Consistent Answers from Integrated Data Sources

Leopoldo Bertossi\(^1\), Jan Chomicki\(^2\)
Alvaro Cortés\(^3\), and Claudio Gutiérrez\(^4\)

\(^1\) Carleton University, School of Computer Science, Ottawa, Canada, bertossi@scs.carleton.ca
\(^2\) State University of New York at Buffalo, Dept. of Computer Science and Engineering, chomicki@cse.buffalo.edu
\(^3\) Pontificia Universidad Católica de Chile, Departamento de Ciencia de Computación, Santiago, Chile, acortes@ing.puc.cl
\(^4\) Universidad de Chile, Center for Web Research, Departamento de Ciencias de la Computación, Santiago, Chile, egutier@sde.uChile.cl

Abstract. When data sources are integrated into a single global system, inconsistencies wrt global integrity constraints are likely to occur. In this paper, the notion of consistent answer to a global query in the context of the local-as-view model of data integration is characterized. Furthermore, a methodology for generating query plans for retrieving consistent answer to global queries is introduced. For this purpose, an extension of the inverse-rules algorithm for deriving query plans is presented. It can be used to answer first order queries posed to data sources integrated according to the local-as-view.

1 Introduction

In last few years, due the increasing number of information sources that are available and may interact, the subject of data integration has been widely studied from different points of view. Topics like mediated schemas, query containment, answering queries using views, etc., have been deeply discussed in this context. However, less attention has attracted the important and natural issue of consistency of data derived from the integration process and from answering queries posed to the integrated system.

A data integration system provides a uniform interface to several information sources. This interface, referred as global schema or mediated schema, is a context-dependent set of virtual relations used to formulate queries to the integrated system. When the user queries the system in terms of the global schema, a query processor or a mediator is in charge of rewriting the global query into a query plan that will eventually access the underlying information sources.

In order to perform this query transformation, the processor needs a mapping between the mediated schema and the information sources. Two general paradigms have been proposed to provide this mapping. One of them, called the Local-as-View (LAV) approach [15], considers each base information source as
a view defined in terms of relations in the global schema. The Global-as-View (GAV) approach, considers each global predicate as a view defined in terms of the source relations [20, 22].

In this paper we concentrate on the LAV approach. This scenario is more flexible than GAV for adding new data sources into a global system. Actually, preexisting data sources in the system do not need to be considered when a new source is introduced. In consequence, inconsistencies are more likely to occur. Furthermore, from the point of view of studying the logical issues around consistency of data, the LAV paradigm seem to be more challenging than the GAV, that can be more easily assimilated to the classical problem of consistency of views defined over relational databases.

In the context of the local-as-view approach, several algorithms have been proposed to rewrite a global query into a query plan that accesses the data source relations to answer the original query [14].

Several approaches to query plan generation assume that certain integrity constraints (ICs) hold at the global level, and they use the ICs in the query plan generation. In [13], a rewriting algorithm that uses functional and inclusion dependencies in the mediated schema is proposed. In [9], another algorithm for query plan generation that uses functional and full dependencies is introduced. This algorithm may take a global query written in Datalog as an input. In [11], a deductive, resolution based approach to data integration is presented. It may also use global integrity constraints in the deductive derivation of the query plan.

There are situations where, without assuming that certain global ICs hold, no query plan can be generated.

**Example 1.** (taken from [9]) Consider the following relations in the global schema

\[
\text{conference}(\text{Paper}, \text{Conference})
\]

\[
\text{year}(\text{Paper}, \text{Year})
\]

\[
\text{location}(\text{Conference}, \text{Year}, \text{Location})
\]

plus the functional dependencies (FDs):

\[
\text{conference} : \text{Paper} \rightarrow \text{Conference}
\]

\[
\text{year} : \text{Paper} \rightarrow \text{Year}
\]

\[
\text{location} : \text{Conference}, \text{Year} \rightarrow \text{Location}
\]

and the following data sources expressed as views over the global schema:

\[
s_1(P, C, Y) \leftarrow \text{conference}(P, C), \text{year}(P, Y)
\]

\[
s_2(P, L) \leftarrow \text{conference}(P, C), \text{year}(P, Y), \text{location}(C, Y, L)
\]

If we want to know the location where PODS89 was held, we can pose the global query \( Q : \text{ans}(L) \leftarrow \text{location}(\text{pods}, 1989, L) \), and we could use the following

\[^{1}\text{It is also possible to specify this mapping using description logics [5].}\]
query plan to answer it:

\[ \text{ans}(L) \leftarrow s_1(P, \text{pods}, 1989), s_2(P, L). \]

This plan proceeds as follows. It first finds some paper presented at PODS89 using source \( s_1 \), and then finds the location of the conference at which this paper was presented using source \( s_2 \). This plan is correct because every paper is presented at one conference and in one year only. In fact, if these dependencies would not hold, there would be no way to answer this query using the given sources.

In the previous example, the ICs are supposed to hold in the global system. Nevertheless, it is not obvious that certain desirable, global ICs will hold at that level. After all, the data is in the sources, the global relations are virtual\(^2\), and there may be no consistency checking mechanism at the global level.

In addition to this, and particularly in the local-as-view approach, it is not clear what it means for the global system to be consistent, because we do not have a global instance. Actually, given a set of data sources, there may be several potential global instances that (re)produce the data sources as views according to the view definitions.

A **global system** will be a global schema plus a collection of materialized data sources that are described as views over the global schema. In this context, it is quite natural to pose queries at the global level, expecting to retrieve those answers that are consistent wrt a given set of global ICs, because, as we mentioned before, global ICs may be easily violated due to the lack of a global maintenance mechanism and the high likelihood of producing inconsistencies when data from different sources is integrated. Actually, as the following example shows, each data source, with its own, independent maintenance mechanism, can be consistent, but inconsistencies may arise when the sources are integrated.

**Example 2.** Consider the global relation \( R(X, Y) \) and two source relations \( \{V_1(a, b), V_1(c, d)\} \) and \( \{V_2(a, c), V_2(d, e)\} \) described by the view definitions:

\[
V_1(X, Y) \leftarrow R(X, Y) \quad V_2(X, Y) \leftarrow R(X, Y).
\]

Then, the global functional dependency (FD) \( R : X \rightarrow Y \) is violated, but not \( V_1 : X \rightarrow Y, V_2 : X \rightarrow Y \).\( \square \)

We will be interested in posing queries to a global system that is inconsistent wrt to certain global ICs. In such a situation of global inconsistency, we would like to retrieve as answers only those tuples that are consistent wrt the global ICs.

\(^2\) Even in the local-as-view approach where, from a theoretical perspective, sources are seen as views over the global relations: sources are still materialized, but global relations are virtual.
Example 3. (example 2 continued) If we pose to the global system the query $Q : \text{ans}(X, Y) \leftarrow R(X, Y)$, we obtain the answers $\{\text{ans}(a, b), \text{ans}(c, d), \text{ans}(a, c), \text{ans}(d, e)\}$. However, only the tuple $(c, d), (d, e)$ should be returned as answers that are consistent wrt the global FD.

In order to address these issues, several semantic problems appear: (a) When and in what sense is global system consistent wrt a given set of global ICs? (b) Which are the answers to a global query that are consistent wrt the given ICs? (c) Is there a computational mechanism to compute those consistent answers?

In [3], the problem of characterizing and computing consistent query answers from an inconsistent relational database instance were addressed. In this case, the database instance $r$ is inconsistent when it does not satisfy a given set of ICs. Intuitively speaking, an answer to a query is considered to be consistent in $r$ if it is an answer to the query in every possible repair of the original instance, where a repair is a new instance that satisfies the ICs and differs from $r$ by a minimal set of tuples under set inclusion. The computational mechanism basically consists in rewriting the query into a new query that, posed to the original, inconsistent database, gets as (normal) answers the consistent answers to the original query.

In this paper we characterize the consistent answers to a query posed to a global, virtual, integrated system. We also consider the problem of deriving mechanisms for computing consistent answers from a global system. This scenario is quite different from the one considered in [3]. There are important differences.

First of all, in the context of virtual data integration we do not have a global instance. As shown in [10], a data integration system may determine a possible finite set of global instances. The notions introduced in [10] will help us define our consistent answers.

