# Consistent Query Answers in Virtual Data Integration Systems

Leopoldo Bertossi[1] and Loreto Bravo[2]

Carleton University,
School of Computer Science,
Ottawa, Canada
{bertossi, lbravo}@scs.carleton.ca

**Abstract.** When data sources are virtually integrated there is no common and centralized mechanism for maintaining global consistency. In consequence, it is likely that inconsistencies with respect to certain global integrity constraints (ICs) will occur. In this chapter we consider the problem of defining and computing those answers that are consistent wrt the global ICs when global queries are posed to virtual data integration systems whose sources are specified following the local-as-view approach. The solution is based on a specification using logic programs with stable model semantics of the minimal legal instances of the integration system. Apart from being useful for computing consistent answers, the specification can be used to compute the certain answers to monotone queries, and minimal answers to non monotone queries.

## 1 Introduction

There is an increasing number of available information sources, many of them online, like organizational databases, library catalogues, scientific data repositories, etc., and in different formats and ranging from highly structured, like relational databases, to semi-structured, like data on the web. Many applications need to access and combine information from several databases, in consequence, a user (or application) is confronted to many different data sources.

One possibility for attacking this problem consists in bringing a possibly huge amount of data -that might be required by the application- into one single, physical, material site; and then making the application interact with this only data repository. This process is costly in term of storage, design, and refreshment, which would be necessary when the original sources are updated. That is, we have complexities that are similar to those involved in the processes associated to data warehouses, but with the difference that updating the repository could be more crucial that in data warehouses, where, most likely, decision support could be achieved without having completely up-to-date data.

An alternative solution consists in keeping the data in their sources. In this way, if the application needs answers to a query, it has to interact with the collection of available sources, first determining and selecting those that contain the relevant information. Next, queries have to be posed to those sources, on an

L. Bertossi et al. (Eds.): Inconsistency Tolerance, LNCS 3300, pp. 42–83, 2004.

individual basis; and the different results have to be combined. This can be a long, tedious, complex and error prone process if performed on an ad hoc basis. It is better to have a general, robust and uniform implementation that supports this process on a permanent and regular basis. Ideally, the application will interact with the data sources via a unique -database like - common interface.

A solution in this line consists in the *virtual integration* of the data sources via a *mediator* [75], that is, a software system that offers a common interface to a set of autonomous, independent and possibly heterogeneous data sources. Under this paradigm for data integration, the integration is virtual in the sense that the data stays in the sources, but the user -who interacts with the mediator- feels like interacting with a single database. The sources most likely do not cooperate with each other, and the mediator, except for the possibility of asking queries, has no control on the individual sources. There is no central control or maintenance mechanism either. It is also desirable that the set of participating sources is flexible and open.

It is clear that combining data from different and independent sources offers many and difficult challenges. If the integrated system is expected to keep some correspondence with the reality it is modelling, then it should keep some general, global semantic constraints satisfied. This is difficult to achieve, because most likely there will be semantic conflicts between pieces of data coming from different sources. Since there is no central, global integrity enforcement mechanism, and there is no possibility of doing any kind of *global* data cleaning, as in the datawarehouse approach to data integration, semantic problems have to be solved when the application interacts with the integration system.

More specifically, in this chapter we describe novel techniques to solve inconsistencies when queries posed to the integration system are answered. That is, only those answers to a global query that are consistent with the given global integrity constraints are returned. Apart from the problem of defining the notion of consistent answers in this scenario, there is the problem of designing query plans to consistently answering queries.

The mediator, in order to design query plans, needs to know the correspondence between the global relations offered by the mediator's interface, which determine an external query language, and the relations in the internal databases. These descriptions of the contents of the internal data sources can be expressed in different ways. In this chapter we will mostly concentrate to the *local as view* approach to data integration, according to which the sources are described as views of the global relations.

Global integrity constraints (ICs) will be expressed as first order formulas, and database instances are seen as first order structures with finite relations. We say that a database instance $D$ is *consistent* wrt to a set $IC$ of ICs if $D$ satisfies $IC$ (what is denoted by $D \models IC$, as usual). Of course, the set of global integrity constraints $IC$ will be assumed to be logically consistent, in the sense that at least one database instance satisfies it.

This chapter is structured as follows. In Section 2 we consider virtual data integration systems, describing in general terms the main elements and issues; in particular, two alternative ways to specify the data contained in the data sources, in such a way that the mediator can make use of it. In Section 3 the semantics of virtual data integration systems with open sources under the local-as-view approach is given in detail. In Section 5 we briefly review the notion of consistent answer to a query posed to a single relational database, and some methodologies for computing them. The notion of consistent answer to a query, but now for an integration system, is defined in Section 6. With the goal of computing consistent answers in integration systems, in Section 7 logic programs with stable model semantics are used to specify the class of minimal instances of open integration systems under LAV. The results presented there are interesting in themselves, independently from consistent query answering, because they can be used to compute (ordinary) answers to both monotonic and non monotonic queries in integration systems, which extend previous results in the area. Section 8 shows how to compute consistent answers to queries posed to integration systems. The specification of minimal instances presented in Section 7 is extended in Section 9 to the case where in addition to open sources also closed and both closed and open sources are available. That specification is presented here for the first time. In Section 10, some open research issues are indicated. In Section 11 we finalize with a discussion of related work.

## 2 Virtual Data Integration Systems

### 2.1 Mediators for Data Integration

The main features of a mediator based system are: (a) The interaction with the system via queries posed to the mediator; (b) Updates via the mediator are not allowed; (c) Data sources are mutually independent and may participate in different mediated systems at the same time; (d) Sources are allowed to get in and out; (e) Data is kept in the local, individual sources, and extracted at the mediator's request.

Since the mediator offers a database like interface to the user or application, it has a *global or mediated schema*, consisting of a set of names for relations (virtual tables) and their attributes. This schema is application dependent and determines a (family of) query language(s), like in a usual relational databases from the user point of view. However, the "database" corresponding to the global schema is virtual.

A user poses queries to the mediator in terms of the relations in the global schema. However, in order to answer those global queries, the mediator needs to knows the correspondence between the global schema and the local schemas. This is achieved by means of a set of source descriptions, i.e. descriptions of what data can be found in the different sources. Having this information, when the mediator receives a query , it develops a *query plan* that determines: (a) the portions of data that are relevant to the query at hand, (b) their locations in the relevant data sources, (c) how to extract that data from the sources via queries, and (d) how to combine the answers received into a final answer for the user.

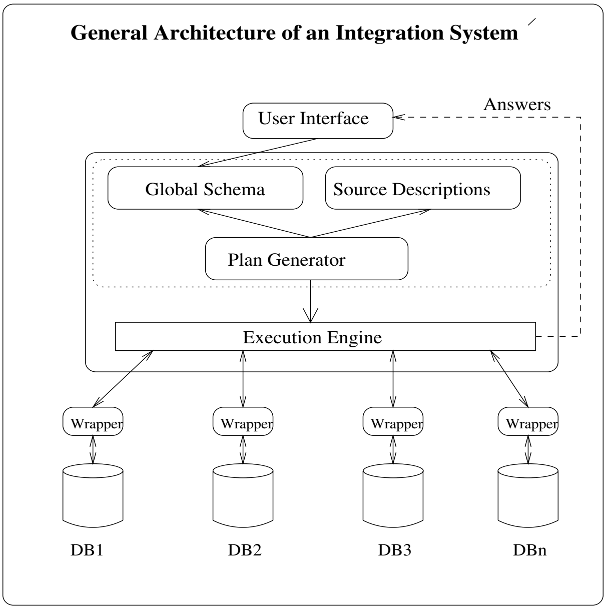

**Fig. 1.** Architecture of an Integration System

Figure 1 shows the main elements in the architecture of a mediator for virtual integration of data sources.

The mediator is responsible of solving problems of redundancy, complementarity, incompleteness, and consistency of data in the integration system. In this chapter we will consider this last problem, a very relevant one in this context. For example, what should the mediator do if it is asked about a person's ID card number and it gets two different numbers, each coming from a different source? The two sources, taken independently and separately, may be consistent, but taken together, possibly not. Such consistency problems are likely and natural in virtual data integration. Notice that consistency problems in virtual integration, unlike the "materialized" approaches to data integration, which offer data reconciliation solutions, cannot be solved a priori, at the physical data level.

Another element shown in Figure 1 is the *wrapper*. This is a module that is responsible for wrapping a data source in such a way that the latter can interact with the rest of integration system. It provides the mediator with data from a source as requested by the execution engine. In consequence, it presents a data source as a convenient database, with the right schema and data, the one that is understood and used by the mediator. Notice that this presentation schema may be different from the real one, the internal to the data source. Actually, it may be the case that the source is not at all internally structured as a database, but this should be transparent to the mediator. All this may require preliminary transformations, cleaning, etc., before the data can be exported to the integration system. There is a wrapper (or more) for each data source. In the following, we will assume that each data source has already a wrapper that presents it as a relational database.

*Example 1.* Consider a global schema for a database "containing" information about music albums: $CD(Album, Artist, Year)$, $Contract(Artist, Year, Label)$, $Songs(Album, Song)$. Now, a user wants to know the name of the label with which Norah Jones had a contract during 2002. This is asked issuing the following query to the global system $Q$: $Ans(L) \leftarrow Contract(NorahJones, 2002, L)$.

Here, predicate *Ans* will contain the answers, that are to be computed using the expression on the RHS of this rule. In this case, this is the simple selection $\mathtt{SELECT}_{X=NorahJones, Y=2002}\ \mathtt{Contract(X, Y, L)}$.

It is a problem that the material data is not in the virtual global relation *Contract*, but in the data sources $DB_1(Album, Artist, Year)$, $DB_2(Album, Artist, Year, Label)$, $DB_3(Album, Song)$. In consequence, a query plan is needed in order to extract and combine the relevant data from the material sources. However, in order to design such a plan, the mediator needs to know the correspondence between the virtual global relations and the data sources. □

A key element in the mediator architecture is the set of *source descriptions*, i.e. the descriptions of the available sources and their contents (as presented by the wrapper), which is achieved by establishing the relationships (mappings) between the global schema and the local schemata. These descriptions are given by means of a set of logical formulas; similar to the way in which views are defined in terms of base tables in a relational database, i.e. using queries written in a query language. Usually those query languages use logical formulas or their SQL versions.

With respect to how mappings are defined, there are two main approaches (and combinations of them): (a) *Global as View* (GAV), under which the relations in the global schema are described as views of the collection of local relations [73]; and (b) *Local as View* (LAV), under which each relation in a local source is described as a view of the global schema [61]. GLAV denotes a combination of GAV and LAV [37] where the rules can have more than one atom in the head. Another approach, called *Both as View* (BAV), consists on a specification of the transformation of the local schema into the given global schema, in such a way that each schema can be seen as defined as in terms of the other schema [65]. In Section 2.2 we describe and compare the GAV and LAV approaches.

The *plan generator* gets a user query in terms of global relations and uses the source descriptions to design a *query plan.* This is achieved by *rewriting* the original query as a set of subqueries that are expressed in terms of the local relations. The query plan includes prescriptions on how the answers from the local sources have to be combined. The query rewriting process executed by the plan generator strongly depends on whether the LAV or the GAV approach is followed. Still much theoretical and technical research is going on in relation to query plan generation. The plan is executed by the *execution engine.* Notice that it should be the plan generator who takes care of anticipating and solving potential inconsistencies. It should solve them in advance, when the plan is being generated. Later in this chapter, we will explore this issue in detail.

## 2.2 Description of Data Sources

The global/local schema mappings or, equivalently, the descriptions of the source contents are expressed through logical formulas that relate the global and local relations.

### Global as View

In this case, the relations in the global schema are described as *views* over the tables in the union of the local schemata. This is conceptually very natural, because views are usually virtual relations defined in terms of material relations (the tables); and here we have global relations that are virtual and local sources that are materialized.

*Example 2.* (example 1 continued) Assume the relation *CD* is defined as the view

$$CD(Album, Artist, Year) \leftarrow DB_1(Album, Artist, Year)$$
$$CD(Album, Artist, Year) \leftarrow DB_2(Album, Artist, Year, Label).$$

Relation *CD* is defined as the union of the projections of $DB_1$ and $DB_2$ on attributes *Album*, *Artist*, *Year*, i.e. in relational terms, defined by

$$CD = \quad \Pi_{Album, Artist, Year}(DB_1) \ \cup \ \Pi_{Album, Artist, Year}(DB_2).$$

The global relation *Songs* and *Label* are defined as follows:

$$Songs(Album, Song) \leftarrow DB_1(Album, Artist, Year), DB_3(Album, Song).$$
$$Contract(Artist, Year, Label) \leftarrow DB_2(Album, Artist, Year, Label).$$

The first view is defined as, first, the join of $DB_1$ and $DB_3$ via attribute *Album*, and then, a projection on *Album*, *Song*. The second view is defined as the projection of $DB_2$ over *Artist*, *Year*, *Label*.

These views have been defined by means of *rules*. Each rule specifies that in order to compute the tuples in the relation in the LHS (the *head* of the rule), one has to go to the RHS (the *body* of the rule) and compute whatever is specified there. The attributes appearing in the head indicate that they are the attributes of interest, thus the others (in the body) can be projected out at the end. If there are more that one rule to compute a same relation, we use all of them and we take the union of the results, as for the relation *CD*.

Instead of using a rule as above, we could have used relational algebra (or relational calculus, or SQL2), in the case of the relation *Songs*,

$$Songs = \Pi_{Album, Song}(DB_1 \bowtie_{Album} DB_3).$$

The language of rules is more expressive than relational algebra, e.g. recursive views can be defined using rules, but not with relational algebra [72]. □

Once the global relations have been defined as views, we may start posing global queries, i.e. queries expressed in terms of the global relations. The problem is to answer them considering that the global relations do not contain material data. Under the GAV approach this is simple, all we need to do is *rule unfolding*.

*Example 3.* (example 2 continued) Consider the following global query about the music albums released in the year 2003, with their artists and songs

$$\mathit{Ans}(\mathit{Album}, \mathit{Artist}, \mathit{Song}) \leftarrow \underline{\mathit{CD}(\mathit{Album}, \mathit{Artist}, 2003)}, \underline{\underline{\mathit{Songs}(\mathit{Album}, \mathit{Song})}}.$$

Since it is expressed in terms of the global schema, the data has to be obtained from the sources, that is, the query has to be *rewritten* in terms of the source relations. We do this by unfolding each global relation, replacing it by its definition in terms of the local relations. We have underlined differently the goals in the body in order to keep track of the rewriting for each of them.

$$\begin{aligned}
\mathit{Ans}'(\mathit{Album}, \mathit{Artist}, \mathit{Song}) \leftarrow\ & \underline{\mathit{DB}_1(\mathit{Album}, \mathit{Artist}, \mathit{2003})},\\
& \underline{\underline{\mathit{DB}_1(\mathit{Album}, \mathit{Artist}, \mathit{Year}), \mathit{DB}_3(\mathit{Album}, \mathit{Song}).}}\\
\mathit{Ans}'(\mathit{Album}, \mathit{Artist}, \mathit{Song}) \leftarrow\ & \underline{\mathit{DB}_2(\mathit{Album}, \mathit{Artist}, \mathit{2003}, \mathit{Label})},\\
& \underline{\underline{\mathit{DB}_1(\mathit{Album}, \mathit{Artist}, \mathit{Year}), \mathit{DB}_3(\mathit{Album}, \mathit{Song}).}}
\end{aligned}$$

These new queries do get answers directly from the sources; and the final answer is the union of two answer sets, one for each of the rules. □

If, in addition to the view definitions, there are ICs that have to be and are satisfied by the system, unfolding is not enough for query answering [17, 19] (see Section 11 for more details).

