Course at ESSLLI 2005

The Logics of Consistent Query Answers in Databases

by

Prof. Leopoldo Bertossi

Carleton University
School of Computer Science
Ottawa, Canada

bertossi@scs.carleton.ca
www.scs.carleton.ca/~bertossi

Abstract: This course addresses the logical aspects of the problems of defining and obtaining consistent information from inconsistent databases, i.e. from databases that violate given semantic (integrity) constraints.

The basic assumption that databases may be inconsistent departs from the everyday practice of database management systems, where typically the system checks the satisfaction of integrity constraints and backs out those updates that violate them. However, present-day database applications have to consider a variety of scenarios in which data is not necessarily consistent. From this perspective, integrity constraints can be seen as constraints on query answers rather than on the data or database states.

The subject of consistent query answering in databases has received the attention of the database and logic programming communities for the last 6 years. We summarize research carried out in the field, starting by the seminal paper presented at the ACM Symposium on Principles of Database Systems (PODS 99) by Arenas, Bertossi, Chomicki.

The logical approaches that has been followed rely on the concepts of "repair" and "consistent query answer" (CQA). We describe (a) logical specifications of the notion of repair and CQA; and (b) methodologies for computing CQAs: query transformation, logic programming, inference in annotated logics, and specialized algorithms. Computational complexity issues are also discussed. Applications to virtual data integration and P2P data exchange will be introduced.

Slides for the course will be posted before and during the course under
http://www.scs.carleton.ca/~bertossi/talks/esslli05.pdf

Contents

Material (papers)

1. 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. **P.1**

2. 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. **P.12**

3. Arenas, M., Bertossi, L. and Kifer, M. “Applications of Annotated Predicate Calculus to Querying Inconsistent Databases”. In *Computational Logic - CL 2000*, Springer LNAI 1861, Springer, 2000, pp. 926 – 941. **P.15**

4. Barcelo, P. and Bertossi, L. “Repairing Databases with Annotated Predicate Logic”. *Proc. NMR’02*, 2002. **P.21**

5. Bertossi, L., Schwind, C. “Database Repairs and Analytic Tableaux”. *Annals of Mathematics and Artificial Intelligence*, 2004, 40(1-2): 5-35. **P.24**

6. 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. **P.33**

7. 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. **P.41**

8. Bravo, L., Bertossi, L. “Consistent Query Answering under Inclusion Dependencies”. *Proc. 14th CASCON Conference*, IBM, Toronto Lab., 2004. **P.51**

9. Arenas, M., Bertossi, L., Chomicki,J., He, X., Raghavan, V. and Spinrad, J. “Scalar Aggregation in Inconsistent Databases”. *Theoretical Computer Science*, Volume 296, Issue 3, 2003, pp. 405-434. **P.55**

10. Bertossi, L. and Bravo, L. “Consistent Query Answers in Virtual Data Integration Systems”. In *Inconsistency Tolerance*, Springer LNCS 3300, 2004, pp. 42-83. **P.62**

11. 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. **P.73**

12. Bravo, L. and Bertossi, L. “Disjunctive Deductive Databases for Computing Certain and Consistent Answers to Queries from Mediated Data Integration Systems”. *Journal of Applied Logic*, 2005, 3(2):329-367. **P.80**

13. Bertossi, L. and Bravo, L. “Query Answering in Peer-to-Peer Data Exchange Systems”. In *Current Trends in Database Technology.* Springer LNCS 3268, 2004, pp. 478-485. (the paper attached here is an extended version) **P.90**

Query Answering in Inconsistent Databases

Leopoldo Bertossi[1] and Jan Chomicki[2]

[1] School of Computer Science, Carleton University, Ottawa, Canada,
bertossi@scs.carleton.ca
[2] Dept. of Computer Science and Engineering, University at Buffalo, State
University of New York, Buffalo, NY, chomicki@cse.buffalo.edu

Abstract. In this chapter, we summarize the research on querying inconsistent databases that we have been conducting during the last five years. The formal framework that we have used is based on two concepts: *repair* and *consistent query answer*. We describe different approaches to the issue of computing consistent query answers: query transformation, logic programming, inference in annotated logics, and specialized algorithms. We also characterize the computational complexity of this problem. Finally, we discuss related research in artificial intelligence, databases, and logic programming.

1 Introduction

In this chapter, we address the issue of obtaining *consistent* information from *inconsistent* databases – databases that violate given integrity constraints. Our basic assumption departs from the everyday practice of database management systems. Typically, a database management system checks the satisfaction of integrity constraints and backs out those updates that violate them. However, present-day database applications have to consider a variety of scenarios in which data is not necessarily consistent:

Integration of autonomous data sources. The sources may separately satisfy the constraints, but when they are integrated the constraints may not hold. For instance, consider different conflicting addresses for the same person in a taxpayer database and a voter registration database. Each of those databases separately satisfies the functional dependency that associates a single address with each person, and yet together they violate this dependency. Moreover, because the sources are autonomous, the violations cannot be simply fixed by removing one of the conflicting tuples.

Unenforced integrity constraints. Even though integrity constraints capture an important part of the semantics of a given application, they may still fail to be enforced for a variety of reasons. A data source may be a legacy system that does not support the notion of integrity checking altogether, or integrity checking may be too costly (this is often the reason for dropping some integrity constraints from a database schema). Finally, the DBMS itself may support only a limited class of constraints.

Temporary inconsistencies. It may often be the case that the consistency of a database is only temporarily violated and further updates or transactions

are expected to restore it. This phenomenon is becoming more and more common, as databases are increasingly involved in a variety of long-running activities or *workflows*.

Conflict resolution. Removing tuples from a database to restore consistency leads to information loss, which may be undesirable. For example, one may want to keep multiple addresses for a person if it is not clear which is the correct one. In general, the process of conflict resolution may be complex, costly, and nondeterministic. In real-time decision-making applications, there may not be enough time to resolve all conflicts relevant to a query.

To formalize the notion of consistent information obtained from a (possibly inconsistent) database in response to a user query, we propose the notion of a *consistent query answer*. A consistent answer is, intuitively, true regardless of the way the database is fixed to remove constraint violations. Thus, answer consistency serves as an indication of its reliability. The different ways of fixing an inconsistent database are formalized using the notion of a *repair*. A repair is another database that is consistent and differs minimally from the original database.

We summarize the results that we and our collaborators have obtained so far in this area. We have studied consistent query answers for first-order and scalar aggregation queries. We have also considered the specification of repairs using logic-based formalisms. We relate our results to similar work undertaken in knowledge representation and logic programming, databases, and philosophical logic. It should be pointed out that we are studying a very specific instance of the logical inconsistency problem: when the data is inconsistent with the integrity constraints. We do not address the issue of how to deal with inconsistent sets of formulas in general. In standard relational databases negative information is represented implicitly (through the Closed World Assumption), and inconsistencies appear only in the presence of integrity constraints.

The trivialization of classical logical inference in the presence of an inconsistency is less of a problem in the database context because database systems typically do not support full-fledged first-order inference. It is more important to be able to distinguish which query answers are affected by the inconsistency and which are not.

This chapter is structured as follows. In Sect. 2, we define the notions of repair and consistent query answer (CQA) in the context of first-order queries. In Sect. 3, we present a corresponding computational methodology based on query transformation. In Sect. 4, we show how to specify database repairs declaratively using logic programming and annotated logics. In Sect. 5, we discuss computational complexity issues. In Sect. 6, we show that in the context of aggregation queries the definition of CQAs has to be slightly modified, and we discuss the corresponding computational mechanisms. In Sect. 7, we discuss other related approaches to handling inconsistent information. In Sect. 8, we present open problems.

2 Consistent Query Answers

Our basic assumption is that an inconsistent database is not necessarily going to be repaired in a way that fully restores its consistency. Therefore, if such a database is to be queried, we have to distinguish between the information in the database that participates in the integrity violations, and one that does not. Typically, only a small part of a database will be inconsistent.

We need to make precise the notion of "consistent" (or "correct") information in an inconsistent database. More specifically, we address the following issues:

1. giving a precise definition of a *consistent answer* to a query in an inconsistent database,
2. finding *computational mechanisms* for obtaining consistent information from an inconsistent database, and
3. studying the *computational complexity* of this problem.

Example 1. Consider the following relational database instance r:

Employee	*Name*	*Salary*
	$J.Page$	5000
	$J.Page$	8000
	$V.Smith$	3000
	$M.Stowe$	7000

The instance r violates the functional dependency f_1 : *Name* → *Salary* through the first two tuples. This is an inconsistent database. Nevertheless, there is still some "consistent" information in it. For example, only the first two tuples participate in the integrity violation. To characterize the consistent information, we notice that there are two possible ways to repair the database in a minimal way if only deletions and insertions of whole tuples are allowed. They give rise to two different repairs:

Employee1	*Name*	*Salary*
	$J.Page$	5000
	$V.Smith$	3000
	$M.Stowe$	7000

Employee2	*Name*	*Salary*
	$J.Page$	8000
	$V.Smith$	3000
	$M.Stowe$	7000

We can see that certain information, for example $(M.Stowe, 7000)$, persists in both repairs because it does not participate in the violation of the FD f_1. On the other hand, some information, for example $(J.Page, 8000)$, does not persist in all repairs because it participates in the violation of f_1.

There are other pieces of information that can be found in both repairs, for example we know that there is an employee with the name *J. Page*. Such information cannot be obtained if we simply discard the tuples that participate in the violation.

□

In the following, we assume that we have a fixed relational database schema R consisting of a finite set of relations. We also have two fixed, disjoint, infinite database domains: D (uninterpreted constants) and N (numbers). We assume that elements of the domains with different names are different. Database instances can be seen as finite, first-order structures over the given schema that share the domains D and N. Every attribute in every relation is typed, thus all the instances of R can contain only elements of either D or N in a single attribute. Because each instance is finite, it has a finite active domain that is a subset of $D \cup N$. As usual, we allow the standard built-in predicates over N $(=, \neq, <, >, \leq, \geq)$ that have infinite, fixed extensions. The domain D has only equality as a built-in predicate. Using all of these elements we can build a first-order language $\mathcal{L}$.

2.1 Integrity Constraints

Integrity constraints are typed, closed first-order $\mathcal{L}$-formulas. We assume that we are dealing with a single set of integrity constraints IC which is consistent as a set of logical formulas. In the sequel, we will denote relation symbols by $P_1, \ldots, P_m$; tuples of variables and constants by $\bar{x}_1, \ldots, \bar{x}_m$; and a quantifier-free formula referring only to built-in predicates by ϕ. We also represent a ground tuple $\bar{a}$ in a relation P as the fact $P(\bar{a})$.

Practically important integrity constraints (called simply *dependencies* in [1, Chapt. 10]) can be expressed as $\mathcal{L}$-sentences of the form

$$\forall \bar{x}\ \exists \bar{y}.\ [\bigvee_{i=1}^{m} P_i(\bar{x}_i)\ \vee \bigvee_{i=m+1}^{n} \neg P_i(\bar{x}_i)\ \vee\ \phi(\bar{x}_1, \ldots, \bar{x}_n)], \tag{1}$$

where $\bar{x}_i \subseteq \bar{x} \cup \bar{y}$, $i = 1, \ldots, n$.

In this chapter, we discuss the following classes of integrity constraints that are special cases of (1):

1. *Universal integrity constraints*: $\mathcal{L}$-sentences
$$\forall \bar{x}_1, \ldots, \bar{x}_n.\ [\bigvee_{i=1}^{m} P_i(\bar{x}_i)\ \vee \bigvee_{i=m+1}^{n} \neg P_i(\bar{x}_i)\ \vee\ \phi(\bar{x}_1, \ldots, \bar{x}_n)].$$
2. *Denial constraints*: $\mathcal{L}$-sentences
$$\forall \bar{x}_1, \ldots, \bar{x}_n.\ [\bigvee_{i=1}^{n} \neg P_i(\bar{x}_i)\ \vee\ \phi(\bar{x}_1, \ldots, \bar{x}_n)].$$
They are a special case of universal constraints.
3. *Binary constraints*: universal constraints with at most two occurrences of database relations.

4. *Functional dependencies (FDs)*: $\mathcal{L}$-sentences

$$\forall \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5.\ [\neg P(\bar{x}_1,\bar{x}_2,\bar{x}_4)\ \vee\ \neg P(\bar{x}_1,\bar{x}_3,\bar{x}_5)\ \vee\ \bar{x}_2 = \bar{x}_3].$$

   They are a special case of binary denial constraints. A more familiar formulation of the above FD is $X \rightarrow Y$, where X is the set of attributes of P corresponding to $\bar{x}_1$ and Y the set of attributes of P corresponding to $\bar{x}_2$ (and $\bar{x}_3$).
5. *Referential integrity constraints*, also known as *inclusion dependencies (INDs)*: $\mathcal{L}$-sentences

$$\forall \bar{x}_1\ \exists \bar{x}_3.\ [\neg Q(\bar{x}_1)\ \vee\ P(\bar{x}_2,\bar{x}_3)],$$

   where the $\bar{x}_i$ are sequences of distinct variables, with $\bar{x}_2$ contained in $\bar{x}_1$; and database relations P, Q. Again, this is often written as $Q[Y] \subseteq P[X]$ where X (respectively, Y) is the set of attributes of P (respectively, Q) corresponding to $\bar{x}_2$. If P and Q are clear from the context, we omit them and write the dependency simply as $Y \subseteq X$. If an IND can be written without any existential quantifiers, then it is called *full*.

Denial constraints, in particular FDs, and INDs are the most common integrity constraints in database practice. In fact, commercial systems typically restrict FDs to key dependencies and INDs to foreign key constraints.

Given a set of FDs and INDs IC and a relation P with attributes U, a *key* of P is a minimal set of attributes X of P such that IC entails the FD $X \rightarrow U$. In that case, we say that each FD $X \rightarrow Y \in IC$ is a *key* dependency and each IND $Q[Y] \subseteq P[X] \in IC$ is a *foreign key constraint*. If, additionally, X is the primary key of P, then both kinds of dependencies are termed *primary*.

We have seen an FD in Example 1. FDs and INDs are also present in Example 4. Below, we show some examples of denial constraints.

Example 2. Consider the relation *Emp* with attributes *Name*, *Salary*, and *Manager*, where *Name* is the primary key. The constraint that *no employee can have a salary greater than that of her manager* is a denial constraint:

$$\forall n, s, m, s', m'.\ [\neg Emp(n, s, m) \vee \neg Emp(m, s', m') \vee s \leq s'].$$

Similarly, single-tuple constraints (`CHECK` constraints in SQL2) are a special case of denial constraints. For example, the constraint that *no employee can have a salary over $200,000* is expressed as

$$\forall n, s, m.[\neg Emp(n, s, m) \vee s \leq 200000].$$

□

Definition 1. Given a database instance r of R and a set of integrity constraints IC, we say that r is *consistent* if $r \vDash IC$ in the standard model-theoretic sense (i.e. IC is true in r); *inconsistent* otherwise. □

Reiter [88] characterized relational databases as first-order theories by axiomatizing the Unique Names, Domain Closure, and Closed World Assumptions. Each such theory is categorical in the sense that it admits the original database, seen as a first-order structure, as its only model. In consequence, satisfaction in a model can be replaced by first-order logical entailment. In this context, a database is consistent with respect to a set of integrity constraints if it entails (as a theory in the sense of Reiter) the set of integrity constraints. There is an alternative notion of database consistency [91]: a database is consistent if its union (as a theory consisting of the atoms in the database) with the set of integrity constraints is consistent in the usual logical sense. All three notions of a consistent relational database, namely the two just presented and Definition 1, turn out to be equivalent for relational databases but may differ for "open" knowledge bases (see [89,90] for a discussion).

Example 3. Consider a binary relation $P(AB)$ and a functional dependency $A \rightarrow B$. An instance p of P consisting of two tuples (a, b) and (a, c) is inconsistent according to Definition 1. The following set of formulas,

$$\{P(a, b), P(a, c), b \neq c, \forall x, y, z.\ [\neg P(x, y)\ \vee\ \neg P(x, z)\ \vee\ y = z]\},$$

is inconsistent in the standard logic sense. □

2.2 Repairs

Given a database instance r, the *set* $\Sigma(r)$ *of facts* of r is the set of ground atomic formulas $\{P(\bar{a}) \mid r \vDash P(\bar{a})\}$, where P is a relation symbol and $\bar{a}$ a ground tuple. The *distance* $\Delta(r, r')$ between database instances r and r' is defined as the symmetric difference of r and r':

$$\Delta(r, r')\ =\ (\Sigma(r) - \Sigma(r')\ \cup\ (\Sigma(r') - \Sigma(r)).$$

Definition 2. [3] A database instance r' is a *repair* of a database instance r w.r.t. a set of integrity constraints IC if

1. r' is over the same schema and domain as r,
2. r' satisfies IC,
3. the *distance* $\Delta(r, r')$ is minimal under set containment among the instances satisfying the first two conditions.

□

We note that for denial constraints all repairs of an instance r are subsets of r (see Example 1). However, for more general constraints, repairs may contain tuples that do not belong to r. For instance, removing violations of referential integrity constraints can be done not only by deleting but also by inserting tuples.

Example 4. Consider a database with two relations *Personnel(SSN,Name)* and *Manager(SSN)*. There are FDs $SSN \rightarrow Name$ and $Name \rightarrow SSN$, and an IND $Manager[SSN] \subseteq Personnel[SSN]$. The relations have the following instances:

Personnel	*SSN*	*Name*
	123456789	*Smith*
	555555555	*Jones*
	555555555	*Smith*

Manager	*SSN*
	123456789
	555555555

The instances do not violate the IND but violate both FDs. If we consider only the FDs, there are two repairs: one obtained by removing the third tuple from *Personnel*, and the other by removing the first two tuples from the same relation. However, the second repair violates the IND. This can be fixed by removing the first tuple from *Manager*. So if we consider all three constraints, there are two repairs obtained by deletion:

Personnel	*SSN*	*Name*
	123456789	*Smith*
	555555555	*Jones*

Manager	*SSN*
	123456789
	555555555

and

Personnel	*SSN*	*Name*
	555555555	*Smith*

Manager	*SSN*
	555555555

Additionally, there are infinitely many repairs, obtained by a combination of deletions and insertions, of the form

Personnel	*SSN*	*Name*
	123456789	c
	555555555	*Smith*

Manager	*SSN*
	123456789
	555555555

where c is an arbitrary element of the database domain D different from *Smith*. □

Definition 2 reflects the assumption that the information in the database may not only be incorrect but also *incomplete*. This assumption is warranted in some information integration approaches [73]. On the other hand, restricting repairs to subsets of the original database (as in [29]) is based on the

assumption that the information in the database is *complete*, although not necessarily correct. That assumption seems appropriate in the context of data warehousing where dirty data coming from many sources is cleaned for use as part of the warehouse itself.

Another variation of the notion of repair assumes a different notion of minimality: instead of minimizing the symmetric difference, we may minimize its *cardinality*. We discuss this issue in Sect. 7. Still another dimension of the repair concept was recently introduced by Wijsen [96] who proposed repairs obtained by modifying selected tuple components.

2.3 Queries and Consistent Query Answers

Queries are formulas over the same language $\mathcal{L}$ as the integrity constraints. A query is *closed* (or a *sentence*) if it has no free variables. A closed query without quantifiers is also called *ground*. *Conjunctive queries* [1,27] are queries of the form

$$\exists \bar{x}_1, \ldots \bar{x}_m.\ [P_1(\bar{x}_1)\ \wedge\ \cdots\ \wedge\ P_m(\bar{x}_m)\ \wedge\ \phi(\bar{x}_1, \ldots, \bar{x}_m)],$$

where $\phi(\bar{x}_1, \ldots, \bar{x}_m)$ is a conjunction of built-in predicate atoms. If a conjunctive query has no repeated relation symbols, it is called *simple*.

The following definition is standard:

Definition 3. A tuple $\bar{t}$ is an *answer* to a query $Q(\bar{x})$ in r iff $r \models Q(\bar{t})$, i.e. the formula Q with $\bar{x}$ replaced by $\bar{t}$ is true in r. □

Given a query $Q(\bar{x})$ to an instance r, we want as *consistent* answers those tuples that are unaffected by the violations of the integrity constraints present in r.

Definition 4. [3] A tuple $\bar{t}$ is a *consistent answer* (CQA) to a query $Q(\bar{x})$ in a database instance r w.r.t. a set of integrity constraints IC iff $\bar{t}$ is an answer to the query $Q(\bar{x})$ in every repair r' of r w.r.t. IC. An $\mathcal{L}$-sentence Q is *consistently true* in r w.r.t. IC if it is true in every repair of r w.r.t. IC. In symbols,

$$r \models_{IC} Q(\bar{t})\ \equiv\ r' \models Q(\bar{t})\ \text{ for every repair } r' \text{ of } r \text{ w.r.t. } IC.$$

□

Example 5. Continuing Example 1, we have the following consistently true formulas:

1. $r \models_{\{f_1\}}\ Employee(M.Stowe, 7000)$
2. $r \models_{\{f_1\}}\ (Employee(J.Page, 5000)\ \vee\ Employee(J.Page, 8000))$

3. $r \models_{\{f_1\}} \exists x.\ [Employee(J.Page, x)]$

□

Notice that through Definition 4 our approach leads to a stronger notion of inference from inconsistent databases than an approach based on simply discarding conflicting data. In the latter approach, the last two inferences in Example 5 would not be possible.

For universal integrity constraints, the number of repairs of a finite database is also finite. However, referential integrity constraints may lead to infinitely many repairs (see Example 4). Having infinitely many repairs is a problem for those approaches to computing consistent query answers that construct a representation of all repairs, as do the approaches based on logic programming (Sect. 4). Therefore, they use a slightly different notion of repair by allowing tuples with *nulls* to be inserted into the database. This reflects common SQL2 database practice. But that approach does not always work, as the *entity integrity* constraint inherent in the relational data model prevents null values from appearing in the primary key.

Example 6. Consider Example 4 again. Infinitely many repairs can be replaced by a single repair:

Personnel	*SSN*	*Name*
	123456789	*null*
	555555555	*Smith*

Manager	*SSN*
	123456789
	555555555

only if it is the *SSN* attribute which is designated the primary key, not the *Name* attribute (which still remains a key). □

One can also avoid dealing with infinitely many repairs by restricting repairs to subsets of the original instance, as in [29].

If a notion of repair different from than that in Definition 2 is used, the notion of consistent query answer changes, too. In general, the more restricted the repairs, the stronger the consistent query answers, as illustrated by the following example.

Example 7. Consider a database schema consisting of two relations $P(AB)$ and $S(C)$. The integrity constraints are FD $A \rightarrow B$ and IND $B \subseteq C$. Assume that the database instance $r_1 = \{P(a,b), P(a,c), S(b)\}$. Under Definition 2, there are two repairs: $r_2 = \{P(a,b), S(b)\}$ and $r_3 = \{P(a,c), S(b), S(c)\}$. Thus, $P(a,b)$ is not consistently true in the original instance r_1 according to Definition 4. Note that $P(a,c)$ is not consistently true in r_1 either. Therefore, $P(a,b)$ and $P(a,c)$ are treated symmetrically from the point of view of consistent query answering. However, intuitively there is a difference between them. Think of A as the person's name, B her address, and S a list of valid addresses. The difference between $P(a,b)$ and $P(a,c)$ is captured under

a more restrictive definition of repair requiring that a repair be a subset of the original instance. In the latter sense, only r_2 is a repair and $P(a,b)$ is consistently true in r_1. □

In the sequel, we will mostly use the notion of repair from Definition 2, clearly indicating the cases where a different notion is applied.

2.4 Computing CQAs

What we have so far is a semantic definition of a consistent query answer in a (possibly inconsistent) database, based on the notion of database repair. However, retrieving CQAs via the computation of *all* database repairs is not feasible. Even for FDs, the number of repairs may be too large.

Example 8. Consider the functional dependency $A \rightarrow B$ and the following family of relation instances r_n, $n > 0$, each of which has $2n$ tuples (represented as columns) and 2^n repairs:

r_n							
A	a_1	a_1	a_2	a_2	$\cdots$	a_n	a_n
B	b_0	b_1	b_0	b_1	$\cdots$	b_0	b_1

□

Therefore, we develop various methods for computing CQAs *without explicitly computing all repairs*. Such methods can be split into two categories:

1. *Query transformation.* Given a query Q and a set of integrity constraints IC, construct a query Q' such that for every database instance r, the set of answers to Q' in r is equal to the set of consistent answers to Q in r w.r.t. IC. This approach was first proposed in [3] for first-order queries. In that case, the transformed query is also first-order, thus after a straightforward translation to SQL2, it can be evaluated by any relational database engine. Note that the construction of all repairs is entirely avoided. In [25], the implementation of an extended version of the method of [3] was described.
2. *Compact representation of repairs.* Given a set of integrity constraints IC and a database instance r, construct a space-efficient representation of all repairs of r w.r.t. IC, and then, use this representation to answer queries. Different representations have been considered in this context:

2.1. Repairs are answer sets of a logic program [4,6,14,15]. The compact representation is the program, and to obtain consistent answers, one runs the program.

2.2. Repairs are some distinguished minimal models of a theory written in annotated predicate logic [8,14].

2.3. Repairs are maximal independent sets in a hypergraph whose nodes are database tuples and whose edges are sets of tuples participating in a violation of a denial constraint. This approach has been applied in [29] to quantifier-free first-order queries and in [5,7] to aggregation queries.

2.4. The interaction of the database instance and the integrity constraints is represented as an analytical tableau that becomes closed due to the mutual inconsistency of the database and the integrity constraints. The implicit "openings" of the tableau are the repairs [19]. Implementation issues around consistent query answering based on analytic tableaux for nonmonotonic reasoning are discussed in [20].

In the next sections, we describe some of these approaches.

3 Query Transformation

Here we consider first-order queries and universal integrity constraints. Given a query, we rewrite it, preserving the original database instance. The query is transformed by qualifying it with appropriate information derived from the interaction between the query and the integrity constraints. This forces the (local) satisfaction of the integrity constraints and makes it possible to discriminate between the tuples in the answer set. The technique is inspired by *semantic query optimization* [26].

More precisely, given a query $\varphi(\bar{x})$, a new query $T^\omega(\varphi(\bar{x}))$ is computed by iterating an operator T which transforms the query by conjoining the corresponding *residues* to each database literal appearing in the query, until a fixed point is reached. (If there are no residues, then $T(Q) = Q$.) The residues of a database literal force the satisfaction of the integrity constraints for the tuples satisfying the literal and are obtained by resolving the literal with the integrity constraints.

Example 9. Consider the following integrity constraints:

$$IC\ = \{\forall x.[R(x) \vee \neg P(x) \vee \neg Q(x)],\ \forall x.[P(x) \vee \neg Q(x)]\}$$

and the query $Q(x)$. The residue of $Q(x)$ w.r.t. the first constraint is $R(x) \vee \neg P(x)$, because if both $Q(x)$ and the constraint are to be satisfied, then that residue has to be true. Similarly, the residue of $Q(x)$ w.r.t. the second constraint is $P(x)$. In consequence, instead of the query $Q(x)$, one rather asks the transformed query $Q(x) \wedge (R(x) \vee \neg P(x)) \wedge P(x)$. The literal $\neg Q(x)$ does not have any residues w.r.t. the given integrity constraints, because the integrity constraints do not constrain it. □

If we want the CQAs to an $\mathcal{L}$-query $\varphi(\bar{x})$ in r, we rewrite the query into the new $\mathcal{L}$-query $T^\omega(\varphi(\bar{x}))$, and we pose $T^\omega(\varphi(\bar{x}))$ to r as an ordinary query. We expect that for every ground tuple $\bar{t}$:

$$r \models_{IC} \varphi(\bar{t})\ \equiv\ r \models T^\omega(\varphi(\bar{t})).$$

We explain later under what conditions this equivalence holds.

Example 10. (Example 1, continued) The FD f_1 can be written as the $\mathcal{L}$-formula

$$f_I :\ \ \forall x,y,z.\ [\neg Employee(x,y)\ \vee\ \neg Employee(x,z)\ \vee\ y = z]. \qquad (2)$$

If we are given the query $Q(x,y)$: $Employee(x,y)$, we expect to obtain the consistent answers: $(V.Smith,\ 3000)$, $(M.Stowe,\ 7000)$, but not $(J.Page,\ 5000)$ or $(J.Page,\ 8000)$.