A second problem has to do with deriving query plans for retrieving, hopefully all and only, answers to a global query that are consistent wrt the desired, global ICs. Following the approach in [3], we may rewrite the global query into a new query, and then pose the new query to the global system. The problem is that the rewritten query may not be handled by any of the existing query plan generation algorithms, e.g. [11, 15, 13]. In consequence, we need to develop query plan mechanisms that are appropriate for our rewritten queries. For this purpose, we extend the “inverse rules” algorithm from [9] for the kind of rewritten queries we need to answer, namely Datalog queries with negation, but no recursion. In this part we restrict ourselves to the case of “open” sources [10], the most common scenario.

2 Preliminaries

2.1 Global schemas and view definitions

A global schema, $\mathcal{R}$, is modeled by a finite set of relations $\{R_1, R_2, \ldots, R_n\}$ and a possibly infinite domain $\mathcal{D}$. With these predicate symbols and the elements of $\mathcal{D}$ treated as constants, a first order language $\mathcal{L}^*(\mathcal{R})$ can be defined. This language can be extended with new defined predicates.
A view, denoted by a new global predicate $V$, can be defined by means of an \( \mathcal{L}(\mathcal{R}) \)-formula of the form $\varphi_V: V(\overline{t}) \leftarrow \text{body}(\varphi_V)$, where $\overline{t}$ is a tuple containing variables and/or constants, and $\text{body}(\varphi_V)$ is a conjunction of global atoms. The formula $\varphi_V$ is implicitly universally quantified. That is, a view is defined by a conjunctive query [1).

A database instance $D$ over schema $\mathcal{R}$ can be considered as a first order structure with domain $D$, where the extensions of the interpretations of the predicates $R_i$ are finite (unless they are built-in predicates, in whose case they may have infinite, but fixed extensions). An integrity constraint is a first order sentence $\psi$ written in language $\mathcal{L}(\mathcal{R})$. The instance $D$ satisfies $\psi$, denoted $D \models \psi$, if $\psi$ is true in $D$.

Given a database instance $D$ over schema $\mathcal{R}$, and a view definition $\varphi_V$, $\varphi_V(D)$ denotes the extension of $V$ obtained by applying the definition $\varphi_V$ to $D$.

Assume we are given a definition $\varphi_V$ of a view $V$, a set $v$ of ground atoms on predicate $V$ (think of a fixed, given data source), and a global instance $D$. It is possible that the view extension $v$ of $V$ differs from the extension $\varphi_V(D)$ of $V$ obtained by applying $\varphi_V$ to instance $D$. According to [2, 10], if the view extension $v$ stores all the tuples that satisfy the definition of view $V$, we say that the view extension is closed wrt $D$. On the other hand, if the view extension $v$ is possibly incomplete and stores only some of the tuples that satisfy the definition of $V$, we say that the view extension is open wrt $D$. In example 1, the first source definition (1) has to be read as $s_1(P, C, Y) \subseteq \text{conference}(P, C), \text{year}(P, Y)$ if the source is considered to be open. Most mechanisms for deriving query plans are based on open sources [15, 9, 18].

Following [10], we say that a source, $S$, is a triple $< \varphi, \text{label}, v >$, where $\varphi$ is a view definition, $\text{label} \in \{ \text{open}, \text{closed}, \text{copen} \}$ and $v$ is a view extension $v$ of $\varphi$. Here, copen stands for closed and open. A global system (or source collection in [10]), $\mathcal{G}$, is a finite set of sources. The schema $\mathcal{R}$ of the global system can be read from the bodies of the view definitions. It consists of the predicate names that do not have a definition in the global system. The underlying domain $D$ for $\mathcal{R}$ (maybe properly) contains all the constants appearing in view extension $v$'s in the sources.

### 2.2 Global instances

When we talk about consistency in databases wrt a set of ICs we think of instances satisfying ICs. However, in a global system for data integration there is not such a global instance, at least not explicitly. Instead, a global system $\mathcal{G}$ defines a set of possible instances.

**Definition 1.** [10] Given a global system $\mathcal{G}$, the set of legal global instances is

$$\text{Linst}(\mathcal{G}) = \{ D \text{ instance over } \mathcal{R} | v_i \subseteq \varphi_i(D) \text{ for all open sources } S_i \in \mathcal{G},$$

$$\text{ and } v_j \supseteq \varphi_j(D) \text{ for all closed sources } S_j \in \mathcal{G},$$

$$\text{ and } v_k = \varphi_k(D) \text{ for all copen sources } S_k \in \mathcal{G} \}.$$
Here, $v_i$ is the extension in the source $S_i$ of the view defined by $\varphi$, $\varphi_i(D)$ is the set of tuples obtained by applying the view definition $\varphi_i$ to instance $D$. \qed

As mentioned before, this definition considers the possibility that the set $\varphi_i(D)$ differs from the extension in the original data source $v_i$.

Example 4. (example 2 continued) If both sources were open, a legal instance for the global system would be $D = \{R(a, b), R(a, c), R(c, d), R(d, e)\}$. Another legal instance could be $D' = \{R(a, b), R(a, c), R(c, d), R(d, e), R(m, n)\}$. In general, any superset of $D$ would be a possible instance in $\text{Linst}(\mathcal{G})$. On the other hand, if both view sources were closed, there are no legal instances, except for the empty one. Finally, if both sources were clopen, there would be no legal instances. \qed

If all sources are open in a global system $\mathcal{G}$, we say that $\mathcal{G}$ is an open global system. In this case, following [2], we may say that we are working under the open world assumption. On the other hand, if all sources are clopen, we say that we are working under the closed world assumption.

Remark 1. In this paper we will concentrate on open global systems, as in [9]. In section 7 we make some comments about other sources. In consequence, if we do not label the sources, we assume they are open. \qed

Since this paper deals with the notion of consistent answers to queries, we first need a notion of answer to a query in a global system.

Definition 2. (a) [2] The ground tuple $\tilde{a}$ is a certain answer to a query $Q$ posed to a global system $\mathcal{G}$ if for every instance $D \in \text{Linst}(\mathcal{G})$, $\tilde{a} \in Q(D)$, where $Q(D)$ is the answer set for $Q$ in $D$.

(b) We denote by $\text{Certain}_\mathcal{G}(Q)$ the set of certain answers to query $Q$ in $\mathcal{G}$.

2.3 Short review of the inverse rules method

The inverse-rule algorithm [9] for generating query plans under the local as view paradigm assumes that sources are open and each source relation $V$ has a source description that defines it as a view of the global schema

$$V(\bar{X}) \leftarrow P_1(\bar{X}_1), \ldots, P_n(\bar{X}_n).$$

Then, for $j = 1, \ldots, n$,

$$P_j(\bar{X}_j) \leftarrow V(\bar{X})$$

is an inverse rule for $V$. The tuple $\bar{X}_j$ is transformed to obtain the tuple $\bar{X}_j'$ as follows: if $X$ is a constant or is a variable in $\bar{X}$, then $X$ is unchanged in $\bar{X}_j'$. Otherwise, $X$ is one of the variables $X_i$ appearing in the body of the definition of $V$, but not in $\bar{X}$. In this case, $X$ is replaced by a Skolem function term $f_{s,i}(\bar{X})$ in $\bar{X}_j'$. We denote the set of inverse rules of the collection $V$ of source descriptions in $\mathcal{G}$ by $V^{-1}$.

Given a Datalog query $Q$ and a set of conjunctive source descriptions in $\mathcal{G}$, the construction of the query plan is as follows. All the rules from $Q$ that
contain global relations that cannot be defined (directly or indirectly) in terms of the global relations appearing in the source descriptions are deleted. To the resulting query, denoted as $Q^-$, the rules in $\mathcal{V}^{-1}$ are added; and the query so obtained is denoted by $(Q^-, \mathcal{V}^{-1})$. Notice that the global predicates can be seen as EDB predicates in the rules for $Q$. However, they become IDB predicates in $(Q^-, \mathcal{V}^{-1})$, because they appear in heads of the rules in $\mathcal{V}^{-1}$. In consequence, the query plan is given essentially in terms of the source predicates.

3 Global Systems and Consistency

We assume from here on that we have a fixed set of static first order integrity global constraints, $IC$, on a global schema. We also assume that the set of ICs is consistent as a set of logical sentences. Furthermore, we will also assume that the set $IC$ is general, in the sense that there is no ground literal $L$ in the language of the global schema such that $IC \models L$. The ICs used in database praxis are always general.