### Local as View

Under the LAV approach, each table in each local data source is described as a view (i.e. as a query expression) in terms of the global relations. This may seem somehow unnatural or unusual from the conceptual point of view, and from perspective of databases practice, because here the views contain the data, but not the "base tables". However, as we will see, this approach has some advantages.

More precisely, in the general situation we have a collection of material data sources (think of a collection of material relational tables) $S_1, \ldots, S_n$, and a global schema $G$ for the system that integrates data from $S_1, \ldots, S_n$. Tables in $S_1, \ldots, S_n$ are seen as *views* over $G$, and in consequence, they can be defined by query expressions over the global schema.

*Example 4.* Consider the sources $S_1, S_2$ that are defined by the view expressions

$$\begin{aligned}
S_1\!: \quad V_1(\mathit{Album}, \mathit{Artist}, \mathit{Year}) \leftarrow\ & \mathit{CD}(\mathit{Album}, \mathit{Artist}, \mathit{Year}),\\
& \mathit{Contract}(\mathit{Artist}, \mathit{Year}, \mathit{emi}), \mathit{Year} \geq 1990\\
S_2\!: \qquad V_2(\mathit{Album}, \mathit{Song}) \leftarrow\ & \mathit{Songs}(\mathit{Album}, \mathit{Song}).
\end{aligned}$$

Source $S_1$ contains a table whose entries are albums produced after 1990 by the label EMI with their artists and years. Source $S_2$ contains one table with songs and their albums.

Those relations that are not defined as views belong to the global schema $G$, in this case, we have the relations: $\mathit{CD}(\mathit{Album}, \mathit{Artist}, \mathit{Year})$, $\mathit{Songs}(\mathit{Album}, \mathit{Song})$, $\mathit{Contract}(\mathit{Artist}, \mathit{Year}, \mathit{Label})$. □

Notice that from the perspective of $S_1$, there could be other sources containing information about albums produced by EMI after 1990, and that complementary information could be exported to the global system. In this sense, the information in $S_1$ could be considered as "incomplete" wrt what $G$ contains (or might contain). In other words, $S_1$ contains only a part of the data of the same kind in the global system. We will elaborate on this later on. Finally, also notice that in the example, and this is a general situation under LAV, the definition of each source does not depend on other sources.

Now we want to answer global queries under LAV.

*Example 5.* (example 4 continued) The following query posed to $G$ asks for the songs with its album and the year they were released:

$$Ans(Album, Song, Year) \leftarrow CD(Album, Artist, Year), Songs(Album, Song).$$

This query is expressed as usual, in terms of global relations only, however, it is not possible to obtain the answers by a simple and direct computation of the RHS of the query. Now, there is no direct rule unfolding mechanism for the relations in the body, because we do not have explicit definitions for them. And the data resides in the sources, which are now defined as views.

We can see that plan generation to extract information from the sources becomes more complex under LAV than under GAV. Since a query plan is a rewriting of the query as a set of queries to the sources and a prescription on how to combine their answers (what is needed in this example), the following could be a query plan to answer the original query:

$$Ans'(Album, Song, Year) \leftarrow V_1(Album, Artist, Year), V_2(Album, Song).$$

The query has been rewritten in terms of the views; and in order to obtain the final answer, we first extract values for *Album*, *Year* from $V_1$; then we extract the tuples from $V_2$; finally, at the mediator level, we compute the join via *Album*.

Notice that due to the limited contents of the sources, we only obtain albums produced by EMI after 1990. □

In LAV we pose a query in terms of certain relations (the global ones), but we have to answer using the contents of certain views only (the local relations). In consequence, query plan generation becomes an instance of a more general and traditional problem in databases, the one of *query rewriting using views*.

To see this connection more clearly, assume we have a collection of views $V_1, \ldots, V_n$, whose contents have already been computed, and cached or materialized. When a new query $Q$ arrives, instead of computing its answers directly, we try to use the answers (contents) to (of) $V_1, \ldots, V_n$. A problem to consider consists in determining how much from the real answer do we get by using the pre-computed views only; and also determining what is the maximum we can get in terms of the kind of views we have available. The research carried out in query answering using views [60, 2, 49, 51, 50, 35] and query containment [2, 56, 67, 23] has become quite relevant to the area of data integration.

### 2.3 Comparison of Paradigms

We have seen that under GAV, rule unfolding makes plan generation simple and direct. On the other hand, GAV is not flexible to accept new sources or eliminate sources into/from the system. Actually, adding or deleting sources might imply modifying the definitions of the global relations.

LAV offers more flexibility to add new sources or delete old ones into/from the integration system, because a new source is just a new view definition. Other sources do not need to be considered at this point, because there are no other sources interfering in the process. Only the plan generator has to be aware of these changes. On the other side, plan generation is provably more difficult [2, 58, 18, 73].

### 2.4 Data Integration and Consistency

Notice that, so far, we have not considered any integrity constraints at the global schema level. Since the data sources are autonomous and possibly updated independently from the integration system in which they participate and from other data sources, there is not much we can do wrt to data maintenance at the global level. However, in virtual data integration, one usually assumes that certain integrity constraints hold at the global level, and they are used in the plan generation process [48, 30, 45]. Even more, in some cases the generation of a query plan is possible because certain integrity constraints (are supposed to) hold [30].

In general, we cannot be sure that such global integrity constraints hold, because they are not maintained at the global level. A more natural scenario is the one where integrity constraints are considered when queries are posed to the system. In this case, we have the problem -to be addressed in Section 6- of retrieving information from the global system that is consistent wrt certain global constraints, but the problem has to be solved at query time, as opposed to the usual approach in single databases, where all the data in the database is kept and maintained consistent, independently from potential queries.[1] This is an interesting point of view wrt integrity constraints: they constitute constraints on the answers to queries rather than on the database states.

Notice that the flexibility to add/remove sources, in particular under LAV, is likely to introduce extra sources of inconsistencies we have to take care of.

The global ICs we will consider are first order sentences written in the language of the global schema. In particular, they will be *universal integrity constraints*, i.e. sentences of the form $\forall \bar{x} \varphi(\bar{x})$, where $\varphi(\bar{x})$ is a quantifier-free formula; and also *referential integrity constraints* of the form $\forall \bar{x}(P(\bar{x}) \rightarrow \exists y(Q(\bar{x}', y)))$, where $\bar{x}' \subseteq \bar{x}$.

[1] Work reported in [11] departs from this practice and considers a more flexible approach to query answering in databases where databases may be inconsistent, but only answers to queries are expected to be consistent.

## 3 Semantics of Virtual Data Integration Systems

In the rest of this paper, unless otherwise stated, we will concentrate on the LAV approach (see Section 11 for references on the GAV approach). The semantics of virtual data integration systems is given in terms of the intended global instances. This does not mean that such instances are to be computed, but they will allow us to give a model theoretic semantics to global integrity constraint satisfaction, to query answers, etc.

A data integration system $\mathcal{G}$ under the LAV approach is specified by a set of view definitions, plus a set of material tables $v_i$ corresponding to the views $V_i$ defined:

$$\begin{aligned} \mathcal{G}\colon \qquad & V_1(\bar{X}_1) \leftarrow \varphi_1(\bar{X}_1'); \quad v_1 \\ & \cdots\cdots\cdots \\ & V_n(\bar{X}_n) \leftarrow \varphi_n(\bar{X}_n'); \quad v_n \end{aligned} \tag{1}$$

Here, $\bar{X}_j \subseteq \bar{X}_j'$, and each $v_i$ is an extension (a material relation) for view $V_i$, which in its turn is defined as a conjunctive view.

*Until further notice we will assume that the system has all its sources open (also called sound).* This means that the information stored in the sources might be incomplete. The description in (1) plus the openness assumption will determine a *a set of legal global instances*. Now we describe how.[2]

Let $D$ be a global instance, i.e. its domain contains at least the constants appearing in the source extensions and the view definitions; and has relations (and contents) for the global schema. We denote with $\varphi_i(D)$ the set of tuples obtained by applying to $D$ the definition of view $V_i$. This gives an extension for $V_i$ in (wrt) global instance $D$, which can be compared with $v_i$. We call a global instance $D$ *legal* if the computed extension on $D$ of each view $V_i$ contains the originally given extension $v_i$:

$$\mathit{Legal}(\mathcal{G}) := \{ \text{ global } D \mid v_i \subseteqq \varphi_i(D);\ i = 1, \ldots, n\},$$

which captures the incompleteness of the sources, because if a view is applied to a legal instance, the result will be a superset of the elements in the source. Only legal instances will determine the semantics of $\mathcal{G}$.

*Example 6.* Consider the system $\mathcal{G}_1$ with global relation $R(X, Y)$ and the following open sources

$$\begin{aligned} V_1(X,Y) \leftarrow R(X,Y); \qquad & v_1 = \{(a,b),(c,d)\} \\ V_2(X,Y) \leftarrow R(X,Y); \qquad & v_2 = \{(a,c),(d,e)\}. \end{aligned}$$

The global instance $D$ for which the relation $R$ has the extension $R^D = \{(a,b),(c,d),(a,c),(d,e)\}$[3] is legal, because: (a) $v_1 \subseteq \varphi_1(D) = \{(a,b),(c,d),(a,c),$

[2] A similar semantics can be given in the case of the GAV approach [58].

[3] In the rest of this chapter we will use a simpler description for an instance of this kind. We simple write $D = \{(a,b),(c,d),(a,c),\ (d,e)\}$, because there is only one global relation. If there were another relation, we write $D = \{R(a,b), R(c,d), ...\}$.

$(d,e)\}$; and (b) $v_2 \subseteq \varphi_2(D) = \{(a,b),(c,d),(a,c),(d,e)\}$. All supersets of $D$ are also legal global instances; e.g. $\{(a,b),(c,d),(a,c),(d,e),\ (c,e)\} \in \mathit{Legal}(\mathcal{G})$, but no subset of $D$ is legal, e.g. $\{(a,b),(c,d),(a,c)\} \notin \mathit{Legal}(\mathcal{G})$. □

*Example 7.* Let $\mathcal{D} = \{a,b,c,\dots\}$ be the underlying domain. Consider the integration system $\mathcal{G}_2$ defined by

$$V_1(X,Z) \leftarrow P(X,Y), R(Y,Z); \qquad v_1 = \{(a,b)\}$$
$$V_2(X,Y) \leftarrow P(X,Y); \qquad\qquad v_2 = \{(a,c)\}.$$

Each global instance $D$ of the form $\{P(a,c), P(a,z), R(z,b)\}$, with $z \in \mathcal{D}$ is a legal instance, because $v_1 \subseteq \varphi_1(D) = \{(a,b)\}$ and $v_2 \subseteq \varphi_2(D) = \{(a,c),(a,z)\}$. Any superset of $D$ is also legal, but none of its subsets is. □

Now we can define the intended answers to a global query $Q$. They are the *certain answers*, those that can be obtained from every legal global instance [2]:

$$\mathit{Certain}_{\mathcal{G}}(Q) := \{\bar{t} \mid \bar{t} \text{ is an answer to } Q \text{ in } D \text{ for all } D \in \mathit{Legal}(\mathcal{G})\}.$$

*Example 8.* (example 6 continued) Consider the following global query $Q$ posed to system $\mathcal{G}_1$: $\mathit{Ans}(X,Y) \leftarrow R(X,Y)$. In this case, $\mathit{Certain}_{\mathcal{G}_1}(Q) = \{(a,b),(c,d),(a,c),(d,e)\}$. □

The algorithms for constructing query plans should be sound and complete wrt this semantics, more precisely they should be able to produce plans whose execution will allow us to get all and only the certain answers from a data integration system; of course, without explicitly computing all the legal instances and querying them.

## 4 Query Plans

There are several algorithms for generating query plans. See [62, 51] for survey of different techniques. In [45] a deductive methodology is presented. Here we will briefly describe the *inverse rules algorithm* (IRA) [29, 30]. This algorithm is conceptually simple, shows the main issues, and will be used later in this chapter in our solution to the problem of consistent query answering in integration systems.

Our framework is as follows. We are given a global query $Q$ posed in terms of the global schema, but we need to go to the sources for the data required to evaluate $Q$. The problem is how to do this, or more precisely, how to rewrite $Q$ in terms of the views available, i.e. in terms of the relations in the sources.

We will assume that we have a set of rules describing the source relations as conjunctive (Select-Project-Join) views of the global schema [1]. We also assume that the sources are open.

The input to our problem is a global query expressed, e.g. in Datalog (may be recursive, but without negation). The expected output is a new Datalog program expressed in terms of the source relations.

*Example 9.* Consider the local relations $V_1, V_2$ in sources $S_1, S_2$, resp., and the global relations $R_1, R_2, R_3$. The set of source descriptions contains

$$S_1: \quad V_1(X,Z) \leftarrow R_1(X,Y), R_2(Y,Z), \tag{2}$$

$$S_2: \quad V_2(X,Y) \leftarrow R_3(X,Y). \tag{3}$$

The idea behind IRA consists in obtaining, from these descriptions, "inverse rules" describing the global relations. Let us start from (3). Since $V_2$ is open, it is contained in the "extension" of the global relation $R_3$. That is, the only way to get tuples for $V_2$ is by going to pick up tuples from the RHS of (3). In other terms, we can say that $V_2$ "$\subseteq$" $R_3$, or, equivalently, $V_2$ "$\Rightarrow$" $R_3$. More precisely, we invert the rule in the description of $V_2$, obtaining

$$R_3(X,Y) \leftarrow V_2(X,Y),$$

now, a rule describing $R_3$, which we wanted. If there are (not in this case though) other rules of this kind describing $R_3$ (from other source description rules containing $R_3$ on the RHS), we just take the union.

Now, wrt inverting rule (2), a first attempt could be

$$R_1(X,Y), R_2(Y,Z) \leftarrow V_1(X,Z),$$

but this is a strange rule, with a strange head. There are several problems. If the head is seen as a conjunction, then we may split it into two rules, namely $R_1(X,Y) \leftarrow V_1(X,Z)$ and $R_2(Y,Z) \leftarrow V_1(X,Z)$, but now the two occurrences of variable $Y$ are independent, and before it was a shared variable that allowed us to combine tables $R_1, R_2$ by means of a join. This connection is lost now. Another problem has to do with the unrestricted occurrence of $Y$ in the heads; there are no conditions on $Y$ in the bodies (this kind of rules are considered *unsafe* in databases [72]). It should not be the case that any value for $Y$ is admissible.