The residue obtained by resolving the query with the FD f_1 is

$$\forall z.\ [\neg Employee(x,z)\ \vee\ y = z].$$

Note that we get the same residue by resolving the query with the first or second literal of the constraint. Thus, the rewritten query $T(Q(x,y))$ is as follows:

$$T(Q(x,y)) := Employee(x,y)\ \wedge\ \forall z.\ [\neg Employee(x,z)\ \vee\ y = z],$$

and returns exactly $(V.Smith,\ 3000)$ and $(M.Stowe,\ 7000)$ as answers, i.e. the consistent answers to the original query. □

In general, T needs to be iterated because we may need to consider the residues of residues and so on. In consequence, depending on the integrity constraints and the original query, we may need to iterate T until the infinite fixed point T^ω is obtained. In Example 10, this was not necessary, because the literal $\neg Employee(x,z)$ in the appended residue does not have a residue w.r.t. f_1 itself. We stop after the first iteration.

Example 11. (Example 9, continued) The following are the sets of residues for the relevant literals (the other literals have no residues):

Literal	*Residues*
$P(x)$	$: \{R(x) \vee \neg Q(x)\}$
$Q(x)$	$: \{R(x) \vee \neg P(x),\ \ P(x)\}$
$\neg P(x)$	$: \{\neg Q(x)\}$
$\neg R(x)$	$: \{\neg P(x) \vee \neg Q(x)\}.$

The query is transformed into $T(Q(x)) = Q(x) \wedge (R(x) \vee \neg P(x)) \wedge P(x)$. Now, we apply T again, to the appended residues, obtaining

$$\begin{aligned} T^2(Q(x)) &= Q(x) \wedge (T(R(x)) \vee T(\neg P(x))) \wedge T(P(x)) \\ &= Q(x) \wedge (R(x) \vee (\neg P(x) \wedge \neg Q(x))) \wedge P(x) \wedge (R(x) \vee \neg Q(x)). \end{aligned}$$

And once more

$$\begin{aligned} T^3(Q(x)) = Q(x) \wedge\ & (R(x) \vee (\neg P(x) \wedge T(\neg Q(x)))) \wedge \\ & P(x) \wedge (T(R(x)) \vee T(\neg Q(x))). \end{aligned}$$

Because $T(\neg Q(x)) = \neg Q(x)$ and $T(R(x)) = R(x)$, we obtain $T^2(Q(x)) = T^3(Q(x))$, and we have reached a fixed point. □

The fundamental properties of the transformation-based approach are: soundness, completeness, and termination [3]. *Soundness* means that every answer to $T^\omega(Q)$ is a consistent answer to Q. *Completeness* means that every consistent answer to Q is an answer to $T^\omega(Q)$. *Termination* means that there is an n such that for all $m \geq n$, $\forall \bar{x}(T^n(Q(\bar{x})) \equiv T^m(Q(\bar{x}))$ is a valid formula.

In [3], some very general sufficient conditions for soundness of the transformation-based approach are defined, encompassing essentially all integrity constraints that occur in practice. Completeness is much harder to achieve. In [3], the completeness of the transformation-based approach is proved for binary, generic integrity constraints and queries that are conjunctions of literals. (A constraint is *generic* if it does not imply any ground database literal.) For example, we may have the query $R(u,v) \wedge \neg P(u,v)$, and the binary integrity constraints

$$IC = \{\forall x,y.[\neg P(x,y) \vee R(x,y)], \forall x,y,z.[\neg P(x,y) \vee \neg P(x,z) \vee y = z]\}.$$

However, with disjunctive or existential queries, we may lose completeness.

Example 12. In Example 10, if we pose the ground disjunctive query

$$Q\colon \mathit{Employee}(J.\mathit{Page},\ 5000) \vee \mathit{Employee}(J.\mathit{Page},\ 8000),$$

the application of the operator T produces the rewritten query $T(Q)$:

$$\begin{aligned} &(\mathit{Employee}(J.\mathit{Page}, 5000)\ \wedge\ \forall z\ (\neg \mathit{Employee}(J.\mathit{Page}, z)\ \vee\ z = 5000))\ \vee \\ &(\mathit{Employee}(J.\mathit{Page}, 8000)\ \wedge\ \forall z\ (\neg \mathit{Employee}(J.\mathit{Page}, z)\ \vee\ z = 8000)). \end{aligned}$$

that has the answer (truth value) *false* in the original database instance, but, according to the definition of consistent answer, is consistently true in this instance. □

Termination can be guaranteed if there is an n such that $T^n(Q(\bar{x}))$ and $T^{n+1}(Q(\bar{x}))$ are syntactically the same. Reference [3] shows that this property holds for any kind of queries iff the set of integrity constraints IC is *acyclic*, where IC is acyclic if there exists a function

$$f : \{P_1, \ldots, P_n, \neg P_1, \ldots, \neg P_n\} \longrightarrow \mathbb{N},$$

such that for every constraint,

$$\forall(\bigvee_{i=1}^{k} l_i(\bar{x}_i) \vee \phi(\bar{x})) \in IC,$$

and every $1 \leq i,j \leq k$, if $i \neq j$, then $f(\neg l_i) > f(l_j)$. Here, f is a *level mapping*, similar to the mappings associated with stratified or hierarchical logic programs, except that complementary literals get values independently of each other. Any set of denial constraints – thus also FDs – is acyclic.

For example, termination is syntactically guaranteed for any query if

$$IC = \{\forall x,y.[\neg P(x,y) \vee R(x,y)], \forall x,y,z.[\neg P(x,y) \vee \neg P(x,z) \vee y = z]\}.$$

Reference [3] provides further, nonsyntactic sufficient criteria for termination of the transformation-based approach. In particular, termination for multivalued dependencies is obtained.

In [25], an implementation of the operator T^ω is presented. The implementation is done on top of the XSB deductive database system [92], whose tabling techniques make it possible to keep track of previously computed residues and their subsumption. In this way, redundant computation of residues is avoided, and termination is detected for a wider class of integrity constraints than those presented in [3]. Using XSB also allows a real interaction with the IBM DB2 DBMS.

The query transformation approach to CQAs, as presented in [3,25], has some limitations. First of all, the methodology is designed to handle only universal integrity constraints, whereas existential quantifiers are necessary for specifying referential integrity constraints. Furthermore, as we have shown the transformation-based approach fails (it is sound but not complete) for disjunctive or existentially quantified queries. This failure can be partially explained by complexity-theoretic reasons. Except for very restricted classes of constraints and queries, adding an existential quantifier leads to co-NP-completeness of CQAs. This issue is discussed in more depth in Sect. 5.

4 Specifying Database Repairs

So far we have presented a model-theoretic definition of CQAs and a computational methodology to obtain such answers for some classes of queries and

integrity constraints. Nevertheless, what is still missing is a logical specification Spec_r of *all database repairs* of an instance r, satisfying the following property for all queries Q and tuples $\bar{t}$:

$$\mathit{Spec}_r\ \vdash\ Q(\bar{t}) \quad \equiv \quad r \models_{IC} Q(\bar{t}), \tag{3}$$

where $\vdash$ is a new, suitable consequence relation. If we had such a specification, we could consistently answer every query $Q(\bar{x})$ by asking for those $\bar{t}$ such that $\mathit{Spec}_r\ \vdash\ Q(\bar{t})$.

As the following example shows, $\vdash$ has to be *nonmonotonic*.

Example 13. The database that contains the table

Employee	*Name*	*Salary*
	J.Page	5000
	V.Smith	3000
	M.Stowe	7000

is consistent w.r.t. the FD f_1 of Example 1. In consequence, the set of CQAs to the query $Q(x,y):\ \mathit{Employee}(x,y)$ is

$$\{(J.\mathit{Page},\ 5000), (V.\mathit{Smith},\ 3000), (M.\mathit{Stowe},\ 7000)\}.$$

If we add the tuple $(J.\mathit{Page},\ 8000)$ to the database, the set of CQAs to the same query is reduced to

$$\{(V.\mathit{Smith},\ 3000),\ (M.\mathit{Stowe},\ 7000)\}.$$

□

A specification Spec_r may provide new ways of computing CQAs and shed some light on the computational complexity of this problem.

4.1 Logic Programs

We show here how to specify the database repairs of an inconsistent database r by means of a logic program $\mathit{\Pi}_r$ [4,6]. To pose and answer a first-order query $Q(\bar{x})$, a stratified logic program $\mathit{\Pi}(Q)$ plus a new goal query atom $G(\bar{x})$ is obtained by a standard methodology [81,1], and the query $G(\bar{x})$ is evaluated against the program $\mathit{\Pi}_r \cup \mathit{\Pi}(Q)$. The essential part is the program $\mathit{\Pi}_r$.

The first observation is that when a database is repaired, most of the data *persists*, except for some tuples. More precisely, by default, all the positive and implicit negative data (the latter derived through the Closed World Assumption) persist from r to the repairs, except for some tuples that have to be added or removed to restore the consistency of the database. To capture this idea, we may use *logic programs with exceptions* [70], in this case containing:

- *default rules* capturing the persistence of the data, and
- *exception rules* stating that certain changes have to be made and the integrity of the database has to be restored.

The exception rules should have higher priority than the default rules. The semantics is that of e-answer sets, based on answer set semantics for extended disjunctive logic programs [49]. A logic program with exceptions can be eventually translated into an extended disjunctive normal logic program with answer set semantics [70]. Now, we give an example of this transformed version, where default rules have been replaced by *persistence rules*, so that the whole program has an answer set semantics. (In addition to disjunction, the program has two kinds of negation: classical negation $\neg$ and negation-as-failure *not*.)

Example 14. Consider the full inclusion dependency $\forall x.[\neg P(x) \vee Q(x)]$ and the inconsistent database instance $r = \{P(a)\}$. The program $\mathit{\Pi}_r$ that specifies the repairs of r contains two new predicates, P' and Q', corresponding to the repaired versions of P, Q, respectively, and the following sets of rules:

1. *Persistence rules*:

$$P'(x)\ \leftarrow\ P(x), \mathit{not}\ \neg P'(x); \qquad Q'(x)\ \leftarrow\ Q(x), \mathit{not}\ \neg Q'(x)$$

$$\neg P'(x)\ \leftarrow\ \ \mathit{not}\ P(x), \mathit{not}\ P'(x); \quad \neg Q'(x)\ \leftarrow\ \ \mathit{not}\ Q(x), \mathit{not}\ Q'(x).$$

   The defaults say that all data persists from the original tables to their repaired versions.
2. *Triggering exception*: $\neg P'(x) \vee Q'(x)\ \ \leftarrow\ \ P(x),\ \mathit{not}\ Q(x)$.
   This rule is needed as a first step toward the repair of r. It states that to "locally" repair the constraint, $P(x)$ needs to be deleted or $Q(x)$ inserted.
3. *Stabilizing exceptions*: $Q'(x)\ \leftarrow\ P'(x)$; $\neg P'(x) \leftarrow \neg Q'(x)$.
   The rules say that eventually the constraint has to be satisfied in the repairs. This kind of exception rules is important if there are interacting integrity constraints and local repairs alone are not enough.
4. *Database facts*: $P(a)$.

If we instantiate the rules in all possible ways in the underlying domain, we obtain a ground program $\mathit{\Pi}_r$. A set of ground literals M is an answer set of $\mathit{\Pi}_r$ if it is a minimal model of $\mathit{\Pi}$, where $\mathit{\Pi} = \{A_1 \vee \cdots \vee A_n \leftarrow B_1, \cdots, B_m \mid A_1 \vee \cdots \vee A_n \leftarrow B_1, \cdots, B_m, \mathit{not}\ C_1, \cdots, \mathit{not}\ C_k\ \in \mathit{\Pi}_r$ and $C_i \notin M$ for $1 \leq i \leq k\}$. If M has complementary literals, then M is a trivial answer set containing all ground literals.

In this example, the answer sets of the program correspond to the expected database repairs: $\{\underline{\neg P'(a)}, \neg Q'(a),\ P(a)\}$; $\{P'(a),\ \underline{Q'(a)}, P(a)\}$. The first one indicates through the underlined literal that $P(a)$ has to be deleted from the database; the second one – that $Q(a)$ has to be inserted in the database. □

In [6], it is proved that for the class of binary integrity constraints (defined in Sect. 2), there exists a one-to-one correspondence between answer sets and database repairs. In consequence, in (3) we can take $Spec_r$ as the appropriate extended disjunctive logic program and the notion of logical consequence there as being true w.r.t. all answer sets of the program (i.e. the skeptical answer set semantics).

¿From the correspondence results just mentioned, we can obtain a method to compute database repairs by using any implementation of the answer set semantics for extended disjunctive logic programs. To compute CQAs, one needs to have a way to obtain atoms true in every answer set of the logic program. In [6], experiments with the deductive database system *DLV* [40] are reported. It is also possible to extend the methodology to include referential integrity constraints containing existentially quantified variables [4,6].

The logic programming approach is very general because it applies to arbitrary first-order queries. However, the systems computing answer sets work typically by grounding the logic program. In the database context, this may lead to huge ground programs and be impractical.

Logic programs for repairing databases and computing CQAs w.r.t. arbitrary universal constraints have been independently introduced in [57]. That work is further discussed in Sect. 7.

4.2 Annotated Logics

As explained at the beginning of this section, we would like to have a logical specification of database repairs. Such a specification must contain information about the database and the integrity constraints – two pieces of information that will be mutually inconsistent if the database does not satisfy the integrity constraints. So including them in a classical first-order theory would lead to an inconsistent theory and the trivialization of reasoning. In consequence, if we want a first-order theory, we have to depart from classical logic, moving to nonclassical logic, where reasoning in the presence of classical inconsistencies does not necessarily collapse. Following [8], we show here how to generate a consistent first-order theory with a nonclassical semantics. We use *Annotated Predicate Calculus* (APC) [68].

In APC, database *atoms* are annotated with truth values taken from a truth-value lattice. The most common annotations are: true ($\mathbf{t}$), false ($\mathbf{f}$), contradictory ($\top$), and unknown ($\bot$). In [8], a lattice was used to capture the preference for integrity constraints when they conflict with the data: the integrity constraints cannot be given up but the database can be repaired. These are the new truth values in the lattice:

- *Database values*: $\mathbf{t_d}$ and $\mathbf{f_d}$, used to annotate the atoms in the original database, respectively, outside of it.
- *Constraint values*: $\mathbf{t_c}$ and $\mathbf{f_c}$, used to annotate, depending on their sign, the database literals appearing in the disjunctive normal form of the integrity constraints. The built-in atoms appearing in the integrity constraints are annotated with the classical annotations $\mathbf{t}$ and $\mathbf{f}$.
- *Advisory values*: $\mathbf{t_a}$ and $\mathbf{f_a}$, used to solve the conflicts between the database and the integrity constraints, always in favor of the integrity constraints that are not to be given up, whereas the data is subject to changes. This is represented in the lattice *Latt* in Fig. 1. Intuitively, if a ground atom becomes annotated with both $\mathbf{t_d}$ and $\mathbf{f_c}$, then it gets the value $\mathbf{f_a}$ (the least upper bound of the first two values in the lattice), meaning that the advice is to make it false, as suggested by the integrity constraints, that is, the facts for which the advisory truth values $\mathbf{f_a}$ and $\mathbf{t_a}$ are derived are to be removed from, respectively, inserted into, the database to satisfy the integrity constraints.

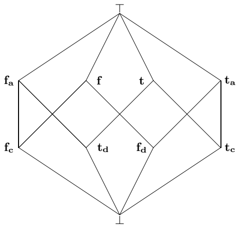


Fig. 1. The truth-value lattice *Latt*

In this lattice, the top element is $\top$, that is reached as the least upper bound (lub) of any pair of contradictory annotations. The annotations $\mathbf{t_d}$ and $\mathbf{f_c}$, for example, are not considered definitely contradictory (i.e. with lub $\top$) if we can still make them compatible by passing to their lub $\mathbf{f_a}$. If there is no conflict between a data and a constraint annotation, then we pass to their lubs, i.e. $\mathbf{t}$ or $\mathbf{f}$.

Now, both the database r and the integrity constraints IC, with the appropriate annotations taken from the lattice, can be embedded into a single and consistent APC theory $Th(r, IC)$. We show this embedding using an example.

Example 15. (Example 1, continued) The integrity constraint

$$\forall x, y, z.[\neg Employee(x, y) \ \vee\ \neg Employee(x, z) \ \vee\ y = z]$$

is translated into

$$\forall x, y, z.[Employee(x, y)\colon \mathbf{f_c} \ \ \vee\ \ Employee(x, z)\colon \mathbf{f_c} \ \ \vee\ \ y = z\colon \mathbf{t}].$$

Each of the database facts is appropriately annotated. For example, the fact $Employee(J.Page, 5000)$ is annotated as $Employee(J.Page, 5000) : \mathbf{t_d}$. We also introduce annotated axioms representing the Unique Names Assumption and the Closed World Assumption [88]. In this way, we generate the annotated first-order theory $Th(r, IC)$. □

As mentioned before, navigation in the lattice and an adequate definition of APC formula satisfaction help solve the conflicts between the database and the integrity constraints. The notion of formula satisfaction in a Herbrand interpretation I (now containing annotated ground atoms) is defined as in classical first-order logic, except for atomic formulas. By definition, for such formulas, $I \models p\!:\!\mathbf{s}$, with $\mathbf{s} \in Latt$, iff for some $\mathbf{s}'$ such that $\mathbf{s} \leq_{Latt} \mathbf{s}'$, $p\!:\!\mathbf{s}' \in I$.

In [8], it is shown that for every database r, there is a one-to-one correspondence between the repairs of r w.r.t. IC and the models of $Th(r, IC)$ that make true a minimal set of atoms annotated with $\mathbf{t_a}$ or $\mathbf{f_a}$ (corresponding to the fact that a minimal set of database atoms is changed). In consequence, the specification $Spec_r$, postulated in (3) at the beginning of this section, is simply $Th(r, IC)$, and the corresponding (nonmonotonic) notion of consequence is truth in all $\{\mathbf{t_a}, \mathbf{f_a}\}$-minimal annotated models of the theory. The approach of [8] produces a set of *advisory clauses* from $Th(r, IC)$. The clauses are then processed by specialized algorithms. The approach is applicable to queries that are conjunctions or disjunctions of positive literals and to universal constraints.

4.3 Logic Programs with Annotation Constants

In [15] a method to obtain a disjunctive logic program $\Pi^{ann}(r, IC)$ from $Th(r, IC)$ is presented. This program, having a stable model semantics, specifies the database repairs. The program has annotations as additional predicate arguments; thus it is a standard, not an annotated [69] logic program, and the standard results and techniques apply to it. We give here an example only.

Example 16. Consider the same database r and integrity constraints IC as in example 14. The logic program should have the effect of repairing the database. Single, local repair steps are obtained, as before, by deriving the annotations $\mathbf{t_a}$ or $\mathbf{f_a}$. This is done when each constraint is considered in isolation, but there may be interacting integrity constraints, and the repair process may take several steps and should stabilize at some point. To achieve this, we need additional, auxiliary annotations $\mathbf{t}^\star$, $\mathbf{f}^\star$, $\mathbf{t}^{\star\star}$, and $\mathbf{f}^{\star\star}$ that are new special constants in the language.

The annotation $\mathbf{t}^\star$, for example, groups together the annotations $\mathbf{t_d}$ and $\mathbf{t_a}$ for the same atom (rules 1 and 4 below). This new, derived annotation can be used to provide 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 (respectively, 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 $\Pi^{ann}(r, IC)$:

1. $P(x, \mathbf{f}^\star) \leftarrow P(x, \mathbf{f_a}). \quad P(x, \mathbf{t}^\star) \leftarrow P(x, \mathbf{t_a}). \quad P(x, \mathbf{t}^\star) \leftarrow P(x, \mathbf{t_d}).$

   $Q(x, \mathbf{f}^\star) \leftarrow Q(x, \mathbf{f_a}). \quad Q(x, \mathbf{t}^\star) \leftarrow Q(x, \mathbf{t_a}). \quad Q(x, \mathbf{t}^\star) \leftarrow Q(x, \mathbf{t_d}).$

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

   $P(x, \mathbf{f}^{\star\star}) \leftarrow P(x, \mathbf{f_a}). \quad P(x, \mathbf{f}^{\star\star}) \leftarrow\ not\ P(x, \mathbf{t_d}),\ not\ P(x, \mathbf{t_a}).$

   $Q(x, \mathbf{t}^{\star\star}) \leftarrow Q(x, \mathbf{t_a}). \quad Q(x, \mathbf{t}^{\star\star}) \leftarrow Q(x, \mathbf{t_d}),\ not\ Q(x, \mathbf{f_a}).$

   $Q(x, \mathbf{f}^{\star\star}) \leftarrow Q(x, \mathbf{f_a}). \quad Q(x, \mathbf{f}^{\star\star}) \leftarrow\ not\ Q(x, \mathbf{t_d}),\ not\ Q(x, \mathbf{t_a}).$

Rule 2 is the only rule dependent on the integrity constraints. It says how to repair the constraint when an inconsistency is detected. If there were other integrity constraints interacting with this constraint, having passed to the annotations $\mathbf{t}^\star$ and $\mathbf{f}^\star$ will allow the system to keep repairing the constraint if it becomes violated due to the repair of a different constraint. Rules 3 contain the database atoms. Rules 4 capture the Closed World Assumption. Rules 5 are denial constraints for coherence, that is, coherent models do not contain atoms annotated with both $\mathbf{t_a}$ and $\mathbf{f_a}$.

Stable models are defined exactly as the answer sets in Example 14, but considering sets of ground atoms only because there is no classical negation in programs with annotations.

The program in this example has two stable models:

$$\{P(a, \mathbf{t_d}), P(a, \mathbf{t}^\star), Q(a, \mathbf{f}^\star), Q(a, \mathbf{t_a}), \underline{P(a, \mathbf{t}^{\star\star})}, Q(a, \mathbf{t}^\star), \underline{Q(a, \mathbf{t}^{\star\star})}\}$$

and

$$\{P(a, \mathbf{t_d}), P(a, \mathbf{t}^\star), P(a, \mathbf{f}^\star), Q(a, \mathbf{f}^\star), \underline{P(a, \mathbf{f}^{\star\star})}, \underline{Q(a, \mathbf{f}^{\star\star})}, P(a, \mathbf{f_a})\};$$

the first one says, through its underlined atoms, that $Q(a)$ is to be inserted into the database; the second one – that $P(a)$ is to be deleted. □

In [15], a one-to-one correspondence between the stable models of the programs $\Pi^{ann}(r, IC)$ and the repairs of r w.r.t. IC is established. Consistent answers can thus be obtained by "running" a query program together with the repair program $\Pi^{ann}(r, IC)$, under the skeptical stable model semantics.

The programs with annotations obtained are simpler than those in Sect. 4.1 in the sense that they contain one change-triggering rule per constraint (rule 2 in the example), whereas the natural extension to arbitrary universal constraints of the approach in Sect. 4.1 may produce programs with the number of rules that is exponential in the number of disjuncts in the disjunctive normal forms of the (universal) integrity constraints [6,57]. The method of [15] can also capture repairs of referential integrity constraints (under the notion of repair allowing tuples with nulls, as discussed in Sect. 2). Thus, the approach in [15] is the most general considered so far, because it applies to arbitrary first-order queries and arbitrary universal or referential integrity constraints (with the exception of the cases that may lead to the violations of the entity integrity constraint; see Example 6).

In some cases, optimizations of the program are possible. For example, the program we just gave is *head-cycle-free* [16]. In consequence, it can be transformed into a nondisjunctive normal program, reducing the complexity of its evaluation from Π_2^p to co-NP [35,75]. Not every repair program with annotations will be head-cycle-free though.

5 Computational Complexity

We summarize here the results about the computational complexity of consistent query answers [3,7,29,30]. We will adopt the *data complexity* assumption [1,66,95] that measures the complexity of the problem as a function of the number of tuples in a given database instance. The given query and integrity constraints are considered fixed.

The query transformation approach [3] – when it terminates – provides a direct way to establish PTIME-computability of consistent query answers. If the original query is first-order, so is the transformed version. In this way, we obtain a PTIME (or, more precisely AC^0) procedure for computing CQAs: transform the query and evaluate it in the original database. Note that the transformation of the query is done independently of the database instance and therefore, does not affect the data complexity. For example, in Example 8 the query $R(x, y)$ will be transformed (similarly to the query in Example 10) to another first-order query and evaluated in PTIME, despite the presence of an exponential number of repairs. However, the query transformation approach is sound, complete and terminating only for restricted classes of queries and constraints. More specifically, the results of [3] imply that for binary denial constraints and full inclusion dependencies, consistent answers can be computed in PTIME for queries that are conjunctions of literals. The logic programming approaches described in Sect. 4 do not have good asymptotic complexity properties because they are all based on Π_2^p-complete classes of logic programs [35]. So it was an open question how far the boundary between tractable and intractable could be pushed in this context.

The paper [29] ([30] is an earlier version containing only some of the results) shows how the complexity of computing CQAs depends on the *type* of the constraints considered, their *number*, and the *size* of the query. Several new classes for which consistent query answers are in PTIME are identified:

- ground queries and arbitrary denial constraints;
- closed simple (without repeated relation symbols) conjunctive queries and functional dependencies with at most one FD per relation;
- ground or closed simple conjunctive queries together with key functional dependencies and foreign key constraints with at most one key per relation.

Additionally, the paper [29] analyzes the data complexity of *repair checking*, the problem of testing whether one database is a repair of another. (The paper [29] assumes that repairs are subsets of the original instance.) It is shown that repair checking is in PTIME for all of the above classes of constraints, as well as for arbitrary FDs together with acyclic INDs. The results obtained are tight in the sense that relaxing any of the above restrictions leads to co-NP-hard problems. (This, of course, does not preclude the possibility that introducing *additional* orthogonal restrictions could lead to more PTIME cases.) To complete the picture, it is shown that for arbitrary sets of FDs and INDs, repair checking is co-NP-complete, and consistent query answering is Π_2^p-complete.

We outline now the proof of the first result listed above because it is done by using a technique different from query transformation. We introduce first the notion of a *conflict hypergraph* that will serve as a succinct representation of all repairs of a given instance.

Definition 5. The *conflict hypergraph* $\mathcal{G}_{F,r}$ is a hypergraph whose set of vertices is the set $\Sigma(r)$ of facts of an instance r and whose set of edges consists of all sets

$$\{P_1(\bar{t}_1), P_2(\bar{t}_2), \ldots P_l(\bar{t}_l)\}$$

such that $P_1(\bar{t}_1), P_2(\bar{t}_2), \ldots P_l(\bar{t}_l) \in \Sigma(r)$, and there is a constraint

$$\forall \bar{x}_1, \bar{x}_2, \ldots \bar{x}_l.\ [\neg P_1(\bar{x}_1)\ \vee\ \neg P_2(\bar{x}_2)\ \vee\ \ldots\ \vee\ \neg P_l(\bar{x}_l)\ \vee\ \phi(\bar{x}_1, \bar{x}_2, \ldots \bar{x}_l)]$$

in F such that $P_1(\bar{t}_1), P_2(\bar{t}_2), \ldots P_l(\bar{t}_l)$ violate together this constraint, which means that there exists a substitution ρ such that $\rho(\bar{x}_1) = \bar{t}_1, \rho(\bar{x}_2) = \bar{t}_2, \ldots \rho(\bar{x}_l) = \bar{t}_l$ and that $\phi(\bar{t}_1, \bar{t}_2, \ldots \bar{t}_l)$ is false.

By an *independent set* in a hypergraph we mean a subset of its set of vertices that does not contain any edge. Clearly, each repair of r w.r.t. F corresponds to a maximal independent set in $\mathcal{G}_{F,r}$.

We prove here that for every set F of denial constraints and ground query Φ, the data complexity of checking whether Φ is consistently true w.r.t. F in an instance r is in PTIME. We assume that the sentence Φ is in CNF, i.e. of the form $\Phi = \Phi_1 \wedge \Phi_2 \wedge \ldots \Phi_l$, where each Φ_i is a disjunction of ground literals. Φ is true in every repair of r if and only if each of the clauses Φ_i is true in every repair. So it is enough to provide a polynomial algorithm that will check if a given ground clause is consistently true.

It is easier to think that we are checking if a ground clause is **not** consistently true. This means that we are checking whether there exists a repair in which $\neg\Phi_i$ is true for some i. But $\neg\Phi_i$ is of the form

$$P_1(\bar{t}_1) \wedge P_2(\bar{t}_2) \wedge \ldots \wedge P_m(\bar{t}_m) \wedge \neg P_{m+1}(\bar{t}_{m+1}) \wedge \ldots \wedge \neg P_n(\bar{t}_n),$$

where the $\bar{t}_j$'s are tuples of constants. (We assume that all facts in the set $\{P_1(\bar{t}_1), \ldots, P_n(\bar{t}_n)\}$ are mutually distinct.)