We also have an open global system adapted to schema $\mathcal{R}$. In general, the global system $\mathfrak{G}$ may determine a possibly infinite set, $\text{Linst}(\mathfrak{G})$, of global instances $D$, and each of them may or may not satisfy $IC$.

We could say that a global system $\mathfrak{G}$ is consistent if every $D \in \text{Linst}(\mathfrak{G})$ satisfies $IC$.

**Example 5.** Consider the global system $\mathfrak{G}_1 = \{S_1, S_2\}$, with

\[
S_1 = \langle V_1(X,Y) \leftarrow R(X,Y), \{V_1(a, b)\} \rangle, \\
S_2 = \langle V_2(X,Y) \leftarrow R(X,Y), \{V_2(c, d)\} \rangle,
\]

and $IC$, the functional dependency $R(X,Y) : X \rightarrow Y$. $D = \{R(a, b), R(c, d)\}$ is an instance in $\text{Linst}(\mathfrak{G}_1)$ that satisfies $IC$, and $D' = \{R(a, b), R(c, d), R(a, e)\}$ is another instance in $\text{Linst}(G_1)$ that does not satisfy $IC$. In consequence, $\mathfrak{G}_1$ would be inconsistent wrt $IC$, because $D'$ violates $IC$.

In this example, the global system $\mathfrak{G}_1$ determines an infinite number of instances in which the ICs should be checked in order to sanction the system as consistent. We can see that, under such definition of consistency, it could be very easy for a global system to become inconsistent. We will have many global instances that will violate the ICs due to tuples that have no relation to the original data sources. In this sense, the notion of consistent system we suggested seems not to be the natural and useful one. We need a precise definition of consistency of a global system that somehow captures the intuitive notion of consistency related to the definitions of the sources and the only available data, namely the one in the sources.

**Example 6.** (Example 5 continued) The global system $\mathfrak{G}_1$ should be consistent wrt to the FD, because the data in the global instance $D$ that comes from the sources, namely $\{R(a, b), R(c, d)\}$, does not violate the FD. In contrast, the global
system $\mathfrak{G}_3$, defined exactly as $\mathfrak{G}_1$, but with the extensions $<\ldots,\{V_1(a,b)\}>$ and $<\ldots,\{V_2(a,c)\}>$ for the views, should be inconsistent, due to the data $\{R(a,b), R(a,c)\}$ obtained from the sources that violates the FD.

**Definition 3.** Given a global system, $\mathfrak{G}$, a minimal global instance of $\mathfrak{G}$ is an instance $D \in \text{Linst}(\mathfrak{G})$ that is minimal wrt set inclusion, i.e., there is no other instance in $\text{Linst}(\mathfrak{G})$ that is a proper subset of $D$ (as a set of atoms). We denote by $\text{mininst}(\mathfrak{G})$ the set of minimal legal global instances of $\mathfrak{G}$ wrt set inclusion.

This definition is particularly relevant in the case of open sources. In this case, there is only one minimal instance if the intersection of the elements in $\text{Linst}(\mathfrak{G})$ is again an element of $\text{Linst}(\mathfrak{G})$.

**Definition 4.** A global system $\mathfrak{G}$ is consistent wrt to a set of global integrity constraints, $IC$, if every minimal legal global instance of $\mathfrak{G}$ satisfies $IC$: for all $D \in \text{mininst}(\mathfrak{G})$, $D \models IC$.

**Example 7.** (example 5 continued) The only minimal legal global instance $D$ satisfies the FD. In consequence, the global system $\mathfrak{G}_1$ is consistent. Nevertheless, the global system $\mathfrak{G}_3$ in example 6 is inconsistent wrt the same FD. This is because its only minimal legal global instance does not satisfy FD.

**Example 8.** Consider $\mathfrak{G} = \{S_1, S_2\}$, with

$$S_1 = \langle V_1(X,Y) \leftarrow P(X,Z) \land Q(Z,Y), \{V_1(a,b)\} \rangle$$

$$S_2 = \langle V_2(X,Y) \leftarrow P(X,Y), \{V_2(a,c)\} \rangle.$$ 

In this case, the elements of $\text{mininst}(\mathfrak{G})$ are of the form $D_z = \{P(a,z), Q(z,b), P(a,c)\}$. The global FD $P(X,Y) : X \rightarrow Y$ is violated exactly in those minimal legal instances $D_z$ for which $z \neq c$. Thus, the global system is inconsistent. Notice that it would be consistent if in definition 4 we require that at least one global instance is consistent. The only certain answer to the query $\text{Ans}(X,Y) \leftarrow P(X,Y)$ is $\{(a,c)\}$. 

A global system $\mathfrak{G}$ could be inconsistent in the sense of not satisfying the given set of ICs, but still be possible or realizable in the sense that $\text{Linst}(\mathfrak{G}) \neq \emptyset$.

**Definition 5.** (a) The ground tuple $\bar{a}$ is a minimal answer to a query $Q$ posed to a global system $\mathfrak{G}$ if for every minimal instance $D \in \text{mininst}(\mathfrak{G})$, $\bar{a} \in \text{Q}(D)$, where $\text{Q}(D)$ is the answer set for $Q$ in $D$.

(b) We denote by $\text{Minimal}_{\mathfrak{G}}(Q)$ the set of minimal answers to query $Q$ in $\mathfrak{G}$.

In general, $\text{Certain}_{\mathfrak{G}}(Q) \subseteq \text{Minimal}_{\mathfrak{G}}(Q)$. For monotone queries [1], the notions of minimal and certain answers coincide. Nevertheless, in example 8 the query $\text{Ans}(X,Y) \leftarrow \neg P(X,Y)$ has $(b,a)$ as a minimal answer, but $(b,a)$ is not a certain answer, because there are legal instances that contain $P(b,a)$. Later on our queries will be allowed to contain negation. Since consistent answers have been defined relative to minimal global instances, for us the relevant notion of answer is that of minimal answer. Notice that this assumption is like imposing a form of closed world assumption to global instances associated to local sources.

---

3 For closed sources, the only minimal instance is empty.
3.1 Repairs of global systems

Given a database instance $D$, we denote by $\Sigma(D)$ the set of ground formulas $\{P(\bar{a}) \mid P \in \mathcal{R} \text{ and } D \models P(\bar{a})\}$.

**Definition 6.** [3] (a) Let $D, D'$ be database instances over the same schema and domain. The distance, $\Delta(D, D')$, between $D$ and $D'$ is the symmetric difference:

$$\Delta(D, D') = (\Sigma(D) \setminus \Sigma(D')) \cup (\Sigma(D') \setminus \Sigma(D)).$$

(b) For database instances $D, D', D''$, we define $D' \leq_D D''$ if $\Delta(D, D') \subseteq \Delta(D, D'')$, i.e., if the distance between $D$ and $D'$ is less than or equal to the distance between $D$ and $D''$.

Given a possibly inconsistent global system $\mathfrak{G}$, we want to define the notion of consistent answer from $\mathfrak{G}$ to the a query. This will be done on the basis of the possible global instances and their repairs. More precisely, given a global database instance $D$, we will be interested in those instances $D'$ that satisfy the given, global ICs and are minimal wrt the order $\leq_D$, that is, that have a minimal difference with $D$ wrt set inclusion\(^4\).

**Definition 7.** Let $\mathfrak{G}$ be a global system and $IC$ a set of global ICs. A repair of $\mathfrak{G}$ wrt $IC$ is a global database instance $D'$, i.e., an instance over global schema $\mathcal{R}$, such that:

(a) $D' \models IC$ and
(b) $D'$ is $\leq_D$-minimal for some $D \in \text{mininst}(\mathfrak{G})$.

We can see that a repair of a global system is a global database instance that minimally differs from a minimal legal global database instance. Notice that if $\mathfrak{G}$ is consistent (definition 4), then the repairs are exactly the elements in $\text{mininst}(\mathfrak{G})$.