A better approach is as follows: $V_1(X,Z) \leftarrow R_1(X,Y), R_2(Y,Z)$ is equivalent to $V_1(X,Z) \leftarrow \exists Y\ (R_1(X,Y) \wedge R_2(Y,Z))$ (a join followed by a projection). Inverting, we obtain $\exists Y(R_1(X,Y) \wedge R_2(Y,Z)) \leftarrow V_1(X,Z)$. This rule has an implicit universal quantification on $X, Z$, then each value for $Y$ possibly depends on the values for $X, Z$, i.e. $Y$ is a function of $X, Z$. To capture this dependence, we replace $Y$ by a function symbol $f(X,Z)$ (a so-called "Skolem function"), obtaining

$$R_1(X, f(X,Z)) \wedge R_2(f(X,Z), Z) \ \leftarrow\ V_1(X,Z).$$

As before, we split the conjunction, obtaining the rules $R_1(X, f(X,Z)) \leftarrow V_1(X,Z)$ and $R_2(f(X,Z),Z) \leftarrow V_1(X,Z)$. In this way, we obtain the following set $\mathcal{V}^{-1}$ of inverse rules

$$\begin{aligned} R_1(X, f(X,Z)) &\leftarrow V_1(X,Z) \\ R_2(f(X,Z),Z) &\leftarrow V_1(X,Z) \\ R_3(X,Y) &\leftarrow V_2(X,Y), \end{aligned}$$

which can be used to compute answers to global queries.

Notice that we may need other symbolic functions, for dependencies between variables in the same or other rules. More precisely, we introduce one function symbol for each variable in the body of a view definition that is not in the head; and that function appears evaluated in the variables in the head.

Now, assume the following global query $Q$ is posed to the integration system

$$\begin{aligned} Ans(X,Z) &\leftarrow R_1(X,Y), R_2(Y,Z), R_4(X) \\ R_4(X) &\leftarrow R_3(X,Y) \\ R_4(X) &\leftarrow R_7(X) \\ R_7(X) &\leftarrow R_1(X,Y), R_6(X,Y). \end{aligned}$$

We can see that the goal $R_6$ cannot be computed, because there is no definition for it in $\mathcal{V}^{-1}$. Then, $R_7$ cannot be evaluated either; and the rule defining it can be deleted. For the same reason, the third rule in the query cannot be evaluated; and can be deleted. In this way we obtain a pruned query $Q^-$:

$$\begin{aligned} Ans(X,Z) &\leftarrow R_1(X,Y), R_2(Y,Z), R_4(X) \\ R_4(X) &\leftarrow R_3(X,Y). \end{aligned}$$

In consequence, the final query produced by the plan generator, using the IRA, is $Q^- \cup \mathcal{V}^{-1}$. This is a sort of Datalog program, but with functions.

This is all and the best we have to answer the original query. With the new query program we can compute *some* answers to $Q$, but actually, "the most" we can. The plan can be evaluated, e.g. bottom-up, from concrete source contents [72]. The final answer may contain some tuples with the function symbol $f$ in them; but they are eventually deleted.

We will illustrate this process with a different query. Assume that the source contents are $v_1 = \{(a,b),(a,a),(c,a),(b,a)\}$ and $v_2 = \{(a,c),(a,a),(c,d),(b,b)\}$; and the query is now $Q'$:

$$\begin{aligned} Ans(X) &\leftarrow R_1(X,Y), R_2(Y,Z), R_4(X) \\ Ans(X) &\leftarrow R_2(X,Y) \\ R_4(X) &\leftarrow R_3(X,Y) \\ R_4(X) &\leftarrow R_1(X,a). \end{aligned}$$

We have the same set $\mathcal{V}^{-1}$ of inverse rules as above, they are the same for all the queries. So, first we prune the query rules that cannot be evaluated from the inverse rules. We delete the last rule in the query, because it does not contribute to $R_4$ ($a$ cannot be an $f$-value). We obtain the final query consisting of the rules in $\mathcal{V}^{-1}$ plus the first three rules in $Q'$. It can be evaluated bottom-up. The mediator will use the inverse rules applied to the sources, which requires sending one query to each source, and will obtain

$$\begin{aligned} R_1 &= \{(a,f(a,b)),(a,f(a,a)),(c,f(c,a)),(b,f(b,a))\} \\ R_2 &= \{(f(a,b),b),(f(a,a),a),(f(c,a),a),(f(b,a),a)\} \\ R_3 &= \{(a,c),(a,a),(c,d),(b,b)\}. \end{aligned}$$

Using the third rule of $Q'$, we obtain $R_4 = \{a, c, b\}$. Now we can evaluate the first rule in $Q'$, whose body becomes $\Pi_X(R_1 \bowtie R_2) \cap R_4 = \{a, c, b\} \cap \{a, c, b\} = \{a, c, b\}$. Then, $a, c, b \in Ans$. From the second rule in $Q'$ we obtain $f(a, b), f(a, a), f(c, a), f(b, a) \in Ans$, but these tuples are not considered, because all the tuples containing function symbols are eliminated from the final answer set. So, finally $Ans = \{a, c, b\}$. □

Given a Datalog query, the query plan obtained for it is a new Datalog program, but may contain function symbols (strictly speaking, for this reason, it is not a Datalog program). If the original query does not contain recursion, neither does the final query. The query plan: (a) does not contain negation, (b) can be evaluated in a bottom-up manner and always has a unique fix point, (c) can be constructed in polynomial time in the size of the original query and the source descriptions.

The plan obtained is the best we can get under the circumstances, i.e. given the query, the sources and their descriptions. More precisely, for a Datalog query $Q$ and a set of sources defined as conjunctive views, the query plan generated with the IRA is *maximally contained* [2] in the original query $Q$ [30]. In other words, there is no other query plan that retrieves a set of answers to $Q$ that is a proper superset of *answers* to $Q$ produced by IRA.

It is possible to prove [2] that for conjunctive views and Datalog queries (and open sources), a maximally contained query plan computes all the certain answers. In consequence, the inverse rules algorithm returns all the certain answers to Datalog queries [30].

We have seen in this section and also in Section 2.2 for the GAV approach, that the query plan prescribes how to rewrite the original, global, conjunctive query as a new query expressed in terms of the source relations. The new query is also a first order or Datalog query. However, for more complex queries, the "rewriting" may need to be expressed in more expressive languages, e.g. disjunctive logic programs with stable model semantics, as in Section 7, in order to capture a higher data complexity of query answering (see [22] for a discussion about what should qualify as a query rewriting).

Now, if in addition to the source descriptions, we have a set $IC$ of global integrity constraints; it is quite likely that they are not going to be satisfied by (all) the legal instances. In consequence, instead of retrieving the certain answers to a global query, we might be interested in retrieving those answers that are *consistent wrt* $IC$ . This notion is still to be formalized (see Section 6), but having done that, we would expect that the query plans generated by the mediator should incorporate new elements, responsible for enforcing the satisfaction of the ICs at the query answer level.

In order to formally define what is a consistent answer to a query to the integration system, we will appeal to some notions and techniques introduced, in the context of single, stand alone relational databases, to characterize and compute answers to queries that are consistent wrt to integrity constraints that the database may fail to satisfy. We review some of those relevant notions and techniques in Section 5.

## 5 Consistent Query Answering for Single Databases

Assume we have a single relational database instance $D$ and a set of integrity constraints (ICs) that $D$ may fail to satisfy. This inconsistent database can still give us "correct" answers to queries, because not all the data in it participates in the violation of the ICs. It becomes necessary to define in precise terms what is the "correct" or "consistent" information in the database; and in particular, which are the "correct answers" to a query. Having done this, it is necessary to develop mechanisms for retrieving such consistent answers; but without changing the database, restoring its consistency. See [11] for an extended discussion about why this is a natural and important problem. Here we briefly review some notions and techniques that have been given to attack these problems.

Given a relational database instance $D$, a query $Q$, and a set $IC$ of ICs, we say that a tuple $\bar{t}$ is a *consistent answer* to $Q$ in $D$ wrt $IC$ whenever $\bar{t}$ is an answer to $Q$ in every *repair* of $D$, where a repair of instance $D$ is a database instance $D'$, over the same schema and domain, that satisfies $IC$, and differs from $D$ by a minimal set of changes (insertions/deletions of whole tuples) wrt to set inclusion [3].

Intuitively speaking, consistent answers are invariant under minimal ways of restoring consistency. Repairs are just an auxiliary concept, used to characterize the consistent answers, but we *we are not interested in repairs per se*. Actually we may try to avoid to (explicitly and completely) compute them whenever possible, because this is an expensive process. In consequence, the ideal situation is the one in which we are able to compute the consistent answers to $Q$ by posing a -hopefully- simple new query $Q'$ to the inconsistent instance $D$, in such a way that the standard answers to $Q'$ are precisely the consistent answers to $Q$. In some cases it is possible to generate a new first order query $Q'$ with that property, however in other situations, the query $Q'$ has to be written in some extension of Datalog, possibly as disjunctive normal programs [41, 27].

*Example 10.* Consider the database instance $D = \{P(a), P(b), R(a), R(c)\}$ and the integrity constraint $IC: \forall x(\neg P(x) \vee \neg R(x))$, stating that tables $P$ and $R$ do not intersect. The instance is inconsistent wrt to $IC$. The two repairs of $D$ are $D_1 = \{P(a), P(b), R(c)\}$, $D_2 = \{P(b), R(a), R(c)\}$. The query $Q(x): Ans(x) \leftarrow P(x)$ has $b$ as only consistent answer, because $P$ becomes true only of $b$ in both repairs. The query $Q'$ consisting of the rules $Ans(x) \leftarrow P(x)$ and $Ans(x) \leftarrow R(x)$, has $a, b, c$ as consistent answers, what shows that data is not cleaned from inconsistencies: the problematic tuple $a$ is still recovered. □

In [11], an alternative repair based semantic was used in the presence of referential integrity constraints. There, if a tuple is inconsistent (participates in a violation), the possible ways to repair are deleting the inconsistent tuple or adding a tuple with null values in the existentially quantified attributes of the constraint.

In order to compute the consistent answers to queries, two main approaches have been introduced. One of them is first order (FO) query rewriting (if the original query is first order) [3, 25, 13]; and the other consists in specification of database repairs using disjunctive logic programs with stable model semantics [4, 47, 7]. The later approach is more general, but more expensive than FO query rewriting. Despite their higher data complexity, disjunctive programs have to be applied, also to some first order queries, because in some cases, for complexity reasons, there is no FO rewriting [26, 20, 38].

### 5.1 Query Rewriting

*Example 11.* (example 10 continued) Consider again query $Q$. Notice that a tuple $\bar{t}$ is an answer to the query and at the same time consistent wrt to $IC$ if it is not in $R$. In consequence, instead of posing the original query to the original database, we pose the new query $(P(x) \wedge \neg R(x))$, which gives us the expected answer, $b$, in $D$.

The extra condition $\neg R(x)$ imposed on the original query is the so-called *residue* of the literal $P(x)$ wrt the $IC$. Notice that this residue can be obtained by resolution between the query literal and the $IC$. We write $T^1(Q) = (P(x) \wedge \neg R(x))$. In principle, the new literal appended may have residues of its own wrt $IC$. We do not have any in this case, but if we had, we would append its residues, obtaining $T^2(Q)$, etc. Here, the iteration stopped and we write $T^\omega(Q) = (P(x) \wedge \neg R(x))$. See [3, 25] for details. □

The FO query rewriting based methodology introduced in [3] via the $T$ operator has some limitations [3, 25]. It cannot be applied to existential or disjunctive queries, like query $Q'$ in Example 10, and only universal integrity constraints can be involved.

### 5.2 Logic Programming

The second approach consists in representing in a compact form the collection of all database repairs. This is like axiomatizing a class of models, namely as the intended models of a disjunctive logic program under the stable model semantics [41]. That is, the repairs correspond to certain distinguished models of the program, namely, to its stable models.

Once the specification has been given, in order to obtain consistent answers to a, say, FO query $Q$, the latter is transformed into a query written as logic program, which is a standard process [64, 1]; and then, this query program is "run" together with the program that specifies the repairs. This evaluation can be implemented on top of DLV, for example; a logic programming system that computes according to the stable models semantics [31, 59]. We illustrate the methodology presented in [6] by means of an example. In order to capture the repair process, the program uses annotation constants, whose intended semantics is shown in Table 1.

*Example 12.* (example 10 continued) The repair program $\Pi(r, IC)$ consists of:

1. Facts: $P(a, \mathbf{t_d}), P(b, \mathbf{t_d}), R(a, \mathbf{t_d}), R(c, \mathbf{t_d})$.

Whatever was true (false) or becomes true (false), gets annotated with $\mathbf{t}^\star$ $(\mathbf{f}^\star)$:

**Table 1.** Semantic of Annotation Constants

| Annotation | Atom | The tuple $P(\bar{a})$ is... |
|---|---|---|
| $\mathbf{t_d}$ | $P(\bar{a}, \mathbf{t_d})$ | a fact of the database |
| $\mathbf{f_d}$ | $P(\bar{a}, \mathbf{f_d})$ | a fact not in the database |
| $\mathbf{t_a}$ | $P(\bar{a}, \mathbf{t_a})$ | advised to be made true |
| $\mathbf{f_a}$ | $P(\bar{a}, \mathbf{f_a})$ | advised to be made false |
| $\mathbf{t}^\star$ | $P(\bar{a}, \mathbf{t}^\star)$ | true or becomes true |
| $\mathbf{f}^\star$ | $P(\bar{a}, \mathbf{f}^\star)$ | false or becomes false |
| $\mathbf{t}^{\star\star}$ | $P(\bar{a}, \mathbf{t}^{\star\star})$ | true in the repair |
| $\mathbf{f}^{\star\star}$ | $P(\bar{a}, \mathbf{f}^{\star\star})$ | false in the repair |

2. $$\begin{array}{l} P(X, \mathbf{t}^\star) \leftarrow P(X, \mathbf{t_d}) \\ P(X, \mathbf{t}^\star) \leftarrow P(X, \mathbf{t_a}) \\ P(X, \mathbf{f}^\star) \leftarrow \ \mathit{not}\ P(X, \mathbf{t_d}) \\ P(X, \mathbf{f}^\star) \leftarrow P(X, \mathbf{f_a}) \qquad \text{... the same for } R \text{ ...} \end{array}$$

3. $$P(X, \mathbf{f_a}) \ \vee \ R(X, \mathbf{f_a}) \ \leftarrow \ P(X, \mathbf{t}^\star), R(X, \mathbf{t}^\star)$$

   One rule per IC; that says how to repair the IC, in this case, if $x$ belongs both to $P$ and $R$, either delete the tuple from $P$ or from $R$. Passing to annotations $\mathbf{t}^\star$ and $\mathbf{f}^\star$ allows to keep repairing the DB wrt to all the ICs until the whole process stabilizes.

Repairs must be *coherent*: we use denial constraints at the program level, to prune the models that do not satisfy them

4. $$\begin{array}{l} \leftarrow P(X, \mathbf{t_a}), P(X, \mathbf{f_a}) \\ \leftarrow R(X, \mathbf{t_a}), R(X, \mathbf{f_a}) \end{array}$$

Finally, annotations constants $\mathbf{t}^{\star\star}$ and $\mathbf{f}^{\star\star}$ are used to read off the literals that are inside (outside) a repair, i.e. they are used to interpret the stable models of the program as database repairs.