The nonderministic algorithm selects an edge $E_j \in \mathcal{G}_{F,r}$ for every j, such that $m + 1 \leq j \leq n$, $P_j(\bar{t}_j) \in \Sigma(r)$, and $P_j(\bar{t}_j) \in E_j$, and constructs a set of facts S, such that

$$S = \{P_1(\bar{t}_1), \ldots, P_m(\bar{t}_m)\}\ \cup \bigcup_{m+1 \leq j \leq n, P_j(\bar{t}_j) \in \Sigma(r)} (E_j - \{P_j(\bar{t}_j)\})$$

and *there is no edge* $E \in \mathcal{G}_{F,r}$ *such that* $E \subseteq S$. If the construction of S succeeds, then a repair in which $\neg\Phi_i$ is true can be built by adding to S new facts from $\Sigma(r)$ until the set is maximal independent. The algorithm needs $n - m$ nondeterministic steps, a number that is independent of the size of the database (but dependent on Φ), and in each of its nondeterministic steps, selects one possibility from a set whose size is polynomial in the size of the database. So there is an equivalent PTIME deterministic algorithm.

6 Aggregation Queries

So far we have considered only first-order queries, but in databases, aggregation queries are also important. In fact, aggregation is essential in scenarios, like data warehousing, where inconsistencies are likely to occur, and keeping inconsistent data may be useful. Only some aggregation queries, for example, computing a maximum or minimum value of an attribute in a relation can be expressed as first-order queries. Even in this case, due to its syntax, the resulting first-order query cannot be handled by the query transformation methodology described earlier.

We will consider here a restricted scenario: the integrity constraints will be limited to functional dependencies, and the aggregation queries will consist of single applications of one of the standard SQL-2 aggregation operators (`MIN`, `MAX`, `COUNT(*)`, `COUNT(A)`, `SUM`, and `AVG`). Even in this case, it was shown [5] that computing consistent query answers to aggregation queries is a challenging problem for both semantic and complexity-theoretic reasons.

Example 17. Consider again the instance r of *Employee* from Example 1. It is inconsistent w.r.t. the FD f_1: *Name* → *Salary*.

Employee	*Name*	*Salary*
	J.Page	5000
	J.Page	8000
	V.Smith	3000
	M.Stowe	7000

The repairs are

Employee1	*Name*	*Salary*
	J.Page	5000
	V.Smith	3000
	M.Stowe	7000

Employee2	*Name*	*Salary*
	J.Page	8000
	V.Smith	3000
	M.Stowe	7000

If we pose the query

```
SELECT MIN(Salary) FROM Employee
```

we should get 3000 as a consistent answer: `MIN(Salary)` returns 3000 in each repair. Nevertheless, if we ask

```
SELECT MAX(Salary) FROM Employee
```

then the maximum, 8000, comes from a tuple that participates in the violation of f_1. Actually, `MAX(Salary)` returns a different value in each repair: 7000 or 8000. Thus, there is no consistent answer in the sense of Definition 4. □

We give a new, slightly weakened definition of a consistent answer to an aggregation query that addresses the above difficulty.

Definition 6. [5] (a) A consistent answer to an aggregation query Q in a database instance r w.r.t. a set of integrity constraints F is the minimal interval $I = [a, b]$ such that for every repair r' of r w.r.t. F, the scalar value $Q(r')$ of query Q in r' belongs to I.
(b) The left and right end-points of the interval I are the *greatest lower bound* (glb) and *least upper bound* (lub), respectively, answers to Q in r. □

According to this definition, in Example 17, the interval $[7000, 8000]$ is the consistent answer to the query

```
SELECT MAX(Salary) FROM Employee
```

and 7000 and 8000 are the *glb-answer* and *lub-answer*, respectively. Notice that the consistent query answer interval represents in a *succinct* form a superset of the values that the aggregation query can take in all possible repairs of the database r w.r.t. a set of FDs. The representation of the interval is always polynomially sized because the numeric values of the end-points can be represented in binary.

Example 18. Along the lines of Example 8, consider the functional dependency $A \to B$ and the following family of relation instances S_n, $n > 0$:

r_n									
A	a_1	a_1	a_2	a_2	a_3	a_3	$\cdots$	a_n	a_n
B	0	1	0	2	0	4	$\cdots$	0	2^n

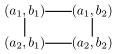
The aggregation query `SUM(B)` takes all of the exponentially many values between 0 and $2^{n+1} - 1$ in the (exponentially many) repairs of the database [5]. An explicit representation of the possible values of the aggregation function would then be exponentially large. Moreover, it would violate the 1NF assumption. On the other hand, the interval representation is of polynomial size. □

Next, we consider the complexity of the problem of computing the *glb-* and *lub-answers*. The complexity results are given in terms of data complexity [1,66,95]. To classify the problems of consistent answering to different aggregation queries in terms of complexity and to find polynomial time algorithms in tractable cases, it is useful to use a graph representation of the set of all repairs. Because we are dealing with functional dependencies, we can specialize the notion of *conflict hypergraph* (Definition 5) to that of a *conflict graph* (edges contain two vertices).

Example 19. Consider the schema $R(AB)$, a set F of two functional dependencies $A \to B$ and $B \to A$, and the inconsistent instance

$$r = \{(a_1, b_1), (a_1, b_2), (a_2, b_2), (a_2, b_1)\}$$

over this schema. The following is the conflict graph $\mathcal{G}_{F,r}$:



In this graph, the two maximal independent sets $\{(a_1, b_1), (a_2, b_2)\}$ and $\{(a_1, b_2), (a_2, b_1)\}$ correspond to the two possible repairs of the database. □

The paper [7] contains a complete classification of tractable and intractable cases of the problem of computing consistent query answers (in the sense of Definition 6) to aggregation queries. Its results can be summarized as follows:

- For all the aggregate operators except `COUNT(A)`, the problem is in PTIME if the set of integrity constraints contains at most one nontrivial FD.

- For `COUNT(A)`, the problem is NP-complete, even for one nontrivial FD (one can encode the HITTING SET problem [48]).
- For more than one nontrivial FD, even the problem of checking whether the glb-answer to a query is $\leq k$ (respectively, the problem of checking whether the lub-answer to a query is $\geq k$) is NP-complete.

For aggregate operators `MIN`, `MAX`, `COUNT(*)`, and `SUM` and a single FD, the glb- and lub-answers are computed by SQL2 queries (so this is in a sense an analogue of the query transformation approach for first-order queries discussed earlier). For `AVG`, however, the PTIME algorithm is iterative and cannot be formulated in SQL2.

Example 20. Continuing Example 17, the greatest lower bound answer to the query

```
SELECT MAX(Salary) FROM Employee
```

is computed by the following SQL2 query

```
SELECT MAX(C) FROM
  (SELECT MIN(Salary) AS C
   FROM Employee
   GROUP BY Name).
```

□

In [5,7], some special properties of conflict graphs were identified, paving the way to more tractable cases. For example, for two FDs and the relation schema in Boyce–Codd Normal Form, the conflict graphs are claw-free and perfect [22], and computing lub-answers to `COUNT(*)` queries can be done in PTIME.

Given the intractability results, it seems appropriate to find approximations to consistent answers to aggregation queries. Unfortunately, "maximal independent set" seems to have bad approximation properties [61].

7 Related Work

We discuss here related work on dealing with inconsistency in artificial intelligence, databases and logic programming. We will attempt to characterize various approaches along several common dimensions, including

- *Semantics:* What is the underlying notion of inconsistency? Are the notions of repair and consistent query answer supported in any sense?
- *Scope:* What classes of databases, integrity constraints, and queries can be handled?
- *Computational mechanisms:* How is consistent information obtained in the presence of inconsistency?

- *Computational complexity.*

To be able to delineate the scope of different approaches, one has to observe whether they are first-order or propositional, and if they are first-order – whether they can be *reduced* to propositional. The approaches presented in this chapter so far are first-order. However, they can be reduced to the propositional case for universal integrity constraints and ground queries because the constraints themselves can be grounded using the constants in the database and the query. For more general classes of queries and constraints, for example, referential integrity constraints, such a reduction does not apply. Moreover, the approaches which do not require grounding, for example, query transformation in Sect. 3, are preferable from the efficiency point of view.

A specific dimension of the computational mechanisms under consideration is support for the *locality* of inconsistency. Locality in our context means that the consistency violations that are irrelevant to a given query are ignored in the process of obtaining consistent answers to the query. Clearly, locality is desirable, but it is supported by only a few approaches. The query transformation approach (Sect. 3) supports locality because the violations occurring in the relations not mentioned in the transformed query are irrelevant for evaluating this query and are ignored. The approaches based on some form of specification of all repairs (Section 4) do not support locality because they require resolving all violations by constructing all answer sets (or minimal models). The algorithm described in Sect. 5 is based on constructing the conflict hypergraph of the given instance and though it does not resolve all conflicts, it has to detect them all. Other PTIME algorithms mentioned in that section support locality. It seems possible to refine the nonlocal approaches mentioned above to obtain locality.

7.1 Belief Revision and Update

Semantically, our approach to inconsistency handling corresponds to some of the approaches followed by the belief revision/update community [46,47]. Database repairs (Definition 2) coincide with revised models defined by Winslett [97]. Both use the same notion of minimality. Comparing our framework with that of belief revision, we have an empty domain theory, one model: a database instance, and a revision by a set of integrity constraints. The revision of a database instance by the integrity constraints produces new database instances – repairs of the original database. The scenario adopted by most belief revision papers is thus more general than ours because such papers typically assume that it is a formula (or, equivalently, the set of its models) that is undergoing the revision, and that the domain theory is nonempty. On the other hand, the research on belief revision is typically limited to the propositional case.

Our implicit notion of revision satisfies the postulates (R1) – (R5),(R7) and (R8) introduced by Katsuno and Mendelzon [67]. Dalal [33] postulated a different notion of revision, based on minimizing the cardinality of the set of changes, as opposed to minimizing the set of changes under set inclusion [3,97]. In [6] it is shown how to capture repairs under Dalal's notion of revision by logic programs for consistent query answering.

The belief revision community has adopted a notion of inference called *counterfactual inference* [46] that corresponds to our notion of a formula being consistently true. Counterfactual inference is based on the *Ramsey test* for conditionals: a formula $\beta > \gamma$ is a counterfactual consequence of a set of beliefs K if for every closest context in which K is revised so that β is true, γ is also true. In our case, K is a database, β is the set of integrity constraints, and γ is the query.

Winslett's approach [97] is mainly propositional, but a preliminary extension to the first-order ground case can be found in [31]. Those papers concentrate on the computation of the models of the revised theory, i.e. the repairs in our case. Inference or query answering is not addressed. The complexity of belief revision and counterfactual inference was exhaustively classified by Eiter and Gottlob [41]. They deal with the propositional case only. We have outlined above how to reduce – in some cases – consistent query answering to the propositional case by grounding. However, grounding of integrity constraints results in an update formula which is *unbounded*, i.e. whose size depends on the size of the database. This prevents the transfer of any of the PTIME upper bounds from [41] into our framework. Similarly, the lower bounds from [41] require kinds of formulas different from those that we use. The classic paper on updating logical theories by Fagin et al. [42] focuses on the semantics of updates but does not address computational issues. Moreover, the proposed framework is also limited to the propositional case. It is interesting that [42] proposes yet another notion of *repair minimality* by giving priority to minimizing deletions over minimizing insertions.

The approaches pursued by the belief revision community are nonlocal in the sense of having to resolve all inconsistencies in the database, even those that are irrelevant to the query.

7.2 Reasoning in the Presence of Inconsistency

There are many approaches to handling inconsistency in the literature.[1] Many of them have been proposed by the logic community; the most prominent is the family of *paraconsistent logics* [32,62]. Such logics protect reasoning from *triviality* (the property that an inconsistent theory entails every formula) in the presence of classical inconsistencies . Their applicability in the context of inconsistent databases is, however, limited. First, they typically do not address the issue of the special role of integrity constraints whose truth cannot be given up during the inference process. Second, most paraconsistent logic are monotonic and thus fail to capture the nonmonotonicity inherent in the

[1] For recent collections of papers, see [17,37].

notion of a consistent query answer (Example 13). Third, they are mostly nonlocal. Below we discuss those paraconsistency-based approaches that are closest to ours.

Bry [24] was, to our knowledge, the first author to consider the notion of a consistent query answer in inconsistent databases. He defined consistent query answers using provability in minimal logic. The proposed inference method is nonmonotonic but fails to capture minimal change (thus Bry's notion of a consistent query answer is weaker than ours). Moreover, Bry's approach is entirely proof-theoretic and does not provide a computational mechanism to obtain consistent answers to first-order queries. Other formalisms, for example, [79], are also limited to propositional inference. Moreover, they do not distinguish between integrity constraints and database facts. Thus, if the data in the database violates an integrity constraint, the constraint itself can no longer be inferred (which is not acceptable in the database context).

Example 21. Assume that the integrity constraint is $(\neg p \vee \neg q)$ and the database contains the facts p and q. In the approach of Lin [79], $p \vee q$ can be inferred (minimal change is captured correctly), but p, q and $(\neg p \vee \neg q)$ can no longer be inferred (they are all involved in an inconsistency). □

Several papers by Lozinskii, Kifer, Arieli, and Avron [9,68,82] studied the problem of making inferences from a possibly inconsistent, propositional, or first-order knowledge base. The basic idea is to infer the classical consequences of all maximal consistent subsets of the knowledge base [82] or all *most consistent* models of the knowledge base [9,68] (where the order on models is defined on the basis of atom annotations drawing values from a lattice or a bilattice). This provides a nonmonotonic consequence relation but the special role of integrity constraints (whose truth cannot be given up) is not captured. Also, no computational mechanisms for answering first-order (or aggregation) queries are proposed, nor are computational complexity issues addressed. In section 4, we described how the approach of Kifer and Lozinskii [68] can be adapted to the task of computing consistent query answers.

In [36], a logical framework based on a three-valued logic is used to distinguish between consistent and inconsistent (controversial) information. A database instance is a finite set of tuples, each tuple associated with the value 1 (safe), 0 (false, does not need to be stored) or $\frac{1}{2}$ (controversial). Integrity constraints are expressed in a first-order language and have three-valued semantics. A repair J of I is an instance satisfying a set of integrity constraints IC, which is $\leq_I$-minimal among all the instances satisfying IC, where $\leq_I$ is defined as follows. The distance between I and J is the sum over all tuples u of $|I(u) - J(u)|$, where $I(u)$ and $J(u)$ are the values associated with u in I and J, respectively. Then, $J \leq_I K$ if the distance between I and J is less than or equal to the distance between I and K. Furthermore, in [36], an algorithm for computing repairs is introduced. This algorithm is based on the tableau proof system for the three-valued logic used in the framework.

A related approach of Arieli et al. [10] introduces executable specifications of repairs using abductive logic programming [65]. In both approaches, however, no notion analogous to consistent query answers is proposed, and no complexity analysis is provided.

Pradhan [86,87] introduced a logic for reasoning in the presence of *contestations* that are conflicts of different kinds: logical, semantical, domain dependent, etc. They are declared together with a domain specification which is, for example, a logical theory or a normal logic program. The logic has a non-classical, four-valued semantics that allows inferring conflict-free sets of consequences. For example, if conflicts have been declared as classical logical conflicts, no logical contradiction will be found in the set of consequences. A deductive evaluation mechanism is developed for ground queries that are strong consequences of the specification, i.e. that hold in all conflict-free models. Furthermore, it is also shown how to represent – as a set of conflicts – the inconsistency of a deductive database w.r.t. a set of integrity constraints. An interesting approach to integrity constraints in databases is taken: the constraints should restrict the possible answers one can get from the database, rather than capture the semantics of the domain or restrict the states of the database. This view is quite compatible with the approach in [3] and could be used as another motivation for it. However, the general deductive system for strong consequences is not explicitly applied nor specialized to consistent (conflict-free) query answering in databases. Complexity issues are not addressed.

Further related treatments of inconsistency have been developed in the areas of knowledge representation [44] and formal specifications in software engineering [12,85].

7.3 Databases

The approaches discussed here and in the next subsection are applicable to relational databases and to first-order queries and integrity constraints.

Asirelli et al. [11] treat integrity constraints as views over a deductive database. In that way, queries can be answered "through the views," so that the resulting answers satisfy the integrity constraints and answers that do not satisfy them are filtered out. This approach is the closest to the transformation-based approach presented in Sect. 3 and also supports locality. However, the approach [11] is a deductive, resolution-based, direct query answering method, similar to the approaches to query answering in deductive databases in the presence of integrity constraints [71,91]. Moreover, queries are restricted to be conjunctions of literals. No computational complexity issues are addressed.

Wijsen [96] studies the problem of consistent query answering in the context of universal constraints. In contrast to Definition 2, he considers repairs obtained by modifying individual tuple components. Notice that a modification of a tuple component cannot necessarily be simulated as a deletion

followed by an insertion, because this might not be minimal under set inclusion. Wijsen proposes to represent all repairs of an instance by a single *trustable tableau*. From this tableau, answers to conjunctive queries can be efficiently obtained. It is not clear, however, what is the computational complexity of constructing the tableau, or even whether the tableau is always of polynomial size.

Franconi et al. [43] also discuss repairs based on updating individual values in the context of a data cleaning application. The aim is to compute all possible repairs, in this case of a particular kind of databases storing census data, rather than consistent query answering. The issues addressed consist of detecting and solving conflicts inside the database and conflicts between answers to questionnaires and the intended declarative semantics of the latter, as opposed to conflicts between data and integrity constraints. This work is a specific case of *data cleaning* [45].

It has been widely recognized that in database integration, the integrated data may be inconsistent with the integrity constraints. A typical (theoretical) solution to the problem of database inconsistency in this context is augmenting the data model to represent disjunctive information. Different disjuncts correspond to different ways of resolving an inconsistency. The following example explains the need for a solution of this kind.

Example 22. Consider the functional dependency *"every person has a single salary"* in Example 1. It is violated by the first two tuples. Each of those tuples may be coming from a different data source that satisfies the dependency. Thus, both tuples are replaced by their disjunction

$$Employee(J.Page, 5000) \vee Employee(J.Page, 8000)$$

in the integrated database. Now, the functional dependency is no longer violated. □

To solve this kind of problem, Agarwal et al. [2] introduced the notion of *flexible relation*, a non-1NF relation that contains tuples with sets of nonkey values (where such a set stands for *one* of its elements). This approach is limited to primary key functional dependencies and was subsequently generalized to other key functional dependencies by Dung [38]. In the same context, Baral et al. [13,54] proposed to use disjunctive Datalog, and Lin and Mendelzon [80] tables with OR-objects [63,64]. Agarwal et al. [2] introduced flexible relational algebra to query flexible relations, and Dung [38] introduced flexible relational calculus (a proper subset of the calculus can be translated to flexible relational algebra). The remaining papers did not discuss query language issues, relying on the existing approaches to query disjunctive Datalog or tables with OR-objects.

There are several important differences between the above approaches and ours. First, they rely on the construction of a single (disjunctive) instance and the deletion of conflicting tuples. The integrity constraints are

used solely for conflict resolution. However, all conflicts need to be resolved, and thus the above approaches are nonlocal. In our transformation-based approach, the underlying databases are incorporated into the integrated one in toto, without any changes. There is no need for introducing disjunctive information. The integrity constraints are used only during querying. Second, the above approaches do not generalize to arbitrary functional dependencies and other kinds of integrity constraints. Imielinski et al. [64] provide a comprehensive characterization of the computational complexity of evaluating conjunctive queries in databases with OR-objects. Those results carry over into our framework only in very limited cases, as discussed in [7,29].

Motro [84] addressed the issue of integrating data from possibly mutually inconsistent sources in a fashion different from the above and closer to our approach. He proposed, among others, the notion of *sound* query answers – the answers present in the query result in every source. For functional dependencies and single-literal queries, every sound answer (in Motro's sense) is a consistent answer (in our sense). However, the converse is not true, since a tuple that appears only in a single source will not be a sound answer, although it is a consistent answer if it does not conflict with any other tuple. Also, for general denial constraints, there may be sound answers that are not consistent. The computational mechanism proposed in [84] consists of simply taking the intersection of the query answers in individual sources, and thus it is local. No complexity analysis is provided.

Gertz [50,51] described techniques based on model-based diagnosis for detecting causes of inconsistencies in databases and computing the corresponding repairs. However, he did not address the issue of query answering in the presence of an inconsistency.

Cholvy [28] introduced a deductive approach based on modal logic that allows limiting the impact of inconsistent information that is related to a query. The logic distinguishes *sure* and *doubtful* information. From the original inconsistent deductive database that includes integrity constraints, a new database consisting of modal formulas is constructed. There are modalities S and D for the *sure* and *doubtful* formulas. Then, the idea is to derive the sure answers from the deductive system, so query processing consists of constructing a proof in an appropriate modal logic. Integrity constraints are considered at the same level of reliability as data, and in consequence, they could be considered "doubtful." No complexity analysis is provided.

7.4 Logic Programming

Greco et al. [57,59] independently developed a logic-programming-based approach to inconsistency handling in databases, alternative to those presented in Sect. 4.1 and 4.2. In that approach, disjunctive logic programs with stable model semantics are used to specify the sets of changes that lead to database repairs in the sense of [3]. The authors present a general solution based on a compact schema for generating repair programs for universal integrity constraints. The application of such a schema leads to rules whose heads involve essentially all possible disjunctions of database literals that occur together in a constraint. Thus, a single constraint can produce exponentially many clauses. The approach of [4,6] can be generalized to nonbinary constraints along the same lines. (In contrast to [4,6,57,59], the approach of [15] does not lead to an exponential blowup.) The approach of [57] is concentrated mainly on producing the sets of changes, rather than the repaired databases explicitly. In particular, there are no persistence rules in the generated program. In consequence, the program cannot be directly used to obtain consistent query answers. An additional contribution of [57] is the notion of *repair constraints* that specify preferences for certain kinds of repairs (for example, deletion over insertion).

Another approach to database repairs based on logic programming semantics consists of *revision programs* proposed by Marek and Truszczynski [83]. The rules in those programs explicitly declare how to enforce the satisfaction of an integrity constraint, rather than explicitly stating the integrity constraints, for example,

$$in(a) \leftarrow in(a_1) \ldots, in(a_k), out(b_1), \ldots, out(b_m)$$

has the intended procedural meaning of inserting the database atom a whenever $a_1, \ldots, a_k$ but not $b_1, \ldots, b_m$ are in the database. Also a declarative, stable model semantics is given to revision programs (thus also providing a computational mechanism). Preferences for certain kinds of repair actions can be captured by including the corresponding rules in the revision program and omitting the rules that could lead to other forms of repairs. No notion analogous to consistent query answers is proposed.

There are several proposals for language constructs specifying nondeterministic queries that are related to our approach: *witness* [1] and *choice* [52,53,58]. Essentially, the idea is to construct a maximal subset of a given relation that satisfies a given set of functional dependencies. Because there is usually more than one such subset, the approach yields nondeterministic queries in a natural way. Clearly, maximal consistent subsets (choice models [52]) correspond to repairs of Definition 2. Stratified Datalog with choice [52] combines enforcing functional dependencies with inference using stratified Datalog programs. Answering queries in all choice models ($\forall G$-queries [58]) corresponds to our notion of computation of consistent query answers for first-order queries (Definition 4). However, in [58], the former problem is shown co-NP-complete and no tractable cases are identified. One of the sources of complexity in this case is the presence of intensional relations defined by Datalog rules. Such relations are absent from our approach. Moreover, the procedure proposed in [58] runs in exponential time if there are exponentially many repairs, as in Example 8. Also, only conjunctions of literals are considered as queries in [58]. Arbitrary first-order or aggregation queries are not studied. Neither is the approach generalized beyond functional dependencies.

Blair and Subrahmanian [21] introduced paraconsistent logic programming. Paraconsistent logic programs have nonclassical semantics, inspired by paraconsistent first-order semantics. Kifer and Subrahmanian [69] discussed annotated logic programs with lattice-based, nonclassical semantics. Atoms in clauses have annotations, as in [68], but now they may also contain variables and functions, providing a stronger representation formalism. The implementation of annotated logic programs and query answering mechanisms are discussed in [72]. Subrahmanian [93] further generalized annotated programs to use them for amalgamating databases by resolving potential conflicts between integrated data. For this purpose, the product of the lattices underlying each database is constructed as the semantic basis for the integrated database. Conflict resolutions and preferences are captured by means of function-based annotations. Other approaches to paraconsistent logic programming are discussed in [34]. These works do not define notions analogous to repairs and consistent query answers.

8 Open Problems and Ongoing Work

8.1 Flexible Repairs

The existing notions of a consistent query answer [3–5,57,59,73] are based on the notion of database repair from [3]. Database repairs of an inconsistent database instance are new instances that satisfy the integrity constraints but differ from the original instance by a minimal set of *whole* database tuples, where minimality is understood under set inclusion. In Sect. 2, we mentioned some alternative notions of repair. In particular, in some situations, it may be more natural to consider more *flexible* repairs that allow modifications of individual tuple components [96].

Other alternatives, independently of how repairs are defined, should consider more flexibility w.r.t. the class of all repairs, for example, considering an answer consistent if it is true in the majority of the database repairs, or true in some preferred repairs, under some predefined notion of preference. Majority-based approaches to consistency have been studied in [80] and [82] in the context of data integration. The whole issue of preferences for certain changes and repairs still remains to be investigated. Some work in this direction is presented in [57].

8.2 Data Integration

Assume that we have a collection of (materialized) data sources $S_1, \ldots, S_n$, and a global, virtual database G that integrates data from $S_1, \ldots, S_n$. According to the *local-as-view* approach [76,84,94], we can look at the data sources, S_i, as *views* of the global schema G. Now, given a query Q to G, one can generate a *query plan* that extracts the information from the sources [55,76–78]. In the global-as-view approach [74], the global database is defined as a view over the data sources.

Sometimes one assumes that certain integrity constraints hold in the global system, and those integrity constraints are used in generating the query plan; actually, there are situations where without integrity constraints, no query plan can be generated [39,56,60]. The problem is that we cannot be sure that such global integrity constraints hold. Even in the presence of consistent data sources, a global system that integrates them may become inconsistent. The global integrity constraints are not maintained and could easily be violated. In consequence, data integration is a natural scenario to apply the methodologies presented before. What we have to do is to retrieve consistent information from the global system.

Several new interesting issues appear, among them are (a) What is a consistent answer in this context? (b) If we are going to base this notion on a notion of repair, what is a repair? Notice that we do not have global *instances* to repair. (c) How can the consistent answers be retrieved from the global systems? What kind of query plans do we need? These and other issues are addressed in [18,23] for the local-as-view approach and in [73] for the global-as-view approach.

8.3 Other Problems

An important achievement of this line of research would be integrating the mechanisms for retrieving consistent query answers with a full-fledged DBMS. In such a system, it should be possible to specify, in SQL, *soft* integrity constraints (constraints that are not explicitly maintained) and pose the usual SQL queries. The consistent answers to those queries would be obtained by an enhanced SQL engine. Note that different users, having different perceptions, could specify different soft constraints.

So far, we have developed our notions of consistent answer and repair in the context of relational databases. Nevertheless, it would be interesting to extend these notions and the corresponding computational mechanisms to other kinds of databases: semistructured, deductive, spatiotemporal, etc.

Acknowledgments

This paper is dedicated to the memory of Ray Reiter whose pioneering work on the logical foundations of databases laid the foundation for and influenced many research efforts in that area, including our own.

The contributions of the following researchers are gratefully acknowledged: Marcelo Arenas, Pablo Barcelo, Alexander Celle, Alvaro Cortes, Claudio Gutierrez, Xin He, Michael Kifer, Jerzy Marcinkowski, Francisco Orchard, Vijay Raghavan, Camilla Schwind, and Jeremy Spinrad. The anonymous reviewers are thanked for their detailed and helpful comments.