**Example 9.** Consider the global system $\mathfrak{G}_4 = \{S_1, S_2\}$, with

\[ S_1 = \langle V_1(X) \leftarrow R(X,Y), \{V_1(a)\} \rangle, \]
\[ S_2 = \langle V_2(X) \leftarrow R(X,Y), \{V_2(a)\} \rangle, \]

and the global FD $R(X,Y): X \rightarrow Y$. In this case, $D = \{R(a,b_1), R(a,b_2)\} \in \text{Linst}(\mathfrak{G}_4)$, but $D \not\models IC$. However, $D \not\in \text{mininst}(\mathfrak{G}_4)$. Actually, the elements in $\text{mininst}(\mathfrak{G}_4)$ are of the form $\{R(a,b)\}$, for some $b$ in the global database domain. The elements in $\text{mininst}(\mathfrak{G}_4)$ coincide with the repairs. Notice that $\mathfrak{G}_4$ is a consistent global system.

Notice that in this definition of repair we are not requiring that a repair respects the (open) labels, i.e. that the instantiation of each view definition in the repair contains the corresponding view extension in the source. That is it may be the case that a repair -still a global instance- does not belong to $\text{Linst}(\mathfrak{G})$. If we do not allow this kind of label violation, then a global system might not be repairable.

\(^4\) Notice from definition 6 that built-in predicates do not contribute to the $\Delta$s, because they have fixed extensions, identical in every database instance.
Example 10. Consider the global system $\mathcal{G}_6 = \{S_1, S_2\}$, with

$$S_1 = \{V_1(X, Y) \leftarrow R(X, Y), \{V_1(a, b)\}\},$$
$$S_2 = \{V_2(X, Y) \leftarrow R(X, Y), \{V_2(a, b)\}\},$$

and the FD $R(X, Y) : X \rightarrow Y$. The only element in $\text{mininst}(\mathcal{G}_6)$ is $D_0 = \{R(a, b_1), R(a, b_2)\}$, that does not satisfy IC. The global system is inconsistent. The only repairs are the global instances that minimally differ from $D_0$ and satisfy the FD, namely $D^1_0 = \{R(a, b_1)\}$ and $D^2_0 = \{R(a, b_2)\}$. Notice that they do not belong to $\text{Liminst}(\mathcal{G}_6)$.

Example 11. Consider the global system $\mathcal{G}_6 = \{S_1, S_2\}$, with

$$S_1 = \{V_1(X, Y, Z) \leftarrow R_1(X, Y, Z), R_2(X, U), \{V_1(a, b, 1), V_1(e, f, -5), V_1(l, m, 3)\}\},$$
$$S_2 = \{V_2(X, Y) \leftarrow R_3(X, Y), \{V_2(a, b)\}\},$$

and the inclusion dependency IC: $\forall X Y (R_1(X, Y, Z), Z > 0 \rightarrow R_0(X, Y))$. The elements in $\text{mininst}(\mathcal{G}_6)$ are of the form

$$D_{c_1, c_2, c_3} = \{R_1(a, b, 1), R_1(e, f, -5), R_1(l, m, 3), R_2(a, c_1), R_2(e, c_2), R_2(l, c_3), R_3(a, b)\},$$

where $c_1, c_2, c_3$ are any elements in the underlying domain. They do not satisfy IC because $R_0(l, m) \notin D_{c_1, c_2, c_3}$. The global system is inconsistent. The next two global instances minimally differ from $D_{c_1, c_2, c_3}$ and satisfy IC, in consequence they are repairs of $D_{c_1, c_2, c_3}$:

$$D^1_{c_1, c_2, c_3} = \{R_1(a, b, 1), R_1(e, f, -5), R_2(a, c_1), R_2(e, c_2), R_2(l, c_3), R_3(a, b)\}$$
$$D^2_{c_1, c_2, c_3} = \{R_1(a, b, 1), R_1(e, f, -5), R_1(l, m, 3), R_2(a, c_1), R_2(e, c_2), R_2(l, c_3), R_3(a, b), R_3(l, m)\}$$

Notice that, as in [3], we are not interested in the repairs by themselves. We are not interested in repairing the global system, neither its instances of any kind. Repairs will be used as an auxiliary notion to define consistent answers.

3.2 Consistent answers to global queries

There are algorithms in the literature for obtaining certain answers to a global query $Q$ from a global system $\mathcal{G}$. This system may violate desired, global integrity constraints IC. Our goal is to characterize those answers to $Q$ that are consistent with IC, even when the global system is inconsistent as a whole. As in [3], we formalize this notion appealing to the repairs of the global system. Nevertheless, in this new scenario, we do not start form a single, possibly inconsistent relational databases instance, but from a possibly inconsistent global system, with a collection of implicit instances.
Definition 8. (a) Given a global system $\mathcal{G}$, a set of global integrity constraints $IC$, and a global first order query $Q(\bar{X})$, we say that a (ground) tuple $\bar{t}$ is a consistent answer to $Q$ wrt $IC$, denoted by $\mathcal{G} \models e \cdot Q[\bar{t}]$, iff for every repair $D$ of $\mathcal{G}$, $D \models Q(\bar{t})$.
(b) We denote by $\text{Cons}_{\mathcal{G}}(Q)$ the set of consistent answers to query $Q$ in $\mathcal{G}$.

Example 12. (example 10 continued) For the query $Q_1(X) : \exists Y \ R(X,Y)$, $a$ is a consistent answer, i.e. $\mathcal{G}_a \models e \cdot \exists Y \ R(X,Y)[a]$. Instead, the query $Q_2(X,Y) : R(X,Y')$, does not have any consistent answer. Nevertheless, a query plan for $Q_2$, e.g. found according to [10], without caring about inconsistency, would be:

$$\text{Ans}(X,Y) \leftarrow R(X,Y)$$
$$R(X,Y) \leftarrow V_1(X,Y)$$
$$R(X,Y) \leftarrow V_2(X,Y),$$

that evaluated on the data sources would give the answers $[a, b_1], [a, b_2]$, that are not consistent answers.

Example 13. (example 11 continued) For the query $Q_1(X,Y,Z) : R_1(X,Y,Z)$, $[a, b, 1]$ and $[e, f, -5]$ are consistent answers, i.e. $\mathcal{G}_a \models e \cdot R_1(X,Y,Z)[a,b,1]$ and $\mathcal{G}_e \models R_1(X,Y,Z)[e,f,-5]$. However the certain answers to query $Q_1(X,Y,Z)$ are $[a,b,1],[e,f,-5]$ and $[k,m,3]$.

Proposition 1. Given an open global system $\mathcal{G}$, a set of global integrity constraints $IC$ such that $IC \not\models L$ for every ground literal $L$, and a global query $Q$, it holds:
(a) Every consistent answer is a minimal answer.
(b) If $\mathcal{G}$ is consistent wrt $IC$, then every minimal answer is consistent.

4 Computing Consistent Answers

After having given a semantic definition of consistent answer to a global query from a global system, we concentrate on the computational problem of consistent query answering (CQA). For this purpose, in [3], but in the case of a stand alone relational database, the operator $\mathcal{T}^*$ was introduced. It does the following: Given a first order query $Q(\bar{X})$, a modified query $\mathcal{T}^*(Q(\bar{X}))$ is computed. The new query $\mathcal{T}^*(Q(\bar{X}))$ is posed to the original database, and the returned answers are consistent in the semantic sense. In section 4.2 we will use this operator to transform queries posed to a global systems. In section 4.1 we give a short description of the rewriting algorithm as introduced in [3].

4.1 The rewriting operator for CQA

We consider universal first order integrity constraints expressed in the so-called “standard format” [3]: $\forall \left( \bigvee_{i=1}^m l_i(\bar{x}_i) \right) \lor \varphi$, where $l_i$ is a database literal and $\varphi$ is a formula containing built-in predicates only.
The new query $T^\omega(\varphi(x))$ is computed via the iterative $T$ operator, which transforms an arbitrary query by appending the corresponding residue to each database literal appearing in the query until a fixed point is reached. The residue of a database literal forces the local satisfaction of the ICs for the tuples satisfying the literal. Residues can be obtained by resolution between the table and the ICs.

