach is based on describing the complete (or valid) parts of the database as views. Given a query Q, if there is a rewriting of the query using the complete (resp. valid) views, then the answer is complete (resp. valid). He describes an algorithm that finds rewritings of queries using views, but it is not always guaranteed to find one if one exists. Although complete algorithms for rewriting queries using views have been developed since (e.g., [LMSS95]), finding a rewriting of the query using views has not been shown to be a necessary condition for answer-completeness. Motro also does not consider the problem of determining answer-completeness w.r.t. a specific database instance.

Recently, Etzioni et al. [EGW94] considered the problem of answer-completeness in order to avoid redundant information gathering actions in the Internet Softbot system [EW94], and demonstrated experimentally the value of detecting answer-completeness. They show that answer-completeness is closed under conjunction and partial instantiation of queries, and use these properties as a basis for their algorithm for determining answer-completeness. As they show, their algorithm is not guaranteed to always detect answer-completeness when it holds. They do not allow existential variables in local-completeness statements, and they do not make use of the semantics of comparison predicates in their algorithms (though comparison predicates are allowed to appear). Reasoning about local-complete information is also related to the problem of reasoning with the *closed world assumption.* The bulk of previous work on the topic (see [Gin87] for a collection of articles) has concerned itself with the *logic* of reasoning with the closed world assumption (i.e., which conclusions are appropriate to derive) rather than efficient algorithms for doing so.

Levy and Sagiv [LS93] present sound and complete algorithms for query independence for queries that are unions of conjunctive queries with comparison predicates and for recursive queries. Theorem 4.1 is a case that has not been considered specifically in [LS93] for which there is a polynomial time algorithm for query independence. Elkan [Elk90] describes an algorithm for query independence whose time-complexity is polynomial in the case considered by Theorem 4.1. However, his algorithms apply only to queries with no self-joins (i.e., at most one occurrence of every relation in the query). In [LSK95] an algorithm for pruning redundant sources based on sources completeness was described. However, the question of whether the answer to a given query is complete given local-completeness statements was not addressed.

8 Conclusions

We considered the problem of answering queries from databases that may be incomplete or incorrect, and presented algorithms that decide whether the answer to a query is complete or correct. We provided a complete characterization of the problem by relating it to the problem of determining independence of queries from updates. Whereas determining completeness of an answer is translated to independence of insertion updates, determining correctness of an answer is translated to independence of deletion updates. Consequently, we can deal uniformly with both cases. This characterization yields a better understanding of the problem's complexity and decidability, and in particular, it yields novel sound and complete algorithms for answer-completeness and answer-correctness that generalize previous treatments of the problem. In particular, the algorithm for determining answer-completeness based on query containment can also be applied in wider contexts, such as in the presence of integrity constraints on the database. We identified an important case in which independence can be determined efficiently, which, aside from being a result of independent interest, yields an efficient algorithm for the answer-completeness problem. Finally, we presented an algorithm that considers the current state of the database to determine whether the answer is complete.

A related question one can pose about incomplete databases (which was considered in [EGW94]) is what happens when the partial-completeness completeness information changes. In particular, is the answer to a query still complete even if parts of the database that were assumed to be complete may not be complete anymore. It can be shown that this problem can be reformulated as a problem of independence of queries from deletion updates, thereby giving it a uniform treatment with the problem we considered in this paper.

There are several interesting directions of future work to pursue. One is to consider other ways of specifying local-completeness information that cannot be captured by the statements we allow here. An interesting direction is to extend the algorithm described in Section 5 by considering whether more information about completeness can be obtained by issuing additional queries to the database.

Acknowledgements

I would like to thank Oren Etzioni and Dan Weld for discussions that inspired me to do this work.

References

[ACHK94] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems*, 1994.

[ASU79a] Alfred Aho, Yehoshua Sagiv, and Jeffrey D. Ullman. Efficient optimization of a class of relational expressions. *ACM Transactions on Database Systems*, (4)4:435–454, 1979.

[ASU79b] Alfred Aho, Yehoshua Sagiv, and Jeffrey D. Ullman. Equivalence of relational expressions. *SIAM Journal of Computing*, (8)2:218–246, 1979.

[BCL89] J. A. Blakeley, N. Coburn, and P. A. Larson. Updating derived relations: detecting irrelevant and autonomously computable updates. *Transactions of Database Systems*, 14(3):369–400, 1989.

[CGMH$^+$94] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogenous information sources. In proceedings of IPSJ, Tokyo, Japan, October 1994.

[CM77] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[CV92] Surajit Chaudhuri and Moshe Vardi. On the equivalence of recursive and nonrecursive datalog programs. In *The Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, CA.*, pages 55–66, 1992.

[CV94] Surajit Chaudhuri and Moshe Vardi. On the complexity of equivalence between recursive and nonrecursive datalog programs. In *The Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 55–66, 1994.

[EGW94] Oren Etzioni, Keith Golden, and Daniel Weld. Tractable closed world reasoning with updates. In *Proceedings of the Conference on Principles of Knowledge Representation and Reasoning, KR-94.*, 1994. Extended version to appear in *Artificial Intelligence*.

[Elk90] Charles Elkan. Independence of logic database queries and updates. In *Proceedings of the 9th ACM Symp. on Principles of Database Systems*, pages 154–160, 1990.

[EW94] Oren Etzioni and Dan Weld. A softbot-based interface to the internet. *CACM*, 37(7):72–76, 1994.

[Gin87] Matthew Ginsberg. *Readings in Nonmonotonic reasoning.* Morgan Kaufmann Publishers, Inc., San Mateo, California, 1987.

[JK83] D. S. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependecies. *Journal of Computer and System Sciences*, (28):1:167–189, 1983.

[Klu88] A. Klug. On conjunctive queries containing inequalities. *Journal of the ACM*, pages 35(1): 146–160, 1988.

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Jose, CA*, 1995.

[LRO96a] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query answering algorithms for information agents. In *Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence*, 1996.

[LRO96b] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd VLDB Conference, Bombay, India.*, 1996.

[LS93] Alon Y. Levy and Yehoshua Sagiv. Queries independent of updates. In *Proceedings of the 19th VLDB Conference, Dublin, Ireland*, pages 171–181, 1993.

[LSK95] Alon Y. Levy, Divesh Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems, Special Issue on Networked Information Discovery and Retrieval*, 5 (2), September 1995.

[Mot89] Amihai Motro. Integrity = validity + completeness. *ACM Transactions on Database Systems*, 14(4):480–502, December 1989.

[Sag88] Yehoshua Sagiv. Optimizing datalog programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 659–698. Morgan Kaufmann, Los Altos, CA, 1988.

[Shm93] Oded Shmueli. Equivalence of datalog queries is undecidable. *Journal of Logic Programming*, 15:231–241, 1993.

[SY81] Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1981.

[Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems, Volumes I, II.* Computer Science Press, Rockville MD, 1989.

[vdM92] Ron van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *The Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, CA.*, pages 331–345, 1992.