This research was supported by FONDECYT Grant 1000593, joint NSF Grant INT-9901877/CONICYT Grant 1998-02-083, NSF Grant IIS-0119186, Carleton University Start-Up Grant 9364-01, and NSERC Grant 250279-02.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, Reading, Mass., 1995.
2. S. Agarwal, A. M. Keller, G. Wiederhold, and K. Saraswat. Flexible Relation: An Approach for Integrating Data from Multiple, Possibly Inconsistent Databases. In *11th IEEE International Conference on Data Engineering*, 1995.
3. M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *18th ACM Symposium on Principles of Database Systems*, pp. 68–79, 1999.
4. M. Arenas, L. Bertossi, and J. Chomicki. Specifying and Querying Database Repairs Using Logic Programs with Exceptions. In *4th International Conference on Flexible Query Answering Systems*, pp. 27–41. Springer-Verlag, 2000.
5. M. Arenas, L. Bertossi, and J. Chomicki. Scalar Aggregation in FD-Inconsistent Databases. In *8th International Conference on Database Theory*, pp. 39–53. Springer-Verlag, LNCS 1973, 2001.
6. M. Arenas, L. Bertossi, and J. Chomicki. Answer Sets for Consistent Query Answering in Inconsistent Databases. Technical Report arXiv:cs.DB/0204094, arXiv.org e-Print archive, July 2002. To appear in *Theory and Practice of Logic Programming.*
7. M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science*, 296(3): 405–434, 2003. Special issue: selected papers from ICDT 2001.
8. M. Arenas, L. Bertossi, and M. Kifer. Applications of Annotated Predicate Calculus to Querying Inconsistent Databases. In *International Conference on Computational Logic*, pp. 926–941. Springer-Verlag, LNCS 1861, 2000.
9. O. Arieli and A. Avron. A Model-Theoretic Approach for Recovering Consistent Data from Inconsistent Knowledge Bases. *Journal of Automated Reasoning*, 22(2):263–309, 1999.
10. O. Arieli, M. Denecker, B. Van Nuffelen, and M. Bruynooghe. Repairing Inconsistent Databases: A Model-Theoretic Approach. In Decker et al. [37].
11. P. Asirelli, P. Inverardi, and G. Plagenza. Integrity Constraints as Views in Deductive Databases. In S. Conrad, H-J. Klein, and K-D. Schewe, editors, *International Workshop on Foundations of Models and Languages for Data and Objects*, pp. 133–140, September 1996.
12. B. Balzer. Tolerating Inconsistency. In *13th International Conference on Software Engineering*, pp. 158–165. IEEE Computer Society Press, 1991.
13. C. Baral, S. Kraus, J. Minker, and V. S. Subrahmanian. Combining Knowledge Bases Consisting of First-Order Theories. *Computational Intelligence*, 8:45–71, 1992.
14. P. Barcelo and L. Bertossi. Repairing Databases with Annotated Predicate Logic. In S. Benferhat and E. Giunchiglia, editors, *Ninth International Workshop on Non-Monotonic Reasoning (NMR02), Special Session: Changing and Integrating Information: From Theory to Practice.*, pages 160–170, 2002.

15. P. Barcelo and L. Bertossi. Logic Programs for Querying Inconsistent Databases. In *International Symposium on Practical Aspects of Declarative Languages*, pp. 208–222. Springer-Verlag, LNCS 2562, 2003.
16. R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1994.
17. L. Bertossi and J. Chomicki, editors. *Working Notes of the IJCAI'01 Workshop on Inconsistency in Data and Knowledge*. AAAI Press, 2001.
18. L. Bertossi, J. Chomicki, A. Cortes, and C. Gutierrez. Consistent Answers from Integrated Data Sources. In *5th International Conference on Flexible Query Answering Systems*, pp. 71–85, Springer-Verlag, LNCS 2522, 2002.
19. L. Bertossi and C. Schwind. Analytic Tableaux and Database Repairs: Foundations. In *2nd International Symposium on Foundations of Information and Knowledge Systems*, pp. 32–48. Springer-Verlag, LNCS 2284, 2002.
20. L. Bertossi and C. Schwind. Database Repairs and Analytic Tableaux. Technical Report arXiv:cs.DB/0211042, arXiv.org e-Print archive, November 2002. To appear in a special issue of Annals of Mathematics and Artificial Intelligence.
21. H. Blair and V. S. Subrahmanian. Paraconsistent Logic Programming. *Theoretical Computer Science*, 68(2):135–154, 1989.
22. A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey*. SIAM, 1999.
23. L. Bravo and L. Bertossi. Logic Programs for Consistently Querying Data Integration Systems. In *International Joint Conference on Artificial Intelligence*, 2003. To appear.
24. F. Bry. Query Answering in Information Systems with Integrity Constraints. In *IFIP WG 11.5 Working Conference on Integrity and Control in Information Systems*. Chapman &Hall, 1997.
25. A. Celle and L. Bertossi. Querying Inconsistent Databases: Algorithms and Implementation. In *International Conference on Computational Logic*, pp. 942–956. Springer-Verlag, LNCS 1861, 2000.
26. U. S. Chakravarthy, J. Grant, and J. Minker. Logic-Based Approach to Semantic Query Optimization. *ACM Transactions on Database Systems*, 15(2):162–207, 1990.
27. A. Chandra and P. Merlin. Optimal Implementation of Conjunctive Queries in Relational Databases. In *9th ACM SIGACT Symposium on the Theory of Computing*, pp. 77–90, 1977.
28. L. Cholvy. Querying an Inconsistent Database. In *Proceedings of Artificial Intelligence : Methodology, Systems and Applications (AIMSA)*. North-Holland, 1990.
29. J. Chomicki and J. Marcinkowski. Minimal-Change Integrity Maintenance Using Tuple Deletions. Technical Report cs.DB/0212004, arXiv.org e-Print archive, December 2002.
30. J. Chomicki and J. Marcinkowski. On the Computational Complexity of Consistent Query Answers. Technical Report arXiv:cs.DB/0204010, arXiv.org e-Print archive, April 2002.
31. T. Chou and M. Winslett. A Model-Based Belief Revision System. *Journal of Automated Reasoning*, 12:157–208, 1994.
32. N. C. A. da Costa, J-Y. Beziau, and O. Bueno. Paraconsistent Logic in a Historical Perspective. *Logique & Analyse*, 150/151/152:111–125, 1995.
33. M. Dalal. Investigations into a Theory of Knowledge Base Revision. In *7th National Conference on Artificial Intelligence*, August 1988.

34. C. V. Damasio and L. M. Pereira. A Survey of Paraconsistent Semantics for Extended Logic Programs. In Gabbay and Smets [44], pp. 241–320.
35. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001.
36. S. de Amo, W. A. Carnielli, and J. Marcos. A Logical Framework for Integrating Inconsistent Information in Multiple Databases. In *2nd International Symposium on Foundations of Information and Knowledge Systems*, pp. 67–84, 2002.
37. H. Decker, J. Villadsen, and T. Waragai, editors. *ICLP 2002 Workshop on Paraconsistent Computational Logic (PCL)*, July 2002.
38. Phan Minh Dung. Integrating Data from Possibly Inconsistent Databases. In *1st International Conference on Cooperative Information Systems*, Brussels, Belgium, 1996.
39. O. M. Duschka, M. R. Genesereth, and A. Y. Levy. Recursive Query Plans for Data Integration. *Journal of Logic Programming*, 43(1):49–73, 2000.
40. T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative Problem-Solving in DLV. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pp. 79–103. Kluwer, 2000.
41. T. Eiter and G. Gottlob. On the Complexity of Propositional Knowledge Base Revision, Updates, and Counterfactuals. *Artificial Intelligence*, 57(2-3):227–270, 1992.
42. R. Fagin, J. D. Ullman, and M. Y. Vardi. On the semantics of updates in databases. In *2nd ACM Symposium on Principles of Database Systems*, 1983.
43. E. Franconi, A. L. Palma, N. Leone, S. Perri, and F. Scarcello. Census Data Repair: a Challenging Application of Disjunctive Logic Programming. In *9th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, pp. 561–578. Springer-Verlag, LNCS 2250, 2002.
44. D. M. Gabbay and P. Smets, editors. *Handbook of Defeasible Reasoning and Uncertain Information*, Vol. 2. Kluwer, Dordrecht, The Netherlands, 1998.
45. H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C-A. Saita. Declarative Data Cleaning: Language, Model, and Algorithms. In *27th International Conference on Very Large Data Bases*, pp. 371–380, 2001.
46. P. Gärdenfors. *Knowledge in Flux*. MIT Press, Cambridge, Mass., 1990.
47. P. Gärdenfors and H. Rott. Belief Revision. In D. M. Gabbay, J. Hogger, C, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, Vol. 4, pp. 35–132. Oxford University Press, 1995.
48. M. R. Garey and D. S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.
49. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9(3/4):365–386, 1991.
50. M. Gertz. A Extensible Framework for Repairing Constraint Violations. In S. Conrad, H-J. Klein, and K-D. Schewe, editors, *International Workshop on Foundations of Models and Languages for Data and Objects*, pp. 41–56, September 1996.
51. M. Gertz. *Diagnosis and Repair of Constraint Violations in Database Systems*. PhD thesis, Universität Hannover, 1996.
52. F. Giannotti, S. Greco, D. Sacca, and C. Zaniolo. Programming with Nondeterminism in Deductive Databases. *Annals of Mathematics and Artificial Intelligence*, 19(3-4), 1997.

53. F. Giannotti and D. Pedreschi. Datalog with Nondeterministic Choice Computes NDB-PTIME. *Journal of Logic Programming*, 35:75–101, 1998.
54. P. Godfrey, J. Grant, J. Gryz, and J. Minker. Integrity Constraints: Semantics and Applications. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, Chapt. 9. Kluwer Academic Publishers, Boston, 1998.
55. G. Grahne and A. O. Mendelzon. Tableau Techniques for Querying Information Sources through Global Schemas. In *7th International Conference on Database Theory*, pp. 332–347. Springer-Verlag, LNCS 1540, 1999.
56. J. Grant and J. Minker. A Logic-Based Approach to Data Integration. *Theory and Practice of Logic Programming*, 2(3):323–368, 2002.
57. G. Greco, S. Greco, and E. Zumpano. A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. In *17th International Conference on Logic Programming*, pp. 348–364. Springer-Verlag, LNCS 2237, 2001.
58. S. Greco, D. Sacca, and C. Zaniolo. Datalog Queries with Stratified Negation and Choice: from P to D^P. In *5th International Conference on Database Theory*, pp. 82–96. Springer-Verlag, 1995.
59. S. Greco and E. Zumpano. Querying Inconsistent Databases. In *7th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, pp. 308–325. Springer-Verlag, LNCS 1955, 2000.
60. J. Gryz. Query Rewriting using Views in the Presence of Functional and Inclusion Dependencies. *Information Systems*, 24(7):597–612, 1999.
61. D. S. Hochbaum. Approximating Covering and Packing Problems: Set Cover, Vertex Cover, Independent Set, and Related Problems. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*. PWS, 1997.
62. A. Hunter. Paraconsistent Logics. In Gabbay and Smets [44], pp. 13–44.
63. T. Imieliński, S. Naqvi, and K. Vadaparty. Incomplete Objects - A Data Model for Design and Planning Applications. In *ACM SIGMOD International Conference on Management of Data*, pp. 288–297, Denver, Colorado, May 1991.
64. T. Imieliński, R. van der Meyden, and K. Vadaparty. Complexity Tailored Design: A New Design Methodology for Databases With Incomplete Information. *Journal of Computer and System Sciences*, 51(3):405–432, 1995.
65. C. Kakas, A, R. A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.
66. P. C. Kanellakis. Elements of Relational Database Theory. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, Vol. B, Chapt. 17, pp. 1073–1158. Elsevier/MIT Press, 1990.
67. H. Katsuno and A. O. Mendelzon. Propositional Knowledge Base Revision and Minimal Change. *Artificial Intelligence*, 52(3):263–294, 1992.
68. M. Kifer and E. L. Lozinskii. A Logic for Reasoning with Inconsistency. *Journal of Automated Reasoning*, 9(2):179–215, 1992.
69. M. Kifer and V. S. Subrahmanian. Theory of Generalized Annotated Logic Programming and its Applications. *Journal of Logic Programming*, 12(4):335–368, 1992.
70. R. Kowalski and F. Sadri. Logic Programs with Exceptions. *New Generation Computing*, 9(3/4):387–400, 1991.
71. R. Kowalski, F. Sadri, and P. Soper. Integrity Checking in Deductive Databases. In *International Conference on Very Large Data Bases*, pp. 61–69. Morgan Kaufmann Publishers, 1987.

72. S. M. Leach and J. J. Lu. Query Processing in Annotated Logic Programming: Theory and Implementation. *Journal of Intelligent Information Systems*, 6:33–58, 1996.
73. D. Lembo, M. Lenzerini, and R. Rosati. Source Inconsistency and Incompleteness in Data Integration. In *9th International Workshop on Nonmonotonic Reasoning (NMR'02)*, Toulouse, France, 2002.
74. M. Lenzerini. Data Integration: A Theoretical Perspective. In *21st ACM Symposium on Principles of Database Systems*, 2002. Invited talk.
75. N. Leone, P. Rullo, and F. Scarcello. Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics, and Computation. *Information and Computation*, 135(2):69–112, 1997.
76. A. Y. Levy. Combining Artificial Intelligence and Databases for Data Integration. In *Artificial Intelligence Today*, pp. 249–268. Springer-Verlag, LNCS 1600, 1999.
77. A. Y. Levy, A. Rajaraman, and J. J. Ordille. Query-Answering Algorithms for Information Agents. In *13th National Conference on Artificial Intelligence*, pp. 40–47, 1996.
78. A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *22nd International Conference on Very Large Data Bases*, pp. 251–262, 1996.
79. J. Lin. A Semantics for Reasoning Consistently in the Presence of Inconsistency. *Artificial Intelligence*, 86(1-2):75–95, 1996.
80. J. Lin and A. O. Mendelzon. Merging Databases under Constraints. *International Journal of Cooperative Information Systems*, 7(1):55–76, 1996.
81. J. W. Lloyd. *Foundations of Logic Programming*. 2nd edition, Springer-Verlag, 1987.
82. E. L. Lozinskii. Resolving Contradictions: A Plausible Semantics for Inconsistent Systems. *Journal of Automated Reasoning*, 12(1):1–32, 1994.
83. V. W. Marek and M. Truszczynski. Revision Programming. *Theoretical Computer Science*, 190(2):241–277, 1998.
84. A. Motro. Multiplex: A Formal Model for Multidatabases and Its Implementation. In *4th International Workshop on Next Generation Information Technology and Systems*, pp. 138–158. Springer-Verlag, LNCS 1649, 1999.
85. B. A. Nuseibeh, S. M. Easterbrook, and Russo A. Leveraging Inconsistency in Software Development. *IEEE Computer*, 33(4):24–29, 2000.
86. S. Pradhan. Semantics of Normal Logic Programs and Contested Information. In *11th IEEE Symposium on Logic in Computer Science*, pp. 406–415, 1996.
87. S. Pradhan. *Reasoning with Conflicting Information in Artificial Intelligence and Database Theory*. Ph.D. thesis, Department of Computer Science, University of Maryland, College Park, Md., 2001.
88. R. Reiter. Towards a Logical Reconstruction of Relational Database Theory. In M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, editors, *On Conceptual Modeling*, pp. 191–233. Springer-Verlag, 1984.
89. R. Reiter. On Integrity Constraints. In *2nd International Conference on Theoretical Aspects of Rationality and Knowledge*, pp. 97–111, 1988.
90. R. Reiter. What Should A Database Know? In *6th ACM Symposium on Principles of Database Systems*, pp. 302–304, 1988.
91. F. Sadri and R. Kowalski. A Theorem-Proving Approach to Database Integrity. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pp. 313–362. Morgan Kaufmann, 1988.

92. K. F. Sagonas, T. Swift, and D. S. Warren. XSB as an Efficient Deductive Database Engine. In *ACM SIGMOD International Conference on Management of Data*, pp. 442–453, 1994.
93. V. S. Subrahmanian. Amalgamating Knowledge Bases. *ACM Transactions on Database Systems*, 19(2):291–331, 1994.
94. J. D. Ullman. Information Integration Using Logical Views. In *6th International Conference on Database Theory*, pp. 19–40. Springer-Verlag, LNCS 1186, 1997.
95. M. Y. Vardi. The Complexity of Relational Query Languages. In *14th ACM Symposium on Theory of Computing*, pp. 137–146, 1982.
96. J. Wijsen. Condensed Representation of Database Repairs for Consistent Query Answering. In *9th International Conference on Database Theory*, pp. 378–393, Springer-Verlag, LNCS 2572, 2003.
97. M. Winslett. Reasoning about Action using a Possible Models Approach. In *7th National Conference on Artificial Intelligence*, 1988.

Consistent Query Answers in Inconsistent Databases

Marcelo Arenas
Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de Computación
Casilla 306, Santiago 22, Chile
marenas@ing.puc.cl

Leopoldo Bertossi
Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de Computación
Casilla 306, Santiago 22, Chile
bertossi@ing.puc.cl

Jan Chomicki
Monmouth University
Department of Computer Science
West Long Branch, NJ 07764
chomicki@monmouth.edu

Abstract

In this paper we consider the problem of the logical characterization of the notion of consistent answer in a relational database that may violate given integrity constraints. This notion is captured in terms of the possible repaired versions of the database. A method for computing consistent answers is given and its soundness and completeness (for some classes of constraints and queries) proved. The method is based on an iterative procedure whose termination for several classes of constraints is proved as well.

1 Introduction

Integrity constraints capture an important normative aspect of every database application. However, it is often the case that their satisfaction cannot be guaranteed, allowing for the existence of inconsistent database instances. In that case, it is important to know which query answers are consistent with the integrity constraints and which are not. In this paper, we provide a logical characterization of consistent query answers in relational databases that may be inconsistent with the given integrity constraints. Intuitively, an answer to a query posed to a database that violates the integrity constraints will be consistent in a precise sense: It should be the same as the answer obtained from any minimally repaired version of the original database. We also provide a method for computing such answers and prove its properties. On the basis of a query Q, the method computes, using an iterative procedure, a new query $T_\omega(Q)$ whose evaluation in an arbitrary, consistent or inconsistent, database returns the set of consistent answers to the original query Q. We envision the application of our results in a number of areas:

Data warehousing. A data warehouse contains data coming from many different sources. Some of it typically does not satisfy the given integrity constraints. The usual approach is thus to *clean* the data by removing inconsistencies before the data is stored in the warehouse [6]. Our results make it possible to determine which data is already clean and proceed to safely remove unclean data. Moreover, a different scenario becomes possible, in which the inconsistencies are not removed but rather query answers are marked as "consistent" or "inconsistent". In this way, information loss due to data cleaning may be prevented.

Database integration. Often many different databases are integrated together to provide a single unified view for the users. Database integration is difficult since it requires the resolution of many different kinds of discrepancies of the integrated databases. One possible discrepancy is due to different sets of integrity constraints. Moreover, even if every integrated database *locally* satisfies the same integrity constraint, the constraint may be *globally* violated. For example, different databases may assign different addresses to the same student. Such conflicts may fail to be resolved at all and inconsistent data cannot be "cleaned" because of the autonomy of different databases. Therefore, it is important to be able to find out, given a set of local integrity constraints, which query answers returned from the integrated database are consistent with the constraints and which are not.

Active and reactive databases. A violation of integrity constraints may be acceptable under the provision that it will be repaired in the near future. For example, the stock level in a warehouse may be allowed to fall below the required minimum if the necessary replenishments have been ordered. During this temporary inconsistency, however, query answers should give an indication whether they are consistent with the constraints or not. This problem is particularly acute in active databases that allow such consistency lapses. The result of evaluating a trigger condition that is consistent with the integrity constraints should be treated differently from the one that isn't.

The following example presents the basic intuitions behind the notion of consistent query answer.

Example 1. Consider a database subject to the following *IC*:

$$\forall x(P(x) \supset Q(x)).$$

The instance

$$\{P(a),P(b),Q(a),Q(c)\}$$

violates this constraint. Now if the query asks for all x such that $Q(x)$, only a is returned as an answer consistent with the integrity constraint.

The plan of this paper is as follows. In section 2 we introduce the basic notions of our approach, including those of *repair* and *consistent query answer*. In section 3 we show a method how to compute the query $T_\omega(Q)$ for a given first-order query Q. In subsequent sections, the properties of this method are analyzed: soundness in section 4, completeness in section 5, and termination in section 6. In section 7 we discuss related work. In section 8 we conclude and outline some of the prospects for future work in this area. The proofs are given in the appendix.

2 Basic Notions

In this paper we assume we have a fixed database schema and a fixed infinite database domain D. We also have a first order language based on this schema with names for the elements of D. We assume that elements of the domain with different names are different. The instances of the schema are finite structures for interpreting the first order language. As such they all share the given domain D, nevertheless, since relations are finite, every instance has a finite active domain which is a subset of D. As usual, we allow built-in predicates that have infinite extensions, identical for all database instances. There is also a set of integrity constraints *IC*, expressed in that language, which the database instances are expected to satisfy. We will assume that *IC* is consistent in the sense that there is a database instance that makes it true.

Definition 1. (*Consistency*) A database instance r is *consistent* if r satisfies *IC* in the standard model-theoretic sense, that is, $r \quad IC$; r is inconsistent otherwise.

This paper addresses the issue of obtaining meaningful and useful query answers in *any*, consistent or inconsistent, database. It is well known how to obtain query answers in consistent databases. Therefore, the challenging part is how to deal with the inconsistent ones.

2.1 Repairs

Given a database instance r, we denote by $\Sigma(r)$ the set of formulas $\{P(\bar{a})|r \quad P(\bar{a})\}$, where the Ps are relation names and $\bar{a}$ is ground tuple.

Definition 2. (*Distance*) The *distance* $\Delta(r,r')$ between database instances r and r' is the symmetric difference:

$$\Delta(r,r') = (\Sigma(r) \quad \Sigma(r')) \cup (\Sigma(r') \quad \Sigma(r)).$$

Definition 3. For the instances r,r',r'', $r' \quad_r r''$ if $\Delta(r,r') \quad \Delta(r,r'')$, i.e., if the distance between r and r' is less than or equal to the distance between r and r''.

Notice that built-in predicates do not contribute to the Δs because they have fixed extensions, identical in every database instance.

Definition 4. (*Repair*) Given database instances r and r', we say that r' is a *repair* of r if $r' \quad IC$ and r' is $\quad_r$*-minimal* in the class of database instances that satisfy the ICs.

Clearly, what constitutes a repair depends on the given set of integrity constraints. In the following we assume that this set is fixed.

Example 2. Let us consider a database schema with two unary relations P and Q and domain $D = \{a,b,c\}$. Assume that for an instance r, $\Sigma(r) = \{P(a),P(b),Q(a),Q(c)\}$, and let $IC = \{\forall x(P(x) \supset Q(x))\}$. Clearly, r does not satisfy *IC* because $r \quad P(b) \wedge \neg Q(b)$.

In this case we have two possibles repairs for r. First, we can falsify $P(b)$, obtaining an instance r' with $\Sigma(r') = \{P(a),Q(a),Q(c)\}$. As a second alternative, we can make $Q(b)$ true, obtaining an instance r'' with $\Sigma(r'') = \{P(a),P(b), Q(a),Q(b),Q(c)\}$.

The definition of a repair satisfies certain desirable and expected properties. Firstly, a consistent database does not need to be repaired, because if r satisfies *IC*, then, by the minimality condition wrt the relation $\quad_r$, r is the only repair of itself (since $\Delta(r,r)$ is empty). Secondly, any database r can always be repaired because there is a database r' that satisfies *IC*, and $\Delta(r,r')$ is finite.

Example 3. (motivated by [19]) Consider the IC saying that C is the only supplier of items of class T_4:

$$\forall(x,y,z)(Supply(x,y,z) \wedge Class(z,T_4) \supset x = C). \qquad (1)$$

The following database instance r_1 violates the IC:

Supply		
C	D_1	I_1
D	D_2	I_2

Class	
I_1	T_4
I_2	T_4

The only repairs of this database are

Supply		
C	D_1	I_1

Class	
I_1	T_4
I_2	T_4

and

Supply		
C	D_1	I_1
D	D_2	I_2

Class	
I_1	T_4

Example 4. (motivated by [19]) Consider the IC:

$$\forall(x,y)(Supply(x,y,I_1) \supset Supply(x,y,I_2)), \qquad (2)$$

saying that item I_2 is supplied whenever item I_1 is supplied; and the following inconsistent instance, r_2, of the database

Supply		
C	D_1	I_1
C	D_1	I_3

This instance has two repairs:

Supply		
C	D_1	I_1
C	D_1	I_2
C	D_1	I_3

and

Supply		
C	D_1	I_3

Example 5. Consider a student database. $Student(x,y,z)$ means that x is the student number, y is the student's name, and z is the student's address. The two following ICs state that the first argument is a key of the relation

$$\forall(x,y,z,u,v)(Student(x,y,z) \wedge Student(x,u,v) \supset y = u),$$
$$\forall(x,y,z,u,v)(Student(x,y,z) \wedge Student(x,u,v) \supset z = v).$$

The inconsistent database instance r_3

Student		
S_1	N_1	D_1
S_1	N_2	D_1

Course		
S_1	C_1	G_1
S_1	C_2	G_2

has two repairs:

Student		
S_1	N_1	D_1

Course		
S_1	C_1	G_1
S_1	C_2	G_2

and

Student		
S_1	N_2	D_1

Course		
S_1	C_1	G_1
S_1	C_2	G_2

2.2 Consistent query answers

We assume all queries are in prefix disjunctive normal form.

Definition 5. A formula Q is a *query* if it has the following syntactical form:

$$\bar{Q}\bigvee_{i=1}^{s}(\bigwedge_{j=1}^{m_i} P_{i,j}(\bar{u}_{i,j}) \wedge \bigwedge_{j=1}^{n_i} \neg Q_{i,j}(\bar{v}_{i,j}) \wedge \psi_i),$$

where $\bar{Q}$ is a sequence of quantifiers and every ψ_i contains only built-in predicates. If $\bar{Q}$ contains only universal quantifiers, then we say that Q is a *universal query*. If $\bar{Q}$ contains existential (and possibly universal) quantifiers, we say that Q is *non-universal* query.

Definition 6. (*Query answer*) A (ground) tuple $\bar{t}$ is an *answer* to a query $Q(\bar{x})$ in a database instance r if $r \models Q(\bar{t})$. A (ground) tuple $\bar{t}$ is an *answer* to a set of queries $\{Q_1,\ldots,Q_n\}$ if $r \models Q_1 \wedge \quad \wedge Q_n$.

Definition 7. (*Consistent answer*) Given a set of integrity constraints, we say that a (ground) tuple $\bar{t}$ is a *consistent answer* to a query $Q(\bar{x})$ in a database instance r, and we write $r \models_c Q(\bar{t})$ (or $r \models_c Q(\bar{x})[\bar{t}]$), if for every repair r' of r, $r' \quad Q(\bar{t})$. If Q is a sentence, then *true* (*false*) is a *consistent answer* to Q in r, and we write $r \models_c Q$ $(r \not\models_c Q)$, if for every repair r' of r, $r' \quad Q$ $(r' \not\models Q)$.

Example 6. (example 3 continued) The only consistent answer to the query $Class(z,T_4)$, posed to the database instance r_1, is I_1 because $r_1 \models_c Class(z,T_4)[I_1]$.

Example 7. (example 4 continued) The only consistent answer to the query $Supply(C,D_1,z)$, posed to the database instance r_2, is I_3 because $r_2 \models_c Supply(C,D_1,z)[I_3]$.

Example 8. (example 5 continued) By considering all the repairs of the database instance r_3, we obtain C_1 and C_2 as the consistent answers to the query $\exists z Course(S_1,y,z)$, posed to r_3. For the query $\exists(u,v)(Student(u,N_1,v) \wedge Course(u,x,y))$, we obtain no (consistent) answers.

3 The General Approach

We present here a method to compute consistent answers to queries. Given a query Q, the query $T_\omega(Q)$ is defined based on the notion of *residue* developed in the context of semantic query optimization (SQO) [5]. In the context of deductive databases, SQO is used to optimize the process of answering queries using the semantic knowledge about the domain that is contained in the ICs. In this case, the basic assumption is that the ICs are satisfied by the database. In our case, since we allow inconsistent databases, we do not assume the satisfaction of the ICs while answering queries. A first attempt to obtain consistent answers to a query $Q(\bar{x})$ may be to use *query modification*, i.e., ask the query $Q(\bar{x}) \wedge IC$. However, this does not work, as we obtain *false* as the answer if the DB is inconsistent. Instead, we iteratively modify the query Q using the residues. As a result, we obtain the query $T_\omega(Q)$ with the property that the set of all answers to $T_\omega(Q)$ is the same as as the set of consistent answers to Q. (As shown later, the property holds only for restricted classes of queries and constraints.)