**Example 14.** Consider $IC = \{ R(X) \lor \neg P(x) \lor \neg Q(x), \ P(x) \lor \neg Q(x) \}$ and the query $\varphi(x) = Q(x)$. The residue of $Q(x)$ wrt the first IC is $(R(x) \lor \neg P(x))$, because if $Q(x)$ is to be satisfied, then that residue has to be true if the IC is to be satisfied too. Similarly, the residue of $Q(x)$ wrt the second IC is $P(x)$. The operator $T$ iteratively appends the residues to the tables in the queries.

\[
T^1(\varphi(x)) = Q(x) \land (R(x) \lor \neg P(x)) \land P(x).
\]

\[
T^2(\varphi(x)) = T(T^1(\varphi(x))) = Q(x) \land (T(R(x)) \lor T(\neg P(x))) \land T(P(x)).
\]

\[
T^3(\varphi(x)) = Q(x) \land (R(x) \lor (\neg P(x) \land \neg Q(x))) \land P(x) \land (T(R(x)) \lor T(\neg Q(x))).
\]

Since $T(\neg Q(x)) = \neg Q(x)$ and $T(R(x)) = R(x)$, we obtain $T^2(\varphi(x)) = T^3(\varphi(x))$, and a fixed point has been reached. Since $T^\omega(\varphi(x)) := \bigcup_{n<\omega} \{ T_n(\varphi(x)) \}$, here we obtain $T^\omega(\varphi(x)) = T_1(\varphi(x)) \land T_2(\varphi(x))$. \qed

The methodology based on the $T$ operator works for conjunctive queries only. In the rest of this paper we will concentrate on this case. Notice that sometimes the fixed point may be hard or impossible to detect. To address this problem and other implementation issues, an improved algorithm, QUECA, to compute $T^\omega$ was presented in [6].

### 4.2 Computing consistent answer to global queries

In the context of integration of open data sources, under the local-as-view paradigm, with global universal integrity constraints, $IC$, we now present an algorithm to compute consistent answers to conjunctive queries. Given a global query $Q(\bar{x})$, the algorithm starts applying the $T$ operator, and next the rewritten query is answered by producing an appropriate query plan. At a high level, the algorithm is as follows:

**Algorithm:**

- Input: a conjunctive global query $Q$.
- Output: a query plan to obtain consistent answers to $Q$.

1. Rewrite $Q(\bar{x})$ into the first order query $T^\omega(Q(\bar{x}))$ applying the algorithm presented in 4.1 using $IC$ as the input set of ICs.
2. Using a standard methodology [1,16], transform $T^\omega(Q(\bar{x}))$ into a Datalog$^-$ program, i.e. a Datalog program with negation, $\Pi(T^\omega(Q))$.

3. Find a query plan to answer $\Pi(T^\omega(Q))$ seen as a query to the global system.

4. Evaluate the query plan on the view extensions of $\mathcal{G}$ to compute a set of answers.

The most problematic step, apart from the limitations of operator $T$, is 3., because the resulting Datalog program may contain negation. There are no mechanisms to obtain query plans for global queries containing negation (although some heuristics are sketched in [11]). On the positive side, the generated Datalog$^-$ program does not contain recursion.

In some cases, when no negation appears in the rewritten query, e.g. when we obtain a positive conjunctive query, we can use the views in $\mathcal{G}$, and apply the inverse rule algorithm in [9] to obtain the query plan $P(Q)$. Otherwise, in the presence of negation, we need to extend that methodology to generate query plans. In section 5 we present an extension of the algorithm in [9], considering negation (but no recursion). We first present an example where we do not need an extended query plan generator. Next we show an example, where negation is obtained.

**Example 15.** Consider the global system $\mathcal{G}_7 = \{S_1, S_2\}$, with

$$S_1 = \langle V_1(X,Y) \leftarrow R_1(X,Y), \{V_1(a,b), V_1(c,d)\} \rangle$$

$$S_2 = \langle V_2(X,Y) \leftarrow R_2(X,Y), \{V_2(a,b), V_2(m,n)\} \rangle,$$

and the global integrity constraint $IC : R_1(X,Y) \rightarrow R_2(X,Y)$

In this case there is only one instance in $\text{mininst}(\mathcal{G}_7)$:

$$D_0 = \{R_1(a,b), R_1(c,d), R_2(a,b), R_2(m,n)\}.$$ 

Clearly, this instance does not satisfy the $IC$, because $R_2(c,d) \notin D$. Therefore, the global system $G_7$ is inconsistent and repairs need to be considered. They are

$$D_0^1 = \{R_1(a,b), R_2(a,b), R_2(m,n)\}$$

$$D_0^2 = \{R_1(a,b), R_1(c,d), R_2(a,b), R_2(m,n), R_2(c,d)\}.$$ 

If we have the global query $Q(X,Y) : R_1(X,Y)$ and we do not care about consistency, a query plan can be obtained applying the inverse rule algorithm [9] to query $Q$:

$$\text{Ans}(X,Y) \leftarrow R_1(X,Y).$$

$$R_1(X,Y) \leftarrow V_1(X,Y).$$

$$R_2(X,Y) \leftarrow V_2(X,Y).$$
Evaluating this query plan on the view extensions, tuples $[a, b]$ and $[c, d]$ are obtained as certain answers to $Q(X, Y)$. However, tuple $[c, d]$ is not a consistent answer according to definition 8.

Let us now apply the Algorithm for CQA. After the first step, using $IC$ and $Q$, we obtain:

$$T^c(Q(X, Y)) = R_1(X, Y) \land R_2(X, Y).$$

Now, we proceed with step 2, translating the first order query $T^c(Q(X, Y))$ into the Datalog program $\Pi(T^c(Q(X, Y)))$:

$$\text{Ans}(X, Y) \leftarrow R_1(X, Y), R_2(X, Y).$$

Because this Datalog program is a conjunctive query, we can proceed with step 3; a query plan $P(Q)$ for $\Pi(T^c(Q))$ using the view definitions in $\mathcal{G}_7$ is generated:

$$\text{Ans}(X, Y) \leftarrow R_1(X, Y), R_2(X, Y)$$

$$R_1(X, Y) \leftarrow V_1(X, Y)$$

$$R_2(X, Y) \leftarrow V_2(X, Y).$$

Finally, at step 4, we evaluate this query plan on the view extensions (in a bottom-up manner). The tuple $[a, b]$ is obtained, which is the only consistent answer to $Q(X, Y)$. □

**Example 16.** Consider the global system $\mathcal{G}_8 = \{S_1, S_2\}$, with

$$S_1 = \langle V_1(X, Y) \leftarrow R(X, Y), \{V_1(a, b)\} \rangle$$

$$S_2 = \langle V_2(X, Y) \leftarrow R(X, Y), \{V_2(a, c), V_2(c, d)\} \rangle,$$

the FD $R(X, Y): X \rightarrow Y$, and the global query $Q(X, Y): R(X, Y)$.

Here, the only element in $\text{mininst}(\mathcal{G}_8)$ is $D_0 = \{R(a, b), R(a, c), R(c, d)\}$. The only minimal instance violates FD through the tuples $[a, b], [a, c]$. In consequence, the global system $\mathcal{G}_8$ is inconsistent. It has two repairs, $D_1^0 = \{R(a, b), R(c, d)\}$ and $D_2^0 = \{R(a, c), R(c, d)\}$. The only consistent answer to the query is the tuple $[c, d]$.

A query plan Plan($Q$) for query $Q$ obtained using the inverse rules method [9], without caring about consistency, is:

$$\text{Ans}(X, Y) \leftarrow R(X, Y)$$

$$R(X, Y) \leftarrow V_1(X, Y)$$

$$R(X, Y) \leftarrow V_2(X, Y),$$

which evaluated on the view extensions in $\mathcal{G}$, retrieves inconsistent certain answers, namely $[a, b]$ and $[a, c]$.

We now apply the Algorithm for CQA. First of all, we need the FD expressed in a first order language, it becomes $\forall XYZ(R_1(X, Y) \land R_1(X, Z) \rightarrow Y = Z)$. 
At step 1, the query has to be rewritten. We obtain

\[ T^\omega(Q(\bar{X})) \land \neg \exists Z(R(X, Z) \land Z \neq Y). \]  

(3)

This query can also be translated into the following Datalog^- program \( \Pi(T^\omega(Q)) \):

\[
\begin{align*}
\text{Ans}(X, Y) & \leftarrow R(X, Y), \text{ not } S(X, Y) \\
S(X, Y) & \leftarrow R(X, Z), Y \neq Z.
\end{align*}
\]

We need a query plan to answer this query program containing negation. \[ \square \]

5 Plans for Queries with Negation

In [9], the inverse-rule algorithm for generating a query plan for a global Datalog query \( Q \) is presented. This plan produces an answer set that is maximally-contained in the answer set for \( Q \). In [17], it is shown that such a maximally-contained query plan retrieves all certain answers to the query \( Q \).