5. $$\begin{array}{l} P(X, \mathbf{t}^{\star\star}) \leftarrow P(X, \mathbf{t_a}) \\ P(X, \mathbf{t}^{\star\star}) \leftarrow P(X, \mathbf{t_d}),\ \mathit{not}\ P(X, \mathbf{f_a}) \\ P(X, \mathbf{f}^{\star\star}) \leftarrow P(X, \mathbf{f_a}) \\ P(X, \mathbf{f}^{\star\star}) \leftarrow \ \mathit{not}\ P(X, \mathbf{t_d}),\ \mathit{not}\ P(X, \mathbf{t_a}). \quad \text{... etc.} \end{array}$$

The program has two stable models (and two repairs):

$$\{P(a, \mathbf{t_d}), P(a, \mathbf{t}^\star), P(a, \mathbf{t}^{\star\star}), P(b, \mathbf{t_d}), P(b, \mathbf{t}^\star), P(b, \mathbf{t}^{\star\star}), R(a, \mathbf{t_d}), R(a, \mathbf{f_a}), \\ R(a, \mathbf{f}^\star), R(a, \mathbf{f}^{\star\star}),\ R(c, \mathbf{t_d}), R(c, \mathbf{t}^\star), R(c, \mathbf{t}^{\star\star})\} \equiv \{P(a), P(b), Q(c)\}.$$

$$\{P(a, \mathbf{t_d}), P(a, \mathbf{f_a}), P(a, \mathbf{f}^\star), P(a, \mathbf{f}^{\star\star}), P(b, \mathbf{t_d}), P(b, \mathbf{t}^\star), P(b, \mathbf{t}^{\star\star}), \\ R(a, \mathbf{t_d}), R(a, \mathbf{t}^\star), R(a, \mathbf{t}^{\star\star}),\ R(c, \mathbf{t_d}), R(c, \mathbf{t}^\star), R(c, \mathbf{t}^{\star\star})\} \equiv \{P(b), Q(a), Q(c)\}.$$

If we want the consistent answers to the query $(P(\bar{x}) \wedge R(\bar{x}))$, for example, we run the repair program $\Pi(r, IC)$ together with query program $Ans(X) \leftarrow P(X, \mathbf{t}^{\star\star}), Q(X, \mathbf{t}^{\star\star})$, obtaining the answer $Ans = \emptyset$, as expected. With the query $Ans(X) \leftarrow P(X, \mathbf{t}^{\star\star}), Q(X, \mathbf{f}^{\star\star})$, we obtain the answer $Ans = \{b\}$. Finally, we can pose the disjunctive query $Q'$ we had in Example 10 by means of the two rules $Ans(X) \leftarrow P(X, \mathbf{t}^{\star\star})$ and $Ans(X) \leftarrow R(X, \mathbf{t}^{\star\star})$, obtaining $Ans = \{a, b, c\}$. □

This approach can be used for Datalog$^{\vee,\neg}$ queries and universal constraints. The extension for referential constraints can be found in [11]. We have successfully experimented with consistent query answering (CQA) based on specification of database repairs using the DLV system [31].

## 6 Semantics of CQA in Integration Systems

In this section we will assume that we are working under the LAV approach. Actually, this scenario is more challenging than GAV and inconsistency issues are more relevant due to the flexibility to insert/delete sources into/from the system.

Let us first consider an example that will help us motivate our notions of consistency of an integration system and consistent query answering.

*Example 13.* (example 8 continued) We found for query $Q$: $R(X, Y)$, that $Certain_{\mathcal{G}_1}(Q) = \{(a, b), (c, d), (a, c), (d, e)\}$. Now assume that we have global functional dependency $FD\colon X \rightarrow Y$. It is not satisfied by $D = \{(a, b), (c, d), (a, c), (d, e)\}$, nor by its supersets, i.e. no legal instance satisfies it. Since the tuples $(a, b), (a, c)$ participate in the violation of $FD$, only $(c, d), (d, e)$ should be consistent answers to the query.

Notice that the local functional dependencies $V_1\colon\ X \rightarrow Y,\quad V_2\colon\ X \rightarrow Y$ are satisfied by the sources. □

A virtual integration system does not have data at the global level. In spite of this, we would like to be able to characterize such a system as consistent or not, but we would like to do this on the basis of the data at hand, the one that is forced to be in the system, avoiding problems of consistency caused by data that is only potentially contained in the integration system. In this direction we concentrate on the *minimal* instances. We will see that this shift of semantics does not have an impact on query answering for relevant classes of queries in comparison to the semantics based on the whole class of legal instances.

**Definition 1.** [10] (a) A *minimal global instance* of an integration system $\mathcal{G}$ is a legal instance that does not properly contain any other legal instance. We denote by $Mininst(\mathcal{G})$ the set of minimal instances of $\mathcal{G}$.
(b) We say $\mathcal{G}$ is *consistent* wrt a set of global ICs $IC$ if for every $D \in Mininst(\mathcal{G})$ it holds $D \models IC$. □

*Example 14.* (example 13 continued) System $\mathcal{G}_1$ has only $D = \{(a,b),(c,d),(a,c),(d,e)\}$ as minimal instance. There FD does not hold; in consequence, $\mathcal{G}_1$ is inconsistent. □

The minimal instances will play a special role in our treatment of inconsistent integration systems. Since we have a well defined subclass of legal instances, it is natural to consider those answers to queries that hold for all the instances in the class.

**Definition 2.** [10] The *minimal answers* to a global query $Q$ posed to an integration system $\mathcal{G}$ are those answers that can be obtained from every minimal instance. We denote them by $\mathit{Minimal}_{\mathcal{G}}(Q)$. □

*Example 15.* (example 14 continued) For the query $Q\colon \mathit{Ans}(X,Y) \leftarrow R(X,Y)$, we have $\mathit{Minimal}_{\mathcal{G}_1}(Q) = \{(a,b),(c,d),(a,c),(d,e)\}$, which can be obtained by querying the only minimal instance. In this case the minimal answers coincide with the certain answers.

Now consider the query $Q'\colon \mathit{Ans}(X,Y) \leftarrow \neg R(X,Y)$. On the basis of the underlying domain, we have $(a,e) \in \mathit{Minimal}_{\mathcal{G}_1}(Q')$, because the minimal instance does not contain the tuple $(a,e)$. However, $(a,e) \notin \mathit{Certain}_{\mathcal{G}_1}(Q')$, because there are -non minimal- legal instances that contain the tuple $(a,e)$. □

What was shown in the previous example holds in general, namely $\mathit{Certain}_{\mathcal{G}}(Q) \subsetneqq \mathit{Minimal}_{\mathcal{G}}(Q)$; and for monotone queries [1] they coincide; but for queries with negation, possibly not.

As in the case of a single database, consistent answers will be the answers that are invariant under the repairs of the system. We make these intuitions precise.

**Definition 3.** [10] Let $\mathcal{G}$ be an integration system and $\mathit{IC}$ a set of global ICs.

(a) A *repair* of $\mathcal{G}$ wrt to $\mathit{IC}$ is a global instance that satisfies $\mathit{IC}$, and minimally differs from a minimal instance (wrt to inclusion of sets of tuples). We denote by $\mathit{Repairs}^{\mathit{IC}}(\mathcal{G})$ the set of repairs of $\mathcal{G}$ wrt $\mathit{IC}$.
(b) A ground tuple $\bar{t}$ is a *consistent answer* to a global query $Q$ wrt $\mathit{IC}$ if for every $D \in \mathit{Repairs}^{\mathit{IC}}(\mathcal{G})$, it holds $D \models Q[\bar{t}]$, i.e. $\bar{t}$ is an answer to $Q$ in $D$. We denote by $\mathit{Consis}^{\mathit{IC}}_{\mathcal{G}}(Q)$ set of consistent answers to $Q$. □

*Example 16.* (example 14 continued) Consider system $\mathcal{G}_1$ with the global $\mathit{FD}\colon X \rightarrow Y$. Since $D = \{(a,b),(c,d),(a,c),(d,e)\}$ is the only minimal instance, and it does not satisfy $\mathit{FD}$, the system has two repairs wrt $\mathit{FD}$, namely $D^1 = \{(a,b),(c,d),(d,e)\}$ and $D^2 = \{(c,d),(a,c),(d,e)\}$.

Now, for the query $Q : \mathit{Ans}(X,Y) \leftarrow R(X,Y)$, we have $\mathit{Consis}^{\mathit{FD}}_{\mathcal{G}_1}(Q) = \{(c,d),(d,e)\}$, as expected. For the existential query $Q''(X) : \mathit{Ans}(X) \leftarrow R(X,Y)$, we have $\mathit{Consis}^{\mathit{FD}}_{\mathcal{G}_1}(Q'') = \{a,c,d\}$. This shows that the value $a$ is not lost through the repair process and is still recovered as a consistent answer. □

This example shows that repairs may not be legal instances. The two repairs in it are not. This flexibility is necessary to make the system repairable. Remember that the repairs are just an auxiliary notion that we use to define the consistent answers to queries.

Here we are considering repairs that treat deletions and insertions of tuples symmetrically. Other approaches may privilege certain kinds of changes, e.g. in [20] insertions are preferred to deletions in the presence of referential ICs, with the purpose of giving a better account of the openness (or incompleteness) of the sources (see Section 11 for a more detailed discussion of alternative approaches). However, adapting our specifications and methodologies for query answering to this kind of special repairs is rather straightforward.

Also notice that an alternative definition of consistent answer in terms of being true in all consistent legal instances does not always work, because, in the presence of functional dependencies, most likely there won't be any consistent legal instances (see Example 13). Nevertheless, this alternative direction is studied in [57].

Except for strange cases -that we will exclude- where the set of ICs is *non generic* [11], i.e. it entails by itself (independently from the data) that a ground literal belong (or does not belong) to the database, the consistent answers are real answers. More precisely, for generic ICs, we have $\mathit{Consis}_{\mathcal{G}}^{IC}(Q) \subseteqq \mathit{Minimal}_{\mathcal{G}}(Q)$ [10]. If $\mathcal{G}$ is consistent wrt $IC$, then $\mathit{Consis}_{\mathcal{G}}^{IC}(Q) = \mathit{Minimal}_{\mathcal{G}}(Q)$. The problem with non generic ICs is that they force specific data items, which may have not been in the original instance, to belong (not to belong) to every (any) repair, something that can be easily achieved without appealing to ICs. This situation is illustrated in the following example.

*Example 17.* (example 16 continued) Assume that, in addition to the functional dependency, $IC$ also contains the non generic constraint $\forall x \forall y(x = a \wedge y = e \rightarrow R(x, y))$, saying that tuple $(a, e)$ belongs to $R$. In this case, there is only one repair for $\mathcal{G}_1$, namely $D^3 = \{(a, e), (c, d), (d, e)\}$. Now, $\mathit{Consis}_{\mathcal{G}_1}^{IC}(Q) = \{(a, e), (c, d), (d, e)\} \not\subseteq \mathit{Minimal}_{\mathcal{G}_1}(Q)$. □

Having defined what a consistent answer is, we need to find mechanisms for computing them.

## 7 Logic Programming Specification of Minimal Instances

In this section we will show how to specify the minimal instances of a virtual integration system under LAV using logic programs with stable model semantics [41, 42]. This specification is -as we will see- interesting and useful in itself, but in Section 8 it will also be used as the basis for computing consistent answers to queries.

### 7.1 The Simple Specification

We will start by giving a preliminary version of the specification program. This version is simpler to explain than the general, definitive one, and already contains the key ideas.

*Example 18.* (example 7 continued) It is easy to verify that the class of minimal instances for the system is $Mininst(\mathcal{G}) = \{\{P(a,c), P(a,z), R(z,b)\} \mid z \in \mathcal{D}\}$. Now, the set $\mathcal{V}^{-1}$ of inverse rules is

$$\begin{aligned} P(X, f(X,Z)) &\leftarrow V_1(X,Z) \\ R(f(X,Z), Z) &\leftarrow V_1(X,Z) \\ P(X,Y) &\leftarrow V_2(X,Y). \end{aligned}$$

Inspired by these inverse rules, we give the following specification program $\Pi(\mathcal{G}_2)$:

- Facts: $dom(a)$, $dom(b)$, $dom(c)$, $\ldots, V_1(a,b)$, $V_2(a,c)$.
- $P(X,Y) \leftarrow V_1(X,Z), F_1^Y(X,Z,Y)$,
  $R(Y,Z) \leftarrow V_1(X,Z), F_1^Y(X,Z,Y)$,
  $P(X,Y) \leftarrow V_2(X,Y)$.
- $F_1^Y(X,Z,Y) \leftarrow V_1(X,Z), dom(Y), choice((X,Z),(Y))$.

Here, $dom(x)$ is a domain predicate with elements in $\mathcal{D}$, $F_1^Y$ is a predicate corresponding to view $V_1$ and the existential variable $Y$ in its definition; and $choice((X,Z),(Y))$ is the choice operator introduced in [39], which non-determin-istically chooses a unique value for $Y$ for each combination of values in $(X,Z)$. In this way, the functional dependency $X, Z \rightarrow Y$ is enforced; and inclusion of redundant tuples in the global instances is (partly) avoided.

A program with choice $\Pi$ can be always transformed into a normal program, $SV(\Pi)$ [39] with stable model semantics [40]. The so-called *choice models* of the original program $\Pi$ are in one-to-one correspondence with the stable models of its *stable version* $SV(\Pi)$.

In our example, the stable models of $SV(\Pi(\mathcal{G}_2))$ are

$$\begin{aligned} \mathcal{M}_a &= \{dom(a), \ldots, V_1(a,b), V_2(a,c), \underline{P(a,c)}, \underline{R(a,b)}, \underline{P(a,a)}\}; \\ \mathcal{M}_b &= \{dom(a), \ldots, V_1(a,b), V_2(a,c), \underline{P(a,c)}, \underline{R(b,b)}, \underline{P(a,b)}\}; \\ \mathcal{M}_c &= \{dom(a), \ldots, V_1(a,b), V_2(a,c), \underline{P(a,c)}, \underline{R(c,b)}\}; \text{ etc.} \end{aligned}$$

Here we show only their relevant parts, skipping domain atoms, and atoms containing the $F_1$ predicate. In this example we find a one-to-one correspondence between the models of $\Pi(\mathcal{G}_2)$ and the minimal instances of $\mathcal{G}_2$. □

More generally, the preliminary version of the specification contains the following elements:

1. Facts: $dom(a)$ for every constant $a \in \mathcal{D}$, and $V_i(\bar{a})$ whenever $\bar{a} \in v_i$ for a source extension $v_i$ in $\mathcal{G}$.

2. For every view (source) predicate $V_i$ with definition $V_i(\bar{X}) \leftarrow P_1(\bar{X}_1), \ldots, P_n(\bar{X}_n)$, the rule
$$P_j(\bar{X}_j) \leftarrow V_i(\bar{X}), \bigwedge_{X_l \in (\bar{X}_j \setminus \bar{X})} F_i^{X_l}(\bar{X}, X_l).$$
3. For every predicate $F_i^{X_l}(\bar{X}, X_l)$ introduced in 2., the rule
$$F_i^{X_l}(\bar{X}, X_l) \leftarrow V_i(\bar{X}), \mathit{dom}(X_l), \mathit{choice}((\bar{X}), (X_l)).$$

It can be proved [16] that
$$\mathit{Mininst}(\mathcal{G}) \subseteq \text{ class of stable models of } SV(\Pi(\mathcal{G})) \subseteq \mathit{Legal}(\mathcal{G}). \tag{4}$$

Queries expressed as logic programs can be answered by running them together with $\Pi(\mathcal{G})$ under the cautious stable model semantics (that sanctions as true what is true of all stable models). As a consequence of (4) we obtain that for monotone queries $Q$ the answers obtained using $\Pi(\mathcal{G})$ coincide with $\mathit{Certain}_{\mathcal{G}}(Q)$ and $\mathit{Minimal}_{\mathcal{G}}(Q)$.