3.1 Generating residues in relational DBs

We consider only universal constraints. We begin by transforming every integrity constraint to the standard format (*expansion* step).

Definition 8. An integrity constraint is in *standard format* if it has the form

$$\forall(\bigvee_{i=1}^{m} P_i(\bar{x}_i) \vee \bigvee_{i=1}^{n} \neg Q_i(\bar{y}_i) \vee \psi),$$

where $\forall$ represents the universal closure of the formula, $\bar{x}_i$, $\bar{y}_i$ are tuples of variables and ψ is a formula that mentions only built–in predicates, in particular, equality.

Notice that in such an IC there are no constants in the P_i, Q_i; if they are needed they can be pushed into ψ.

Many usual ICs that appear in DBs can be transformed to the standard format, e.g. functional dependencies, set inclusion dependencies of the form $\forall \bar{x}(P(\bar{x}) \supset Q(\bar{x}))$, transitivity constraints of the form $\forall x,y,z(P(x,y) \wedge P(y,z) \supset P(x,z))$. The usual ICs that appear in SQO in deductive databases as rules [5] can be also accommodated in this format, including rules with disjunction and logical negation in their heads. An inclusion dependency of the form $\forall \bar{x}(P(\bar{x}) \supset \exists y\, Q(\bar{x},y))$ cannot be transformed to the standard format.

After the expansion of *IC*, rules associated with the database schema are generated. This could be seen as considering an instance of the database as an extensional database expanded with new rules, and so obtaining an associated deductive database where semantical query optimization can be used.

For each predicate, its negative and positive occurrences in the ICs (in standard format) will be treated separately with the purpose of generating corresponding residues and rules. First, a motivating example.

Example 9. Consider the IC $\forall x\, (\neg P(x) \vee Q(x))$. If $Q(x)$ is false, then $\neg P(x)$ must be true. Then, when asking about $\neg Q(x)$, we make sure that $\neg P(x)$ becomes true. That is, we generate the query $\neg Q(x) \wedge \neg P(x)$ where $\neg P(x)$ is the residue attached to the query.

For each IC in standard format

$$\forall(\bigvee_{i=1}^{m} P_i(\bar{x}_i) \vee \bigvee_{i=1}^{n} \neg Q_i(\bar{y}_i) \vee \psi), \qquad (3)$$

and each positive occurrence of a predicate $P_j(\bar{x}_j)$ in it, the following residue for $\neg P_j(\bar{x}_j)$ is generated

$$\bar{Q}(\bigvee_{i=1}^{j-1} P_i(\bar{x}_i) \vee \bigvee_{i=j+1}^{m} P_i(\bar{x}_i) \vee \bigvee_{i=1}^{n} \neg Q_i(\bar{y}_i) \vee \psi), \qquad (4)$$

where $\bar{Q}$ is a sequence of universal quantifiers over all the variables in the formula not appearing in $\bar{x}_j$.

If $R_1,\ldots,R_r$ are all the residues for $\neg P_j$, then the following rule is generated:

$$\neg P_j(\bar{w}) \longmapsto \neg P_j(\bar{w})\{R_1(\bar{w}),\ldots,R_r(\bar{w})\},$$

where $\bar{w}$ are new variables. If there are no residues for $\neg P_j$, then the rule $\neg P_j(\bar{w}) \longmapsto \neg P_j(\bar{w})$ is generated.

For each negative occurrence of a predicate $Q_j(\bar{y}_j)$ in (3), the following residue for $Q_j(\bar{y}_j)$ is generated

$$\bar{Q}(\bigvee_{i=1}^{m} P_i(\bar{x}_i) \vee \bigvee_{i=1}^{j-1} \neg Q_i(\bar{y}_i) \vee \bigvee_{i=j+1}^{n} \neg Q_i(\bar{y}_i) \vee \psi),$$

where $\bar{Q}$ is a sequence of universal quantifiers over all the variables in the formula not appearing in $\bar{y}_j$.

If $R'_1,\ldots,R'_s$ are all the residues for $Q_j(\bar{y}_j)$, the following rule is generated:

$$Q_j(\bar{u}) \longmapsto Q_j(\bar{u})\{R'_1(\bar{u}),\ldots,R'_s(\bar{u})\}.$$

If there are no residues for $Q_j(\bar{y}_j)$, then the rule $Q_j(\bar{u}) \longmapsto Q_j(\bar{u})$ is generated. Notice that there is exactly one new rule for each positive predicate, and exactly one rule for each negative predicate.

If there are more than one positive (negative) occurrences of a predicate, say P, in an IC, then more then one residue is computed for $\neg P$. In some cases, e.g., for functional dependencies, the subsequent residues will be redundant. In other cases cases, e.g., for *transitivity constraints*, multiple residues are not redundant.

Example 10. If we have the following ICs in standard format

$$IC = \{\forall x(R(x) \vee \neg P(x) \vee \neg Q(x)), \forall x(P(x) \vee \neg Q(x))\},$$

the following rules are generated:

$$\begin{array}{rcl} P(x) & \longmapsto & P(x)\{R(x) \vee \neg Q(x)\} \\ Q(x) & \longmapsto & Q(x)\{R(x) \vee \neg P(x), P(x)\} \\ R(x) & \longmapsto & R(x) \\ \neg P(x) & \longmapsto & \neg P(x)\{\neg Q(x)\} \\ \neg Q(x) & \longmapsto & \neg Q(x) \\ \neg R(x) & \longmapsto & \neg R(x)\{\neg P(x) \vee \neg Q(x)\}. \end{array}$$

Notice that no rules are generated for built-in predicates, but such predicates may appear in the residues. They have fixed extensions and thus cannot contribute to the violation of an IC or be modified to make an IC true. For example, if we have the IC $\forall x,y,z(\neg P(x,y) \vee \neg P(x,z) \vee y = z)$, and the database satisfies $P(1,2), P(1,3)$, the IC cannot be made true by making $2 = 3$.

Once the rules have been generated, it is possible to simplify the associated residues. In every new rule of the form $P(\bar{u}) \mapsto P(\bar{u})\{R_1(\bar{u}),\ldots,R_r(\bar{u})\}$ the auxiliary quantifications introduced in the expansion step are eliminated (both the quantifier and the associated variable in the formula) from the residues by the process inverse to the one applied in the expansion. The same is done with rules of the form $\neg P \mapsto \neg P\{\quad\}$.

3.2 Computing $\mathrm{T}_\omega(Q)$

In order to determine consistent answers to queries in arbitrary databases, we will make use of a family of operators consisting of T_n, $n \quad 0$, and T_ω.

Definition 9. The application of an operator T_n to a query is defined inductively by means of the following rules

1. $\mathrm{T}_n(\Box) := \Box$, $\mathrm{T}_n(\neg\Box) := \neg\Box$, for every $n \quad 0$ ($\Box$ is the empty clause).

2. $T_0(\varphi) := \varphi$.

3. For each predicate $P(\bar{u})$, if there is a rule $P(\bar{u}) \mapsto P(\bar{u})\{R_1(\bar{u}),\ldots,R_r(\bar{u})\}$, then
$$\mathrm{T}_{n+1}(P(\bar{u})) := P(\bar{u}) \wedge \bigwedge_{i=1}^{r} \mathrm{T}_n(R_i(\bar{u})).$$
If $P(\bar{u})$ does not have residues, then $\mathrm{T}_{n+1}(P(\bar{u})) := P(\bar{u})$.

4. For each negated predicate $\neg Q(\bar{v})$, if there is a rule $\neg Q(\bar{v}) \mapsto \neg Q(\bar{v})\{R'_1(\bar{v}),\ldots,R'_s(\bar{v})\}$, then
$$\mathrm{T}_{n+1}(\neg Q(\bar{v})) := \neg Q(\bar{v}) \wedge \bigwedge_{i=1}^{s} \mathrm{T}_n(R'_i(\bar{v})).$$
If $\neg Q(\bar{v})$ does not have any residues, then $\mathrm{T}_{n+1}(\neg Q(\bar{u})) := \neg Q(\bar{u})$.

5. If φ is a formula in prenex disjunctive normal form, that is,
$$\varphi = \bar{Q} \bigvee_{i=1}^{s}(\bigwedge_{j=1}^{m_i} P_{i,j}(\bar{u}_{i,j}) \wedge \bigwedge_{j=1}^{n_i} \neg Q_{i,j}(\bar{v}_{i,j}) \wedge \psi_i),$$
where $\bar{Q}$ is a sequence of quantifiers and ψ_i is a formula that includes only built–in predicates, then for every $n \quad 0$:
$$\mathrm{T}_n(\varphi) := \bar{Q}\bigvee_{i=1}^{s}(\bigwedge_{j=1}^{m_i} \mathrm{T}_n(P_{i,j}(\bar{u}_{i,j}))\wedge$$
$$\bigwedge_{j=1}^{n_i} \mathrm{T}_n(\neg Q_{i,j}(\bar{v}_{i,j})) \wedge \psi_i).$$

Definition 10. The application of operator T_ω on a query is defined as $\mathrm{T}_\omega(\varphi) = \bigcup_{n<\omega}\{\mathrm{T}_n(\varphi)\}$.

Example 11. (example 10 continued) For the query $\neg R(x)$ we have $\mathrm{T}_1(\neg R(x)) = \neg R(x) \wedge (\neg P(x) \vee \neg Q(x))$, $\mathrm{T}_2(\neg R(x)) = \neg R(x) \wedge ((\neg P(x) \wedge \neg Q(x)) \vee \neg Q(x))$ and finally $\mathrm{T}_3(\neg R(x)) = \mathrm{T}_2(\neg R(x))$. We have reached a fixed point and then
$$\mathrm{T}_\omega(\neg R(x)) = \{\neg R(x), \neg R(x) \wedge (\neg P(x) \vee \neg Q(x)),$$
$$\neg R(x) \wedge ((\neg P(x) \wedge \neg Q(x)) \vee \neg Q(x))\}.$$

We show first that the operator T_ω conservatively extends standard query evaluation on consistent databases.

Proposition 1. Given a database instance r and a set of integrity constraints IC, such that $r \quad IC$, then for every query $Q(\bar{x})$ and every natural number n: $r \quad \forall\bar{x}(Q(\bar{x}) \quad \mathrm{T}_n(Q(\bar{x})))$.

Corollary 1. Given a database instance r and a set of integrity constraints IC, such that $r \quad IC$, then for every query $Q(\bar{x})$ and every tuple $\bar{t}$: $r \quad Q(\bar{t})$ if and only if $r \quad \mathrm{T}_\omega(Q(\bar{t}))$.

4 Soundness

Now we will show the relationship between consistent answers to a query Q in a database instance r (definition 7) and answers to the query $\mathrm{T}_\omega(Q)$ (definition 6). We show that $\mathrm{T}_\omega(Q)$ returns only consistent answers to Q.

Theorem 1. (*Soundness*) Let r be a database instance, IC a set of integrity constraints and $Q(\bar{x})$ a query (see definition 5) such that $r \quad \mathrm{T}_\omega(Q(\bar{x}))[\bar{t}]$. If Q is universal or non-universal and domain independent[20], then $\bar{t}$ is a consistent answer to Q in r (in the sense of definition 7), that is, $r \models_c Q(\bar{t})$.

The second condition in the theorem excludes non-universal, but domain dependent queries like $\exists x \neg P(x)$.

Example 12. (example 6 continued) The IC (1) transformed into the standard format becomes
$$\forall(x,y,z,w)(\neg Supply(x,y,z) \vee$$
$$\neg Class(z,w) \vee w \neq T_4 \vee x = C).$$

The following rule is generated:
$$Class(z,w) \mapsto Class(z,w)$$
$$\{\forall(x,y)(\neg Supply(x,y,z) \vee w \neq T_4 \vee x = C)\}.$$

Given the database instance r_1 that violates the IC as before, if we pose the query $Class(z,T_4)$, asking for the items of class T_4, directly to r_1, we obtain I_1 and I_2. Nevertheless, if we pose the query $\mathrm{T}_\omega(Class(z,T_4))$, that is
$$\{Class(z,T_4),$$
$$Class(z,T_4) \wedge \forall(x,y)(\neg Supply(x,y,z) \vee x = C)\}$$

we obtain only I_1, eliminating I_2. I_1 is the only consistent answer.

Example 13. (example 8 continued) In the standard format, the ICs take the form
$$\forall(x,y,z,u,v)(\neg Student(x,y,z) \vee$$
$$\neg Student(x,u,v) \vee y = u),$$
$$\forall(x,y,z,u,v)(\neg Student(x,y,z) \vee$$
$$\neg Student(x,u,v) \vee z = v).$$

The following rule is generated
$$Student(x,y,z) \mapsto Student(x,y,z)$$
$$\{\forall(u,v)(\neg Student(x,u,v) \vee y = u),$$
$$\forall(u,v)(\neg Student(x,u,v) \vee z = v)\}.$$

Given the inconsistent database instance r_3, if we pose the query $\exists z Course(S_1,y,z)$, asking for the names of the courses of the student with number S_1, we obtain C_1 and C_2. If we pose the query
$$\mathrm{T}_\omega(\exists z Course(S_1,y,z)) = \{\exists z Course(S_1,y,z)\}$$

we obviously obtain the same answers which, in this case, are the consistent answers. Intuitively, in this case the T_ω operator helps us to establish that even when the name of the student with number S_1 is undetermined, it is still possible to obtain the list of courses in which he/she is registered. On the other hand, if we pose the query
$$\exists(u,v)(Student(u,N_1,v) \wedge Course(u,x,y))$$

about the courses and grades for a student with name N_1, to r_3, we obtain (C_1,G_1) and (C_2,G_2). Nevertheless, if we ask
$$\mathrm{T}_\omega(\exists(u,v)(Student(u,N_1,v) \wedge Course(u,x,y)))$$

we obtain, in conjunction with the original query, the formula:
$$\exists(u,v)(Student(u,N_1,v) \wedge$$
$$\forall(y',z')(\neg Student(u,y',z') \vee y' = N_1) \wedge$$
$$\forall(y',z')(\neg Student(u,y',z') \vee z' = v) \wedge Course(u,x,y)),$$

from this we obtain the empty set of tuples. This answer is intuitively consistent, because the number of the student with name N_1 is uncertain, and in consequence it is not possible to find out in which courses he/she is registered. The set of answers obtained with the T_ω operator coincides with the set of consistent answers which is empty.

5 Completeness

5.1 Binary ICs

Definition 11. A *binary integrity constraint* (BIC) is a sentence of the form
$$\forall(l_1(\bar{x}_1) \vee l_2(\bar{x}_2) \vee \psi(\bar{x})),$$

where l_1 and l_2 are literals, and ψ is a formula that only contains built-in predicates.

Examples of BICs include: functional dependencies, symmetry constraints, set inclusions dependencies of the form $\forall\bar{x}(P(\bar{x}) \supset Q(\bar{x}))$.

Definition 12. Given a set of sentences Σ in the language of the database schema DB, and a sentence φ, we denote by $\Sigma \quad_{DB} \varphi$ the fact that, for every instance r of the database, if $r \quad \Sigma$, then $r \quad \varphi$.

Theorem 2. (*Completeness for BICs*) Given a set IC of binary integrity constraints, if for every literal $l'(\bar{a})$, $IC \not\models_{DB} l'(\bar{a})$, then the operator T_ω is complete, that is, for every ground literal $l(\bar{t})$, if $r \quad_c l(\bar{t})$ then $r \quad \mathrm{T}_\omega(l(\bar{t}))$.

The theorem says that every consistent answer to a query of the form $l(\bar{x})$ is captured by the T_ω operator. Actually, proposition 2 in the appendix and the completeness theorem can be easily extended to the case of queries that are conjunctions of literals. Notice that the finiteness $\mathrm{T}_\omega(l(\bar{x}))$ is not a part of the hypothesis in this theorem. The hypothesis of the theorem requires that the ICs are not enough to answer a literal query by themselves; they do not contain definite knowledge about the literals.

Example 14. We can see in the example 12 where BICs and queries which are conjunctions of literals appear, that the operator T_ω gave us all the consistent answers, as implied by the theorem.

Corollary 2. If IC is a set of functional dependencies (FDs)
$$IC = \{\forall(\neg P_1(\bar{x}_1,y_1) \vee \neg P_1(\bar{x}_1,z_1) \vee y_1 = z_1), \tag{5}$$
$$\ldots,$$
$$\forall(\neg P_n(\bar{x}_n,y_n) \vee \neg P_n(\bar{x}_n,z_n) \vee y_n = z_n)\},$$

then the operator T_ω is complete for consistent answers to queries that are conjunctions of literals.

Example 15. In example 13 we had FDs that are also BICs. Thus the operator T_ω found all the consistent answers, even for some queries that are not conjunctions of literals, showing that this is not a necessary condition.

Example 16. Here we will show that in general completeness is not obtained for queries that are not conjunctions of literals. Consider the IC: $\forall x,y,z(P(x,y) \wedge P(x,z) \supset y = z)$ and the inconsistent instance r with $\Sigma(r) = \{P(a,b),P(a,c)\}$. This database has two repairs: r' with $\Sigma(r') = \{P(a,b)\}$; and r'' with $\Sigma(r'') = \{P(a,c)\}$. We have that $r \models_c \exists x P(a,x)$, because the query is true in the two repairs.

Now, it is easy to see that $\mathrm{T}_\omega(\exists u P(a,u))$ is logically equivalent to $\exists u(P(a,u) \wedge \forall z(\neg P(a,z) \vee z = u))$. So, we have $r \not\models \mathrm{T}_\omega(\exists x P(a,x))$. Thus, the consistent answer *true* is not captured by the operator T_ω.

5.2 Other Constraints

The following theorem applies to arbitrary ICs and generalizes Theorem 2.

Theorem 3. (*Completeness*) Let IC be a set of integrity constraints, $l(\bar{x})$ a literal, and $\mathrm{T}_n(l(\bar{x}))$ of the form
$$l(\bar{x}) \wedge \bigwedge_{i=1}^{m} \forall(\bar{x}_i, \bar{y}_i)(C_i(\bar{x}, \bar{x}_i) \vee \psi_i(\bar{x}, \bar{y}_i)).$$

If for every $n \quad 0$, there is $S \quad \{1,\ldots,m\}$ such that

1. for every $j \in S$ and every tuple $\bar{a}$: $IC \not\models_{DB} C_j(\bar{a})$, and
2. $\{\forall(\bar{x}_i, \bar{y}_i)(C_i(\bar{x}, \bar{x}_i) \vee \psi_i(\bar{x}, \bar{y}_i)) | i \in S\}$ implies
$$\{\forall(\bar{x}_i, \bar{y}_i)(C_i(\bar{x}, \bar{x}_i) \vee \psi_i(\bar{x}, \bar{y}_i)) | 1 \quad i \quad m\}$$

then $r \quad_c l(\bar{t})$ implies $r \quad \mathrm{T}_\omega(l(\bar{t}))$.

This theorem can be extended to conjunctions of literals. Notice that the theorem requires a condition for every $n \in \mathbb{N}$. Its application is obviously simplified if we know that the iteration terminates. This is an issue to be analyzed in the next section.

6 Termination

Termination means that the operator T_ω returns a finite set of formulas. It is clearly important because then the set of consistent answers can be computed by evaluating a single, finite query. We distinguish between three different notions of termination.

Definition 13. Given a set of ICs and a query $Q(\bar{x})$, we say that $\mathrm{T}_\omega(Q(\bar{x}))$ is

1. *syntactically finite* if there is an an n such that $\mathrm{T}_n(Q(\bar{x}))$ and $\mathrm{T}_{n+1}(Q(\bar{x}))$ are syntactically the same.

2. *semantically finite* if there is an n such that for all $m \quad n$, $\forall\bar{x}(\mathrm{T}_n(Q(\bar{x}) \quad \mathrm{T}_m(Q(\bar{x}))$ is valid.

3. *semantically finite in an instance* r, if there is an n such that for all $m \quad n$, $r \models \forall\bar{x}\,(\mathrm{T}_n(Q(\bar{x}) \quad \mathrm{T}_m(Q(\bar{x}))$.

The number n in cases 2 and 3 is called a *point of finiteness*. It is clear that 1 implies 2 and 2 implies 3. In the full version we will show that all these implications are proper. In all these cases, evaluating $\mathrm{T}_\omega(Q(\bar{x})$ gives the same result as evaluating $\mathrm{T}_n(Q(\bar{x})$ for some n (in the instance r in case 3). If $\mathrm{T}_\omega(Q(\bar{x})$ is semantically finite, sound and complete, then the set of consistent answers to Q is *first-order definable*.

6.1 Syntactical finiteness

The notion of syntactical finiteness is important because then for some n and all $m > n$, $\mathrm{T}_m(Q(\bar{x}))$ will be exactly the same. In consequence, $\mathrm{T}_\omega(Q)$ will be a finite set of formulas. In addition, a point of finiteness n can be detected (if it exists) by syntactically comparing every two consecutive steps in the iteration. No simplification rules need to be considered, because the iterative procedure is fully deterministic.

Here we introduce a necessary and sufficient condition for syntactical finiteness.

Definition 14. A set of integrity constraints IC is *acyclic* if there exists a function f from predicate names plus negations of predicate names in the database to the natural numbers, that is, $f : \{p_1,\ldots,p_n,\neg p_1,\ldots,\neg p_n\} \quad \to \mathbb{N}$, such that for every integrity constraint $\forall(\bigvee_{i=1}^{k} l_i(\bar{x}_i) \vee \psi(\bar{x})) \in IC$ as in (3), and every i and j ($1 \quad i,j \quad k$), if $i \neq j$, then $f(\neg l_i) > f(l_j)$. (Here $\neg l_i$ is the literal complementary to l_i.)

Example 17. The set of ICs
$$IC = \{\forall x(\neg P(x) \vee \neg Q(x) \vee S(x)),$$
$$\forall(x,y)(\neg Q(x) \vee \neg S(y) \vee T(x,y))\}.$$

is acyclic, because the function f defined by
$$f(P) = 2 \quad f(Q) = 2 \quad f(\neg P) = 0 \quad f(\neg Q) = 0$$
$$f(S) = 1 \quad f(T) = 0 \quad f(\neg S) = 1 \quad f(\neg T) = 2,$$
satisfies the condition of definition 14.

Example 18. The set of ICs
$$IC = \{\forall x(\neg P(x) \vee \neg Q(x) \vee S(x)),$$
$$\forall(x,y)(Q(x) \vee \neg S(y) \vee T(x,y))\}.$$

is not acyclic, because for any function f that we may attempt to use to satisfy the condition in definition 14, from the first integrity constraint we obtain $f(Q) > f(S)$, and from the second, we would obtain $f(S) > f(Q)$; a contradiction.

Theorem 4. A set of integrity constraints IC is acyclic iff for every literal name l in the database schema, $\mathrm{T}_\omega(l(\bar{x}))$ is syntactically finite.

The theorem can be extended to any class of queries satisfying Definition 5.

Example 19. The set of integrity constraints in example 18 is not acyclic. In that case $\mathrm{T}_\omega(Q(x))$ is infinite.

Example 20. The ICs in example 17 are acyclic. There we have
$$\mathrm{T}_\omega(P(u)) =$$
$$\{P(u),$$
$$P(u) \wedge (\neg Q(u) \vee S(u)),$$
$$P(u) \wedge (\neg Q(u) \vee S(u) \wedge \forall v(\neg Q(v) \vee T(v,u)))\}$$

$$\mathrm{T}_\omega(Q(u)) =$$
$$\{Q(u),$$
$$Q(u) \wedge (\neg P(u) \vee S(u)) \wedge \forall v(\neg S(v) \vee T(u,v)),$$
$$Q(u) \wedge (\neg P(u) \vee S(u) \wedge \forall w(\neg Q(w) \vee T(w,u))) \wedge$$
$$\forall v(\neg S(v) \wedge (\neg P(v) \vee \neg Q(v)) \vee T(u,v))\}$$

$$\mathrm{T}_\omega(S(u)) = \{S(u), S(u) \wedge \forall(\neg Q(v) \vee T(v,u))\}$$

$$\mathrm{T}_\omega(T(u,v)) = \{T(u,v)\}$$

$$\mathrm{T}_\omega(\neg P(u)) = \{\neg P(u)\}$$

$$\mathrm{T}_\omega(\neg Q(u)) = \{\neg Q(u)\}$$

$$\mathrm{T}_\omega(\neg S(u)) = \{\neg S(u), \neg S(u) \wedge (\neg P(u) \vee \neg Q(u))\}$$

$$\mathrm{T}_\omega(\neg T(u,v)) =$$
$$\{\neg T(u,v),$$
$$\neg T(u,v) \wedge (\neg Q(u) \vee \neg S(v)),$$
$$\neg T(u,v) \wedge (\neg Q(u) \vee \neg S(v) \wedge (\neg P(v) \vee \neg Q(v)))\}.$$

Corollary 3. For functional dependencies and a query $Q(\bar{x})$, $\mathrm{T}_\omega(Q(\bar{x}))$ is always syntactically finite.

6.2 Semantical finiteness

Definition 15. A constraint C in clausal form is *uniform* if for every literal $l(\bar{x})$ in it, the set of variables in $l(\bar{x})$ is the same as the set of variables in $C \quad l(\bar{x})$. A set of constraints is uniform if all the constraints in it are uniform.

Examples of uniform constraints include set inclusion dependencies of the form $\forall\bar{x}(P(\bar{x}) \supset Q(\bar{x}))$, e.g., Example 4.

Theorem 5. If a set of integrity constraints IC is uniform, then for every literal name l in the database schema, $\mathrm{T}_\omega(l(\bar{x}))$ is semantically finite. Furthermore, a point of finiteness n can be bounded from above by a function of the number of variables in the query, and the number of predicates (and their arities) in the query and IC.

Theorem 6. Let l be a literal name. If for some n,
$$\forall\bar{x}(\mathrm{T}_n(l(\bar{x})) \supset \mathrm{T}_{n+1}(l(\bar{x})))$$
is valid, then for all $m \quad n$,
$$\forall\bar{x}(\mathrm{T}_n(l(\bar{x})) \quad \mathrm{T}_m(l(\bar{x})))$$
is valid.

According to Theorem 6, we can detect a point of finiteness by comparing every two consecutive steps wrt logical implication. Although this is undecidable in general, we might try to apply semidecision procedures, for example, automated theorem proving. We have successfully made use of OTTER [17] in some cases that involve sets of constraints that are neither acyclic nor uniform. Examples include multivalued dependencies, and functional dependencies together with set inclusion dependencies. For multivalued dependencies, Theorem 6 together with Theorem 3 gives completeness of $\mathrm{T}_\omega(l(\bar{x}))$ where $l(\bar{x})$ is a negative literal. The criterion from Theorem 6 is also applicable to uniform constraints by providing potentially faster termination detection than the proof of Theorem 5.

6.3 Instance based semantical finiteness

Theorem 7. If $Q(\bar{x})$ is a domain independent query, then for every database instance r there is an n, such that for all $m \quad n$, $r \models \forall\bar{x}(\mathrm{T}_n(Q(\bar{x})) \quad \mathrm{T}_m(Q(\bar{x})))$.

Notice that this theorem does not include the case of negative literals, as in the case of theorem 5.

7 Related work

Bry [4] was, to our knowledge, the first author to consider the notion of consistent query answer in inconsistent databases. He defined consistent query answers based on provability in minimal logic, without giving, however, a proof procedure or any other computational mechanism for obtaining such answers. He didn't address the issues of of semantics, soundness or completeness.

It has been widely recognized that in database integration the integrated data may be inconsistent with the integrity constraints. A typical (theoretical) solution is to augment the data model to represent disjunctive information. The following example explains the need for a solution of this kind.

Example 21. Consider the functional dependency
$$\forall(x,y,z)(P(x,y) \wedge P(x,z) \supset y = z.$$

If the integrated database contains both $P(a,b)$ and $P(a,c)$, then the functional dependency is violated. Each of $P(a,b)$ and $P(a,c)$ may be coming from a different database that satisfies the dependency. Thus, both facts are replaced by their disjunction $P(a,b) \vee P(a,c)$ in the integrated database. Now the functional dependency is no longer violated.