A limitation of the inverse-rules algorithm for query plans is that it allows Datalog queries only. In our case, even if we start with a conjunctive global query, the query program obtained after transforming the query rewritten using the \( T \) operator may contain negation. In consequence, we need to find plans for recursion-free Datalog^- query programs with built-in predicates, like the \( \Pi(T^\omega(Q)) \)'s. Notice that the absence of recursion immediately makes the resulting query programs stratified [1].

Given a recursion-free Datalog^- query \( Q \) in terms of global and defined relations, the extended inverse rules algorithm works analogously to the one presented in [9] (see section 2.3), except that in the case a rule in \( Q \) contains a negated literal in the body, say \( S(\bar{x}) \leftarrow \ldots, L_1(\bar{x}), \neg G(\bar{x}), L_n(\bar{x}), \ldots \), then it is checked if \( G \) can be eventually evaluated in terms of the global relations appearing in the source descriptions (because those are the global relations that can be eventually evaluated using the inverse rules). If this is not the case, then that goal is eliminated, obtaining the modified rule \( S(\bar{x}) \leftarrow \ldots, L_1(\bar{x}), L_n(\bar{x}), \ldots \). We show the methodology by means of an example\(^5\).

Example 17. Consider the global system \( \mathcal{G}_9 = \{S_1, S_2\} \) with

\[
S_1 = \langle V_1(X, Z) \leftarrow R_1(X, Y), R_2(Y, Z), \{V_1(a, b)\} \rangle
\]

\[
S_2 = \langle V_2(X, Y) \leftarrow R_3(X, Y), \{V_2(c, d)\} \rangle.
\]

The global query \( Q \) is:

\[
\begin{align*}
\text{Ans}(X, Z) & \leftarrow R_1(X, Y), R_2(Y, Z), \text{ not } R_4(X, Y) \\
R_4(X, Y) & \leftarrow R_3(X, Y), \text{ not } R_5(X, Y) \\
R_7(X) & \leftarrow R_1(X, Y), R_6(X, Y).
\end{align*}
\]

\( ^5 \) It should be easy to extend this methodology to stratified Datalog^- queries, but we do not need this extension here.
The inverses rules $\mathcal{V}^{-1}$ are obtained from the source descriptions:

$$R_1(X, f_1(X, Z)) \leftarrow V_1(X, Z)$$
$$R_2(f_1(X, Z), Z) \leftarrow V_1(X, Z)$$
$$R_3(X, Y) \leftarrow V_2(X, Y).$$

To compute a query plan for $Q$, we first need $Q^-$:

$$Ans(X, Z) \leftarrow R_1(X, Y), R_2(Y, Z), \text{not } R_4(X, Y)$$
$$R_4(X, Y) \leftarrow R_5(X, Y).$$

The literal not $R_5(X, Y)$ was eliminated from rule (4) because it does not appear in any source description. For the same reason (the literal $R_6(X, Y)$ does not appear in any source description), rule (5) was eliminated. Then, the query plan $P: (Q^-, V^{-1})$ is:

$$Ans(X, Z) \leftarrow R_1(X, Y), R_2(Y, Z), \text{not } R_4(X, Y)$$
$$R_4(X, Y) \leftarrow R_5(X, Y)$$
$$R_1(X, f_1(X, Z)) \leftarrow V_1(X, Z)$$
$$R_2(f_1(X, Z), Z) \leftarrow V_1(X, Z)$$
$$R_3(X, Y) \leftarrow V_2(X, Y).$$

Finally, the query plan can be evaluated in a bottom-up manner to retrieve $Ans(a, b)$ as final answer for the global query $Q$.  

5.1 Containment

In this section we will prove that the resulting query plan is maximally contained in the original query. We will concentrate on recursion free Datalog− programs (including built-ins).

Given a query plan $P$ for $Q$ using $\mathcal{V}$, the expansion $P^{exp}$ of $P$ is obtained from $P$ by replacing all source relations in $P$ with the bodies of the corresponding views $\mathcal{V}$, and using fresh variables for existential variables in the views. That is, the source relations are eliminated by reinverting the inverse rules, in order to be in position to compare the the original query, expressed in terms of global predicates, and the query plan.

**Example 18.** (example 17 continued) To compute $P^{exp}$, we first rename the IDB predicates in $P$.

$$Ans(X, Z) \leftarrow P_1(X, Y), P_2(Y, Z), \text{not } P_4(X, Y)$$
$$P_4(X, Y) \leftarrow P_5(X, Y)$$
$$P_1(X, f_1(X, Z)) \leftarrow V_1(X, Z) \quad (6)$$
$$P_2(f_1(X, Z), Z) \leftarrow V_1(X, Z) \quad (7)$$
$$P_3(X, Y) \leftarrow V_2(X, Y). \quad (8)$$
The expansion is the following Datalog \textsuperscript{−} query $P^{exp}$:

\[
\begin{align*}
\text{Ans}(X, Z) & \leftarrow P_1(X, Y), P_2(Y, Z), \text{ not } P_4(X, Y) \\
P_0(X, Y) & \leftarrow P_5(X, Y) \\
P_1(X, f_1(X, Z)) & \leftarrow R_1(X, U), R_2(U, Z) \\
P_2(f_1(X, Z), Z) & \leftarrow R_1(X, U), R_2(U, Z) \\
P_5(X, Y) & \leftarrow R_3(X, Y) \square \\
\end{align*}
\]

Remark 2. Notice that in the extended plan obtained in the previous example, we could keep the rules (6), (7), (8), plus the source definitions in $S_1, S_2$ of example 17. Sometimes we will do this in the rest of this section.

Theorem 1. For every recursion free Datalog \textsuperscript{−} program $Q$, every set of conjunctive source descriptions $V$, and all finite instances of the source relations, the query plan $(Q^-, V^{-1})$ has a unique finite minimal fixpoint. Furthermore, bottom-up evaluation is guaranteed to terminate, and produces this unique fixpoint. \square

Definition 9. (maximally-contained plan [17]) A query plan $P$ is maximally contained in a query $Q$ using views $V$ if $P^{exp} \subseteq Q$ and for every query plan $P'$ such that $(P')^{exp} \subseteq Q$, it holds $P' \subseteq P$. \square

If $S = \{v_1, \ldots, v_m\}$ is a set of extensions for the sources, and $P$ is a query plan, then $P(S)$ denotes the extension of the query predicate $\text{Ans}$, obtained by evaluating $P$ on $S$. Notice that $P(S)$ may contain tuples with Skolem function symbols. We denote by $P(S) \downarrow$ the set of tuples in $P(S)$ that do not contain function symbols, and by $P_i$ the plan that computes this pruned extension.

In order to prove maximal containment, the notion of proof tree presented in [19] for Datalog queries can be extended to recursion free Datalog \textsuperscript{−} queries (see appendix).

Theorem 2. (compare [9], thm. 8) For every recursion free Datalog \textsuperscript{−} program $Q$ and every set of conjunctive source descriptions $V$, the query plan $(Q^-, V^{-1}) \downarrow$ is maximally contained in $Q$. \square

Remark 3. From the proof of maximality in theorem 2, we can see that the theorem can be restricted to minimal global instances. That is, the query plan $(Q^-, V^{-1}) \downarrow$ is contained in $Q$, i.e. $(Q^-, V^{-1})^{exp}(D) \subseteq Q(D)$ for every minimal global instance $D$ (this is an immediate consequence of the theorem, that holds for arbitrary global instances); and every other plan, whose expansion is contained in $Q$ for minimal global instances, is also contained in $(Q^-, V^{-1}) \downarrow$ (this follows from a particular minimal global instance $D$ constructed in the proof).