The inclusions in (4) suggest that equality may not be achieved. The following example shows that that is the case.

*Example 19.* Let $\mathcal{D} = \{a, b, c, \ldots\}$ be the underlying domain. The system $\mathcal{G}_3$ is defined by
$$\begin{aligned} V_1(X) &\leftarrow P(X, Y); & v_1 &= \{a\} \\ V_2(X, Y) &\leftarrow P(X, Y); & v_2 &= \{(a, c)\}. \end{aligned}$$

Here we have $\mathit{Mininst}(\mathcal{G}_3) = \{\{P(a, c)\}\}$, however, the legal global instances corresponding to stable models of $\Pi(\mathcal{G}_3)$ are of the form $\{\{P(a, c), P(a, z)\} \mid z \in \mathcal{D}\}$, that is, we obtain from the program more legal instances (or stable models) than the minimal instances. The reason is that $V_2$, being open, forces $P(a, c)$ to be in all legal instances, what makes the same condition on $V_1$ being automatically satisfied, i.e. no other values for $Y$ are needed. Nevertheless, the choice operator, as used above, may still choose, and it does, other values $z \in \mathcal{D}$.

As mentioned before, the simple version of the specification program for this system -even not being sound as a specification of the class of minimal instances- can be used to correctly compute minimal and certain answers to monotone queries. For instance, consider the following monotonic queries containing comparisons
$$\mathit{Ans} \leftarrow P(X, Y), Y \neq c \tag{5}$$
$$\mathit{Ans}(Y) \leftarrow P(a, Y), Y \neq c. \tag{6}$$

The boolean query (5) has answer *false* in the class $\{\{P(a, c), P(a, z)\} \mid z \in \mathcal{D}\}$, because it is not true of all the instances in it. Query (6) has empty answer in the same class. In the minimal instance $\{P(a, c)\}$, the queries have answer *false*, and $\emptyset$, respectively. We can see that these queries are correctly answered. □

At this point we could compare what can be obtained using the simple specification of minimal instances and what could we obtain by trying to use the inverse rules algorithm. Notice that the latter algorithm does not consider comparisons other than equalities [30]. The inverse rules can be seen as defining a sort of generic, symbolic instance, which is obtained by propagating the source contents through the inverses rules (from the bodies to the heads in them) and the function symbols.

For example, the set $\mathcal{V}^{-1}$ of inverse rules for the system in Example 19 consists of $P(X, f(X)) \leftarrow V_1(X)$ and $P(X, Y) \leftarrow V_2(X, Y)$. If we propagate the values in the sources, we obtain a "generic instance" containing $f$-values, namely

$$D_f = \{P(a, c), P(a, f(a))\}, \tag{7}$$

that represents a family of legal instances, each of which can be obtained by interpreting $f$ on the underlying domain $\mathcal{D}$. Basically, this class coincides with the class we obtained using the program above (and then it represents a superset of the minimal instances, but a subset of the legal instances). A difference is that with the specification program we obtain the instances explicitly.

If we attempt to use "instance" (7) to evaluate the queries (5) and (6) (this is the idea behind the IRA for conjunctive, built-in free queries in [30]), we obtain, assuming that $f(a)$ is different from $c$ because they are syntactically different, that the answer to (5) is $\mathit{Ans} = \mathit{true}$, whereas query (6) gets the answer $\mathit{Ans} = \{f(a)\}$, which, after elimination of the $f$-value, becomes $\mathit{Ans} = \emptyset$.

The problem with this methodology for query answering based on generic instances with functional values we just attempted, is that it does not capture the minimal instances, actually the only minimal instance $\{P(a, c)\}$ is missed by the assumption that $f(a) \neq c$. In order to make this approach work, we would have to consider alternative values for function $f$. Our explicit approach based on the choice operator achieves this, and can be naturally extended -as we will do in Section 7.2- in such a way that not only monotonic queries, but also non monotonic queries containing negation, can be handled correctly (the latter, wrt the minimal answer semantics).

*Example 20.* (example 19 continued) Assume $\mathcal{G}_3$ is extended with the source definition $V_3(X, Y) \leftarrow R(X, Y)$; $v_3 = \{(a, c)\}$. Then, the minimal instance is $\{\{P(a, c), R(a, c)\}\}$, and the instances obtained from the program are $\{\{P(a, c), P(a, z), R(a, c)\} \mid z \in \mathcal{D}\}$. Now the query

$$\mathit{Ans} \leftarrow P(X, Y),\ \mathit{not}\ R(X, Y) \tag{8}$$

has answer *false* both in the minimal instance and in the class of the instances obtained from the specification program. In the later case, in the sense that the query is not true in all the models of the program. That is, also in this case the simple specification is giving us the right minimal answers.

On the other side, the same query evaluated in the new IRA-induced, generic instance $D_f = \{P(a, c), P(a, f(a)), R(a, c)\}$ has answer *true* if the functional term is assumed to be different from $c$. □

This example shows that even for some non monotonic queries, the simple specification program returns the correct minimal answers. It is an interesting open problem to characterize the class of system descriptions and non monotonic queries for which the simple specification returns the correct minimal answers (however, see [16] for some results in this direction). On the other side, a naive application of the IRA to a query containing negation, as (8), does not give the correct answer.

It is a natural question as to whether the program with Skolem functions introduced by IRA (as in [30]) could be used, instead of the functional predicates, for specifying the repairs, pruning at the end the ground functional terms when queries are answered. In [16] it is shown -and this applies to both the simple and refined version of the specification program- that doing so does not necessarily capture the repairs of the system. The intuitive reason behind is that using the function symbols may prevent us from detecting violations to the ICs by the minimal instances. Actually, as Examples 19 and 20 already show, keeping the functional symbols may fail to properly capture the minimal instances, which is a problem when queries with negations or comparisons are to be answered.

In this work, when we answer non monotone queries, we are interested in the minimal answers. Actually, the consistent answers as defined here are a subset of the minimal answers (see Section 6). Wrt to the certain answers to non monotone queries, we can see that negated sub-queries can always be made false by adding extra data to the legal instances of an integration system with *open* sources. We believe that the notion of minimal answer to a non monotone query posed to an open system is the natural notion to use[4], instead of the notion of certain answer.

### 7.2 The Refined Specification

If we want $\Pi(\mathcal{G})$ to specify only the minimal instances, then the program has to be refined. The new version $\Pi(\mathcal{G})$ detects in which cases it is necessary to use the function predicates. This is achieved by means of a stronger condition, $add_{V_i}(\bar{X})$, in the choice rules, i.e. $F_i^{X_l}(\bar{X}, X_l) \leftarrow add_{V_i}(\bar{X}), dom(X_l), choice((\bar{X}), (X_l))$, where $add_{V_i}(\bar{X})$ is true only when the openness of $V_i$ is not satisfied through other views; and this can be further specified by means of extra rules. The general refined version is described and analyzed in detail in [16]. For it, the class of stable models of the program provably coincides with the minimal instances. In consequence, the program can be used to compute minimal answers to arbitrary queries and certain answers to monotone queries.

The refined version of the program uses annotation constants to be placed in an extra argument added to the global relations. Their intended semantics is given in Table 2. Annotation $\mathbf{t_d}$ is used to read off the atoms in the minimal instances. The others are annotations that are used to compute intermediate atoms. We illustrate the refined version by means of an example.

[4] Assuming, as we have done in this chapter, that the sources are defined as conjunctive views or disjunctions thereof. In particular, they are defined without negation.

**Table 2.** Semantic of Annotation Constants for Minimal Models

| annotation | atom | the tuple $P(\bar{a})$ is ... |
|---|---|---|
| $\mathbf{t_d}$ | $P(\bar{a}, \mathbf{t_d})$ | an atom of the minimal legal instances |
| $\mathbf{o}$ | $P(\bar{a}, \mathbf{o})$ | an obligatory atom in all the minimal legal instances |
| $\mathbf{v_i}$ | $P(\bar{a}, \mathbf{v_i})$ | an optional atom introduced to satisfy the openness of view $V_i$ |
| $\mathbf{nv_i}$ | $P(\bar{a}, \mathbf{nv_i})$ | an optional atom introduced to satisfy the openness of a view other than $V_i$ |

*Example 21.* (example 19 continued) The refined program $\Pi(\mathcal{G}_3)$ is:

$$dom(a), dom(c), ..., V_1(a), V_2(a, c). \tag{9}$$

$$P(X, Y, \mathbf{v_1}) \leftarrow add_{V_1}(X), F_1^Y(X, Y). \tag{10}$$

$$add_{V_1}(X) \leftarrow V_1(X),\ not\ aux_{V_1}(X). \tag{11}$$

$$aux_{V_1}(X) \leftarrow var_{V_1, Z}(X, Z). \tag{12}$$

$$var_{V_1, Z}(X, Z) \leftarrow P(X, Z, \mathbf{nv_1}). \tag{13}$$

$$F_1^Y(X, Y) \leftarrow add_{V_1}(X), dom(Y), choice((X), (Y)). \tag{14}$$

$$P(X, Y, \mathbf{o}) \leftarrow V_2(X, Y). \tag{15}$$

$$P(X, Y, \mathbf{nv_1}) \leftarrow P(X, Y, \mathbf{o}). \tag{16}$$

$$P(X, Y, \mathbf{t_d}) \leftarrow P(X, Y, \mathbf{v_1}). \tag{17}$$

$$P(X, Y, \mathbf{t_d}) \leftarrow P(X, Y, \mathbf{o}). \tag{18}$$

Rules (10) to (13) ensure that if there is an atom in source $V_1$, e.g. $V_1(\bar{a})$, and if an atom of the form $P(\bar{a}, Y)$ was not added by view $V_2$, then it is added by rule (10) with a $Y$ value given by the functional predicate $F_1^Y(\bar{a}, Y)$. This function predicate is calculated by rule (14). Rule (15) enforces the satisfaction of the openness of $V_2$ by adding obligatory atoms to predicate $P$, and rule (16) stores this atoms with the annotation $\mathbf{nv_1}$ implying that they were added by a view different from $V_1$. The last two rules gather with annotation $\mathbf{t_d}$ the elements that were generated by both views. Those are the atoms in the minimal instances.

The only stable model of this program is $\{dom(a), dom(c), \ldots, V_1(a), V_2(a, c), P(a, c, \mathbf{t_d}), P(a, c, \mathbf{o}), P(a, c, \mathbf{nv_1}), aux_{V_1}(a)\}$, which corresponds to the only minimal legal instance $\{P(a, c)\}$. □

We have obtained an answer set programming specification of the minimal instances of an open integration system under LAV. From it, the minimal answers to complex queries, e.g. non stratified Datalog queries [1], can be computed using the cautions or skeptical answer set semantics that sanctions as true what is true of all stable models. Notice that the refined version (and also the simple version) of the specification program $\Pi(\mathcal{G})$ is a non stratified program, whose data complexity [1] is likely to be higher than polynomial [27]. As with the simple program, the refined program can be used to compute the certain answers to monotone queries.

It is interesting to observe that the specification $\Pi(\mathcal{G})$ we just gave can be seen as a considerable extension of the original IRA algorithm since it can be used to obtain the certain answers to monotone queries involving comparisons (see Example 19), and the minimal answers to non-monotone queries.

There are several issues and possible extensions that are discussed in detail in [16]. We briefly mention some of them here. First, we do not need to make any assumption about the underlying domain for the logic programming based specifications of minimal instances to work properly. All we need is that it -possibly properly- contains the active domains of the sources and the constants that may appear in the view definitions. However, if the program is to be run with a system like DLV, we need to have a finite number of elements in the domain. We can always simulate the potential infiniteness of the underlying domain by means of a sufficiently large finite domain [16]. This can be achieved by introducing fresh constants. This subject related to a finite vs. infinite underlying domain certainly deserves further investigation. Any case, computing with infinite stable models has started to receive attention from the answer set programming community [14].

A possible extension, also discussed in [16], consists in having views defined by disjunctions of conjunctive queries. Inspiration for the specification programs can be found in the extension of the IRA to the case of disjunctive sources [29].

We will use the specification of minimal instances as a basis for the computation of consistent answers (see Section 8). In Section 9, the specification is extended to the case where also closed sources participate in the integration system.

## 8 Computing Consistent Answers in Integration Systems

We will see two methodologies for consistently answering queries posed to virtual integration systems under LAV. The first one, in Section 8.1, is based on first-order query rewriting. The second one, to be presented in Section 8.2, is much more general, and provides a solution based on the specification of the repairs of the minimal instances of an open integration systems. Both methodologies eventually rely on the specification of minimal instanced presented in Section 7.

### 8.1 Query Rewriting for CQA

In this section we will describe a methodology, first presented in [10], that provides a partial solution to the problem of CQA under the LAV approach. It builds upon the query rewriting approach to CQA for single relational databases described in Section 5.1. The limitations of that approach are inherited by the solution for the case of integration of data sources. In consequence, this solution applies to queries $Q$ that are conjunctions of literals, but without projection (or existential quantification); and global integrity constraints that are universal. In consequence, referential ICs are excluded.

The high level description of the rewriting based algorithm for CQA in integration system is as follows: Given as input a set $IC$ of global integrity constraints, and global query $Q$ that is a conjunction of literals, we do the following

*Meta–Algorithm* (19)

1. Rewrite $Q(\bar{X})$ into the first-order query $T^\omega(Q(\bar{X}))$ using $IC$.[5]
2. Transform $T^\omega(Q(\bar{X}))$ into a recursion-free Datalog$^\neg$ query program $\Pi(T^\omega(Q))$ (this is straightforward [64]).
3. Find a query plan, $Plan(\Pi(T^\omega(Q)))$ to answer the query $\Pi(T^\omega(Q))$ posed to the global system.
4. Evaluate the query plan on the view extensions of $\mathcal{G}$ to compute the answer set.

A problem with this algorithm is that the program $\Pi(T^\omega(Q))$ may contain negation, that is introduced at the first step. We give some examples.

*Example 22.* Consider the integration system

$$\begin{array}{ll} V_1(X,Y) \leftarrow P(X,Y); & v_1 = \{(a,d)\} \\ V_2(X,Z) \leftarrow P(X,Y), R(Y,Z); & v_2 = \{(a,b),(b,c)\}. \end{array}$$

The minimal instances are of the form $D_{uv} = \{P(a,u), R(u,b), P(b,v), R(v,c), P(a,d)\}$, with $u, v \in \mathcal{D}$. Now consider the global IC $IC: \forall x \forall y(\neg P(x,y) \vee \neg R(x,y))$. The system is inconsistent, because the minimal instances obtained with $u = c, v = a$, i.e. $D_{ca} = \{P(a,c), R(c,b), P(b,a), R(a,c), P(a,d)\}$ is inconsistent. The same happens with $D_{bb}$. The other minimal instances are consistent. Then, the repairs are all the $D_{u,v}$ above, except for the last two combinations, which in their turn contribute with the repairs $D^1_{ca} = \{R(c,b), P(b,a), R(a,c), P(a,d)\}$, $D^2_{ca} = \{P(a,c), R(c,b), P(b,a), P(a,d)\}$, $D^1_{bb} = \{P(a,b), R(b,b), R(b,c), P(a,d)\}$ $D^2_{bb} = \{P(a,b), P(b,b), R(b,c), P(a,d)\}$. Now, consider the query $Q: P(X,Y)?$. The only answer to this query in common to all repairs is $\{P(a,d)\}$, then this is the only consistent answer.