To solve this kind of problems [1] introduced the notion of *flexible relation*, a non-1NF relation that contains tuples with sets of non-key values (with such a set standing for *one* of its elements). This approach is limited to primary key functional dependencies and was subsequently generalized to other key functional dependencies [9]. In the same context, [3, 12] proposed to use disjunctive Datalog and [16] tables with OR-objects. [1] introduced flexible relational algebra to query flexible relations, and [9] - flexible relational calculus (whose subset can be translated to flexible relational algebra). The remaining papers did not discuss query language issues, relying on the existing approaches to query disjunctive Datalog or tables with OR-objects. There are several important differences between the above approaches and ours. First, they rely on the construction of a single (disjunctive) instance and the deletion of conflicting tuples. In our approach, the underlying databases are incorporated into the integrated one *in toto*, without any changes. There is no need for introducing disjunctive information. It would be interesting to compare the scope and the computational requirements of both approaches. For instance, one should note that the single-instance approach is not incremental: Any changes in the underlying databases require the recomputation of the entire instance. Second, our approach seems to be unique, in the context of database integration, in considering tuple insertions as possible repairs for integrity violations. Therefore, in some cases consistent query answers may be different from query answers obtained from the corresponding single instance.

Example 22. Consider the integrity constraint $p \supset q$ and a fact p. The instance consisting of p alone does not satisfy the integrity constraint. The common solution for removing this violation is to delete p. However, in our approach inserting q is also a possible repair. This has consequences for the inferences about $\neg p$ and $\neg q$. Our approach returns *false* in both cases, as p (resp. q) is true in a possible repair. Other approaches return *true* (under CWA) or *undefined* (under OWA).

Our work has connections with research done on belief revision [10]. In our case, we have an implicit notion of revision that is determined by the set of repairs of the database, and corresponds to revising the database (or a suitable categorical theory describing it) by the set of integrity constraints. Thus, querying the inconsistent database expecting only correct answers corresponds to querying the revised theory without restrictions.

It is easy to see that our notion of repair of a relational database is a particular case of the local semantics introduced in [8], restricted to revision performed starting from a single model (the database). From this we obtain that our revision operator satisfies the postulates (R1) – (R5),(R7), (R8) in [13]. For each given database r, the relation $\;_r$ introduced in definition 3 provides the partial order between models that determines the (models of the) revised database as described in [13]. [8] concentrates on the computation of the models of the revised theory, i.e. the repairs in our case, whereas we do not compute the repairs, but keep querying the original, non-revised database and pose a modified query. Therefore, we can view our methodology as a way of representing and querying simultaneously all the repairs of the database by means of a new query. Nevertheless, our motivation and starting point is quite different from belief revision. We attempt to take direct advantage of the semantic information contained in the integrity constraints in order to answer queries, rather than revising the database. Revising the database means repairing all the inconsistencies in it, instead we are interested in the information related to particular queries. For instance, a query referring only to the consistent portion of the database can be answered without repairing the database.

Reasoning in the presence of inconsistency has been an important research problem in the area of knowledge representation. The goal is to design logical formalisms that limit what can be inferred from an inconsistent set of formulas. One does not want to infer all formulas (as required by the classical two-valued logic). Also, one prefers not to infer a formula together with its negation. The formalisms satisfying the above properties, e.g., [15], are usually propositional. Moreover, they do not distinguish between integrity constraints and database facts. Thus, if the data in the database violates an integrity constraint, the constraint itself can no longer be inferred (which is not acceptable in the database context).

Example 23. Assume the integrity constraint is $\neg(p \wedge q)$ and the database contains the facts p and q. In the approach of [15], $p \vee q$ can be inferred (minimal change is captured correctly) but p, q and $\neg(p \wedge q)$ can no longer be inferred (they are all involved in an inconsistency).
Because of the above-mentioned limitations, such methods are not directly applicable to the problem of computing consistent query answers.

Deontic logic [18, 14], a modal logic with operators capturing permission and obligation, has been used for the specification of integrity constraints. [14] used the obligation operator O to distinguish integrity constraints that *have to hold always* from database facts that just *happen to hold*. [18] used deontic operators to describe policies whose violations can then be caught and handled. The issues of possible repairs of constraint violations, their minimality and consistent query answers are not addressed.

Gertz [11] described techniques and algorithms for computing repairs of constraint violations. The issue of query answering in the presence of an inconsistency is not addressed in his work.

8 Conclusions and Further Work

This paper represents a first step in the development of a new research area dealing with the theory and applications of consistent query answers in arbitrary, consistent or inconsistent, databases.

The theoretical results presented here are preliminary. We have proved a general soundness result but the results about completeness and termination are still partial. Also, one needs to look beyond purely universal constraints to include general inclusion dependencies. In a forthcoming paper we will also describe our methodology for using automated theorem proving, in particular, OTTER, for proving termination.

It appears that in order to obtain completeness for disjunctive and existentially quantified queries one needs to move beyond the T_ω operator on queries. Also, the upper bounds on the size of T_ω and the lower bounds on the complexity of computing consistent answers for different classes of queries and constraints need to be studied. In [2] it is shown that in the propositional case, SAT is reducible in polynomial time to the problem of deciding if an arbitrary formula evaluated in the propositional database does not give true as a correct answer, that is it becomes false in some repair. From this it follows that this problem is NP-complete.

There is an interesting connection to modal logic. Consider the definition 7. We could write $r \models \Box Q(\bar{t})$, meaning that $Q(\bar{t})$ is true in all repairs of r, the database instances that are "accessible" from r. This is even more evident from example 16, where, in essence, it is shown that $\Box\exists x Q(\bar{x})$ is not logically equivalent to $\exists x \Box Q(\bar{x})$, which is what usually happens in modal logic.

Acknowledgments

This research has been partially supported by FONDECYT Grants (1971304 & 1980945) and NSF Grant (IRI-9632870). Part of this research was done when the second author was on sabbatical at the Technical University of Berlin (CIS Group) with the financial support from DAAD and DIPUC.

References

[1] S. Agarwal, A.M. Keller, G. Wiederhold, and K. Saraswat. Flexible Relation: An Approach for Integrating Data from Multiple, Possibly Inconsistent Databases. In *IEEE International Conference on Data Engineering*, 1995.

[2] M. Arenas, L. Bertossi, and M. Kifer. APC and Querying Inconsistent Databases. In preparation.

[3] C. Baral, S. Kraus, J. Minker, and V.S. Subrahmanian. Combining Knowledge Bases Consisting of First-Order Theories. *Computational Intelligence*, 8:45–71, 1992.

[4] F. Bry. Query Answering in Information Systems with Integrity Constraints. In *IFIP WG 11.5 Working Conference on Integrity and Control in Information Systems*. Chapman &Hall, 1997.

[5] U.S. Chakravarthy, J. Grant, and J. Minker. Logic-Based Approach to Semantic Query Optimization. *ACM Transactions on Database Systems*, 15(2):162–207, 1990.

[6] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26, March 1997.

[7] J. Chomicki and G. Saake, editors. *Logics for Databases and Information Systems*. Kluwer Academic Publishers, Boston, 1998.

[8] T. Chou and M. Winslett. A Model-Based Belief Revision System. *J. Automated Reasoning*, 12:157–208, 1994.

[9] Phan Minh Dung. Integrating Data from Possibly Inconsistent Databases. In *International Conference on Cooperative Information Systems*, Brussels, Belgium, 1996.

[10] P. Gaerdenfors and H. Rott. Belief Revision. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 4, pages 35–132. Oxford University Press, 1995.

[11] M. Gertz. *Diagnosis and Repair of Constraint Violations in Database Systems*. PhD thesis, Universität Hannover, 1996.

[12] P. Godfrey, J. Grant, J. Gryz, and J. Minker. Integrity Constraints: Semantics and Applications. In Chomicki and Saake [7], chapter 9.

[13] H. Katsuno and A. Mendelzon. Propositional Knowledge Base Revision and Minimal Change. *Artificial Intelligence*, 52:263–294, 1991.

[14] K.L. Kwast. A Deontic Approach to Database Integrity. *Annals of Mathematics and Artificial Intelligence*, 9:205–238, 1993.

[15] J. Lin. A Semantics for Reasoning Consistently in the Presence of Inconsistency. *Artificial Intelligence*, 86(1-2):75–95, 1996.

[16] J. Lin and A. O. Mendelzon. Merging Databases under Constraints. *International Journal of Cooperative Information Systems*, 7(1):55–76, 1996.

[17] W.W. McCune. *OTTER 3.0 Reference Manual and Guide*. Argonne National Laboratory, Technical Report ANL-94/6, 1994.

[18] J.-J. Meyer, R. Wieringa, and F. Dignum. The Role of Deontic Logic in the Specification of Information Systems. In Chomicki and Saake [7], chapter 4.

[19] Jean-Marie Nicolas. Logic for Improving Integrity Checking in Relational Data Bases. *Acta Informatica*, 18:227–253, 1982.

[20] J. Ullman. *Principles of Database and Knowledge-Base Systems, Vol. I.* Computer Science Press, 1988.

Appendix: Proofs of Results

Some technical lemmas are stated without proof. Full proofs can be found in the file `proofspods99.ps` in `http://dcc.ing.puc.cl/~bertossi/`.

Lemma 1. If $r \quad T_\omega(l(\bar{a}))$, where $l(\bar{a})$ is a ground literal, then for every repair r' of r, it holds $r' \quad l(\bar{a})$.

Lemma 2. If $r \quad T_\omega(\bigwedge_{i=1}^{n} l_i(\bar{a}))$, where $l_i(\bar{a})$ is a ground literal, then for every repair r' of r, it holds $r' \quad \bigwedge_{i=1}^{n} l_i(\bar{a})$.

Lemma 3. If $r \quad T_\omega(\bigvee_{i=1}^{n} C_i(\bar{a}))$, with $C_i(\bar{a})$ a conjunction of literals, then for every repair r' of r, $r' \quad \bigvee_{i=1}^{n} C_i(\bar{a})$.

Lemma 4. Let $Q(\bar{x})$ a universal query. If $r \quad T_\omega(Q(\bar{t}))$, for a ground tuple $\bar{t}$, then for every repair r' of r, $r' \quad Q(\bar{t})$.

Lemma 5. Let $Q(\bar{x})$ a domain independent query. If r $T_\omega(Q(\bar{t}))$, for a ground tuple $\bar{t}$, then for every repair r' of r, $r' \quad Q(\bar{t})$.

Proof of Theorem 1: Lemmas 4 and 5.

Proposition 2. Given a set IC of integrity constraints, a ground clause $\bigvee_{i=1}^{m} l_i(\bar{t}_i)$, if $IC \not\models_{DB} \bigvee_{i=1}^{m} l_i(\bar{t}_i)$ and, for every repair r' of r, $r' \quad \bigvee_{i=1}^{m} l_i(\bar{t}_i)$, then $r \quad \bigvee_{i=1}^{m} l_i(\bar{t}_i)$.

Proof of Proposition 2: Assume that $r \quad \neg\bigvee_{i=1}^{m} l_i(\bar{t}_i)$. By hypothesis $IC \not\models_{DB} \bigvee_{i=1}^{m} l_i(\bar{t}_i)$, thus there exists an instance of the database r' such that $r' \quad IC \cup \{\neg\bigvee_{i=1}^{m} l_i(\bar{t}_i)\}$. Let us consider the set of database instances

$$R = \{r^* | r^* \quad IC \text{ and } \Delta(r, r^*) \quad \Delta(r, r')\}.$$

We know that $\Delta(r, r')$ is finite, therefore there exists $r_0 \in R$ such that $\Delta(r, r_0)$ is minimal. Then, r_0 is a repair of r.

For every $1 \quad i \quad m$, if $l_i(\bar{t}_i)$ is $p(\bar{t})$ or $\neg p(\bar{t})$, then $p(\bar{t}) \notin \Delta(r, r')$. Using this fact we conclude that $p(\bar{t}) \notin \Delta(r, r_0)$, Therefore, $r \quad \bigvee_{i=1}^{m} l_i(\bar{t}_i)$ if and only if $r_0 \quad \bigvee_{i=1}^{m} l_i(\bar{t}_i)$. But we assumed that $r \quad \neg\bigvee_{i=1}^{m} l_i(\bar{t}_i)$, then $r_0 \quad \neg\bigvee_{i=1}^{m} l_i(\bar{t}_i)$; a contradiction.

Proof of Theorem 2: From theorem 3.

Proof of Corollary 2: In this case it holds:

1. For every tuple $\bar{a}$, $IC \not\models_{DB} P_i(\bar{a})$, because the empty database instance (which has only empty base relations) satisfies IC, but not $P(\bar{a})$.
2. For every tuple $\bar{a}$, $IC \not\models_{DB} \neg P_i(\bar{a})$, since the database instance $r^i_{\bar{a}}$ where the relation P_i contains only the tuple $\bar{a}$ and the other relations are empty, satisfies IC, but not $\neg P_i(\bar{a})$.

Proof of Theorem 3: Suppose that $r \quad _c\, l(\bar{t})$. Let r' a repair of r, we have that $r' \quad l(\bar{t})$. By proposition 1 we have that $r' \quad T_n(l(\bar{t}))$, that is

$$r' \quad l(\bar{t}) \wedge \bigwedge_{i=1}^{m} \forall (\bigvee_{j=1}^{m_i} l_{i,j}(\bar{t}, \bar{x}_j) \vee \psi_i(\bar{t}, \bar{x})), \tag{6}$$

We want to prove that for every i and for every sequence of ground tuples $a_i, a_{i,1}, \ldots, a_{i,m_i}$:

$$r \quad \bigvee_{j=1}^{m_i} l_{i,j}(\bar{t}, \bar{a}_j) \vee \psi_i(\bar{t}, \bar{a}), \tag{7}$$

To do this, first we are going to prove that for every $i \in S$ and for every sequence of ground tuples $a_i, a_{i,1}, \ldots, a_{i,m_i}$:

$$r \quad \bigvee_{j=1}^{m_i} l_{i,j}(\bar{t}, \bar{a}_j) \vee \psi_i(\bar{t}, \bar{a}), \tag{8}$$

This is immediately obtained when $r \quad \psi_i(\bar{t}, \bar{a})$. Assume that $r \quad \neg\psi_i(\bar{t}, \bar{a})$. We know that ψ_i only mentions built-in predicates, thus for every repair r' of r we have that $r' \quad \neg\psi_i(\bar{t}, \bar{a})$. Therefore, by (6) we conclude that for every repair r' of r:

$$r' \quad \bigvee_{j=1}^{m_i} l_{i,j}(\bar{t}, \bar{a}_j) \vee \psi_i(\bar{t}, \bar{a}),$$

By proposition 2 we conclude (8). Thus we have that

$$r \quad l(\bar{t}) \wedge \bigwedge_{i \in S} \forall (\bigvee_{j=1}^{m_i} l_{i,j}(\bar{t}, \bar{x}_j) \vee \psi_i(\bar{t}, \bar{x})),$$

but by the second condition in the hypothesis of the theorem we conclude that:

$$r \quad l(\bar{t}) \wedge \bigwedge_{i=1}^{m} \forall (\bigvee_{j=1}^{m_i} l_{i,j}(\bar{t}, \bar{x}_j) \vee \psi_i(\bar{t}, \bar{x})).$$

Proof of Theorem 4: ($\Longrightarrow$) Suppose that IC is acyclic, then there exists f as in the definition 14. We are going to prove by induction on k that for every literal name l, if $f(l) = k$, then $T_{k+1}(l(\bar{x})) = T_{k+2}(l(\bar{x}))$
(I) If $k = 0$. We know that that for every literal name l', $f(l') \quad 0$. Therefore, every integrity constraint containing $\neg l$ is of the form $\forall(\neg l(\bar{x}) \vee \psi(\bar{y}))$, where ψ only mentions built-in predicates. This is because if there were any other literal l' in the integrity constraint, we would have $f(l') < f(l) = 0$. Then $T_1(l(\bar{x})) = T_2(l(\bar{x}))$.
(II) Suppose that the property is true for every $m < k$. We know that $T_{k+2}(l(\bar{x}))$ is of the form:

$$l(\bar{x}) \wedge \bigwedge_{i=1}^{m} \bar{Q}_i (\bigvee_{j=1}^{m_i} T_{k+1}(l_{i,j}(\bar{x}_j)) \vee \psi_i(\bar{x})),$$

where $\bar{Q}_i$ is a sequence of quantifiers over all the variables $\bar{x}_1, \ldots, \bar{x}_{m_i}, \bar{x}$ not appearing in $\bar{x}$, and $T_{k+1}(l(\bar{x}))$ is of the form:

$$l(\bar{x}) \wedge \bigwedge_{i=1}^{m} \bar{Q}_i (\bigvee_{j=1}^{m_i} T_k(l_{i,j}(\bar{x}_j)) \vee \psi_i(\bar{x})).$$

By definition of f, we know that for every literal name $l_{i,j}$ in the previous formulas, $f(l_{i,j}) < k$. Then by induction hypothesis $T_k(l(\bar{x}_j)) = T_{k+1}(l_{i,j}(\bar{x}_j))$ (since if $T_m(l'(\bar{x})) = T_{m+1}(l'(\bar{x}))$, then for every $n \quad m$, $T_n(l'(\bar{x})) = T_{n+1}(l'(\bar{x}))$).
($\Longleftarrow$) Suppose that for every literal name l, $T_\omega(l(\bar{x}))$ is finite. The for every literal name l there exists a first natural number k such that $T_k(l(\bar{x})) = T_{k+1}(l(\bar{x}))$. Let us define a function f, from the literal names into the natural number, by $f(l) = k$ (k as before). We can show that this is a well defined function that behaves as in definition 14: since if $\forall(\bigvee_{i=1}^{m} l_i(\bar{x}) \vee \psi(\bar{y})) \in IC$, then for every $1 \quad s \quad m$, $T_{f(\neg l_s)}(\neg l_s(\bar{x}))$ is of the form

$$\neg l_s(\bar{x}) \wedge \bar{Q}(\bigvee_{i=1}^{s-1} T_{f(\neg l_s)-1}(l_i(\bar{x})) \vee \bigvee_{i=s+1}^{m} T_{f(\neg l_s)-1}(l_i(\bar{x})) \vee \psi(\bar{y})) \wedge \theta(\bar{x}_s), \tag{9}$$

where $\bar{Q}$ is a sequence of quantifiers over all the variables $\bar{x}_1, \ldots, \bar{x}_m, \bar{y}$, not appearing in $\bar{x}_s$ and $T_{f(\neg l_s)+1}(\neg l_s(\bar{x}))$ is of the form

$$\neg l_s(\bar{x}) \wedge \bar{Q}(\bigvee_{i=1}^{s-1} T_{f(\neg l_s)}(l_i(\bar{x})) \vee \bigvee_{i=s+1}^{m} T_{f(\neg l_s)}(l_i(\bar{x})) \vee \psi(\bar{y})) \wedge \theta'(\bar{x}). \tag{10}$$

By definition of f, $T_{f(\neg l_s)}(\neg l_s(\bar{x})) = T_{f(\neg l_s)+1}(\neg l_s(\bar{x}))$. Then, by the form of (9) and (10), we conclude that for every $i \neq s$, $T_{f(\neg l_s)-1}(l_i(\bar{x})) = T_{\neg f(l_s)}(l_i(\bar{x}))$, and then, again by definition of f, $f(l_i) < f(\neg l_s)$.

Proof of Corollary 3: The following stratification function from literals to $\mathbb{N}$ can be defined: $f(\neg P_i) = 0$ and $f(P_j) = 1$, where P_i, P_j are relation names.

Proof of Theorem 5: For uniform constraints the residues do not contain quantifiers. Therefore $T_n(l(\bar{x}))$ for every n 0 is quantifier-free and contains only the variables that occur in $\bar{x}$. There are only finitely many inequivalent formulas with this property, and thus $T_\omega(l(\bar{x}))$ is finite.

Lemma 6. If $T_n(l(\bar{x}))$ is of the form:

$$l(\bar{x}) \wedge \bigwedge_{i=1}^{m} \forall(\bar{x}_i, \bar{y}_i)(C_i(\bar{x}, \bar{x}_i) \vee \psi_i(\bar{x}, \bar{y}_i)),$$

then $T_{n+1}(l(\bar{x}))$ is of the form:

$$l(\bar{x}) \wedge \bigwedge_{i=1}^{m} \forall(\bar{x}_i, \bar{y}_i)(T_1(C_i(\bar{x}, \bar{x}_i)) \vee \psi_i(\bar{x}, \bar{y}_i)),$$

Lemma 7. If for a ground tuple $\bar{a}$, $T_n(l(\bar{a})) \quad \forall(\bigvee_{j=1}^{k} l'_j(\bar{a}, \bar{z}_j))$, then $T_{n+1}(l(\bar{a})) \quad \forall(\bigvee_{j=1}^{k} T_1(l'_j(\bar{a}, \bar{z}_j)))$.

Proof of Theorem 6: Suppose that for a natural number n, $\forall\bar{x}(T_n(l(\bar{x})) \supset T_{n+1}(l(\bar{x})))$ is a valid sentence. We are going to prove that for every $m \quad n$, $\forall\bar{x}(T_m(l(\bar{x})) \supset T_{m+1}(l(\bar{x})))$ is a valid sentence, by induction on m.
(I) If $m = n$, by hypothesis.
(II) Suppose that $\forall\bar{x}(T_m(l(\bar{x})) \supset T_{m+1}(l(\bar{x})))$ is a valid sentence. For every clause $\bigvee_{j=1}^{k} l'_j(\bar{x}, \bar{z}_j) \vee \psi(\bar{x}, \bar{z})$ in $T_{m+1}(l(\bar{x}))$ and for every ground tuple $\bar{a}$ we have that

$$T_m(l(\bar{a})) \quad \forall(\bigvee_{j=1}^{k} l'_j(\bar{a}, \bar{z}_j) \vee \psi(\bar{a}, \bar{z})).$$

By lemma 7 and considering that ψ only mentions built-in predicates we have that $T_{m+1}(l(\bar{a})) \quad \forall(\bigvee_{j=1}^{k} T_1(l'_j(\bar{a}, \bar{z}_j)) \vee \psi(\bar{a}, \bar{z}))$, and from this and lemma 6 we can conclude that $\forall\bar{x}(T_{m+1}(l(\bar{x})) \supset T_{m+2}(l(\bar{x})))$ is a valid sentence.

Proof of Theorem 7: Let $Q(\bar{x})$ be a domain independent query and r a database instance. Define $A_n = \{\bar{t} \mid r \quad T_n(Q(\bar{t}))\}$. We know that for every n: $A_{n+1} \quad A_n$, therefore $A = \{A_i \mid i < \omega\}$ is a family of subsets of A_0. But A_0 is finite because $Q(\bar{x})$ is a domain independent query. Thus, there exists a minimal element A_m in A. For this element, it holds that for every $k \quad m$: $A_m = A_k$, since $A_k \quad A_m$.

Applications of Annotated Predicate Calculus to Querying Inconsistent Databases

Marcelo Arenas **Leopoldo Bertossi**
P. Universidad Catolica de Chile
Depto. Ciencia de Computacion
Casilla 306, Santiago 22, Chile
{marenas,bertossi}@ing.puc.cl

Michael Kifer
Department of Computer Science
University at Stony Brook
Stony Brook, NY 11794, USA
kifer@cs.sunysb.edu

Abstract. We consider the problem of specifying and computing consistent answers to queries against databases that do not satisfy given integrity constraints. This is done by simultaneously embedding the database and the integrity constraints, which are mutually inconsistent in classical logic, into a theory in annotated predicate calculus — a logic that allows non trivial reasoning in the presence of inconsistency. In this way, several goals are achieved: (a) A logical specification of the class of all minimal "repairs" of the original database, and the ability to reason about them; (b) The ability to distinguish between consistent and inconsistent information in the database; and (c) The development of computational mechanisms for retrieving consistent query answers, *i.e.*, answers that are not affected by the violation of the integrity constraints.

1 Introduction

Databases that violate stated integrity constraints is an (unfortunate) fact of life for many corporations. They arise due to poor data entry control, due to merges of previously separate databases, due to the incorporation of legacy data, and so on. We call such databases "inconsistent."

Even though the information stored in such a database might be logically inconsistent (and, thus, strictly speaking, *any* tuple should be viewed as a correct query answer), this has not been a deterrent to the use of such databases in practice, because application programmers have been inventing ingenious techniques for salvaging "good" information. Of course, in such situations, what is good information and what is not is in the eyes of beholder, and each concrete case currently requires a custom solution. This situation can be compared to the times before the advent of relational databases, when every database query required a custom solution.

Thus, the problem is: what is the definition of "good information" in an inconsistent database and, once this is settled, what is the meaning of a query in this case. Several proposals to address these problems — both semantically and computationally — are known (*e.g.*, [1]), and we are not going to propose yet another definition for consistent query answers. Instead, we introduce a new *semantic framework*, based on Annotated Predicate Calculus [9], that leads to a different computational solution and provides a basis for a systematic study of the problem.

Ultimately, our framework leads to the query semantics proposed in [1]. According to [1], a tuple $\bar{t}$ is an answer to the query $Q(\bar{x})$ in a possibly inconsistent database instance r, if $Q(\bar{t})$ holds true in all the "repairs" of the original database, that is in all the databases that satisfy the given constraints and can be obtained from r by means of a "minimal" set of changes (where minimality is measured in terms of a smallest symmetric set difference).

In [1], an algorithm is proposed whereby the original query is modified using the set of integrity constraints (that are violated by the database). The modified query is then posed against the original database (with the integrity constraints ignored). In this way, the explicit integrity checking and computation of all database repairs is avoided.

In this paper, we take a more direct approach. First, since the database is inconsistent with the constraints, it seems natural to embed it into a logic that is better suited for dealing with inconsistency than classical logic. In this paper we use *Annotated Predicate Calculus* (abbr. APC) introduced in [9]. APC is a form of "paraconsistent logic," *i.e.*, logic where inconsistent information does not unravel logical inference and where causes of inconsistency can be reasoned about. APC generalizes a number of earlier proposals [12, 11, 3] and its various partial generalizations have also been studied in different contexts (*e.g.*, [10]).

The gist of our approach is to embed an inconsistent database theory in APC and then use APC to define database repairs and query answers. This helps understand the results of [1], leads to a more straightforward complexity analysis, and provides a more general algorithm that covers classes of queries not included in [1]. Furthermore, by varying the semi-lattice underlying the host APC theory, it is possible to control how exactly inconsistency is resolved in the original database.

Section 2 formalizes the problem of querying inconsistent databases. Section 3 reviews the basic definitions of Annotated Predicate Calculus, and Section 4 applies this calculus to our problem. In Section 5, we provide a syntactic characterization for database repairs and discuss the associated computational process. Section 6 studies the problem of query evaluation in inconsistent databases and Section 7 concludes the paper.

2 Preliminaries

We assume we have a fixed database schema $P = \{p_1, \ldots, p_n\}$, where p_1, ..., p_n are predicates corresponding to the database relations; a fixed, possibly infinite database domain $D = \{c_1, c_2, ...\}$; and a fixed set of built-in predicates $B = \{e_1, \ldots, e_m\}$. Each predicate has *arity*, *i.e.*, the number of arguments it takes. An integrity constraint is a closed first-order formula in the language defined by the above components. We also assume a first order language $\mathcal{L} = D \cup P \cup B$ that is based on this schema.

Definition 1. *(Databases and Constraints) A* database instance **DB** *is a finite collection of facts, i.e., of statements of the form $p(c_1, ..., c_n)$, where p is a predicate in P and $c_1, ..., c_n$ are constants in D.*

An integrity constraint *is a clause of the form*

$$p_1(\bar{T}_1) \vee \cdots \vee p_n(\bar{T}_n) \vee \neg q_1(\bar{S}_1) \vee \cdots \vee \neg q_m(\bar{S}_m)$$

where each p_i $(1 \leq i \leq n)$ and q_j $(q \leq j \leq m)$ is a predicate in $P \cup B$ and $\bar{T}_1, ..., \bar{T}_n, \bar{S}_1, ..., \bar{S}_m$ are tuples (of appropriate arities) of constants or variables. As usual, we assume that all variables in a clause are universally quantified, so the quantifiers are omitted.

Throughout this paper we assume that both the database instance **DB** and the set of integrity constraints **IC** are consistent when considered in isolation. However, together $\mathbf{DB} \cup \mathbf{IC}$ might not be consistent.

Definition 2. *(Sentence Satisfaction) We use $\models_{DB}$ to denote the usual notion of formula satisfaction in a database. The subscript DB is used to distinguish this relation from other types of implication used in this paper. In other words,*

- $\mathbf{DB} \models_{DB} p(\bar{c})$*, where* $p \in P$*, iff* $p(\bar{c}) \in \mathbf{DB}$*;*
- $\mathbf{DB} \models_{DB} q(\bar{c})$*, where* $q \in B$*, iff* $q(\bar{c})$ *is true;*
- $\mathbf{DB} \models_{DB} \neg\varphi$ *iff it is not true that* $\mathbf{DB} \models_{DB} \varphi$*;*
- $\mathbf{DB} \models_{DB} \phi \wedge \psi$ *iff* $\mathbf{DB} \models_{DB} \phi$ *and* $\mathbf{DB} \models_{DB} \psi$*;*
- $\mathbf{DB} \models_{DB} (\forall X)\phi(X)$ *iff for all* $d \in D$, $\mathbf{DB} \models_{DB} \phi(d)$*;*

and so on. Notice that the domain is fixed, and it is involved in the above definition.

