Logic Programs for Consistently Querying Data Integration Systems

Loreto Bravo
Pontificia Universidad Católica de Chile
Departamento de Ciencia de Computación
Santiago, Chile.
lbravo@ing.puc.cl

Leopoldo Bertossi
Carleton University
School of Computer Science
Ottawa, Canada
bertossi@scs.carleton.ca

Abstract

We solve the problem of obtaining answers to queries posed to a mediated integration system under the local-as-view paradigm that are consistent wrt to certain global integrity constraints. For this, the query program is combined with logic programming specifications under the stable model semantics of the class of minimal global instances, and of the class of their repairs.

1 Introduction

For several reasons a database may become inconsistent wrt certain integrity constraints (ICs). In such a situation, possibly most of the data is still consistent. In [Arenas *et al.*, 1999] consistent data in a single relational database is characterized as the data that is invariant under all minimal restorations of consistency, i.e. true in all repaired versions of the original instance (the *repairs*). In that paper and others [Arenas *et al.*, 2000; Greco *et al.*, 2001], some mechanisms have been developed for retrieving consistent answer when queries are posed to such an inconsistent database.

When independent data sources are integrated, inconsistencies wrt to global ICs are likely to occur, specially when the sources are virtually integrated by means of a mediator, because the data sources are kept completely independent. The mediator provides a global schema as an interface, and is responsible for generating query plans to answer global queries by retrieving data sets from the sources and combining them into a final answer set for the user.

"Local-as-view" (LAV) is a common paradigm for data integration that describes each data source as a set of views over the global schema. Another one, "global-as-view" (GAV), defines every global relation as a view of the set of relations in the sources ([Lenzerini, 2002] is a good survey). Query answering is harder in LAV [Abiteboul *et al.*, 1998]. On the other side, the LAV approach allows more flexibility when new sources are integrated into an existing system. However, the flexibility to add new sources, without having to consider the other sources in the system, makes inconsistencies wrt global ICs more likely.

Example 1 *Consider the LAV based global integration system $\mathfrak{G}_1$ with a global relation $R(X, Y)$ and two source relations $v_1 = \{V_1(a, b), V_1(c, d)\}$ and $v_2 = \{V_2(a, c), V_2(d, e)\}$ that are described by the view definitions $V_1(X, Y) \leftarrow R(X, Y)$; $V_2(X, Y) \leftarrow R(X, Y)$. The global functional dependency (FD) $R\colon X \rightarrow Y$ is violated through the pair of tuples $\{(a, b), (a, c)\}$.* □

In a virtual integration system, the mediator should solve potential inconsistencies when the query plan is generated. In [Bertossi *et al.*, 2002], under the LAV approach, a methodology for generating query plans to compute answers to limited forms of queries that are consistent wrt an also restricted class of universal ICs was presented. The limitation comes from a first step where a query transformation for consistent query answering (CQA) is performed [Arenas *et al.*, 1999]. Next, query plans are generated for the transformed query. However, [Bertossi *et al.*, 2002] provides the right semantics for CQA in mediated integrated systems (see Section 2).

Example 2 *(example 1 continued) If we pose to the global system the query $Q : Ans(X, Y) \leftarrow R(X, Y)$, we obtain the answers $\{Ans(a, b), Ans(c, d), Ans(a, c), Ans(d, e)\}$. However, only the tuples $Ans(c, d), Ans(d, e)$ should be returned as consistent answers wrt the FD $R\colon X \rightarrow Y$.* □

In this paper, under the LAV approach and assuming that sources are open (or incomplete) [Abiteboul *et al.*, 1998], we solve the problem of retrieving consistent answers to global queries. We consider arbitrary universal ICs and referential ICs, in consequence all the ICs that are used in database praxis [Abiteboul *et al.*, 1995]. View definitions are conjunctive queries, and global queries are expressed in Datalog and its extensions with negation. The methodology can be summarized as follows. First, in Section 3, the minimal legal global instances of a mediated system are specified by means of logic programs with a stable model, or answer sets, semantics. Next, in Section 4, the repairs of the minimal global instances are specified as the stable models of disjunctive logic programs. Those programs contain annotation constants, like those used to specify repairs of single relational databases for CQA [Barcelo *et al.*, 2002]. Finally, in Section 5, consistent answers to queries are obtained by running a query program in combination with the previous two specification programs.

2 Preliminaries

2.1 Global schemas and view definitions

A *global schema* $\mathcal{R}$ is modeled by a finite set of relations $\{R_1, R_2, ..., R_n\}$ over a fixed domain $\mathcal{U}$. With these relation

symbols and the elements of $\mathcal{U}$ treated as constants, a first-order language $\mathcal{L}(\mathcal{R})$ can be defined. This language can be extended with new defined predicates and built-ins. In particular, we will extend the global schema with a *local schema* $\mathcal{S}$, i.e. a finite set of new view predicates $V_1, V_2, ...$, that will be used to describe the relations in the local sources.

Each *view*, denoted by, say a new predicate V, is defined by means of conjunctive query [Abiteboul *et al.*, 1995], i.e. an $\mathcal{L}(\mathcal{R} \cup \mathcal{S})$-formula of the form φ_V: $V(\bar{t}) \leftarrow body(\varphi_V)$, where $\bar{t}$ is a tuple containing variables and/or constants, and $body(\varphi_V)$ is a conjunction of $\mathcal{R}$-atoms.

A *database instance* D over schema $\mathcal{R}$ can be considered as a first-order structure with domain $\mathcal{U}$, where the extensions of the relations R_i are finite. (The extensions of built-in predicates may be infinite, but fixed.) A global *integrity constraint* (IC) is an $\mathcal{L}(\mathcal{R})$-sentence ψ. An instance D satisfies ψ, denoted $D \models \psi$, if ψ is true in D.