Theorem 3. Given an open global system $\mathcal{G}$, $IC$ a set of general ICs, and a recursion free Datalog \textsuperscript{−} query $Q$ with built-ins, the plan $\text{Plan}(Q)$ obtained with the extended inverse rules algorithm retrieves exactly $\text{Minimal}_\mathcal{G}(Q)$. \square

\textsuperscript{6} Here, $P^{exp} \subseteq Q$ means that the extension of the query predicate in $P^{exp}$ is included in the extension of $Q$ for every database instance over the global schema.
6 The CQA Algorithm Revisited

We have presented an algorithm for generation of query plans for global queries that are Datalog\textsuperscript{+\text{-}} queries, more specifically Datalog\textsuperscript{-} queries without recursion. Let us denote by Plan\((Q)\) the query plan for query \(Q\). Then, the algorithm for CQA is as follows:

0. Given a conjunctive query \(Q\) that waits for its consistent answer.
1. Obtain the FO query \(T^\omega(Q)\).
2. Translate \(T^\omega(Q)\) into a Datalog\textsuperscript{-} program \(Q'\).
3. Obtain Plan\((Q')\).
4. Evaluate Plan\((Q')\) on the sources for \(\mathfrak{S}\).
5. Return the answers in 3. as consistent answers to \(Q\).

Example 19. (example 16 cont.) Applying the extended inverse rules methodology to the Datalog\textsuperscript{+\text{-}} we had obtained, the following query plan Plan\((\Pi(T^\omega(Q)))\) can be generated:

\[
\text{Ans}'(X,Y) \leftarrow R(X,Y), \text{ not } S(X,Y)
\]
\[
S(X,Y) \leftarrow R(X,Z), Y \neq Z
\]
\[
R(X,Y) \leftarrow V_1(X,Y)
\]
\[
R(X,Y) \leftarrow V_2(X,Y).
\]

This query plan can be evaluated on the view extensions in \(\mathfrak{S}_8\) to obtain the answer \([c,d]\). This answer is consistent: \(\mathfrak{S}_8 \models c R(X,Y)[c,d]\). Notice that the original query (3) could be evaluated instead of the program using SQL2, defining first \(R\) as a view that is the union of \(V_1\) and \(V_2\).

\[\square\]

Proposition 2. Given an open global system \(\mathfrak{S}\), IC a set of general ICs, the consistent answers to a conjunctive global query \(Q\) correspond to Minimal\((T^\omega(Q))\). Furthermore, if \(T^\omega(Q)\) is monotone, then its certain answers are the consistent answers to \(Q\).

\[\square\]

Theorem 4. Given an open global system \(\mathfrak{S}\), IC a set of general ICs, and a conjunctive global query \(Q\), Plan\((\Pi(T^\omega(Q)))\) retrieves exactly Consist\(_e(Q)\).

\[\square\]

7 Conclusions

In this paper we have concentrated on the local-as-view paradigm. However, consistency issues are present with other approaches as well. Nevertheless, the problem of obtaining consistent answers to global queries becomes more interesting in the local-as-view approach. First, this approach is better suited to easily add new sources, without much consideration for the other sources; in consequence inconsistencies are more likely to occur. Second, with local-as-view approach, we find the issue of multiple global instances we had to deal with; whereas with
the global-as-view approach, the global instances are directly derivable from the materialized sources. Third, already existing techniques for obtaining consistent answers, like [3], cannot be directly applied in the local-as-view scenario. This is because complex rewritten global queries appear from the very beginning, for which techniques for deriving query plans have not being developed yet. Instead, with the global-as-view approach, it is easier to answer global queries by rule unfolding, in particular rewritten queries as obtained with the methodology developed in [3]. Actually, all the notions and techniques presented in [3] for single relational databases are more easily applicable in the global-as-view integration scenario. Note, however, as recently pointed in [8], when global integrity constraints are present, the query folding on the global-as-view approach can be also very difficult.

We have also focused specially on open sources. In the future, it would interesting to extend the semantic and algorithm work presented in this paper to consider open, closed and clopen sources on the global system. The work in [10] introduces an interesting framework to deal with this kind of global system that can be explored further, particularly in presence of global integrity constraints.

The problem of query containment is crucial in the context of data integration. However, most of this work has focused on conjunctive queries. An interesting challenge is to expand the universe to more expressive queries. In this line, the work [21] introduces an improved method for testing query containment for union of general conjunctive queries. The methodology we have presented here is based in its turn on the methodology presented in [3]. In consequence, it applies to a limited class of queries and constraints. Other approaches to consistent query answering based on logic programs with stable model semantics were presented in [4, 7, 12]. They can handle general first order queries in the context of a stand alone relational database. It would be interesting to see how the methodology presented there could be integrated with the methodology presented in this paper to consistently answer conjunctive queries posed to global integrated systems under the local-as-view paradigm.

In [11], a logic-based approach is presented for transforming a query to the global schema into a query plan to the sources. That methodology takes into account, on a deductive basis, the global ICs in the derivation of the query plan. In consequence, it is close in spirit to the methodology developed in [3] to rewrite queries to the inconsistent instance into a new query to be posed to the original database as well. The desired, but possibly violated ICs are used in the rewriting process.

Acknowledgments: Work supported by FONDECYT Grant 1000593, NSF Grant INT-9901877/CONICYT Grant 1998-02-083, NSF Grant IIS-0119186, Carleton University Start-Up Grant 9364-01, NSERC Grant 250279-02, Nucleus Millenium for Web Research (Mideplan, Grant P01-029-F). We are grateful to Alberto Mendelzon for stimulating and useful conversations.
References

Appendix A: Proof trees

Let \( \Pi \) be a Datalog\(^-\) program with built-in predicates, \( C \) a ground literal, \( d \in \{ +1, -1 \} \), and \( I \) a database instance that provides the extensions for the EDB predicates.

**Definition 10.** A \( d \)-proof-tree for \( C \) from \( I \) and \( \Pi \) is a tree, whose nodes have all labels of the form \( (s, A) \), with \( s \in \{ +1, -1 \} \) and \( A \) a (positive, ground) fact, that is constructed as follows:

1. The label of the root is
   (a) \((d, A)\) if \( C = A \), with a positive \( A \)
   (b) \((-d, A)\) if \( C = \neg A \), with a positive \( A \).
2. Each leaf is labeled by either:
   (a) \((+1, A)\), where \( A \) is a fact in \( I \), or
   (b) \((-1, B)\), where \( B \) is not a fact in \( I \) and \( B \) does not appear in the head of any ground instantiation of a rule in \( \Pi \).
3. For each internal vertex with label \((+1, A)\), there exists a ground instantiation \( A \leftarrow \bar{A}_1, \ldots, \bar{A}_n \) of a program rule in \( \Pi \) and for each \( \bar{A}_j \) there is a child which is the root of a \((+1)\)-proof-tree of \( A_j \).
4. For each internal vertex with label \((-1, B)\), it holds that for each ground instantiation \( B \leftarrow \bar{B}_1, \ldots, \bar{B}_n \) of a program rule, there is some \( j \) and a child which is the root of a \((-1)\)-proof-tree of \( B_j \).

A proof-tree will mean in what follows a \( d \)-proof tree for some \( d \). Notice that all positive leaves (i.e. with first component \(+1\)) are EDB atoms, but a negative leave is never an EDB atom and cannot be further expanded using rules in the program. For “EDB leaves”, the set of positive (resp. negative) leaves of a \((d)\)-proof-tree \( T \) is denoted by \( \text{pos}(T) \) (resp. \( \text{neg}(T) \)). In our case, we will apply the notion of proof tree to ground query atoms \( \text{Ans}(\bar{t}) \), where predicate \( \text{Ans} \) is defined by means of a Datalog\(^-\) program.

The proof-tree presented here extend those presented in [19] for Datalog programs and queries. For each query in stratified Datalog\(^-\), and under NAF, a proof for a query atom \( Q \) on an instance \( I \) has as a witness an extended proof-tree.

Notice that every proof-tree has a depth and a branching factor bounded by a constant depending only on the program. Hence the number of leaves is bounded by a constant \( l \). Let \( m \) be the maximum arity of the EDB predicates occurring in \( P \). Let \( A = \{ a_i \}, i < ml \) be a set of different constants.

Appendix B: Proofs

**Proof of Proposition 1** (a) Let \( \bar{t} \in \text{Consis}_\emptyset(Q) \). Then for every repair \( D \) of \( \emptyset \), \( D \models Q(\bar{t}) \). By the hypothesis on \( IC \), \( D \models Q(\bar{t}) \) for every \( D \in \text{mininst}(\emptyset) \), then \( \bar{t} \in \text{Minimal}_\emptyset(Q) \).