Definition 3. *(IC Satisfaction) A database instance* **DB** *satisfies a set of integrity constraints* **IC** *iff for every* $\varphi \in \mathbf{IC}$, $\mathbf{DB} \models_{DB} \varphi$.

If **DB** *does not satisfy* **IC***, we say that* **DB** *is* inconsistent *with* **IC**. *Additionally, we say that a set of integrity constraints is* consistent *if there exists a database instance that satisfies it.*

Next we recall the relevant definitions from [1].

Given two database instances $\mathbf{DB}_1$ and $\mathbf{DB}_2$, the *distance* $\Delta(\mathbf{DB}_1, \mathbf{DB}_2)$ between them is their symmetric difference: $\Delta(\mathbf{DB}_1, \mathbf{DB}_2) = (\mathbf{DB}_1 - \mathbf{DB}_2) \cup (\mathbf{DB}_2 - \mathbf{DB}_1)$. This leads to the following partial order:

$$\mathbf{DB}_1 \leq_{\mathbf{DB}} \mathbf{DB}_2 \ \text{ iff } \ \Delta(\mathbf{DB}, \mathbf{DB}_1) \subseteq \Delta(\mathbf{DB}, \mathbf{DB}_2).$$

That is, $\leq_{\mathbf{DB}}$ determines the "closeness" to **DB**. The notion of closeness forms the basis for the concept of a repair of an inconsistent database.

Definition 4. *(Repair) Given database instances* **DB** *and* $\mathbf{DB}'$*, we say that* $\mathbf{DB}'$ *is a* repair *of* **DB** *with respect to a set of integrity constraints* **IC** *iff* $\mathbf{DB}'$ *satisfies* **IC** *and* $\mathbf{DB}'$ *is* $\leq_{\mathbf{DB}}$*-minimal in the class of database instances that satisfy* **IC**.

Clearly if **DB** is consistent with **IC**, then **DB** is its own repair. Concepts similar to database repair were proposed in the context of database maintenance and belief revision [7, 4].

Example 1. (Repairing a database) Consider a database schema with two unary relations p and q and domain $D = \{a, b, c, \ldots\}$. Let $\mathbf{DB} = \{p(a), p(b), q(a), q(c)\}$ be a database instance over the domain D and let $\mathbf{IC} = \{\neg p(x) \vee q(x)\}$ be a set of constraints. This database does not satisfy **IC** because $\neg p(b) \vee q(b)$ is false.

Two repairs are possible. First, we can make $p(b)$ false, obtaining $\mathbf{DB}' = \{p(a), q(a), q(c)\}$. Alternatively, we can make $q(b)$ true, obtaining $\mathbf{DB}'' = \{p(a), p(b), q(a), q(b), q(c)\}$.

Definition 5. *(Consistent Answers) Let* **DB** *be a database instance,* **IC** *be set of integrity constraints and* $Q(\bar{x})$ *be a query. We say that a tuple of constants* $\bar{t}$ *is a* consistent answer *to the query, denoted* $\mathbf{DB} \models_c Q(\bar{t})$*, if for every repair* $\mathbf{DB}'$ *of* **DB**, $\mathbf{DB}' \models_{DB} Q(\bar{t})$.

If Q is a closed formula, then true (respectively, false) is a consistent answer *to Q, denoted* $\mathbf{DB} \models_c Q$*, if* $\mathbf{DB}' \models_{DB} Q$ *(respectively,* $\mathbf{DB}' \not\models_{DB} Q$*) for every repair* $\mathbf{DB}'$ *of* **DB**.

3 Annotated Predicate Calculus

Annotated predicate calculus (abbr. APC) [9] is a generalization of annotated logic programs introduced by Blair and Subrahmanian [3]. It was introduced in order to study the problem of "causes of inconsistency" in classical logical theories, which is closely related to the problem of consistent query answers being addressed in our present work. This section briefly surveys the basics of APC used in this paper.

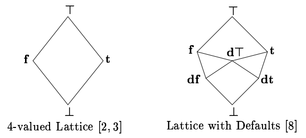
The syntax and the semantics of APC is based on classical logic, except that the classical atomic formulas are annotated with values drawn from a *belief semilattice* (abbr. *BSL*) — an upper semilattice[1] with the following properties:

(i) *BSL* contains at least the following four distinguished elements: **t** (true), **f** (false), $\top$ (contradiction), and $\bot$ (unknown);
(ii) For every $\mathbf{s} \in BSL$, $\bot \leq \mathbf{s} \leq \top$ ($\leq$ is the semilattice ordering);
(iii) $\mathtt{lub}(\mathbf{t}, \mathbf{f}) = \top$, where `lub` denotes the least upper bound.

As usual in the lattice theory, `lub` imposes a partial order on *BSL*: $a \leq b$ iff $b = \mathtt{lub}(a, b)$ and $a < b$ iff $a \leq b$ and a is different from b. Two typical examples of *BSL* (which happen to be complete lattices) are shown in Figure 1. In both of them, the lattice elements are ordered upwards. The specific *BSL* used in this paper is introduced later, in Figure 2.

Thus, the only syntactic difference between APC and classical predicate logic is that the atomic formulas of APC are constructed from the classical atomic

[1] That is, the least upper bound, $\mathtt{lub}(a, b)$, is defined for every pair of elements $a, b \in BSL$.



4-valued Lattice [2, 3]   Lattice with Defaults [8]

Fig. 1. Typical Belief Semilattices

formulas by attaching annotation suffixes. For instance, if $\mathbf{s}$, $\mathbf{t}$, $\top$ are elements of the belief semilattice, then $p(X) : \mathbf{s}$, $q : \top$, and $r(X, Y, Z) : \mathbf{t}$ all are atomic formulas in APC.

We define only the Herbrand semantics of APC (this is all we need here), and we also assume that the language is free of function symbols (because we are dealing with relational databases in this paper). We thus assume that the *Herbrand universe* is D, the set of all domain constants, and the *Herbrand base*, HB, is the set of all ground (*i.e.*, variable-free) atomic formulas of APC.

A *Herbrand interpretation* is any downward-closed subset of HB, where a set $I \subseteq HB$ is said to be *downward-closed* iff $p : \mathbf{s} \in I$ implies that $p : \mathbf{s}' \in I$ for every $\mathbf{s}' \in BSL$ such that $\mathbf{s}' \leq \mathbf{s}$. Formula satisfaction can then be defined as follows, where ν is a variable assignment that gives a value in D to every variable:

- $I \models_v p : \mathbf{s}$, where $\mathbf{s} \in BSL$ and p is a classical atomic formula, if and only if $p : \mathbf{s} \in I$.
- $I \models_v \phi \wedge \psi$ if and only if $I \models_v \phi$ and $I \models_v \psi$;
- $I \models_v \neg\psi$ if and only if not $I \models_v \psi$;
- $I \models_v (\forall X)\psi(X)$ if and only if $I \models_u \psi$, for *every* u that may differ from v only in its X-value.

It is thus easy to see that the definition of $\models$ looks very much classical. The only difference (which happens to have significant implications) is the syntax of atomic formulas and the requirement that Herbrand interpretations must be downward-closed. The implication $a \longleftarrow b$ is also defined classically, as $a \vee \neg b$.

It turns out that whether or not APC has a complete proof theory depends on which semilattice is used. It is shown in [9] that for a very large and natural class of semilattices (which includes all finite semilattices), APC has a sound and complete proof theory.

The reason why APC is useful in analyzing inconsistent logical theories is because classical theories can be embedded in APC in various ways. The most useful types of embeddings are those where theories that are inconsistent in classical logic become consistent in APC. It then becomes possible to reason about the embedded theories and gain insight into the original inconsistent theory.

The two embeddings defined in [9] are called *epistemic* and *ontological.* Under the epistemic embedding, a (classically inconsistent) set of formulas such as $\mathbf{S} = \{p(1),\ \neg p(1),\ q(2)\}$ is embedded in APC as $\mathbf{S}^e = \{p(1) : \mathbf{t},\ p(1) : \mathbf{f},\ q(2) : \mathbf{t}\}$ and under the ontological embedding it is embedded as $\mathbf{S}^o = \{p(1) : \mathbf{t},\ \neg p(1) : \mathbf{t},\ q(2) : \mathbf{t}\}$.[2] In the second case, the embedded theory is still inconsistent in APC, but in the first case it does have a model: the downward closure of $\{p(1) : \top,\ q(2) : \mathbf{t}\}$. In this model, $p(1)$ is annotated with $\top$, which signifies that its truth value is "inconsistent." In contrast, the truth value of $q(2)$ is $\mathbf{t}$. More precisely, while both $q(2)$ and $\neg q(2)$ follow from $\mathbf{S}$ in classical logic, because $\mathbf{S}$ is inconsistent, only $q(2) : \mathbf{t}$ (but not $q(2) : \mathbf{f}$!) is implied by $\mathbf{S}^e$. Thus, $q(2)$ can be seen as a consistent answer to the query $?-\ q(X)$ with respect to the inconsistent database $\mathbf{S}$.

In [9], epistemic embedding has been shown to be a suitable tool for analyzing inconsistent classical theories. However, this embedding does not adequately capture the inherent lack of symmetry present in our setting, where inconsistency arises due to the incompatibility of two distinct sets of formulas (the database and the constraints) and only one of these sets (the database) is allowed to change to restore consistency. To deal with this problem, we develop a new type of embedding into APC. It uses a 10-valued lattice depicted in Figure 2, and is akin to the epistemic embedding of [9], but it also has certain features of the ontological embedding.

The above simple examples illustrate one important property of APC: a set of formulas, $\mathbf{S}$, might be *ontologically consistent* in the sense that it might have a model, but it might be *epistemically inconsistent* (abbr. *e-inconsistent*) in the sense that $\mathbf{S} \models p : \top$ for some p, *i.e.*, $\mathbf{S}$ contains at least one inconsistent fact. Moreover, $\mathbf{S}$ can be e-consistent (*i.e.*, it might not imply $p : \top$ for any p), but each of its models in APC might contain an inconsistent fact nonetheless (this fact must then be different in each model, if $\mathbf{S}$ is e-consistent).

It was demonstrated in [9] that ordering models of APC theories according to the amount of inconsistency they contain can be useful for studying the problem of recovering from inconsistency. To illustrate this order, consider $\mathbf{S} = \{p : \mathbf{t},\ p : \mathbf{f} \vee q : \mathbf{t},\ p : \mathbf{f} \vee q : \mathbf{f}\}$ and some of its models:

$\mathcal{M}_1$, where $p : \top$ and $q : \top$ are true;
$\mathcal{M}_2$, where $p : \top$ and $q : \bot$ are true;
$\mathcal{M}_3$, where $p :\ \mathbf{t}$ and $q : \top$ are true.

Among these models, both $\mathcal{M}_2$ and $\mathcal{M}_3$ contain strictly less inconsistent information than $\mathcal{M}_1$ does. In addition, $\mathcal{M}_2$ and $\mathcal{M}_3$ contain incomparable amounts of information, and they are both "minimal" with respect to the amount of inconsistent information that they have. This leads to the following definition.

Definition 6. *(E-consistency Order) Given $\Delta \subseteqq BSL$, a semantic structure I_1 is more (or equally)* e-consistent *than I_2 with respect to Δ (denoted $I_2 \leq_\Delta I_1$) if and only if for every atom $p(t_1, \ldots, t_k)$ and $\lambda \in \Delta$, whenever $I_1 \models p(t_1, \ldots, t_k) : \lambda$ then also $I_2 \models p(t_1, \ldots, t_k) : \lambda$.*

[2] $\neg p : \mathbf{v}$ is to be always read as $\neg(p : \mathbf{v})$.

I is most e-consistent *in a class of semantic structures with respect to Δ, if no semantic structure in this class is strictly more e-consistent with respect to Δ than I (i.e., for every J in the class, $I \leq_\Delta J$ implies $J \leq_\Delta I$).*

4 Embedding Databases in APC

One way to find reliable answers to a query over an inconsistent database is to find an algorithm that implements the definition of consistent answers. While this approach has been successfully used in [1], it is desirable to see it as part of a bigger picture, because consistent query answers were defined at the meta-level, without an independent logical justification. A more general framework might (and does, as we shall see) help study the problem both semantically and algorithmically.

Our new approach is to embed inconsistent databases into APC and study the ways to eliminate inconsistency there. A similar problem was considered in [9] and we are going to adapt some key ideas from that work. In particular, we will define an embedding, $\mathcal{T}$, such that the repairs of the original database are precisely the models (in the APC sense) of the embedded database. This embedding is described below.

First, we define a special 10-valued lattice, $\mathcal{L}^{\mathbf{db}}$, which defines the truth values appropriate for our problem. The lattice is shown in Figure 2. The values $\bot$, $\top$, $\mathbf{t}$ and $\mathbf{f}$ signify undefinedness, inconsistency, truth, and falsehood, as usual. The other six truth values are explained below.

Informally, values $\mathbf{t_c}$ and $\mathbf{f_c}$ signify the truth values as they should be for the purpose of constraint satisfaction. The values $\mathbf{t_d}$ and $\mathbf{f_d}$ are the truth values as they should be according to the database $\mathbf{DB}$. Finally, $\mathbf{t_a}$ and $\mathbf{f_a}$ are the *advisory* truth values. Advisory truth values are intended as keepers of the information that helps resolve conflicts between constraints and the database.

Notice that $\mathtt{lub}(\mathbf{f_d}, \mathbf{t_c})$ is $\mathbf{t_a}$ and $\mathtt{lub}(\mathbf{t_d}, \mathbf{f_c})$ is $\mathbf{f_a}$. This means that in case of a conflict between the constraints and the database the advise is to change the truth value of the corresponding fact to the one prescribed by the constraints. Intuitively, the facts that are assigned the advisory truth values are the ones that are to be removed or added to the database in order to satisfy the constraints. The gist of our approach is in finding an embedding of $\mathbf{DB}$ and $\mathbf{IC}$ into APC to take advantage of the above truth values.

Embedding the ICs. Given a set of integrity constraints $\mathbf{IC}$, we define a new theory, $\mathcal{T}(\mathbf{IC})$, which contains three kinds of formulas:

1. For every constraint in $\mathbf{IC}$:

$$p_1(\bar{T}_1) \quad \vee \quad \cdots \quad \vee \quad p_n(\bar{T}_n) \quad \vee \quad \neg q_1(\bar{S}_1) \quad \vee \quad \cdots \quad \vee \quad \neg q_m(\bar{S}_m),$$

   $\mathcal{T}(\mathbf{IC})$ has the following formula:

$$p_1(\bar{T}_1) : \mathbf{t_c} \ \vee \cdots \vee\ p_n(\bar{T}_n) : \mathbf{t_c} \ \vee\ q_1(\bar{S}_1) : \mathbf{f_c} \ \vee \cdots \vee\ q_m(\bar{S}_m) : \mathbf{f_c}.$$

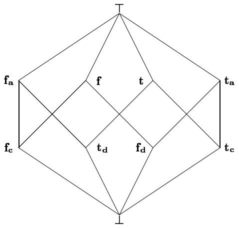


Fig. 2. The lattice $\mathcal{L}^{\mathbf{db}}$ with *constraints values*, *database values* and *advisory values.*

   In other words, positive literals are embedded using the "constraint-true" truth value, $\mathbf{t_c}$, and negative literals are embedded using the "constraint-false" truth value $\mathbf{f_c}$.

2. For every predicate symbol $p \in P$, the following formulas are in $\mathcal{T}(\mathbf{IC})$:

$$p(\bar{x}) : \mathbf{t_c} \ \vee\ p(\bar{x}) : \mathbf{f_c},\ \neg\ p(\bar{x}) : \mathbf{t_c} \ \vee\ \neg\ p(\bar{x}) : \mathbf{f_c}.$$

   Intuitively, this says that every embedded literal must be either constraint-true or constraint-false (and not both).

Embedding database facts. $\mathcal{T}(\mathbf{DB})$, the embedding of the database facts into APC is defined as follows:

1. For every fact $p(\bar{a})$, where $p \in P$: if $p(\bar{a}) \in \mathbf{DB}$, then $p(\bar{a}) : \mathbf{t_d} \in \mathcal{T}(\mathbf{DB})$; if $p(\bar{a}) \notin \mathbf{DB}$, then $p(\bar{a}) : \mathbf{f_d} \in \mathcal{T}(\mathbf{DB})$.

Embedding built-in predicates. $\mathcal{T}(\mathcal{B})$, the result of embedding of the built-in predicates into APC is defined as follows:

1. For every built-in fact $p(\bar{a})$, where $p \in B$, the fact $p(\bar{a}) : \mathbf{t}$ is in $\mathcal{T}(\mathcal{B})$ iff $p(\bar{a})$ is true. Otherwise, if $p(\bar{a})$ is false then $p(\bar{a}) : \mathbf{f}\ \in \mathcal{T}(\mathcal{B})$.
2. $\neg\ p(\bar{x}) : \top\ \in \mathcal{T}(\mathcal{B})$, for every built-in $p \in B$.

The former rule simply says that built-in facts (like 1=1) that are true in classical sense must have the truth value $\mathbf{t}$ and the false built-in facts (*e.g.*, 2=3) must have the truth value $\mathbf{f}$. The second rule states that built-in facts cannot be both true and false. This ensures that theories for built-in predicates are embedded in 2-valued fashion: every built-in fact in $\mathcal{T}(\mathcal{B})$ is annotated with either $\mathbf{t}$ or $\mathbf{f}$, but not both.

Example 2. (Embedding, I) Consider the database $\mathbf{DB} = \{p(a), p(b), q(a)\}$ over the domain $D = \{a, b\}$ and let $\mathbf{IC}$ be $\{\neg p(x) \vee q(x)\}$. Then

$$\mathcal{T}(\mathbf{DB}) = \{p(a) : \mathbf{t_d},\ p(b) : \mathbf{t_d},\ q(a) : \mathbf{t_d},\ q(b) : \mathbf{f_d}\}$$

and $\mathcal{T}(\mathbf{IC})$ consists of:

$$\begin{array}{l} p(x) : \mathbf{f_c} \ \vee\ q(x) : \mathbf{t_c}, \\ p(x) : \mathbf{t_c} \ \vee\ p(x) : \mathbf{f_c},\ \ \neg\ p(x) : \mathbf{t_c} \ \vee\ \neg\ p(x) : \mathbf{f_c}, \\ q(x) : \mathbf{t_c} \ \vee\ q(x) : \mathbf{f_c},\ \ \neg\ q(x) : \mathbf{t_c} \ \vee\ \neg\ q(x) : \mathbf{f_c} \end{array}$$

Example 3. (Embedding, II) Let $\mathbf{DB} = \{p(a,a), p(a,b), p(b,a)\}$, $D = \{a, b\}$, and let $\mathbf{IC}$ be $\{\neg p(x,y) \vee \neg p(x,z) \vee y = z\}$. It is easy to see that this constraint represents the functional dependency $p.1 \rightarrow p.2$. Since this constraint involves the built-in "=", the rules for embedding the built-ins apply.

In this case, $\mathcal{T}(\mathbf{DB}) = \{p(a,a) : \mathbf{t_d},\ \ p(a,b) : \mathbf{t_d},\ \ p(b,a) : \mathbf{t_d},\ \ p(b,b) : \mathbf{f_d}\}$ and $\mathcal{T}(\mathbf{IC})$ is:

$$\begin{array}{l} p(x,y) : \mathbf{f_c} \ \vee\ p(x,z) : \mathbf{f_c} \ \vee\ y = z : \mathbf{t_c}, \\ p(x,y) : \mathbf{t_c} \ \vee\ p(x,y) : \mathbf{f_c},\ \neg\ p(x,y) : \mathbf{t_c} \ \vee\ \neg\ p(x,y) : \mathbf{f_c}. \end{array}$$

The embedded theory $\mathcal{T}(\mathcal{B})$ for the built-in predicate "=" is: $(a = a) : \mathbf{t}$, $(b = b) : \mathbf{t}$, $(a = b) : \mathbf{f}$, $(b = a) : \mathbf{f}$, $\neg\ (x = y) : \top$. □

Finally, we define $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$ as $\mathcal{T}(\mathbf{DB}) \cup \mathcal{T}(\mathbf{IC}) \cup \mathcal{T}(\mathcal{B})$. We can now state the following properties that confirm our intuition about the intended meanings of the truth values in $\mathcal{L}^{\mathbf{db}}$.

Lemma 1. *If $\mathcal{M}$ is a model of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$, then for every predicate $p \in P$ and a fact $p(\bar{a})$, the following is true:*

1. $\mathcal{M} \models \neg\ p(\bar{a}) : \top$.
2. $\mathcal{M} \models p(\bar{a}) : \mathbf{t} \ \vee\ p(\bar{a}) : \mathbf{f} \ \vee\ p(\bar{a}) : \mathbf{t_a} \vee p(\bar{a}) : \mathbf{f_a}$. □

The first part of the lemma says that even if the initial database $\mathbf{DB}$ is inconsistent with constraints $\mathbf{IC}$, every model of our embedded theory is *epistemically consistent* in the sense of [9], *i.e.*, no fact of the form $p(\bar{a}) : \top$ is true in any such model.[3] The second part says that any fact is either true, or false, or it has an advisory value of true or false. This indicates that database repairs can be constructed out of these embeddings by converting the advisory truth values to the corresponding values $\mathbf{t}$ and $\mathbf{f}$. This idea is explored next.

Given a pair of database instances $\mathbf{DB}_1$ and $\mathbf{DB}_2$ over the same domain, we construct the Herbrand structure $\mathcal{M}(\mathbf{DB}_1, \mathbf{DB}_2) = \langle D, I_P, I_B \rangle$, where D is the domain of the database and I_P, I_B are the interpretations for the predicates and the built-ins, respectively. I_P is defined as follows:

$$I_P(p(\bar{a})) = \begin{cases} \mathbf{t} & p(\bar{a}) \in \mathbf{DB}_1,\ p(\bar{a}) \in \mathbf{DB}_2 \\ \mathbf{f} & p(\bar{a}) \notin \mathbf{DB}_1,\ p(\bar{a}) \notin \mathbf{DB}_2 \\ \mathbf{f_a} & p(\bar{a}) \in \mathbf{DB}_1,\ p(\bar{a}) \notin \mathbf{DB}_2 \\ \mathbf{t_a} & p(\bar{a}) \notin \mathbf{DB}_1,\ p(\bar{a}) \in \mathbf{DB}_2 \end{cases} \qquad (1)$$

The interpretation I_B is defined as expected: if q is a built-in, then $I_P(q(\bar{a})) = \mathbf{t}$ iff $q(\bar{a})$ is true in classical logic, and $I_P(q(\bar{a})) = \mathbf{f}$ iff $q(\bar{a})$ is false.

Notice that $\mathcal{M}(\mathbf{DB}_1, \mathbf{DB}_2)$ is not symmetric. The intent is to use these structures as the basis for construction of database repairs. In fact, when $\mathbf{DB}_1$ is inconsistent and $\mathbf{DB}_2$ is a repair, I_P shows how the advisory truth values are to be changed to obtain a repair.

Lemma 2. *Given two database instances $\mathbf{DB}$ and $\mathbf{DB}'$, if $\mathbf{DB}' \models_{DB} \mathbf{IC}$, then $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models \mathcal{T}(\mathbf{DB}, \mathbf{IC})$.* □

The implication of this lemma is that whenever $\mathbf{IC}$ is consistent, then the theory $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$ is also consistent in APC. Since in this paper we are always dealing with consistent sets of integrity constraints, we conclude that $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$ is always a consistent APC theory.

We will now show how to generate repairs out of the models of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$. Given a model $\mathcal{M}$ of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$, we define $\mathbf{DB}_\mathcal{M}$ as:

$$\{p(\bar{a}) \mid p \in P \text{ and } \mathcal{M} \models\ p(\bar{a}) : \mathbf{t} \vee p(\bar{a}) : \mathbf{t_a}\}. \qquad (2)$$

Note that $\mathbf{DB}_\mathcal{M}$ can be an infinite set of facts (but finite when $\mathcal{M}$ corresponds to a database instance).

Lemma 3. *If $\mathcal{M}$ is a model of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$ such that $\mathbf{DB}_\mathcal{M}$ is finite, then $\mathbf{DB}_\mathcal{M} \models_{DB} \mathbf{IC}$.*

Proposition 1. *Let $\mathcal{M}$ be a model of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$. If $\mathcal{M}$ is most e-consistent with respect to $\Delta = \{\mathbf{t_a}, \mathbf{f_a}, \top\}$ (see Definition 6) among the models of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$ and $\mathbf{DB}_\mathcal{M}$ is finite, then $\mathbf{DB}_\mathcal{M}$ is a repair of $\mathbf{DB}$ with respect to $\mathbf{IC}$.*

Proposition 2. *If $\mathbf{DB}'$ is a repair of $\mathbf{DB}$ with respect to the set of integrity constraints $\mathbf{IC}$, then $\mathcal{M}(\mathbf{DB}, \mathbf{DB}')$ is most e-consistent with respect to $\Delta = \{\mathbf{t_a}, \mathbf{f_a}, \top\}$ among the models of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$.*

Example 4. (Repairs as most e-consistent models) Consider a database instance $\mathbf{DB} = \{p(a)\}$ over the domain $D = \{a\}$ and a set of integrity constraints $\mathbf{IC} = \{\neg p(x) \ \vee\ q(x),\ \ \neg q(x) \ \vee\ r(x)\}$. In this case $\mathcal{T}(\mathbf{DB}) = \{p(a) : \mathbf{t_d},\ \ q(a) : \mathbf{f_d},\ \ r(a) : \mathbf{f_d}\}$, and $\mathcal{T}(\mathbf{IC})$ is

$$\begin{array}{ll} p(x) : \mathbf{f_c} \vee q(x) : \mathbf{t_c}, & q(x) : \mathbf{f_c} \vee r(x) : \mathbf{t_c}, \\ p(x) : \mathbf{t_c} \vee p(x) : \mathbf{f_c}, & \neg p(x) : \mathbf{t_c} \vee \neg p(x) : \mathbf{f_c}, \\ q(x) : \mathbf{t_c} \vee q(x) : \mathbf{f_c}, & \neg q(x) : \mathbf{t_c} \vee \neg q(x) : \mathbf{f_c}, \\ r(x) : \mathbf{t_c} \vee r(x) : \mathbf{f_c}, & \neg r(x) : \mathbf{t_c} \vee \neg r(x) : \mathbf{f_c} \end{array}$$

This theory has four models, depicted in the following table:

	$p(a)$	$q(a)$	$r(a)$
$\mathcal{M}_1$	$\mathbf{t}$	$\mathbf{t_a}$	$\mathbf{t_a}$
$\mathcal{M}_2$	$\mathbf{f_a}$	$\mathbf{f}$	$\mathbf{f}$
$\mathcal{M}_3$	$\mathbf{f_a}$	$\mathbf{f}$	$\mathbf{t_a}$
$\mathcal{M}_4$	$\mathbf{f_a}$	$\mathbf{t_a}$	$\mathbf{t_a}$

It is easy to verify that $\mathcal{M}_1$ and $\mathcal{M}_2$ are the most e-consistent models with respect to $\Delta = \{\mathbf{t_a}, \mathbf{f_a}, \top\}$ among the models in the table and the database instance $\mathbf{DB}_{\mathcal{M}_1} = \{p(a), q(a), r(a)\}$ and $\mathbf{DB}_{\mathcal{M}_2} = \emptyset$ are exactly the repairs of $\mathbf{DB}$ with respect to $\mathbf{IC}$.

Example 5. (Example 3 continued) The embedding of the database described in Example 3 has nine models listed in the following table. The table omits the built-in "=", since it has the same interpretation in all models.