Given a database instance D over schema $\mathcal{R}$, and a view definition φ_V, $\varphi_V(D)$ denotes the extension of V obtained by applying the definition φ_V to D. If the view already has an extension v (corresponding to the contents of a data source), it is possible that v is incomplete and stores only some of the tuples in $\varphi_V(D)$; and we say the view extension v is *open* wrt D [Abiteboul *et al.*, 1998]. Most mechanisms for deriving query plans assume that sources are open, e.g. [Duschka *et al.*, 2000].

A *source* S is a pair $\langle \varphi, v \rangle$, where φ is the view definition, and v is an extension for φ. An *open global system* $\mathfrak{G}$ is a finite set of open sources. The global schema $\mathcal{R}$ consists of the relation names that do not have a definition in the global system. The underlying domain $\mathcal{U}$ for $\mathcal{R}$ is a proper superset of the "active" domain. The latter consists of all the constants appearing in the view extensions v_i of the sources, and in their definitions. A global system $\mathfrak{G}$ defines a set of legal global instances [Lenzerini, 2002].

Definition 1 *Given an open global system* $\mathfrak{G} = \{\langle \varphi_1, v_1 \rangle, \ldots, \langle \varphi_n, v_n \rangle\}$*, the set of legal global instances is* $Linst(\mathfrak{G}) = \{D \text{ instance over } \mathcal{R} \mid v_i \subseteq \varphi_i(D),\ i = 1, \ldots, n\}$. □

Example 3 *(example 2 continued) Let us denote by* φ_1, φ_2 *the view definitions of* V_1, V_2, *resp. in* $\mathfrak{G}_1$. $D = \{R(a, b), R(c, d), R(a, c), R(d, e)\}$ *is a legal global instance, because* $v_1 = \{V_1(a, b), V_1(c, d)\} \subseteq \varphi_1(D) = \{V_1(a, b), V_1(c, d), V_1(a, c), V_1(d, e)\}$ *and* $v_2 = \{V_2(a, c), V_2(d, e)\} \subseteq \varphi_2(D) = \{V_2(a, b), V_2(c, d), V_2(a, c), V_2(d, e)\}$. *Supersets of* D *are also legal instances.* □

The semantics of query answers in mediated integration systems is given by the notion of *certain answer*. In this paper we will consider queries expressed in Datalog and its extensions with negation.

Definition 2 [Abiteboul *et al.*, 1998] *Given an open global system* $\mathfrak{G}$ *and a query* $Q(\bar{X})$ *to the system, a tuple* $\bar{t}$ *is a* certain answer *to* Q *in* $\mathfrak{G}$ *if for every global instance* $D \in Linst(\mathfrak{G})$*, it holds* $D \models Q[\bar{t}]$. *We denote with* $Certain_{\mathfrak{G}}(Q)$ *the set of certain answers to* Q *in* $\mathfrak{G}$. □

The inverse-rules algorithm [Duschka *et al.*, 2000] for generating query plans under the LAV approach assumes that sources are open and each source relation V is defined as a conjunctive view over the global schema: $V(\bar{X}) \leftarrow P_1(\bar{X}_1), \ldots, P_n(\bar{X}_n)$, with $\bar{X} \subseteq \bigcup_i \bar{X}_i$.

Then, for $j = 1, \ldots n$, $P_j(\bar{X}'_j) \leftarrow V(\bar{X})$ is an inverse rule for P_j. The tuple $\bar{X}_j$ is transformed to obtain the tuple $\bar{X}'_j$ as follows: if $X \in \bar{X}_j$ is a constant or is a variable appearing in $\bar{X}$, then X is unchanged in $\bar{X}'_j$. Otherwise, X is a variable X_i that does not appear in $\bar{X}$, and it is replaced by the term $f_i(\bar{X})$, where f_i is a fresh Skolem function. We denote the set of inverse rules of the collection $\mathcal{V}$ of source descriptions in $\mathfrak{G}$ by $\mathcal{V}^{-1}$.

Example 4 *Consider the mediated data integration system* $\mathfrak{G}_3$ *with global schema* $\mathcal{R} = \{P, R\}$. *The set* $\mathcal{V}$ *of local view definitions consists of* $V_1(X, Z) \leftarrow P(X, Y), R(Y, Z)$, *and* $V_2(X, Y) \leftarrow P(X, Y)$. *The set* $\mathcal{V}^{-1}$ *consists of the rules* $P(X, f(X, Z)) \leftarrow V_1(X, Z)$; $R(f(X, Z), Z) \leftarrow V_1(X, Z)$; *and* $P(X, Y) \leftarrow V_2(X, Y)$. □

The inverse rules are then used to answer queries expressed in terms of the global relations, that now have definitions in terms of the sources. The answers obtained with the inverse rule algorithm are maximally contained in the queries [Duschka *et al.*, 2000], and coincide with the certain answers [Abiteboul *et al.*, 1998].

2.2 Global systems and consistency

We assume that we have a set of global integrity constraints IC that is consistent as a set of logical sentences, and *generic*, in the sense that it does not entail any ground database literal. ICs used in database praxis are always generic.