On the rewriting side, if we want the consistent answers to the same query relative to $IC$, we rewrite the query as follows $T(Q): (P(X,Y) \wedge \neg R(X,Y))$ (see Example 11), which produces the following query program that contains negation: $Ans(X,Y) \leftarrow P(X,Y),\ not\ R(X,Y)$. □

*Example 23.* (example 6 continued) $FD$ can be written in the form

$$\forall x \forall y \forall z(\neg R(x,y) \vee \neg R(x,z) \vee y = z). \tag{20}$$

If the query $Q$: $R(X,Y)$? is posed to the system, we have to find the residues of $R(X,Y)$ wrt (20), and we obtain after the first step the rewritten query

$$T^\omega(Q(X,Y))\colon\ R(X,Y)\ \wedge\ \neg\exists Z(R(X,Z)\ \wedge\ Z \neq Y). \tag{21}$$

[5] We are assuming here that $T^\omega(Q(\bar{X}))$ produces a finite formula. Conditions for this to happen in terms of $Q$ and $IC$ are studied in [3, 25]. However, those conditions are satisfied by the most common universal ICs found in database practice.

Query (21) is translated into the following Datalog$^\neg$ program $\Pi(T^\omega(Q(X,Y)))$:

$$Ans(X,Y) \leftarrow R(X,Y),\ not\ S(X,Y) \tag{22}$$

$$S(X,Y) \leftarrow R(X,Z), dom(Y), Y \neq Z \tag{23}$$

$$dom(a), dom(b), dom(c), dom(d), dom(e), ... \tag{24}$$

The domain extends the *active domain* [1] that contains the constants in the sources and those that may appear in the view definitions. This is a form of materialization of a domain closure assumption, however we are not necessarily closing wrt the active domain, but wrt a superset of it that contains fresh constants. This allows us to correctly compute certain answers (see [16] for a detailed discussion of this issue). The introduction of the *dom* predicate in programs is a general way to make the rules *safe* [72]. Despite these considerations, in this example, the domain predicate is not necessary, because (21) is logically equivalent to

$$T^\omega(Q(X,Y)){:}\ R(X,Y)\ \wedge\ \neg\exists Z(R(X,Y)\ \wedge\ R(X,Z)\ \wedge\ Z \neq Y).$$

In consequence, program $\Pi(T^\omega(Q(X,Y)))$ can be written as the set of safe rules $Ans(X,Y) \leftarrow R(X,Y),\ not\ S(X,Y)$ and $S(X,Y) \leftarrow R(X,Y), R(X,Z), Y \neq Z$.

At step 3. of algorithm (19), we need a query plan to answer the query expressed by (22)-(24). As we can see, the query contains negation and comparisons. □

Algorithms like IRA are designed to deal with negation-free queries without comparisons [30]. On the other side, $\Pi(T^\omega(Q))$ does not contain recursion but contains negation. In consequence, an algorithm like IRA, if it is going to be applied in this context, has to be extended in order to handle queries that are, e.g. non recursive Datalog programs with negation and comparisons.

Some very limited extensions of the IRA algorithm have been proposed in order to include negation [10, 74, 35]. However, we can use our specification of the minimal instances (see Section 7) as a general query plan mechanism for eventually computing consistent answers to queries. In Algorithm (19) that specification can be used in the third step. All one needs to do is combine the query obtained after the second step (with its predicates expanded with a new, final argument with the annotation $\mathbf{t_d}$ in it) with the specification of the minimal instances. The combined program is run under the cautions stable model semantics.

*Example 24.* (example 22 continued) The query $Ans(X,Y) \leftarrow P(X,Y)$, $not\ R(X,Y)$ has to be combined with the specification of the minimal instances of the integration system, which is essentially the same as the one given in Example 18. If we want or need[6] to use the refined version of the specification of minimal instances, then the query has to be first transformed into $Ans(X,Y) \leftarrow P(X,Y,\mathbf{t_d}),\ not\ R(X,Y,\mathbf{t_d})$. □

[6] In this example this is not necessary, because the simple program correctly specifies the class of minimal instances. In [16] sufficient conditions are identified for this to happen.

## 8.2 CQA from Specifications of Repairs

A more general methodology that the one presented in Section 8.1 is based on a logic programming specifications of the repairs of the minimal instances of an integration system. First results were presented in [15], and full details can be found in [16]. This methodology works for queries expressed in extensions of Datalog, in particular, for first-order queries; and universal ICs combined with acyclic sets of referential ICs. In the rest of this section, we will assume that sources are open, and defined as conjunctive views over the global schema. However the solution can be extended to combinations of closed and open sources (see Section 9), and views defined as disjunctions of conjunctive queries [16].

Figure 2 describes the methodology in general terms. In order to compute the consistent answer to a global query, the query is expressed as a query program, which is run in combination with other programs that specifies, in two layers, the minimal instances of the integration systems, first, and then, the repairs of the minimal instances. Of course, the same specification program can be used with different queries. The specification of minimal instances is the one presented in Section 7.

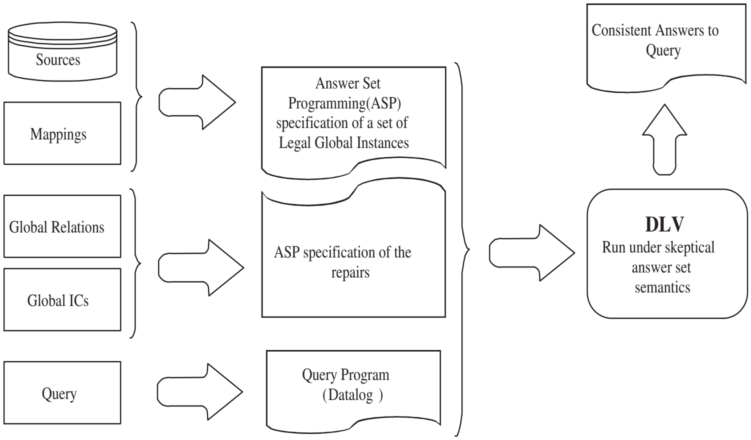
Sources
Mappings
Global Relations
Global ICs
Query
Answer Set Programming(ASP) specification of a set of Legal Global Instances
ASP specification of the repairs
Query Program (Datalog )
DLV
Run under skeptical answer set semantics
Consistent Answers to Query

**Fig. 2.** Computing Consistent Answers

What we have so far is a specification of minimal instances of an open integration system, but they may not satisfy certain global ICs. In consequence, we may consider specifying their repairs wrt those ICs. For this we can apply the ideas and techniques developed to specify repairs of single databases (Section 5). Actually, we can combine into a repair program, $\Pi(\mathcal{G}, IC)$, the program that specifies the minimal instances with a program that specifies the repairs of each minimal instance. This is because a minimal instance can be seen as (or is) a single database instance. Instead of a full treatment (see [16]), we give an example.

*Example 25.* (example 21 continued) Consider system $\mathcal{G}_3$, but now with the global integrity constraint *Sym*: $\forall x \forall y (P(x,y) \rightarrow P(y,x))$. Since $Mininst(\mathcal{G}_3) = \{\{P(a,c)\}\}$, an the only instance does not satisfy *Sym*, the system is inconsistent.

The repair program, $\Pi(\mathcal{G}_3, Sym)$, consists of two layers. The first one is exactly program $\Pi(\mathcal{G}_3)$ in Example 21 that specifies the minimal instances; and the second layer is the following subprogram that repairs the minimal instances; it builds on the atoms annotated with $\mathbf{t_d}$ in the first layer:

$$\begin{aligned}
P(X,Y,\mathbf{t}^\star) &\leftarrow P(X,Y,\mathbf{t_a}), dom(X), dom(Y).\\
P(X,Y,\mathbf{t}^\star) &\leftarrow P(X,Y,\mathbf{t_d}), dom(X), dom(Y).\\
P(X,Y,\mathbf{f}^\star) &\leftarrow dom(X), dom(Y),\ \ not\ P(X,Y,\mathbf{t_d}).\\
P(X,Y,\mathbf{f}^\star) &\leftarrow P(X,Y,\mathbf{f_a}), dom(X), dom(Y).\\
P(X,Y,\mathbf{f_a}) \vee P(Y,X,\mathbf{t_a}) &\leftarrow P(X,Y,\mathbf{t}^\star), P(Y,X,\mathbf{f}^\star), dom(X), dom(Y).\\
P(X,Y,\mathbf{t}^{\star\star}) &\leftarrow P(X,Y,\mathbf{t_a}), dom(X), dom(Y).\\
P(X,Y,\mathbf{t}^{\star\star}) &\leftarrow P(X,Y,\mathbf{t_d}), dom(X), dom(Y),\ \ not\ P(X,Y,\mathbf{f_a}).\\
P(X,Y,\mathbf{f}^{\star\star}) &\leftarrow P(X,Y,\mathbf{f_a}), dom(X), dom(Y).\\
P(X,Y,\mathbf{f}^{\star\star}) &\leftarrow dom(X), dom(Y),\ \ not\ P(X,Y,\mathbf{t_d}),\\
&\qquad not\ P(X,Y,\mathbf{t_a}).\\
&\leftarrow P(X,Y,\mathbf{t_a}), P(X,Y,\mathbf{f_a}).
\end{aligned}$$

The stable models of this program are:

$$\begin{aligned}
\mathcal{M}_1 = \{&dom(a),\ dom(c), \ldots,\ V_1(a),\ V_2(a,c),\ P(a,c,\mathbf{nv_1}),\ P(a,c,\mathbf{v_2}),\\
&P(a,c,\mathbf{t_d}),\ P(a,c,\mathbf{t}^\star),\ aux_{V_1}(a),\ P(a,a,\mathbf{f}^\star),\ P(c,a,\mathbf{f}^\star),\ P(c,c,\mathbf{f}^\star),\\
&P(a,a,\mathbf{f}^{\star\star}),\ P(c,a,\mathbf{t_a}),\ P(c,c,\mathbf{f}^{\star\star}),\ \underline{P(a,c,\mathbf{t}^{\star\star})}, P(c,a,\mathbf{t}^\star),\\
&\underline{P(c,a,\mathbf{t}^{\star\star})}\}.
\end{aligned}$$

$$\begin{aligned}
\mathcal{M}_2 = \{&dom(a),\ dom(c), \ldots,\ V_1(a),\ V_2(a,c),\ P(a,c,\mathbf{nv_1}),\ P(a,c,\mathbf{v_2}),\\
&P(a,c,\mathbf{t_d}),\ P(a,c,\mathbf{t}^\star),\ aux_{V_1}(a),\ P(a,a,\mathbf{f}^\star),\ P(c,a,\mathbf{f}^\star),\ P(a,c,\mathbf{f}^\star),\\
&P(c,c,\mathbf{f}^\star),\ P(a,a,\mathbf{f}^{\star\star}),\ P(c,a,\mathbf{f}^{\star\star}),\ P(a,c,\mathbf{f}^{\star\star}),\ P(c,c,\mathbf{f}^{\star\star}),\\
&P(a,c,\mathbf{f_a})\}.
\end{aligned}$$

By reading the literals annotated with $\mathbf{t}^{\star\star}$, we see that the first model corresponds to the repair $\{P(a,c), P(c,a)\}$; the second one, to the empty repair. □

Repair programs can be given for specifying the repairs of any open integration system under the LAV approach with conjunctive view definitions; and for any set of ICs containing universal and acyclic referential integrity constraints [15, 16].

The restriction to sets of ICs that do not contain cycles in its referential ICs has to do with limitations of the logic programming based approach to the specification of repairs of single relational databases as presented in Section 5. Fundamental, theoretical reasons behind these limitations, that are inherited by our repair programs for integration systems, are studied in depth in [26, 20, 38].

With the repair programs, we can now compute consistent answers to global queries. Let $Q(\bar{x})$ be a query posed to an integration system $\mathcal{G}$. The methodology is as follows. First the query gets its literals annotated with $\mathbf{t}^{\star\star}, \mathbf{f}^{\star\star}$, e.g. if the query is first order, say $Q(\cdots P(\bar{u}) \cdots \neg R(\bar{v}) \cdots)$, we pass to $Q' := Q(\cdots P(\bar{u}, \mathbf{t}^{\star\star}) \cdots R(\bar{v}, \mathbf{f}^{\star\star}) \cdots)$. Next, a query program $\Pi(Q')$ with an $Ans(\bar{X})$ predicate is produced from $Q$ (this is standard [64]). Finally, the program $\Pi := \Pi(Q') \cup \Pi(\mathcal{G}, IC)$ is run under the stable model semantics; and the ground atoms $Ans(\bar{t}) \in \bigcap\{S \mid S$ is a stable model of $\Pi\}$ are collected in the answer set to be returned to the user.

*Example 26.* (example 25 continued) Consider $\mathcal{G}_3$ and the global query $Q: P(X,Y)$? From it we generate $Q': \ P(X,Y,\mathbf{t}^{\star\star})$, which in its turn is transformed into the query program $\Pi(Q'): \ Ans(X,Y) \leftarrow P(X,Y,\mathbf{t}^{\star\star})$. Next, we form $\Pi = \Pi(\mathcal{G}_3, Sym) \cup \Pi(Q')$, with $\Pi(\mathcal{G}_3, Sym)$ as in Example 25.

Now, the models of program $\Pi$ are those of $\Pi(\mathcal{G}_3, Sym)$ but extended with ground $Ans$ atoms, namely they are: $\overline{\mathcal{M}}_1 = \mathcal{M}_1 \cup \{Ans(a,c), Ans(c,a)\}$; $\overline{\mathcal{M}}_2 = \mathcal{M}_2 \cup \emptyset$. Since there are no $Ans$ atoms in common, then query has no consistent answers (as expected). □

*Example 27.* (example 16 continued) The program that computes the consistent answers to query $Q(X,Y)\colon R(X,Y)$? from system $\mathcal{G}_1$ wrt $FD$ is:

Subprogram for minimal instances:

$dom(a). dom(b).\ dom(c). dom(d).\ dom(e).\ \ldots V_1(a,b).\ V_1(c,d).\ V_2(c,a).\ V_2(e,d).$

$$\begin{aligned} R(X,Y,\mathbf{t_d}) &\leftarrow V_1(X,Y).\\ R(Y,X,\mathbf{t_d}) &\leftarrow V_2(X,Y). \end{aligned}$$

Repair subprogram:

$$\begin{aligned} R(X,Y,\mathbf{t}^\star) &\leftarrow R(X,Y,\mathbf{t_a}), dom(X), dom(Y).\\ R(X,Y,\mathbf{t}^\star) &\leftarrow R(X,Y,\mathbf{t_d}), dom(X), dom(Y).\\ R(X,Y,\mathbf{f}^\star) &\leftarrow dom(X), dom(Y),\ \mathit{not}\ R(X,Y,\mathbf{t_d}).\\ R(X,Y,\mathbf{f}^\star) &\leftarrow R(X,Y,\mathbf{f_a}), dom(X), dom(Y).\\ R(X,Y,\mathbf{f_a}) \vee R(X,Z,\mathbf{f_a}) &\leftarrow R(X,Y,\mathbf{t}^\star), R(X,Z,\mathbf{t}^\star), Y \neq Z,\\ &\qquad dom(X), dom(Y), dom(Z).\\ R(X,Y,\mathbf{t}^{\star\star}) &\leftarrow R(X,Y,\mathbf{t_a}), dom(X), dom(Y).\\ R(X,Y,\mathbf{t}^{\star\star}) &\leftarrow R(X,Y,\mathbf{t_d}), dom(X), dom(Y),\ \mathit{not}\ R(X,Y,\mathbf{f_a}).\\ &\leftarrow R(X,Y,\mathbf{f_a}), R(X,Y,\mathbf{t_a}). \end{aligned}$$

Query subprogram:

$$Ans(X,Y) \leftarrow R(X,Y,\mathbf{t}^{\star\star}).$$

The *Ans* atom in common to the two stable models are $Ans(c,d), Ans(d,e)$, then the set of consistent answers to the query is $\{(c,d),(d,e)\}$.