(b) In this case, the repairs coincide with the elements of \( \text{mininst}(\emptyset) \), and then
**Proof of Theorem 1:** $Q^-$ contains negation, but does not introduce function symbols nor recursion. On the other hand, $V^{-1}$ introduces function symbols, but is not recursive nor contains negation. Therefore, every bottom-up evaluation of $(Q^-, V^{-1})$ will necessarily progress in two stages. In the first stage, the extensions of the IDB predicates in $V^{-1}$ are determined. The second stage will then be a normal Datalog evaluation $Q^-$. Because normal Datalog queries have a finite minimal fixpoints, this proves the claims.

**Lemma 1.** Let $Q$ be a global query and $V$ a source description based on the global EDB predicates of $Q$, and $D$ a global instance. Then: $v' \leftarrow e'_1, \ldots, e'_r$ is a function free ground instantiation of a rule of $V$ in $D$ iff $e' \leftarrow v'$ is a function free ground instantiation of an inverse rule in $V^{-1}$.

**Proof of Theorem 2:** (Containment) We need to prove that $(Q^-, V^{-1}) \downarrow^\exp \subseteq Q$. Let $D = d_1, \ldots, d_n$ be instances of the global EDB predicates of $Q$. Let $\text{Ans}(\overline{t}) \in (Q^-, V^{-1})^\exp(D)$; $\overline{t}$ without function symbols. Let $T$ be a proof-tree for $\text{Ans}(\overline{t})$ from $(Q^-, V^{-1})^\exp$ and $D$.

If there is no occurrence of a view predicate $V$ in $\overline{t}$, then this is a proof tree from $Q$ and $D$, and we are done: $\text{Ans}(\overline{t}) \in Q(D)$. If some $V$ occurs in $T$, due to the form of the inverse rules, it must occur as an internal node with children $p_1, \ldots, p_n$ that are ground facts in $D$, in particular, without functions. Due to the form of the inverse rules, its parent must be one of the atoms $p_i$. Notice that there is at most one occurrence of predicate $V$ in a branch. Then, for each such positive occurrence of a view predicate $V$ we can prune the subtree with root $V$. In case of a negative occurrence of $V(\overline{a})$ ($\neg V(\overline{a})$, or equivalently, $(\neg, V(\overline{a}))$, again by the form of the rules, the parent must be of the form $(\neg, \overline{P(\overline{a})})$, with $P$ a global predicate. In this case we can prune the subtree with root $(\neg, V)$, because if we developed the node $(\neg, \overline{P})$ with ist child $(\neg, V)$, that is because $P(\overline{a})$ was not in $D$. Pruning all $V$ nodes in this way, leaves us with a proof tree for $\text{Ans}(\overline{t})$, but now from $Q$ and $D$.

(Maximality) Assume that $P$ is a plan such that $P^\exp \subseteq Q$. We need to prove that $P \subseteq (Q^-, V^{-1}) \downarrow$.

Let $S$ be an instance of $V$, and let $\text{Ans}(\overline{t}) \in P(S)$, $\overline{t}$ without functions. Let $D$ be the global instance produced applying the inverse rules $d \leftarrow v$ from $S$. Then there is a proof for $\text{Ans}(\overline{t})$ with leaves containing ground view literals. Those leaves can be further expanded using the source definitions, and new global ground literal will now appear as leaves. By the construction of $D$, we obtain a proof tree for $\text{Ans}(\overline{t})$ from $P^\exp$ and $D$. Then, $\text{Ans}(\overline{t}) \in P^\exp(D)$. Thus, by the hypothesis, $\text{Ans}(\overline{t}) \in Q(D)$. Now we need to prove that $\text{Ans}(\overline{t}) \in (Q^-, V^{-1}) \downarrow$.

Assuming we kept the view predicates in the expanded plan as indicated in remark 2.
Let \( T \) be a proof-tree for \( \text{Ans}(\tilde{t}) \) from \( Q \) and \( D \). Each fact \( d \in D \) comes from a ground instance \( d^* \leftarrow v' \), where \( v' \) is a fact in the sources in \( S \). Hence we extend the tree \( T \) adding to each positive leaf \( (+, e') \) the child \( v' \), with \( v' \) in \( D \) coming from a ground instance \( e^* \leftarrow v' \) of some rule. With respect to negative leaves \( (-, e') \), just note that if \( e' \notin D \), then no instance of rules \( e \leftarrow v \) can instantiate it (by definition of \( D \)). For negative leaves whose predicates are not in the schema of \( D \), they stay the same, i.e. there are no instantiation for them. Hence we obtain a proof-tree for \( \text{Ans}(\tilde{t}) \) from \( (Q^-, V^{-1}) \) and \( S \).

\[ \square \]

**Proof of Theorem 3:** The proof is exactly as in theorem 4.2 stated in [2] for certain answers. The same proof goes through for minimal answers, because we can apply the property of maximal containment of the query plan \( (Q^-, V^{-1}) \downarrow \), i.e. \( \text{Plan}(Q) \), relative to minimal global instances (see remark 3).

\[ \square \]

**Proof of Proposition 2:** Let \( \tilde{t} \in \text{Consist}_\phi(Q) \). Then, for every \( D \in \text{mininst}(\phi) \) and every repair \( D' \) of \( D \): \( \tilde{t} \in Q(D') \), with \( Q(D') \) the answer set to \( Q \) in \( D' \). Then, by the results in [3], for every \( D \in \text{mininst}(\phi), \tilde{t} \in T^\omega(Q)(D) \), that is, \( \tilde{t} \in \text{Minimal}_\phi(T^\omega(Q)) \). Due to the correctness and completeness of the \( T^\omega \) operator for this kind of queries [3], the other inclusion can be established similarly.

\[ \square \]

**Appendix B: Other Sources**

**Example 20.** Consider now the global system \( \phi_2 = \{ S_1 \} \), with

\[
S_1 = \{ \langle V_1(X,Y) \leftarrow R(X,Y), \text{clopen}, \{ V_1(a,b), V_1(c,d) \} \},
\]

and IC, the functional dependency \( R(X,Y) : X \rightarrow Y \). The only instance in \( Linst(\phi_2) \) is \( D = \{ R(a,b), R(c,d) \} \). Since \( D \) satisfies IC, \( \phi_2 \) would be consistent wrt IC.

\[ \square \]

**Example 21.** (adapted from [10]) Consider the global system \( \phi_7 = \{ S_1, S_2, S_3 \} \) with

\[
S_1 = \{ \langle V_1(U,W) \leftarrow S(U,W), \text{open}, \{ V_1(b,c), V_1(e,f) \} \},
S_2 = \{ \langle V_2(U) \leftarrow R(U,X), \text{open}, \{ V_2(a) \} \},
S_3 = \{ \langle V_3(Z) \leftarrow R(a,Z), \text{closed}, \{ V_3(b), V_3(V') \} \},
\]

and IC, the inclusion dependency \( S(X,U) \rightarrow R(Z,X) \). The elements in \( \text{mininst}(\phi_7) \) are of the form \( D_y = \{ S(b,c), S(e,f), R(a,d) \} \), where \( d \) is any element in the underlying domain. All of them violate IC, so the global system is inconsistent.

\( D_6 \) gives rise to the repairs \( D_6^u = \{ S(b,c), S(e,f), R(a,b), R(u,e) \} \), where \( u \) is any element in the underlying domain, and to \( D_6' = \{ S(b,c), R(a,b) \} \).

\[ ^8 \text{We are applying the Unique Names Assumption, that tells us that } b \neq e, \text{ so cannot make } d \text{ to be equal to both } b \text{ and } e.\]
\(D_e\) gives rise to the repairs \(D'_e = \{S(b, c), S(e, f), R(a, c), R(v, b)\}\), and \(D'_e = \{S(e, f), R(a, c)\}\). Otherwise, \(D_d\), for \(d \neq b, e\), gives rise to the repairs \(D'_d = \{S(b, c), S(e, f), R(a, d), R(u, e), R(v, b)\}\) and \(D'_d = \{R(a, d)\}\). \(\square\)