Definition 3 [Bertossi *et al.*, 2002] *(a) Given a global system,* $\mathfrak{G}$*, an instance* D *is* minimal *if* $D \in Linst(\mathfrak{G})$ *and is minimal wrt set inclusion, i.e. there is no other instance in* $Linst(\mathfrak{G})$ *that is a proper subset of* D *(as a set of atoms). We denote by* $Mininst(\mathfrak{G})$ *the set of minimal legal global instances of* $\mathfrak{G}$ *wrt set inclusion. (b) A global system* $\mathfrak{G}$ *is* consistent *wrt IC, if for all* $D \in Mininst(\mathfrak{G})$, $D \models IC$. □

Example 5 *(example 4 continued) Assume that* $\mathfrak{G}_3$ *has the source contents* $v_1 = \{V_1(a, b)\}$, $v_2 = \{V_2(a, c)\}$, *resp. and that* $\mathcal{U} = \{a, b, c, u\}$. *Then, the elements of* $Mininst(\mathfrak{G}_3)$ *are of the form* $D_z = \{P(a, z), R(z, b), P(a, c)\}$ *for some* $z \in \mathcal{U}$ *. The global FD* $P(X, Y)$: $X \rightarrow Y$ *is violated exactly in those minimal legal instances* D_z *for which* $z \neq c$. *Thus,* $\mathfrak{G}_3$ *is inconsistent.* □

Definition 4 *The ground tuple* $\bar{a}$ *is a* minimal answer *to a query* Q *posed to* $\mathfrak{G}$ *if for every* $D \in Mininst(\mathfrak{G})$, $\bar{a} \in Q(D)$, *where* $Q(D)$ *is the answer set for* Q *in* D. *The set of minimal answers is denoted by* $Minimal_{\mathfrak{G}}(Q)$. □

Clearly $Certain_{\mathfrak{G}}(Q) \subseteq Minimal_{\mathfrak{G}}(Q)$. For monotone queries [Abiteboul *et al.*, 1995], the two notions coincide [Bertossi *et al.*, 2002]. Nevertheless, in Example 5 the query $Ans(X, Y) \leftarrow \neg P(X, Y)$ has (b, a) as a minimal answer, but not as a certain answer, because there are legal instances that contain $P(b, a)$. Since consistency was defined wrt minimal global instances, the notion of minimal answer is particularly relevant.

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

Definition 5 [Arenas *et al.*, 1999] *(a) Let D, D' be database instances over the same schema and domain. The distance, $\Delta(D, D')$, between D and D' is the symmetric difference $\Delta(D, D') = (\Sigma(D) \setminus \Sigma(D')) \cup (\Sigma(D') \setminus \Sigma(D))$. (b) For database instances D, D', D'', we define $D' \leq_D D''$ if $\Delta(D, D') \subseteq \Delta(D, D'')$.* □

Definition 6 *(based on* [Arenas *et al.*, 1999]*) Let $\mathfrak{G}$ be a global system and IC a set of global ICs. A* repair *of $\mathfrak{G}$ wrt IC is a global database instance D', such that $D' \models IC$ and D' is $\leq_D$-minimal for some $D \in Mininst(\mathfrak{G})$.* □

Thus, a repair of a global system is a global database instance that minimally differs from a minimal legal global database instance. If $\mathfrak{G}$ is already consistent, then the repairs are the elements of $Mininst(\mathfrak{G})$. In Definition 6 we are not requiring that a repair respects the property of the sources of being open, i.e. that the extension of each view in the repair contains the corresponding view extension in the source. Thus, it may be the case that a repair – still a global instance – does not belong to $Linst(\mathfrak{G})$. If we do not allow this, a global system might not be repairable. Repairs are used as an auxiliary concept to define the notion of consistent answer.

Example 6 *(example 1 continued) The only element in $Mininst(\mathfrak{G}_1)$ is $D_0 = \{R(a, b), R(c, d), R(a, c), R(d, e)\}$, that does not satisfy IC. Then, $\mathfrak{G}_1$ is inconsistent. The repairs are the global instances that minimally differ from D_0 and satisfy the FD, namely $D_0^1 = \{R(a, b), R(c, d), R(d, e)\}$ and $D_0^2 = \{R(a, c), R(c, d), R(d, e)\}$. Notice that they do not belong to $Linst(\mathfrak{G}_1)$.* □

Definition 7 [Bertossi *et al.*, 2002] *(a) Given a global system $\mathfrak{G}$, a set of global integrity constraints IC, and a global first-order query $Q(\bar{X})$, we say that a (ground) tuple $\bar{t}$ is a consistent answer to Q wrt IC iff for every repair D of $\mathfrak{G}$, $D \models Q[\bar{t}]$. (b) We denote by $Consis_{\mathfrak{G}}(Q)$ the set of consistent answers to Q in $\mathfrak{G}$.* □

Example 7 *(example 6 continued) For the query $Q_1(X)$: $\exists Y$ $R(X, Y)$, a is a consistent answer. $Q_2(X, Y)$: $R(X, Y)$ has $(c, d), (d, e)$ as consistent answers.* □

If $\mathfrak{G}$ is consistent wrt IC, then $Consis_{\mathfrak{G}}(Q) = Minimal_{\mathfrak{G}}(Q)$. Furthermore, if the ICs are generic, then for any $\mathfrak{G}$ it holds $Consis_{\mathfrak{G}}(Q) \subseteq Minimal_{\mathfrak{G}}(Q)$ [Bertossi *et al.*, 2002]. Notice also that the notion of consistent answer can be applied to queries written in Datalog or its extensions with built-ins and stratified negation.

3 Specification of Legal Instances

The specification of the class $Mininst(\mathfrak{G})$ for system $\mathfrak{G}$ is given using normal logic programs, whose rules are inspired by the inverse-rules algorithm. They use auxiliary predicates instead of function symbols, but their functionality is enforced using the *choice predicate* [Giannotti *et al.*, 1991].

Definition 8 *Given an open global system $\mathfrak{G}$, the program, $\Pi(\mathfrak{G})$, contains the following clauses:*

1. Fact $dom(a)$ for every constant $a \in \mathcal{U}$; and the fact $V(\bar{a})$ whenever $V(\bar{a}) \in v_i$ for some source extension v_i in $\mathfrak{G}$.