Here we have used the simple version of the program that specifies the minimal instances. In this case the specification is *sound*, i.e. it does not compute any model that does not correspond to a minimal instance. Classes of system descriptions for which the simple specification has a sound behavior wrt the class of minimal instances are studied in [16]. The example here falls into one of those classes. □

The specifications we have presented are sound and complete for CQA for sets of ICs consisting of universal integrity constraints and acyclic sets of referential integrity constraints [16]. Views can be defined by disjunctions of conjunctive formulas; and queries can be arbitrary Datalog$^{\neg}$ queries.

## 9 Specification of Minimal Instances: Mixed Case

So far we have assumed that all the sources are open. Now we will consider the *mixed case*, where some of the sources may be *closed* or closed and open (*clopen*). In consequence, a virtual data integration system will have a description like the one in (1), but each source will have a label indicating if it is open, closed or clopen [43]. Intuitively speaking, a closed source contains a superset of the data of its kind in the system, and the clopen source contains exactly all the data of its kind in the system.

More precisely, if a material source relation $v$, defined as the view $V(\bar{X}) \leftarrow \varphi_V(\bar{X})$ of the global system, has been defined as a closed (clopen) source, then in any legal instance $D$, it must hold $v \supseteq \varphi_V(D)$ (resp. $v = \varphi_V(D)$).

In this section we will describe how to modify the program that specifies the minimal instances presented in Section 7 when some of the sources are declared closed or clopen.

*Example 28.* For the domain $\mathcal{D} = \{a,b,c,\dots\}$, consider the integration system $\mathcal{G}_4$:

$$V_1(X,Z) \leftarrow P(X,Y), R(Y,Z); \qquad v_1 = \{(a,b)\} \quad \textit{open} \tag{25}$$

$$V_2(X,Y) \leftarrow P(X,Y); \qquad v_2 = \{(a,c)\} \quad \textit{clopen} \tag{26}$$

In Example 18 we had the same sources and definitions, but then they were all declared open; and we had $Mininst(\mathcal{G}_2) = \{\{P(a,c), P(a,z), R(z,b)\} \mid z \in \mathcal{D}\}$. Now, the label on the second sources forces relation $P$ to be $\{(a,c)\}$. In consequence, we obtain $Mininst(\mathcal{G}_4) = \{\{P(a,c), R(c,b)\}\}$. □

It is clear that the closed and clopen labels will impose additional restrictions on the legal instances we had for the purely open case, when all sources are open. In particular, these labels will never force to add new tuples to the legal instances. Actually, if a source is declared closed, then that source will contribute with the empty set of tuples to the minimal instances of the integration system.

With open, closed and clopen sources, the sets of legal and minimal instances will always be subsets of the same sets for the case where the same sources are all declared open. In order to obtain the minimal instances in the mixed case, all we have to do is filter out some of the minimal instances obtained in the purely open case, namely those that violate the closedness condition for some of the sources. This can be captured at the logic program specification level by means of a program denial constraint, which has the effect of discarding some of the stable models.

In the mixed case, the program $\Pi_{mix}(\mathcal{G})$ that specifies the minimal instances consists of the program $\Pi(\mathcal{G})$ we had for the open case in Section 7 (as if all the sources were open) plus a denial constraint of the form $\leftarrow P_1(\bar{X}_1), \ldots, P_n(\bar{X}_n),\ \mathit{not}\ V(\bar{X})$, for each closed (or clopen) source $v$ with view definition $V(\bar{X}) \leftarrow P_1(\bar{X}_1), \ldots, P_n(\bar{X}_n)$. That is, the open sources contribute with rules to the program, the clopen sources both with rules and program constraints, and the closed sources with program constraints only.

With these modifications, the obtain the same correspondence between the stable models of the program $\Pi_{mix}(\mathcal{G})$ and the minimal instances of the mixed integration system $\mathcal{G}$.

*Example 29.* (example 28 continued) The program $\Pi_{mix}(\mathcal{G}_4)$ that specifies the minimal instances of system $\mathcal{G}_4$ is:

$$\mathit{dom}(a).\ \ \mathit{dom}(b).\ \ \mathit{dom}(c).\ \ \ldots\ \ V_1(a,b).\ \ V_2(a,c).$$

$$\begin{aligned}
P(X,Y) &\leftarrow V_1(X,Z), F_1^Y(X,Z,Y)\\
R(Y,Z) &\leftarrow V_1(X,Z), F_1^Y(X,Z,Y)\\
P(X,Y) &\leftarrow V_2(X,Y)\\
F_1^Y(X,Z,Y) &\leftarrow V_1(X,Z), \mathit{dom}(Y), \mathit{choice}((X,Z),(Y))\\
&\leftarrow P(X,Y),\ \mathit{not}\ V_2(X,Y).
\end{aligned}$$

This program, excluding the last denial, coincides with program $\Pi(\mathcal{G}_2)$ in Example 18, where the same sources and definitions are considered, but all the sources are open only. With the denial constraint, that enforces the closeness of source $V_2$, the only stable model of $\Pi_{mix}(\mathcal{G}_4)$ is $\{\mathit{dom}(a), \ldots, V_1(a,b), V_2(a,c), \underline{P(a,c)},\ F_1^Y(a,b,c), \underline{R(c,b)}\}$, which corresponds to the only minimal instance $\{\{P(a,c),\ R(c,b)\}\}$. □

Notice that the solution we have reached via logic programs is similar in spirit to the solution presented in [43], where the mixed case is treated. There tableaux with constraints are used to compactly represent the legal instances and obtain certain answers. The tableaux capture the open part, and the constraints, as in our solution, the closed part.

## 10 Ongoing and Future Work

There are still many relevant open issues in this line of research. Consistency issues have barely investigated in the context of virtual data integration systems. Other research results obtained by other authors in this direction are described in Section 11.

The solution to the problem of *certain and consistent query answering* in virtual data integration system under the LAV approach presented in Sections 7 and 8.2, resp. are quite general, and conceptually clear, however many implementation issues are still open. They have to be addressed in order to use those solutions in real database applications.

A first step would be to implement certain and consistent query answering for the most common queries and constraints found in database practice. Ad hoc mechanisms could be derived from the logic programming specifications. In this direction, [13] shows how to derive, for some classes of queries, first order rewritings from the logic programs that specify repairs of single databases. Of course, by complexity reasons, this is not always possible [11].

In more general terms, the research should be focused on the specialization, optimization, and evaluation of the logic programs we have presented. Specialization has to do with deriving program for particular classes of queries and constrains from the general ones, that are better behaved in terms of evaluation. Optimization has to do with producing equivalent programs that can be more easily evaluated, in particular, the interaction of the logic programming system with the underlying databases has to be optimized. Some optimizations for CQA in single databases are introduced in [7, 13].

Evaluation issues are also extremely relevant. They have to do with splitting the program, caching intermediate results, reusing previous computations, localizing computations to the relevant parts of the data sources. Answering a particular query may not require a full computation of the repairs, but only partial computation could suffice. It becomes important to detect which are the relevant portions of data [32].

Query evaluation is a crucial point. Current implementations of answer set programming are not oriented to the problem of query answering as found in databases, where open queries are usually posed and a set of answers is returned to the user. Instead, the emphasis in answer set programming has been placed on computation of (some) models, and answering ground queries. Actually, the evaluation methodology in such systems is, in general terms, based on massive grounding of the program, full computation of stable models, and recollection of atoms in the intersection of all of them. Grounding is already a problem if the program is to be grounded on the full active domain of the databases, because the ground program generated can be huge. See [31, 59] for a discussion of implementation details.

Query evaluation methodologies that are directed by the query seem to be necessary for applications in databases, in particular, the development and implementation of "magic sets" methods [1] for disjunctive logic programs under the stable model semantics is a promising area of research. Recent research has started addressing this problem [46].

Most of the research around query answering in virtual data integration systems starts from a fixed class of mappings that describe the contents of the sources. Given a class, the semantics and query answering mechanisms are provided. However, in spite of the fact that design issues of data integration systems have been studied [8, 9, 71], the analysis of the impact of particular forms of design on the syntax of the mappings and on query answering has been largely neglected. In particular, if would be interesting to investigate how the integration system is to be designed if certain restrictions on the mappings are to be satisfied. Determining what is a *good design* for a virtual data integration in terms of the query answering features of the system is something that deserves further investigation.

## 11 Related Work

Here we will mention only those papers that more or less explicitly consider consistency issues in virtual data integration systems. Other important papers on virtual data integration have been cited in the main body of this paper, including those that assume that certain integrity constraints hold when query plans are derived.

An early approach to virtual data integration is presented in [68]. There, operations on the relations and attributes in the sources are defined, e.g. meet, join, aggregate, add. These operators applied to a set of source databases generate a global virtual database schema. In this way, mappings are derived and express the global relations as results of a set of operations on the source relations. When a query is posed, it is translated to the sources relations by considering the operators in the inverse order in which they where applied.

In [69], a model is presented where the integration system is considered to have a real global database, and the sources are views obtained by applying projections and selections to this global database. In this framework, the possibility of having inconsistencies in the instances is considered. Inconsistency is reflected in the fact that it can be impossible for the sources to be views of this single global database instance. For example. Consider the global schema with a binary relation $R$ with attributes $A, B$. Let source I have elements $\{a\}$, and source II, elements $\{b\}$, and the respective views $V_1 = \Pi_A(R)$, $V_2 = \Pi_A(R)$. In this case, there is an instance inconsistency, because even though both sources are views of the single global database and they have the same view definitions, their elements are different. In order to handle this situation, the notion of approximate answer is introduced, actually a lower bound and an upper bound are given, corresponding, respectively, to the intersection and union of all the possible answers of the rewriting of the query using the views. No complexity analysis is provided. Global integrity constraints are not considered.

In [19], the use of integrity constraints in a data integration system under the GAV approach for clopen and open sources is studied. In the clopen[7] case, the authors argue that the integration system can be seen as a single database, and therefore, the query answering process in the presence of ICs can be done appealing to the concept of repair [3] and CQA mechanisms for single databases [3, 47, 6]. If the sources are open and there are no ICs, queries can be answered by unfolding. If there are ICs, the semantic is given by the set of legal instances that satisfy both the open mappings and the integrity constraints. Their legal instances can be seen as repairs (in our sense) of the *retrieved global database* that is obtained by propagating the source elements through the mapping. Repairs admit only tuple insertions. Since [19] considers as legal those databases that satisfy the ICs, it holds that their "certain answers" correspond to our consistent answers. If there are no legal instances (in their sense), the integration system is said to be "inconsistent". In this case, tuple deletions are also needed in order to achieve consistency.

In [17] the same semantics as in [19] is consider, for GAV and open sources. There they present an algorithm for rewriting a conjunctive query [1] in order to retrieve the "certain answers" (our consistent answers). This algorithm handles foreign key constraints and assumes that the key constraints are preserved by the mapping, i.e. that the retrieved global instance will not violate the key constraints. For these integrity constraints there will always be legal instances (in their sense), and therefore the integration system is consistent. The rewritten query can be unfolded with the mapping in order to calculate their "certain answers". In [19] an implementation of this method is presented. The complexity of the rewriting is polynomial wrt data complexity.

According to the semantic considered in [17, 19], if a key constraint is not satisfied, then there is no legal instance. This is why in [57] the loosely-sound semantic (in opposition to the previous strictly-sound semantic) is introduced. Now, a database is legal if it is satisfies the integrity constraints and if there is no other database that is *better*. A database is better than another if the portion of the former that is contained in the retrieved global database is greater that the one of the latter. In this way, we have that the inconsistencies wrt foreign key constraints are solved by adding tuples to the retrieved global database, and those wrt key constraints, by deleting a minimal number of tuples from it. The global instances in this case correspond to a subclass of the repairs introduced in [10] for integration systems.

In order to compute the legal instances for the loosely-sound semantic, a Datalog$^\neg$ program under cautious stable model semantics is used. This program calculates a maximal superset of the retrieved global database that satisfies the key constraints. In order to retrieve the certain answers, the query is transformed as defined in [17] and added to that program. This approach works for global relations defined by Datalog queries (and then, GAV is followed). The complexity of retrieving the "certain answers" becomes co-NP-complete.

[7] In several papers, instead of open, clopen and closed, the terms *sound*, *exact* and *complete* are used, resp.

Still under the GAV approach, the results in [57] were extended in [21], considering key constraints and inclusion dependencies, and also queries that are expressed as unions of conjunctive queries. For the strictly-sound semantics two cases are analyzed. In the first case, where only inclusion dependencies (IDs) are considered, the integration system cannot be "inconsistent"; so there is at least one legal database. The rewriting of a query becomes the mapping rules plus the query that is successively unfolded by rules that represent the inclusion dependencies. The second case considers the combination of key dependencies (KDs) and non-key-conflicting IDs (NKC), i.e. IDs where the target (global) relation has no key dependencies or where the target attributes are not a strict superset of the key of the target relation. The rewriting of a query is the same as in the first case plus some rules that enforce that if a global relation violates a KD, then all the tuples are an answer to the query.

For the loosely-sound semantics, the rewriting in [21] is expressed with the same Datalog$^\neg$ program presented in [57]. In order to repair wrt the IDs, this program is coupled with the query rewriting for the case of only IDs and strictly-sound semantics. The data complexity under the strictly-sound semantics for NKC integration systems is PTIME. For loosely-sound semantics, it becomes coNP-complete.

In [32] logic programs for consistent query answering in virtual integration systems are presented. The GAV approach is followed and the global relations can be defined using stratified Datalog$^\neg$ queries. The ICs considered are universal integrity constraints and the queries are expressed in non-recursive Datalog$^\neg$. The specification program is a disjunctive Datalog$^\neg$ program consisting of three hierarchically evaluated modules. The first one uses the mapping and the data sources to compute the "retrieved global database" (as in [19]). The second one enforces the satisfaction of the integrity constraints through repair rules; and the third one corresponds to the query. The structure of each of them depends on the mappings, ICs and query, respectively.

The source of complexity for the program in [32] comes from the second module. In consequence, optimizations are introduced. The optimization process consists of three steps: pruning the rules that are not relevant for computing the answers to the query, next determining and computing the set of facts that need to be repaired, and finally, recombining the repairs in order to compute the answers. The second step decomposes the facts in two sets, those that might be repaired and those that for sure are not going to be repaired. The recombination process presents the repairs in a compact way in order to query them as a relational database. For this, an extra attribute marking each fact is added to each relation. This attribute is a string of zeros and ones. A one (zero) in position $i$ means that the fact is (not) in the repair $i$. The facts for which no repairs are calculated in the second step are marked with '111 . . . 11'. The query needs to be reformulated in order to pose it directly to the marked database. Experiments show that the optimizations significantly improve the performance of the naive and direct techniques.