2. For every view (source) predicate V in the system with description $V(\bar{X}) \leftarrow P_1(\bar{X}_1), \ldots, P_n(\bar{X}_n)$, the rules $P_j(\bar{X}_j) \leftarrow V(\bar{X}),\ \bigwedge_{X_i \in (\bar{X}_j \setminus \bar{X})} F_i(\bar{X}, X_i),\quad j = 1, \ldots n.$

3. For every predicate $F_i(\bar{X}, X_i)$ introduced in 2., the rule $F_i(\bar{X}, X_i) \leftarrow V(\bar{X}), dom(X_i), choice((\bar{X}), (X_i))$. □

The predicate $F_i(\bar{X}, X_i)$ is replacing the Skolem function $f_i(\bar{X}) = X_i$ introduced in Section 2.1, and, through the choice predicate, it assigns values in the domain to the variables in the head of the rule that are not in $\bar{X}$. There is a new Skolem predicate for each pair formed by a description rule as in item 2. above and a different existentially quantified variable in it. The predicate $choice((\bar{X}), (X_i))$ ensures that for every (tuple of) value(s) for $\bar{X}$, only one (tuple of) value(s) for X_i is non deterministically chosen between the constants of the active domain.

Example 8 *(examples 4 and 5 continued) Program $\Pi(\mathfrak{G}_3)$ contains the following rules:*

1. $dom(a).\ dom(b).\ dom(c).\ dom(u).\ \ V_1(a, b).\ V_2(a, c).$

2. $P(X, Z) \leftarrow V_1(X, Y), F_1(X, Y, Z).$
$R(Z, Y) \leftarrow V_1(X, Y), F_1(X, Y, Z).$
$P(X, Y) \leftarrow V_2(X, Y).$

3. $F_1(X, Y, Z) \leftarrow V_1(X, Y), dom(Z), choice((X, Y), (Z)).$ □

For every program Π with the choice operator, there is its *stable version*, $SV(\Pi)$, whose stable models correspond to the so-called *choice models* of Π [Giannotti *et al.*, 1991]. The program $SV(\Pi)$ is obtained as follows: (a) Each choice rule $r:\ H \leftarrow B, choice((\bar{X}), (Y))$ in Π is replaced by the rule $H \leftarrow B, chosen_r(\bar{X}, Y)$. (b) For each rule as in (a), the following rules are added

$$chosen_r(\bar{X}, Y) \leftarrow B, not\ diffChoice_r(\bar{X}, Y).$$
$$diffChoice_r(\bar{X}, Y) \leftarrow chosen_r(\bar{X}, Y'), Y \neq Y'.$$

The rules defined in (b) ensure that, for every tuple $\bar{X}$ where B is satisfied, the predicate $chosen_r(\bar{X}, Y)$ satisfies the functional dependency $\bar{X} \rightarrow Y$. In (a) the *choice* operator is replaced by the new predicate $chosen_r$ that forces the expected functional dependency.

Example 9 *(example 8 continued) Program $SV(\Pi(\mathfrak{G}_3))$ contains the following rules:*

1. $dom(a).\ dom(b).\ dom(c).\ dom(u).\ \ V_1(a, b).\ V_2(a, c).$

2. $P(X, Z) \leftarrow V_1(X, Y), F_1(X, Y, Z).$
$R(Z, Y) \leftarrow V_1(X, Y), F_1(X, Y, Z).$
$P(X, Y) \leftarrow V_2(X, Y).$

3. $F_1(X, Y, Z) \leftarrow V_1(X, Y), dom(Z), chosen_1(X, Y, Z).$

4. $chosen_1(X, Y, Z) \leftarrow V_1(X, Y), dom(Z), not\ diffChoice_1(X, Y, Z).$
$diffChoice_1(X, Y, Z) \leftarrow chosen_1(X, Y, Z'), dom(Z), Z' \neq Z.$

Its stable models are:

$\mathcal{M}_1 = \{dom(a), dom(b), dom(c), dom(u), V_1(a, b), V_2(a, c), P(a, c), diffChoice_1(a, b, a), chosen_1(a, b, b), diffChoice_1(a, b, c), diffChoice_1(a, b, u), F_1(a, b, b), R(b, b), P(a, b)\}$

$\mathcal{M}_2 = \{dom(a), dom(b), dom(c), dom(u), V_1(a, b), V_2(a, c), P(a, c), chosen_1(a, b, a), diffChoice_1(a, b, b), diffChoice_1(a, b, c), diffChoice_1(a, b, u), F_1(a, b, a), R(a, b), P(a, a)\}$

$\mathcal{M}_3 = \{dom(a), dom(b), dom(c), dom(u), V_1(a,b), V_2(a,c), P(a,c), diffChoice_1(a,b,a), diffChoice_1(a,b,b), chosen_1(a,b,c), diffChoice_1(a,b,u), F_1(a,b,c), R(c,b)\}$
$\mathcal{M}_4 = \{domd(a), domd(b), domd(c), domd(u), V_1(a,b), V_2(a,c), P(a,c), diffChoice_1(a,b,a), diffChoice_1(a,b,b), diffChoice_1(a,b,c), chosen_1(a,b,u), F_1(a,b,u), R(u,b), P(a,u)\}$. □

Definition 9 *The instance associated to a stable model $\mathcal{M}$ of $\Pi(\mathfrak{G})$ is $D_{\mathcal{M}} = \{P(\bar{a}) \mid P \in \mathcal{R} \text{ and } P(\bar{a}) \in \mathcal{M}\}$.* □

Example 10 *(example 9 continued) $D_{\mathcal{M}_1}$, $D_{\mathcal{M}_2}$, $D_{\mathcal{M}_3}$, $D_{\mathcal{M}_4}$ are the elements of $Mininst(\mathfrak{G}_3)$, namely $\{P(a,b), R(b,b), P(a,c)\}$, $\{P(a,a), R(a,b), P(a,c)\}$, $\{P(a,c), R(c,b)\}$, $\{P(a,u), R(u,b), P(a,c)\}$, respectively.* □

Theorem 1 *If $\mathcal{M}$ is a stable model of program $SV(\Pi(\mathfrak{G}))$, then $D_{\mathcal{M}} \in Mininst(\mathfrak{G})$. Furthermore, the minimal legal instances obtained in this way are all the minimal legal instances of $\mathfrak{G}$.* □

The program $\Pi(\mathfrak{G})$ (or its stable version) can be used to compute $Minimal_{\mathfrak{G}}(Q)$, where Q is a query expressed as a, say Datalog^{not} program $\Pi(Q)$. This can be done by running the combined program under the skeptical or cautious stable stable model semantics, that sanctions as true wrt the program what is true in all choice (resp. stable) models. If the query Q is monotone, e.g. a Datalog query, then in the same way $Certain_{\mathfrak{G}}(Q)$ can be computed.

4 Specification of Repairs of a Global System

In [Barcelo *et al.*, 2002] repairs of single relational databases are specified using disjunctive logic programs with stable model semantics. The approach works for arbitrary universal and referential ICs in the sense that the repairs of the database correspond to the stable models of the program. We briefly explain these programs, because they will be used to specify repairs of integration systems.

First, the database predicates in the program are expanded with an extra argument to be filled with one of a set of new annotation constants. An atom in (outside) the original database is annotated with $\mathbf{t_d}$ ($\mathbf{f_d}$). Annotations $\mathbf{t_a}$ and $\mathbf{f_a}$ are considered *advisory* values, to solve conflicts between the database and the ICs. If an atom gets the derived annotation $\mathbf{f_a}$, it means an advise to make it false, i.e. to delete it from the database. Similarly, an atom that gets the annotation $\mathbf{t_a}$ must be inserted into the database.

Example 11 *Consider the integrity constraint $\forall x(P(x) \rightarrow R(x))$, and the inconsistent database instance $r = \{P(a)\}$. The logic program should have the effect of repairing the database. Single, local repair steps are obtained by deriving the annotations $\mathbf{t_a}$ or $\mathbf{f_a}$. This is done when each IC is considered in isolation, but there may be interacting ICs, and the repair process may take several steps and should stabilize at some point. In order to achieve this, we use annotations $\mathbf{t}^\star$, $\mathbf{f}^\star$. The latter, for example, groups together the annotations $\mathbf{f_d}$ and $\mathbf{f_a}$ for the same atom (rules 1. and 4. below). These derived annotations are used to give a feedback to the bodies of the rules that produce the local, single repair steps, so that a propagation of changes is triggered (rule 2. below). The annotations $\mathbf{t}^{\star\star}$ and $\mathbf{f}^{\star\star}$ are just used to read off the literals that are inside (resp. outside) a repair. This is achieved by means of rules 6. below, that are used to interpret the models as database repairs. The following is the program:*

1. $P(x, \mathbf{f}^\star) \leftarrow P(x, \mathbf{f_a})$. $\quad P(x, \mathbf{t}^\star) \leftarrow P(x, \mathbf{t_a})$. $\quad P(x, \mathbf{t}^\star) \leftarrow P(x, \mathbf{t_d})$. *(similarly for R)*
2. $P(x, \mathbf{f_a}) \vee R(x, \mathbf{t_a}) \leftarrow P(x, \mathbf{t}^\star), R(x, \mathbf{f}^\star)$.
3. $P(a, \mathbf{t_d}) \leftarrow$.
4. $P(x, \mathbf{f}^\star) \leftarrow not\ P(x, \mathbf{t_d})$. $\quad R(x, \mathbf{f}^\star) \leftarrow not\ R(x, \mathbf{t_d})$.
5. $\leftarrow P(\bar{x}, \mathbf{t_a}), P(\bar{x}, \mathbf{f_a})$. $\quad \leftarrow R(\bar{x}, \mathbf{t_a}), R(\bar{x}, \mathbf{f_a})$.
6. $P(x, \mathbf{t}^{\star\star}) \leftarrow P(x, \mathbf{t_a})$. $\quad P(x, \mathbf{f}^{\star\star}) \leftarrow P(x, \mathbf{f_a})$. $P(x, \mathbf{t}^{\star\star}) \leftarrow P(x, \mathbf{t_d}),\ not\ P(x, \mathbf{f_a})$. $P(x, \mathbf{f}^{\star\star}) \leftarrow not\ P(x, \mathbf{t_d}),\ not\ P(x, \mathbf{t_a})$. *(similarly for R)*

Only rules 2. depend on the ICs. They say how to repair them when violations are found. Rules 3. contain the database atoms. Rules 4. capture the CWA. Rules 5. are denial program constraints to discard models that contain an atom annotated with both $\mathbf{t_a}$ and $\mathbf{f_a}$. The program has two stable models: $\{P(a, \mathbf{t_d}), P(a, \mathbf{t}^\star), R(a, \mathbf{f}^\star), R(a, \mathbf{t_a}), \underline{P(a, \mathbf{t}^{\star\star})}, R(a, \mathbf{t}^\star), \underline{R(a, \mathbf{t}^{\star\star})}\}$, and $\{P(a, \mathbf{t_d}), P(a, \mathbf{t}^\star), \underline{P(a, \mathbf{f}^\star)}, R(a, \mathbf{f}^\star), \underline{P(a, \mathbf{f}^{\star\star})}, \underline{R(a, \mathbf{f}^{\star\star})}, P(a, \mathbf{f_a})\}$, the first one saying (look at underlined atoms) that $R(a)$ is inserted into the database; the second one, that $P(a)$ is deleted. □

The next definition combines into one program the specification of the minimal legal instances and their repairs.