It seems that the optimizations presented in [32] can be adapted to the logic programs we have presented for CQA.

Finally, we will just mention that there seem to be interesting connections between the area of consistently querying virtual data integration systems and other areas, like querying incomplete databases [66, 44], merging inconsistent theories [63, 5], semantic reconciliation of data [54], schema mapping [71, 28, 70], data exchange [33, 34], and query answering in peer-to-peer systems [55, 52, 53, 36, 12, 24].

**Acknowledgements:** This chapter reports on research funded by DIPUC, CONICYT, FONDECYT, Carleton University Start-Up Grant 9364-01, NSERC Grant 250279-02, CoLogNet. L. Bertossi is Faculty Fellow of the IBM Center for Advanced Studies, Toronto Lab. We are grateful to Jan Chomicki, Alvaro Cortes, Claudio Gutierrez, Alberto Mendelzon, Pablo Barcelo, Alon Halevy, Enrico Franconi, Andrei Lopatenko, Ariel Fuxman, and Giuseppe De Giacomo for collaboration, useful conversations and remarks. Comments received from anonymous referees are highly appreciated.

## References

1. Abiteboul, S.; Hull, R. and Vianu, V. *Foundations of Databases.* Addison-Wesley, 1995.
2. Abiteboul, A. and Duschka, O. Complexity of Answering Queries Using Materialized Views. In *Proc. ACM Symposium on Principles of Database Systems (PODS 98)*, 1998, pp. 254-263.
3. Arenas, M., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. In *Proc. 18th ACM Symposium on Principles of Database Systems (PODS 99)*, 1999, pp. 68–79.
4. Arenas, M., Bertossi, L. and Chomicki, L. Answer Sets for Consistent Query Answering in Inconsistent Databases. *Theory and Practice of Logic Programming*, 2003, 3(4-5): 393-424.
5. Baral, C., Kraus, S., Minker, J. and Subrahmanian, V. S. Combining Knowledge Bases Consisting of First-Order Theories. *Computational Intelligence*,1992, 8:45-71.
6. Barcelo, P. and Bertossi, L. Logic Programs for Querying Inconsistent Databases. In *Proc. International Symposium on Practical Aspects of Declarative Languages (PADL 03)*, Springer LNCS 2562, 2003, pp. 208–222.
7. Barcelo, P., Bertossi, L. and Bravo, L. Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. Chapter in book *Semantics of Databases*, Springer LNCS 2582, 2003, pp. 1–27.
8. Batini, C., Lenzerini, M. and Navathe, S.B. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 1986, 18(4): 323-364.
9. Bergamaschi, S., Castano, S., Vincini, M., and Beneventano, D. Semantic Integration of Heterogeneous Information Sources. *Data and Knowledge Engineering*, 2001, 36(3):215-249.
10. Bertossi, L., Chomicki, J., Cortes, A. and Gutierrez, C. Consistent Answers from Integrated Data Sources. In *Flexible Query Answering Systems*, Springer LNAI 2522, 2002, pp. 71–85.

11. Bertossi, L. and Chomicki, J. Query Answering in Inconsistent Databases. Chapter in book *Logics for Emerging Applications of Databases*, J. Chomicki, G. Saake and R. van der Meyden (eds.), Springer, 2003.
12. Bertossi, L. and Bravo, L. Query Answering in Peer-to-Peer Data Exchange Systems. arXiv.org paper cs.DB/0401015. To appear in *Proc. International Workshop on Peer-to-Peer Computing & DataBases (P2P&DB 04)*, Springer LNCS.
13. Bertossi, L. and Bravo, L. In preparation.
14. Bonatti, P. Reasoning with Infinite Stable Models. *Artificial Intelligence*, 2004, 156(1):75-111.
15. Bravo, L. and Bertossi, L. Logic Programs for Consistently Querying Data Integration Systems. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI 03)*, Morgan Kaufmann, 2003, pp. 10–15.
16. Bravo, L. and Bertossi, L. Disjunctive Deductive Databases for Computing Certain and Consistent Answers to Queries from Mediated Data Integration Systems. To appear in Journal of Applied Logic (extended version of [15])
17. Cali, A., Calvanese, D., De Giacomo, G. and Lenzerini, M. Data Integration Under Integrity Constraints. In *Proc. Conference on Advanced Information Systems Engineering (CAISE 02)*, Springer LNCS 2348, 2002, pp. 262–279.
18. Cali, A., Calvanese, D., De Giacomo, G. and Lenzerini, M. On the Expressive Power of Data Integration Systems. In *Proc. of the International Conference on Conceptual Modeling (ER 02)*, Springer LNCS 2503, 2002, pp. 338–350.
19. Cali, A., Calvanese, D., De Giacomo, G. and Lenzerini, M. On the Role of Integrity Constraints in Data Integration. *IEEE Data Engineering Bulletin*, 2002, 25(3): 39-45.
20. Cali, A., Lembo, D. and Rosati, R. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *Proc. of the ACM Symposium on Principles of Database Systems (PODS 03)*, ACM Press, 2003, pp. 260-271.
21. Cali, A., Lembo, D. and Rosati, R. Query Rewriting and Answering under Constraints in Data Integration Systems. In *Proc. of the International Joint Conference on Artificial Intellience (IJCAI 03)*, Morgan Kaufmann, 2003, pp. 16-21.
22. Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. Y. What is Query Rewriting? In *Proc. of the International Workshop on Knowledge Representation meets Databases (KRDB 00)*, CEUR Electronic Workshop Proceedings, 2000, pp. 17-27.
23. Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. Y. View-based Query Containment. In *Proc. of the ACM Symposium on Principles of Database Systems (PODS 03)*, ACM Press, 2003, pp. 56–67.
24. Calvanese, D., De Giacomo, G., Lenzerini, M. and Rosati, R. Logical Foundations of Peer-To-Peer Data Integration. In *Proc. of the ACM Symposium on Principles of Database Systems (PODS 04)*, ACM Press, 2004, pp. 241-251.
25. Celle, A. and Bertossi, L. Querying Inconsistent Databases: Algorithms and Implementation. In *Computational Logic - CL 2000, Stream: International Conference on Rules and Objects in Databases (DOOD 00)*, Springer LNAI 1861, 2000, pp. 942-956.
26. Chomicki, J. and Marcinkowski, J. Minimal-Change Integrity Maintenance Using Tuple Deletions. arXiv.org paper cs.DB/0212004. To appear in *Information and Computation.*
27. Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. Complexity And Expressive Power Of Logic Programming. *ACM Computer Surveys*, 2001, 33(3):374-425.

28. Doan, A., Domingos, P. and Halevy, A. Learning to Match the Schemas of Data Sources: A Multistrategy Approach. *Machine Learning*, 2003, 50(3): 279-301.
29. Duschka, O. Query Planning and Optimization in Information Integration. PhD Thesis, Stanford University, December 1997.
30. Duschka, O., Genesereth, M. and Levy, A. Recursive Query Plans for Data Integration. *Journal of Logic Programming*, 2000, 43(1):49-73.
31. Eiter, T., Faber, W.; Leone, N. and Pfeifer, G. Declarative Problem-Solving in DLV. Chapter in book *Logic-Based Artificial Intelligence*, J. Minker (ed.), Kluwer, 2000, pp. 79-103.
32. Eiter, T., Fink, M., Greco, G. and Lembo, D. Efficient Evaluation of Logic Programs for Querying Data Integration Systems. In *Proc. International Conference on Logic Programming (ICLP 03)*, Springer LNCS 2916, 2003, pp. 163-177.
33. Fagin, R., Kolaitis, P., Miller, R. and Popa, L. Data Exchange: Semantics and Query Answering. In *Proc. Int. Conf on Database Theory (ICDT 03)*, Springer LNCS 2572, 2003, pp. 207-224.
34. Fagin, R., Kolaitis, P. and Popa, L. Data Exchange: Getting to the Core. In *Proc. of the ACM Symposium on Principles of Database Systems (PODS 03)*, ACM Press, 2003, pp. 90-101.
35. Flesca, S. and Greco, S. Rewriting Queries Using Views. *Transactions on Knowledge and Data Engineering*, 2001, 13(6): 980-995.
36. Franconi, E., Kuper, G., Lopatenko, L., Serafini, L. A Robust Logical and Computational Characterisation of Peer-to-Peer Database Systems. In *Proc. International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 03)*, Springer LNCS 2944, 2004, pp. 64-76.
37. Friedman, M., Levy, A. and Millstein, T. Navigational Plans for Data Integration. In *Proc. National Conference on Artificial Intelligence (AAAI 99)*, AAAI Press, 1999, pp. 67-73.
38. Fuxman, A. and Miller, R.J. Towards Inconsistency Management in Data Integration Systems. In *Proceedings of the IJCAI-03 Workshop on Information Integration on the Web.*
39. Giannotti, F., Pedreschi, D., Sacca, D. and Zaniolo, C. Non-Determinism in Deductive Databases. In *Proc. International Conference on Deductive and Object-Oriented Databases (DOOD 91)*, Springer LNCS 566, 1991, pp. 129–146.
40. Gelfond, M. and Lifschitz, V. The Stable Model Semantics for Logic Programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium (ICLP/SLP 88)*, MIT Press, 1988, pp. 1070-1080.
41. Gelfond, M. and Lifschitz, V. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 1991, 9:365–385.
42. Gelfond, M. and Leone, N. Logic Programming and Knowledge Representation - The A-Prolog Perspective. *Artificial Intelligence*, 2002, 138(1-2):3-38.
43. Grahne, G. and Mendelzon, A. Tableau Techniques for Querying Information Sources through Global Schemas. In *Proc. of the International Conference on Database Theory (ICDT 99)*, Springer LNCS 1540, 1999, pp. 332–347.
44. Grahne, G. Information Integration and Incomplete Information. *IEEE Computer Society Bulletin on Data Engineering*, September 2002, pp. 46-52.
45. Grant, J. and Minker, M. A Logic-based Approach to Data Integration. *Theory and Practice of Logic Programming*, 2002, 2(3):323-368.
46. Greco, S. Binding Propagation Techniques for the Optimization of Bound Disjunctive Queries. *IEEE Transactions on Knowledge and Data Engineering*, 2003, 15(2):368-385.

47. Greco, G., Greco, S. and Zumpano, E. A Logical Framework for Querying and Repairing Inconsistent Databases. *IEEE Transactions on Knowledge and Data Engineering*, 2003, 15(6):1389-1408.
48. Gryz, J. Query Rewriting Using Views in the Presence of Functional and Inclusion Dependencies. *Information Systems*, 1999, 24(7):597–612.
49. Gupta, A. and Singh Mumick, I. (eds.) *Materialized Views: Techniques, Implementations, and Applications*. MIT Press, 1999.
50. Halevy, A.Y. Theory of Answering Queries Using Views. *SIGMOD Record*, 2000, 29(4) 40-47.
51. Halevy, A.Y. Answering Queries Using Views: A Survey. *VLDB Journal*, 2001, 10(4): 270-294.
52. Halevy, A., Ives, Z., Suciu, D. and Tatarinov, I. Schema Mediation in Peer Data Management Systems. In *Proc. of the International Conference on Data Engineering (ICDE 03)*, IEEE Computer Society, 2003, pp. 505-518.
53. Halevy, A.Y. Corpus-Based Knowledge Representation. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI 03)*, Morgan Kaufmann, 2003, pp. 1567-1572.
54. Hull, R. Managing Semantic Heterogeneity in Databases: A Theoretical Perspective. In *Proc. of the ACM Symposium on Principles of Database Systems (PODS 97)*, ACM Press, 1997, pp. 51-61.
55. Kementsietsidis, A., Arenas, M. and Miller, R.J. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *Proc. of the ACM International Conference on Management of Data (SIGMOD 03)*, ACM Press, 2003, pp. 325-336.
56. Kolaitis, Ph. and Vardi, M. Conjunctive-Query Containment and Constraint Satisfaction. *J. Computer and Systems Sciences*, 2000, 61(2): 302-332.
57. Lembo, D., Lenzerini, M. and Rosati, R. Source Inconsistency and Incompleteness in Data Integration. In *Proc. International Workshop Knowledge Representation meets Databases (KRDB 02)*, CEUR Electronic Workshop Proceedings, 2002.
58. Lenzerini, M. Data Integration: A Theoretical Perspective. In *Proc. ACM Symposium on Principles of Database Systems (PODS 02)*, ACM Press, 2002, pp. 233-246.
59. Leone, N. et al. The DLV System for Konwledge Representation and Reasoning. arXiv.org paper cs.LO/0211004. To appear in *ACM Transactions on Computational Logic*.
60. Levy, A.Y., Mendelzon, A., Sagiv, Y. and Srivastava, D. Answering Queries Using Views. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS 95)*, ACM Press, 1995, pp. 95-104.
61. Levy, A., Rajaraman, A. and Ordille, J. Querying Heterogeneous Information Sources using Source Descriptions. In *Proc. International Conference on Very Large Databases (VLDB 96)*, Morgan Kaufmann, 1996, pp. 251–262.
62. Levy, A. Logic-Based Techniques in Data Integration. Chapter in *Logic Based Artificial Intelligence*, J. Minker (ed.), Kluwer Publishers, 2000.
63. Lin, J. and Mendelzon, A. Merging Databases under Constraints. *International Journal of Cooperative Information Systems*, 1996, 7(1):55-76.
64. Lloyd, J.W. *Foundations of Logic Programming*. Second ed., Springer-Verlag, 1987.
65. McBrien, P. and Poulovassilis, A. Data Integration by Bi-Directional Schema Transformation Rules. In *Proc. International Conference on Data Engineering (ICDE 03)*, IEEE Computer Society, 2003, pp. 227–238.

66. Meyden, R.v.d. Logical Approaches to Incomplete Information: A Survey. Chapter in *Logics for Databases and Information Systems*, J.Chomicki and G. Saake (eds.), Kluwer, 1998, pp. 307-356.
67. Millstein, T., Halevy, A. and Friedman, M. Query Containment for Data Integration Systems. *Journal of Computer and Systems Sciences*, 2003, 66(1): 20-39.
68. Motro A. Superviews: Virtual Integration of Multiple Databases. *IEEE Transactions on Software Engineering*, 1987, 13(7):785–798.
69. Motro A. Multiplex: A Formal Model for Multidatabases and Its Implementation. In *Proc. International Workshop on Next Generation Information Technology and Systems*, Springer LNCS 1649, 1999, pp. 138–158.
70. Pottinger, R., and Bernstein, Ph. Creating a Mediated Schema Based on Initial Correspondences. *IEEE Data Engineering Bulletin*, 2002, 25(3): 26-31.
71. Rahm, E. and Bernstein, Ph.A. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 2001, 10:334-350.
72. Ullman, J.D. *Principles of Database and Knowledge-Base Systems.* Computer Science Press, 1988.
73. Ullman, J.D. Information Integration Using Logical Views. *Theoretical Computer Science*, 2000, 239(2): 189-210.
74. Wei, F. and Lausen, G. Containment of Conjunctive Queries with Safe Negation. In *Proc. International Conference of Database Theory (ICDT 03)*, Springer LNCS 2572, 2003, pp. 346-360
75. Wiederhold, G. and Genesereth, M. The Conceptual Basis for Mediation Services. *IEEE Expert*, 1997, 12(5): 38-47.