Definition 10 *The* repair program*, $\Pi(\mathfrak{G}, IC)$, of $\mathfrak{G}$ wrt IC contains the following clauses:*

1. *Facts as in 1. in Definition 8.*
2. *Each of the rules 2. in Definition 8 is replaced by $P_j(\bar{X}_j, \mathbf{t_d}) \leftarrow V(\bar{X}),\ \bigwedge_{X_i \in (\bar{X}_j \setminus \bar{X})} F_i(\bar{X}, X_i)$.*
3. *Exactly the same rules as in 3. in Definition 8.*
4. *For every predicate $P \in \mathcal{R}$, the clauses*
   $P(\bar{X}, \mathbf{t}^\star) \leftarrow P(\bar{X}, \mathbf{t_d}),\ dom(\bar{X})$.[1]
   $P(\bar{X}, \mathbf{t}^\star) \leftarrow P(\bar{X}, \mathbf{t_a}),\ dom(\bar{X})$.
   $P(\bar{X}, \mathbf{f}^\star) \leftarrow P(\bar{X}, \mathbf{f_a}),\ dom(\bar{X})$.
   $P(\bar{X}, \mathbf{f}^\star) \leftarrow dom(\bar{X}),\ not\ P(\bar{X}, \mathbf{t_d})$.
5. *For every first-order global universal IC of the form $\overline{\forall}(Q_1(\bar{y}_1) \vee \cdots \vee Q_n(\bar{Y}_n) \leftarrow P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m) \wedge \varphi)$, where $P_i, Q_j \in \mathcal{R}$, and φ is a conjunction of built-in atoms, the clause*
   $\bigvee_{i=1}^n P_i(\bar{X}_i, \mathbf{f_a}) \bigvee_{j=1}^m Q_j(\bar{Y}_j, \mathbf{t_a}) \leftarrow \bigwedge_{i=1}^n P_i(\bar{X}_i, \mathbf{t}^\star), \bigwedge_{j=1}^m Q_j(\bar{Y}_j, \mathbf{f}^\star), dom(\bar{X}),\ \varphi$;
   where $\bar{X}$ is the tuple of all variables appearing in database atoms in the rule.
6. *For every referential IC of the form $\forall \bar{x}(P(\bar{x}) \rightarrow \exists y Q(\bar{x}', y))$, with $\bar{x}' \subseteq \bar{x}$, the clauses*
   $P(\bar{X}, \mathbf{f_a}) \vee Q(\bar{X}', null, \mathbf{t_a}) \leftarrow P(\bar{X}, \mathbf{t}^\star), not\ aux(\bar{X}'), not\ Q(\bar{X}', null, \mathbf{t_d}), dom(\bar{X})$.
   $aux(\bar{X}') \leftarrow Q(\bar{X}', Y, \mathbf{t_d}),\ not\ Q(\bar{X}', Y, \mathbf{f_a}), dom(\bar{X}', Y)$.
   $aux(\bar{X}') \leftarrow Q(\bar{X}', Y, \mathbf{t_a}), dom(\bar{X}', Y)$.
7. *For every predicate $P \in \mathcal{R}$, the interpretation clauses:*
   $P(\bar{a}, \mathbf{f}^{\star\star}) \leftarrow P(\bar{a}, \mathbf{f_a})$.
   $P(\bar{a}, \mathbf{f}^{\star\star}) \leftarrow not\ P(\bar{a}, \mathbf{t_d}),\ not\ P(\bar{a}, \mathbf{t_a})$.

[1] If $\bar{X} = (X_1, \ldots, X_n)$, we abbreviate $dom(X_1) \wedge \cdots \wedge dom(X_n)$ with $dom(\bar{X})$.

$P(\bar{a}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{a}, \mathbf{t_a})$.
$P(\bar{a}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{a}, \mathbf{t_d}),\ not\ P(\bar{a}, \mathbf{f_a})$. □

Rules 6. repair referential ICs by deletion of tuples or insertion of null values that are not propagated through other ICs [Barcelo *et al.*, 2003]. For this purpose, we consider that the new constant $null \notin \mathcal{U}$, in particular $dom(null)$ is not a fact. The choice models of program $\Pi(\mathfrak{G}, IC)$ that do not contain a pair of literals of the form $\{P(\bar{a}, \mathbf{t_a}), P(\bar{a}, \mathbf{f_a})\}$ are called *coherent models*.

Definition 11 *The instance associated to a choice model $\mathcal{M}$ of $\Pi(\mathfrak{G}, IC)$ is $D_{\mathcal{M}} = \{P(\bar{a}) \mid P(\bar{a}, \mathbf{t}^{\star\star}) \in \mathcal{M}\}$.* □

Theorem 2 *If $\mathcal{M}$ is a coherent choice model $\Pi(\mathfrak{G}, IC)$, then $D_{\mathcal{M}}$ is a repair of $\mathfrak{G}$ wrt IC. Furthermore, the repairs obtained in this way are all the repairs of $\mathfrak{G}$.* □

5 Consistent Answers

Now, we can obtain the answers to queries posed to a system $\mathfrak{G}$ that are consistent wrt to IC. We do the following:
1. We start with a query Q that is expressed as a stratified Datalog program, $\Pi(Q)$, whose extensional predicates are elements of the global schema $\mathcal{R}$. Each positive occurrence of those predicates, say $P(\bar{t})$, is replaced by $P(\bar{t}, \mathbf{t}^{\star\star})$; and each negative occurrence, say $not\ P(\bar{t})$, by $P(\bar{t}, \mathbf{f}^{\star\star})$. This program has a query predicate Ans that collects the answers to Q. In particular, first order queries can be expressed as stratified Datalog programs [Abiteboul *et al.*, 1995].
2. Program $\Pi(Q)$ is appended to the program $SV(\Pi(\mathfrak{G}, IC))$, the stable version of the repair program.
3. The consistent answers to Q are the ground Ans atoms in the intersection of all stable models of $\Pi(Q) \cup SV(\Pi(\mathfrak{G}, IC))$.

Example 12 *(example 9 cont.) Consider the global symmetry integrity constraint $sim : \forall x \forall y(R(x, y) \rightarrow R(y, x))$ on $\mathfrak{G}_3$. We want the consistent answers to the query $Q : P(x, y)$. First, the query is written as the query program clause $Ans(X, Y) \leftarrow P(X, Y, \mathbf{t}^{\star\star})$. This query program, $\Pi(Q)$, is run with $SV(\Pi(\mathfrak{G}_3, sim))$, which on its turn has five stable models with the following associated repairs: (a) $D_{\mathcal{M}_1^r} = \{ P(a, b), R(b, b), P(a, c) \}$, the repair of the already consistent minimal instances $D_{\mathcal{M}_1}$ in Example 10; (b) $D_{\mathcal{M}_2^r} = \{ P(a, a), P(a, c) \}$ and $D_{\mathcal{M}_3^r} = \{ R(a, b), R(b, a), P(a, a), P(a, c) \}$, the repairs of the inconsistent instance $D_{\mathcal{M}_2}$; (c) $D_{\mathcal{M}_4^r} = \{ P(a, c) \}$ and $D_{\mathcal{M}_5^r} = \{ R(c, b), R(b, c), P(a, c) \}$, the repairs of instance $D_{\mathcal{M}_3}$; and (d) $D_{\mathcal{M}_6^r} = \{ P(a, u), P(a, c) \}$ and $D_{\mathcal{M}_7^r} = \{ R(u, b), R(b, u), P(a, u), P(a, c) \}$, the repairs of $D_{\mathcal{M}_4}$.*

The corresponding stable models of $\Pi(Q) \cup SV(\Pi(\mathfrak{G}_3, sim))$ are: (a) $\overline{\mathcal{M}}_1^r = \mathcal{M}_1^r \cup \{Ans(a, b), Ans(a, c)\}$; (b) $\overline{\mathcal{M}}_2^r = \mathcal{M}_2^r \cup \{Ans(a, a), Ans(a, c)\}$; $\overline{\mathcal{M}}_3^r = \mathcal{M}_3^r \cup \{Ans(a, a), Ans(a, c)\}$; (c) $\overline{\mathcal{M}}_4^r = \mathcal{M}_4^r \cup \{Ans(a, c)\}$; $\overline{\mathcal{M}}_5^r = \mathcal{M}_5^r \cup \{Ans(a, c)\}$; (d) $\overline{\mathcal{M}}_6^r = \mathcal{M}_6^r \cup \{Ans(a, u), Ans(a, c)\}$; $\overline{\mathcal{M}}_7^r = \mathcal{M}_7^r \cup \{Ans(a, u), Ans(a, c)\}$. $Ans(a, c)$ is the only query atom in all stable models, then the tuple (a, c) is the only consistent answer to the query. □

If $\mathfrak{G}$ is consistent, then the consistent answers to Q computed with this method coincide with the minimal answers to Q, and then to the certain answers if Q is monotone. If we are interested in just the minimal answers to Q, without considering consistency issues, then they can be computed as above, but using $\Pi(\mathfrak{G})$ introduced in Definition 8 instead of $\Pi(\mathfrak{G}, IC)$.

6 Conclusions

We have presented the most general approach so far to specifying, by means of disjunctive logic programs with a stable model semantics, the database repairs of a mediated integration system with open sources under the LAV approach. Then, consistent answers to queries posed to such a system are computed by running a query program together with the specification of database repairs under the skeptical or cautious stable model semantics. The specification of the repairs is achieved by first specifying the class of minimal global legal instances of the integration system. To the best of our knowledge, this is also the first specification, under the LAV paradigm, of such global instances in a logic programming formalism. This specification is inspired by the inverse rules algorithms, where auxiliary functions are replaced by auxiliary predicates that are forced to be functional by means of the non deterministic choice operator.

The methodology works for conjunctive view definitions, but can be extended to disjunctive views using the corresponding extension of the inverse rules algorithm [Duschka, 1997]. Wrt the ICs and queries it can handle, the approach works for arbitrary universal and referential integrity constrains and queries expressed as Datalog^{not} programs. In consequence, the current approach to consistent query answering (CQA) subsumes and extends the methodologies presented in [Bertossi *et al.*, 2002] for integration systems, and the one in [Barcelo *et al.*, 2002] for stand alone relational databases.

For reasons of space, we just mention a few optimizations of the specification programs and their execution. The materialization of the CWA present in $\Pi(\mathfrak{G}, IC)$ can be avoided by program transformation. Classes of common ICs can be identified for which $SV(\Pi(\mathfrak{G}, IC))$ becomes head-cycle-free, and in consequence, can be transformed into a non disjunctive program [Ben-Eliyahu *et al.*, 1994; Barcelo *et al.*, 2003]. The program for CQA can be split [Lifschitz *et al.*, 1994] into the program that specifies minimal legal instances, the program that specifies their repairs and the query program. The first is non stratified, but its models can be computed bottom-up as fixpoints of an iterative operator [Giannotti *et al.*, 2001]; and the second one is locally stratified [Przymusinski, 1991]. Finally, if the query program is stratified, e.g. if the original query is first-order, then the consistent answers can be eventually computed by a bottom-up evaluation mechanism.

For CQA we have successfully experimented with *DLV* [Eiter *et al.*, 2000]. The current implementations of the disjunctive stable models semantics would be much more effective in database applications if it were possible to evaluate open queries in a form that is guided by the query rather than based on, first, massive grounding of the whole program and, second, considering what can be found in every (completely constructed) stable model of the program.

Wrt related papers, query answering in mediated integration systems *under the assumption* that certain global ICs hold has been treated in [Gryz, 1999; Duschka *et al.*, 2000; Grant *et al.*, 2002; Cali *et al.*, 2002]. However, in CQA, we do not assume that global ICs hold. Logic programming specifications of repairs of single relational databases have been presented in [Arenas *et al.*, 2000; Greco *et al.*, 2001; Barcelo *et al.*, 2002].

[Lembo *et al.*, 2002] considers integration systems under the GAV approach that do not satisfy global key dependencies. There, legal instances are allowed to be more flexible, allowing their computed views to accommodate the satisfaction of the ICs. In this sense, the notion of repair is implicit; and the legal instances are the repairs we have considered here. View definitions are expressed as Datalog queries; and the queries to the global system are conjunctive. The "repairs" of the global system are specified by normal programs under stable model semantics.

In [Bertossi *et al.*, 2002], CQA in possibly inconsistent integration systems under the LAV approach is considered. There, the notion of repair of a minimal legal instance is introduced. The algorithm for CQA is based on a query transformation mechanism [Arenas *et al.*, 1999] applied to first-order queries. The resulting query may contain negation, and is run on top of an extension of the inverse algorithm to the case of stratified Datalog^{not} queries. This approach is limited by the restrictions of the query transformation methodology.

Acknowledgments

Work funded by DIPUC, MECESUP, CONICYT, Carleton University Start-Up Grant 9364-01, NSERC Grant 250279-02. L. Bertossi is Faculty Fellow of the IBM Center for Advanced Studies, Toronto Lab. We are grateful to Pablo Barceló, Alvaro Cortés, Ariel Fuxman, Nicola Leone, and Alberto Mendelzon for useful conversations.

References

[Abiteboul *et al.*, 1995] Abiteboul, S.; Hull, R. and Vianu, V. *Foundations of Databases*. Addison-Wesley, 1995.

[Abiteboul *et al.*, 1998] Abiteboul, A. and Duschka, O. Complexity of Answering Queries Using Materialized Views. In *Proc. ACM PODS*, 1998, pp. 254-263.

[Arenas *et al.*, 1999] Arenas, M., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. In *Proc. ACM PODS*, 1999, pp. 68–79.

[Arenas *et al.*, 2000] Arenas, M.; Bertossi, L. and Chomicki, J. Specifying and Querying Database Repairs Using Logic Programs with Exceptions. In *Flexible Query Answering Systems. Recent Developments*. H. Larsen et al. (eds.), Springer, 2000, pp. 27–41.

[Barcelo *et al.*, 2002] Barcelo, P. and Bertossi, L. Repairing Databases with Annotated Predicate Logic. In *Proc. International Workshop on Non-Monotonic Reasoning*. S. Benferhat and E. Giunchiglia (eds.), 2002, pp. 160–170.

[Barcelo *et al.*, 2003] Barcelo, P., Bertossi, L. and Bravo, L. Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. In 'Semantics of Databases', Springer LNCS 2582, 2003, pp. 1–27.

[Ben-Eliyahu *et al.*, 1994] Ben-Eliyahu, R. and Dechter, R. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics in Artificial Intelligence*, 1994, 12:53-87.

[Bertossi *et al.*, 2002] 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.

[Cali *et al.*, 2002] Cali, A.; Calvanese, D.; De Giacomo, G. and Lenzerini, M. Data integration Under Integrity Constraints. In *Proc. CAISE*, Springer LNCS 2348, 2002, pp. 262–279.

[Duschka, 1997] Duschka, O. *Query Planning and Optimization in Information Integration*. PhD Thesis, Stanford University, December 1997.

[Duschka *et al.*, 2000] Duschka, O., Genesereth, M. and Levy, A. Recursive Query Plans for Data Integration. *J. Logic Programming*, 2000, 43(1):49-73.

[Eiter *et al.*, 2000] Eiter, T., Faber, W.; Leone, N. and Pfeifer, G. Declarative Problem-Solving in DLV. In *Logic-Based Artificial Intelligence*. J. Minker (ed.), Kluwer, 2000, pp. 79-103.

[Giannotti *et al.*, 1991] Giannotti, F.; Pedreschi, D.; Sacca, D. and Zaniolo, C. Non-Determinism in Deductive Databases. In *Proc. DOOD*, Springer LNCS 566, 1991, pp. 129–146.

[Giannotti *et al.*, 2001] Giannotti, F.; Pedreschi, D. and Zaniolo, C. Semantics and Expressive Power of Nondeterministic Constructs in Deductive Databases. *J. Computer and System Sciences*, 62(1):15–42, 2001.

[Grant *et al.*, 2002] Grant, J. and Minker, M. A Logic-based Approach to Data Integration. *Theory and Practice of Logic Programming*, 2002, 2(3):323-368.

[Greco *et al.*, 2001] Greco, G., Greco, S. and Zumpano, E. A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. In *Proc. ICLP*, Springer LNCS 2237, 2001, pp. 348–364.

[Gryz, 1999] Gryz, J. Query Rewriting Using Views in the Presence of Functional and Inclusion Dependencies. *Information Systems*, 1999, 24(7):597–612.

[Lembo *et al.*, 2002] Lembo, D.; Lenzerini, M. and Rosati, R. Source Inconsistency and Incompleteness in Data Integration. In *Proc. KRDB*, 2002.

[Lenzerini, 2002] Lenzerini, M. Data Integration: A Theoretical Perspective. In *Proc. ACM PODS*, 2002, pp. 233-246.

[Lifschitz *et al.*, 1994] Lifschitz, V. and Turner, H. Splitting a Logic Program. In *Proc. ICLP*. The MIT Press, 1994, pp. 23–37.

[Przymusinski, 1991] Przymusinski, T. Stable Semantics for Disjunctive Programs. *New Generation Computing*, 9(3/4):401–424, 1991.