	$p(a,a)$	$p(a,b)$	$p(b,a)$	$p(b,b)$
$\mathcal{M}_1$	$\mathbf{t}$	$\mathbf{f_a}$	$\mathbf{t}$	$\mathbf{f}$
$\mathcal{M}_2$	$\mathbf{t}$	$\mathbf{f_a}$	$\mathbf{f_a}$	$\mathbf{f}$
$\mathcal{M}_3$	$\mathbf{t}$	$\mathbf{f_a}$	$\mathbf{f_a}$	$\mathbf{t_a}$
$\mathcal{M}_4$	$\mathbf{f_a}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{f}$
$\mathcal{M}_5$	$\mathbf{f_a}$	$\mathbf{t}$	$\mathbf{f_a}$	$\mathbf{f}$
$\mathcal{M}_6$	$\mathbf{f_a}$	$\mathbf{t}$	$\mathbf{f_a}$	$\mathbf{t_a}$
$\mathcal{M}_7$	$\mathbf{f_a}$	$\mathbf{f_a}$	$\mathbf{t}$	$\mathbf{f}$
$\mathcal{M}_8$	$\mathbf{f_a}$	$\mathbf{f_a}$	$\mathbf{f_a}$	$\mathbf{f}$
$\mathcal{M}_9$	$\mathbf{f_a}$	$\mathbf{f_a}$	$\mathbf{f_a}$	$\mathbf{t_a}$

It is easy to see that $\mathcal{M}_1$ and $\mathcal{M}_4$ are the most e-consistent models with respect to $\Delta = \{\mathbf{t_a}, \mathbf{f_a}, \top\}$ among the models in the table, and the database instances $\mathbf{DB}_{\mathcal{M}_1} = \{p(a,a),\ \ p(b,a)\}$, and $\mathbf{DB}_{\mathcal{M}_4} = \{p(a,b),\ \ p(b,a)\}$ are exactly the repairs of $\mathbf{DB}$ with respect to $\mathbf{IC}$.

5 Repairing Inconsistent Databases

To construct all possible repairs of a database, $\mathbf{DB}$, that is inconsistent with the integrity constraints $\mathbf{IC}$, we need to find the set of all ground clauses of the form

$$\mathbf{p}_1 : ?_\mathbf{a} \vee \cdots \vee \mathbf{p}_n : ?_\mathbf{a}, \qquad (3)$$

that are implied by $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$, where each $?_\mathbf{a}$ is either $\mathbf{t_a}$ or $\mathbf{f_a}$. Such clauses are called *a-clauses*, for advisory clauses.[4]

A-clauses are important because one of the disjuncts of such a clause must be true in each model of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$. Suppose that, say, $\mathbf{p} : ?_\mathbf{a}$ is true in some model I. This means that the truth value of $\mathbf{p}$ with respect to the database is exactly the opposite of what is required in order for I to satisfy the constraints. This observation can be used to construct a repair of the database by reversing the truth value of $\mathbf{p}$ with respect to the database. We explore this idea next.

Constructing database repairs. Let $\mathcal{T}^\mathbf{a}(\mathbf{DB}, \mathbf{IC})$ be the set of all *minimal a-clauses* that are implied by $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$. "Minimal" here means that no disjunct can be removed from any clause in $\mathcal{T}^\mathbf{a}(\mathbf{DB}, \mathbf{IC})$ and still have the clause implied by $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$.

In general, this can be an infinite set, but in most practical cases this set is finite. Conditions for finiteness of $\mathcal{T}^\mathbf{a}(\mathbf{DB}, \mathbf{IC})$ are given in Section 5.1. If $\mathcal{T}^\mathbf{a}(\mathbf{DB}, \mathbf{IC})$ is finite, it can be represented as the following set of clauses:

$$\begin{array}{rcl} C_1 & = & \mathbf{p}_{1,1} : \mathbf{a}_{1,1} \vee \cdots \vee \mathbf{p}_{1,n_1} : \mathbf{a}_{1,n_1} \\ \ddots & & \ddots \qquad\qquad \ddots \\ C_k & = & \mathbf{p}_{k,1} : \mathbf{a}_{k,1} \vee \cdots \vee \mathbf{p}_{k,n_k} : \mathbf{a}_{k,n_k} \end{array}$$

Here, the $\mathbf{p}_{i,j} : \mathbf{a}_{i,j}$ are ground positive literals and their annotations, $\mathbf{a}_{i,j}$, are always of the form $\mathbf{t_a}$ or $\mathbf{f_a}$.

It can be shown that all *a-clauses* can be generated using the APC resolution inference rule [9] between $\mathcal{T}(\mathbf{IC})$, $\mathcal{T}(\mathbf{DB})$, and $\mathcal{T}(\mathcal{B})$. It can be also shown that all a-clauses generated in this way are ground and do not contain built-in predicates.

Given $\mathcal{T}^\mathbf{a}(\mathbf{DB}, \mathbf{IC})$ as above, a *repair signature* is a set of APC literals that contains at least one literal from each clause C_i and is minimal in the sense that no proper subset has a literal from each C_i. In other words, a repair signature is a minimal hitting set of the family of clauses $C_1, \ldots, C_k$ [6].

Notice that if the clauses C_i do not share literals, then each repair signature contains exactly k literals and every literal appearing in a clause C_i belongs to some repair signature.

It follows from the construction of repairs in (2) and from Propositions 1 and 2 that there is a one to one correspondence between repair signatures and repairs of the original database instance $\mathbf{DB}$. Given a repair signature `Repair`, a repair $\mathbf{DB}'$ can be obtained from $\mathbf{DB}$ by removing the tuples $\mathbf{p}(\bar{t})$, if $\mathbf{p}(\bar{t}) : \mathbf{f_a} \in$ `Repair`, and inserting the tuples $\mathbf{p}(\bar{t})$, if $\mathbf{p}(\bar{t}) : \mathbf{t_a} \in$ `Repair`. It can be shown that it is not possible for any fact, $\mathbf{p}$, to occur in $\mathcal{T}^\mathbf{a}(\mathbf{DB}, \mathbf{IC})$ with two different annotations. Therefore, it is not possible that the same fact will be inserted and then removed (or vice versa) while constructing a repair as described here.

5.1 Finiteness of $\mathcal{T}^\mathbf{a}(\mathbf{DB}, \mathbf{IC})$

We now examine the issue of finiteness of the set $\mathcal{T}^\mathbf{a}(\mathbf{DB}, \mathbf{IC})$.

Definition 7. *(Range-restricted Constraints) An integrity constraint, $p_1(\bar{T}_1) \vee \cdots \vee p_n(\bar{T}_n) \vee \neg q_1(\bar{T}'_1) \vee \cdots \vee \neg q_m(\bar{T}'_m)$, is* range-restricted *if and only if every variable in $\bar{T}_i$ $(1 \leq i \leq n)$ also occurs in some $\bar{T}'_j$ $(1 \leq j \leq m)$. Both p_i and q_j can be built-in predicates.*

A set $\mathbf{IC}$ of constraints is range-restricted *if so is every constraint in $\mathbf{IC}$.*

Lemma 4. *Let $\mathbf{IC}$ be a set of range-restricted constraints over a database $\mathbf{DB}$. Then every a-clause implied by $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$ (i.e., every clause of the form (3)) mentions only the constants in the active domain of $\mathbf{DB}$.*[5]

[3] Note that an APC theory *can* entail $p(\bar{a}) : \top$ and be consistent in the sense that it can have a model. However, such a model must contain $p(\bar{a}) : \top$, which makes it epistemically inconsistent.

[4] Here, bold face symbols, *e.g.*, $\mathbf{p}$, denote classical ground atomic formulas.

[5] The active domain consists of the constants in D that appear in some database table.

Corollary 1. *If* $\mathbf{IC}$ *is range-restricted, then* $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$ *is finite.*

6 Queries to Inconsistent Databases

In general, the number of all repair signatures can be exponential in the size of $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$, so using this theory directly is not likely to produce a good query engine. In fact, for the propositional case, [5] shows that the problem of deciding whether a formula holds in all models produced by Winslett's theory of updates [4] is Π_2^P*-complete.* Since, as mentioned before, our repairs are essentially Winslett's updated models, the same result applies to our case.

However, there are cases when complexity is manageable. It is easy to see that if k is the number of clauses in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$ and n_1, ..., n_k are the numbers of disjuncts in C_1, ..., C_k, respectively, then the number of repair signatures is $O(n_1 \times \ldots \times n_k)$. Therefore, two factors affect the number of repairs:

1. The number of clauses in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$;
2. The number of disjuncts in each clause in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$.

So, we should look into those types of constraints where either k is bound or all but a bound number of n_i's equal 1.

Other cases when query answering is feasible arise when the set of *a-clauses* $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$ is precomputed. Precomputing this set might be practical for read-only databases. In other cases, $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$ might be easy to compute because of the special form of constraints (and in this case, the size of $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$ turns out to be P-bounded). For instance, suppose $\mathbf{IC}$ consists of range-restricted formulas and is closed under the resolution inference rule (e.g. if $\mathbf{IC}$ is a set of functional dependencies). In this case, a-clauses can be generated by converting each constraint into a query that finds all tuples that violate the constraint. For instance, the constraint $p(\bar{x}) \supset q(\bar{x})$ can be converted into the query $p(\bar{x}) \wedge \neg q(\bar{x})$ (which is the denial form of this constraint). If the tuple $\bar{a}$ is an answer, then one *a-clause* is $p(\bar{a}) : \mathbf{f_a} \vee q(\bar{a}) : \mathbf{t_a}$.

Answering ground conjunctive queries. To consistently answer a ground conjunctive query of the form $\mathbf{p}_1 \wedge \ldots \wedge \mathbf{p}_k \wedge \neg\mathbf{q}_1 \wedge \ldots \wedge \neg\mathbf{q}_m$, we need to check the following:

For each $\mathbf{p}_i$: if $\mathbf{p}_i \in \mathbf{DB}$ and $\mathbf{p}_i : \mathbf{f_a}$ is not mentioned in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$; or if $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$ has a clause of the form $\mathbf{p}_i : \mathbf{t_a}$.
For each $\mathbf{q}_j$: if $\mathbf{q}_j \notin \mathbf{DB}$ and $\mathbf{q}_j : \mathbf{t_a}$ is not mentioned in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$; or if $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$ has a clause of the form $\mathbf{q}_j : \mathbf{f_a}$.

If all of the above holds, *true* is a consistent answer to the query. Otherwise, the answer is *not true*, meaning that there is at least one repair where our conjunctive query is false. (Note that this is not the same as answering *false* in definition 5).

Non-ground conjunctive queries. Let $\mathbf{DB}$ have the relations $p_1, \ldots, p_n$. We construct a new database, $\mathbf{DB}^{O,U}$, with relations $p_1^O, \ldots, p_n^O, p_1^U, \ldots, p_n^U$ (where O and U stand for "original" and "unknown", resp.), as follows:

p_i^O *consists of*: all the tuples such that $p_i(\bar{t}) \in \mathbf{DB}$ and $p_i(\bar{t}) : \mathbf{f_a}$ is not mentioned in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$ plus the tuples $\bar{t}$ such that $p(\bar{t}) : \mathbf{t_a}$ is a clause in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$.
p_j^U *consists of*: all the tuples $\bar{t}$ such that $p_j(\bar{t}) : \mathbf{t_a}$ or $p_j(\bar{t}) : \mathbf{f_a}$ appear in a clause in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$ that has more than one disjunct.

To answer an open conjunctive query, for example, $p(x) \wedge \neg q(x)$, we pose the query $p^O(x) \wedge \neg q^O(x) \wedge \neg q^U(x)$ to $\mathbf{DB}^{O,U}$. This can be done in polynomial time in the database size plus the size of the set of *a-clauses.*

Ground disjunctive queries. Sound and complete query evaluation techniques for various types of queries and constraints are developed in [1]. Our present framework extends the results in [1] to include disjunctive queries. We concentrate on ground disjunctive queries of the form

$$\mathbf{p}_1 \vee \cdots \vee \mathbf{p}_k \vee \neg\mathbf{q}_1 \cdots \neg\mathbf{q}_r. \tag{4}$$

First, for each p_i we evaluate the query $\mathbf{p}_i^O$ and for each q_j we evaluate the query $\neg\mathbf{q}_j^O \wedge \neg\mathbf{q}_j^U$ against the database $\mathbf{DB}^{O,U}$. If at least one *true* answer is obtained, the answer to (4) is *true.* Otherwise, if all these queries return *false*, we evaluate the queries of the form $\neg\mathbf{p}_i^O \wedge \neg\mathbf{p}_i^U$ and $\mathbf{q}_j^O$ against $\mathbf{DB}^{O,U}$. For each answer *true*, the corresponding literal is eliminated from (4). Let $\mathbf{p}_{i_1} \vee \cdots \vee \mathbf{p}_{i_s} \vee \neg\mathbf{q}_{j_1} \cdots \neg\mathbf{q}_{j_t}$ be the resulting query. If this query is empty, then the answer to the original query is *false*, *i.e.*, the original query is false in every repair. If the resulting query is not empty, we must check if there is a minimal hitting set for $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$, that contains $\{\neg\mathbf{p}_{i_1}, \ldots, \neg\mathbf{p}_{i_s}, \mathbf{q}_{j_1}, \ldots, \mathbf{q}_{j_t}\}$. If such a hitting set exists, the answer to the original query is *maybe*, meaning that there is at least one repair where the answer is *false*. Otherwise, the answer to the query is *true.*

Therefore, the problem of answering disjunctive queries for a given $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$ is equivalent to the problem of deciding whether a given set can be extended to a minimal hitting set of the family. Since this is an *NP-complete* problem, we have the following result.

Proposition 3. *Suppose that* $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$ *has been precomputed. Then the problem of deciding whether true is a consistent answer to a disjunctive ground query is NP-complete with respect to the size of* $\mathbf{DB}$ *plus* $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$.

7 Conclusions

We presented a new semantic framework, based on Annotated Predicate Calculus [9], for studying the problem of query answering in databases that are inconsistent with integrity constraints. This was done by embedding both the database instance and the integrity constraints into a single theory written in an APC with an appropriate truth values lattice. In this way, we obtain a general logical specification of database repairs and consistent query answers.

With this new framework, we are able to provide a better analysis of the computational complexity of query answering in such environments and to develop a more general query answering mechanism than what was known previously [1]. We also identified certain classes of queries and constraints that have lower complexity, and we are looking into better query evaluation algorithms for these classes.

The development of the specific mechanisms for consistent query answering in the presence of universal ICs, and the extension of our methodology to constraints that contain existential quantifiers (*e.g.*, referential integrity constraints) is left for future work.

Acknowledgements. We would like to thank the anonymous referees for their valuable comments. Work supported by Fondecyt Grants # 1980945, # 1000593; and ECOS/CONICYT Grant C97E05.

References

1. M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *Proc. ACM Symposium on Principles of Database Systems (ACM PODS'99, Philadelphia)*, pages 68–79, 1999.
2. N. Belnap. A Useful Four-Valued Logic. In M. Dunn and G. Epstein, editors, *Modern Uses of Multi-Valued Logic*, pages 8–37. Reidel Publ. Co., 1977.
3. H.A. Blair and V.S. Subrahmanian. Paraconsistent Logic Programming. *Theoretical Computer Science*, 68:135–154, 1989.
4. T. Chou and M. Winslett. A Model-Based Belief Revision System. *J. Automated Reasoning*, 12:157–208, 1994.
5. T. Eiter and G. Gottlob. On the Complexity of Propositional Knowledge Base Revision, Updates, and Counterfactuals. *J. Artificial Intelligence*, 57(2-3):227–270, 1992.
6. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Co., 1979.
7. M. Gertz. *Diagnosis and Repair of Constraint Violations in Database Systems.* PhD thesis, Universität Hannover, 1996.
8. M.L. Ginsberg. Multivalued Logics: A Uniform Approach to Reasoning in Artificial Intelligence. *Computational Intelligence*, 4:265–316, 1988.
9. M. Kifer and E.L. Lozinskii. A Logic for Reasoning with Inconsistency. *Journal of Automated Reasoning*, 9(2):179–215, November 1992.
10. M. Kifer and V.S. Subrahmanian. Theory of Generalized Annotated Logic Programming and its Applications. *Journal of Logic Programming*, 12(4):335–368, April 1992.
11. V.S. Subrahmanian. On the Semantics of Quantitative Logic Programs. In *IEEE Symposium on Logic Programming*, pages 173–182, 1987.
12. M.H. van Emden. Quantitative Deduction and its Fixpoint Theory. *Journal of Logic Programming*, 1(4):37–53, 1986.

A Proofs and Intermediate Results

Proof of lemma 1:

1. If $\mathcal{M} \models p(\bar{a}) : \top$, then $\mathcal{M} \models p(\bar{a}) : \mathbf{t_c}$ and $\mathcal{M} \models p(\bar{a}) : \mathbf{f_c}$. Thus, $\mathcal{M} \not\models \mathcal{T}(\mathbf{IC})$, a contradiction.
2. We know that $\mathcal{M} \models p(\bar{a}) : \mathbf{t_c} \vee p(\bar{a}) : \mathbf{f_c}$ and $\mathcal{M} \models p(\bar{a}) : \mathbf{t_d} \vee p(\bar{a}) : \mathbf{f_d}$ (since $p(\bar{a}) : \mathbf{t_d} \in \mathcal{T}(\mathbf{DB}, \mathbf{IC})$ or $p(\bar{a}) : \mathbf{f_d} \in \mathcal{T}(\mathbf{DB}, \mathbf{IC})$). Thus, one of the following cases must be true: (1) $\mathcal{M} \models p(\bar{a}) : \mathbf{t_c}$ and $\mathcal{M} \models p(\bar{a}) : \mathbf{t_d}$, and therefore $\mathcal{M} \models p(\bar{a}) : \mathbf{t}$, (2) $\mathcal{M} \models p(\bar{a}) : \mathbf{t_c}$ and $\mathcal{M} \models p(\bar{a}) : \mathbf{f_d}$, and therefore $\mathcal{M} \models p(\bar{a}) : \mathbf{t_a}$, (3) $\mathcal{M} \models p(\bar{a}) : \mathbf{f_c}$ and $\mathcal{M} \models p(\bar{a}) : \mathbf{t_d}$, and therefore $\mathcal{M} \models p(\bar{a}) : \mathbf{f_a}$, (4) $\mathcal{M} \models p(\bar{a}) : \mathbf{f_c}$ and $\mathcal{M} \models p(\bar{a}) : \mathbf{f_d}$, and therefore $\mathcal{M} \models p(\bar{a}) : \mathbf{f}$.

□

Proof of lemma 2: We have to prove that $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models \mathcal{T}(\mathbf{DB})$ and $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models \mathcal{T}(\mathbf{IC})$.

1. Let us consider $p(\bar{a}) : \mathbf{a} \in \mathcal{T}(\mathbf{DB})$. If $\mathbf{a} = \mathbf{t_d}$, then $p(\bar{a}) \in \mathbf{DB}$, and then by considering (1) we obtain that $I_P(p(\bar{a})) = \mathbf{t}$ or $I_P(p(\bar{a})) = \mathbf{f_a}$, and therefore $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models p(\bar{a}) : \mathbf{a}$. If $\mathbf{a} = \mathbf{f_d}$, then $p(\bar{a}) \notin \mathbf{DB}$, and then by considering (1) we obtain that $I_P(p(\bar{a})) = \mathbf{f}$ or $I_P(p(\bar{a})) = \mathbf{t_a}$, and therefore $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models p(\bar{a}) : \mathbf{a}$.
2. (a) Let us suppose that $p_1(\bar{T}_1) : \mathbf{t_c} \vee \cdots \vee p_n(\bar{T}_n) : \mathbf{t_c} \vee q_1(\bar{T}_1) : \mathbf{f_c} \vee \cdots \vee q_m(\bar{T}_m) : \mathbf{f_c} \in \mathcal{T}(\mathbf{IC})$, and let us assume that $p_1(\bar{a}_1) : \mathbf{t_c} \vee \cdots \vee p_n(\bar{a}_n) : \mathbf{t_c} \vee q_1(\bar{b}_1) : \mathbf{f_c} \vee \cdots \vee q_m(\bar{a}_m) : \mathbf{f_c}$ was obtained from this constraint by instantiating in the domain of the database. In this case we have that $p_1(\bar{T}_1) \vee \cdots \vee p_n(\bar{T}_n) \vee \neg q_1(\bar{T}_1) \vee \cdots \vee \neg q_m(\bar{T}_m)$ is an element of $\mathbf{IC}$, and therefore we have that $\mathbf{DB}' \models_{\mathrm{DB}} p_1(\bar{a}_1) \vee \cdots \vee p_n(\bar{a}_n) \vee \neg q_1(\bar{b}_1) \vee \cdots \vee \neg q_m(\bar{b}_m)$.
   Firstly, we are going to consider what happens if $\mathbf{DB}' \models_{\mathrm{DB}} p_i(\bar{a}_i)$ $(1 \leq i \leq n)$. If p_i is a built-in predicate, then $I_R(p_i(\bar{a}_i)) = \mathbf{t}$, since $\mathcal{M}(\mathbf{DB}, \mathbf{DB}')$ gives to the built-in predicates in the database the appropriate truth values, and therefore $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models p_i(\bar{a}_i) : \mathbf{t_c}$. If p_i is not a built-in predicate, then $I_P(p_i(\bar{a}_i)) = \mathbf{t}$ or $I_P(p_i(\bar{a}_i)) = \mathbf{t_a}$, since $p_i(\bar{a}_i) \in \mathbf{DB}'$, and therefore $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models p_i(\bar{a}_i) : \mathbf{t_c}$.
   Secondly, we are going to consider what happens if $\mathbf{DB}' \models_{\mathrm{DB}} \neg q_i(\bar{b}_i)$ $(1 \leq i \leq m)$. If q_i is a built-in predicate, then $I_R(q_i(\bar{b}_i)) = \mathbf{f}$, since $\mathcal{M}(\mathbf{DB}, \mathbf{DB}')$ gives to the built-in predicates in the database the appropriate truth values, and therefore $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models q_i(\bar{b}_i) : \mathbf{f_c}$. If q_i is not a built-in predicate, then $I_P(q_i(\bar{b}_i)) = \mathbf{f}$ or $I_P(q_i(\bar{b}_i)) = \mathbf{f_a}$, since $q_i(\bar{b}_i) \notin \mathbf{DB}'$, and therefore $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models q_i(\bar{b}_i) : \mathbf{f_c}$.
   (b) Let us consider a predicate p in P. By considering (1) we know that for every tuple $\bar{a}$ (of appropriate arity) $I_P(p(\bar{a})) = \mathbf{t}$, $I_P(p(\bar{a})) = \mathbf{f}$, $I_P(p(\bar{a})) = \mathbf{t_a}$ or $I_P(p(\bar{a})) = \mathbf{f_a}$, and therefore $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models p(\bar{a}) : \mathbf{t_c} \vee p(\bar{a}) : \mathbf{f_c}$. Thus, we conclude that $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models \forall \bar{x}(p(\bar{x}) : \mathbf{t_c} \vee p(\bar{x}) : \mathbf{f_c})$. Additionally, if $I_P(p(\bar{a})) = \mathbf{t}$ or $I_P(p(\bar{a})) = \mathbf{t_a}$, then $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \not\models p(\bar{a}) : \mathbf{f_c}$, and if $I_P(p(\bar{a})) = \mathbf{f}$ or $I_P(p(\bar{a})) = \mathbf{f_a}$, then $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \not\models p(\bar{a}) : \mathbf{t_c}$. Thus, we also conclude that $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models \forall \bar{x}(\neg p(\bar{x}) : \mathbf{t_c} \vee \neg p(\bar{x}) : \mathbf{f_c})$.

□

Proof of lemma 3: We are going to prove that $\mathbf{DB}_{\mathcal{M}} \models_{\mathrm{DB}} \mathbf{IC}$. Let us suppose that $p_1(\bar{T}_1) \vee \cdots \vee p_n(\bar{T}_n) \vee \neg q_1(\bar{T}_1) \vee \cdots \vee \neg q_m(\bar{T}_m)$ is an integrity constraint in $\mathbf{IC}$, and let us assume that $p_1(\bar{a}_1) \vee \cdots \vee p_n(\bar{a}_n) \vee \neg q_1(\bar{b}_1) \vee \cdots \vee \neg q_m(\bar{b}_m)$ was obtained from it by instantiated in the domain of the database. In this case we have that $p_1(\bar{a}_1) : \mathbf{t_c} \vee \cdots \vee p_n(\bar{a}_n) : \mathbf{t_c} \vee q_1(\bar{b}_1) : \mathbf{f_c} \vee \cdots \vee q_m(\bar{b}_m) : \mathbf{f_c}$ could be obtained by instantiated an integrity constraint in $\mathcal{T}(\mathbf{IC})$. Thus, we have that $\mathcal{M} \models p_1(\bar{a}_1) : \mathbf{t_c} \vee \cdots \vee p_n(\bar{a}_n) : \mathbf{t_c} \vee q_1(\bar{b}_1) : \mathbf{f_c} \vee \cdots \vee q_m(\bar{b}_m) : \mathbf{f_c}$.

Firstly, we are going to consider what happens if $\mathcal{M} \models p_i(\bar{a}_i) : \mathbf{t_c}$ $(1 \leq i \leq n)$. If p_i is a built-in predicate, then $I_R(p_i(\bar{a}_i)) = \mathbf{t}$, since $\mathcal{M}$ gives to the built-in predicates in the database the value $\mathbf{t}$ or $\mathbf{f}$, and if in this case we suppose that $I_R(p_i(\bar{a}_i)) = \mathbf{f}$ then $\mathcal{M} \not\models p_i(\bar{a}_i) : \mathbf{t_c}$, a contradiction. Therefore $\mathbf{DB}_{\mathcal{M}} \models_{\mathrm{DB}} p_i(\bar{a}_i)$. If p_i is not a built-in predicate, then $p_i(\bar{a}_i) : \mathbf{t_d} \in \mathcal{T}(\mathbf{DB})$ or $p_i(\bar{a}_i) : \mathbf{f_d} \in \mathcal{T}(\mathbf{DB})$. In the first case we have that $\mathcal{M} \models p_i(\bar{a}_i) : \mathbf{t}$, and therefore $p_i(\bar{a}_i) \in \mathbf{DB}_{\mathcal{M}}$. In the second case $\mathcal{M} \models p_i(\bar{a}_i) : \mathbf{t_a}$, and therefore $p_i(\bar{a}_i) \in \mathbf{DB}_{\mathcal{M}}$.

Secondly, we are going to consider what happens if $\mathcal{M} \models q_i(\bar{b}_i) : \mathbf{f_c}$ $(1 \leq i \leq m)$. If q_i is a built-in predicate, then $I_R(q_i(\bar{b}_i)) = \mathbf{f}$, since $\mathcal{M}$ gives to the built-in predicates in the database the value $\mathbf{t}$ or $\mathbf{f}$, and if in this case we suppose that $I_R(q_i(\bar{b}_i)) = \mathbf{t}$ then $\mathcal{M} \not\models q_i(\bar{b}_i) : \mathbf{f_c}$, a contradiction. Therefore $\mathbf{DB}_{\mathcal{M}} \models_{\mathrm{DB}} \neg q_i(\bar{b}_i)$. If q_i is not a built-in predicate, then $q_i(\bar{b}_i) : \mathbf{t_d} \in \mathcal{T}(\mathbf{DB})$ or $q_i(\bar{b}_i) : \mathbf{f_d} \in \mathcal{T}(\mathbf{DB})$. In the first case we have that $\mathcal{M} \models q_i(\bar{b}_i) : \mathbf{f_a}$, and therefore $q_i(\bar{b}_i) \notin \mathbf{DB}_{\mathcal{M}}$. In the second case $\mathcal{M} \models q_i(\bar{b}_i) : \mathbf{f}$, and therefore $q_i(\bar{b}_i) \notin \mathbf{DB}_{\mathcal{M}}$. □

Proof of proposition 1:

1. By Lemma 3, we conclude that $\mathbf{DB}_{\mathcal{M}} \models_{\mathrm{DB}} \mathbf{IC}$.
2. Now, we need to prove that $\mathbf{DB}_{\mathcal{M}}$ is minimal. Let us suppose this is not true. Then, there is a database instance $\mathbf{DB}^*$ such that $\mathbf{DB}^* \models_{\mathrm{DB}} \mathbf{IC}$ and $\Delta(\mathbf{DB}, \mathbf{DB}^*) \subsetneq \Delta(\mathbf{DB}, \mathbf{DB}_{\mathcal{M}})$.
   (a) From Lemma 2, we conclude that $\mathcal{M}(\mathbf{DB}, \mathbf{DB}^*) \models \mathcal{T}(\mathbf{DB}, \mathbf{IC})$.
   (b) Now, we are going to prove that $\mathcal{M}(\mathbf{DB}, \mathbf{DB}^*) <_{\Delta} \mathcal{M}$.
   If $\mathcal{M}(\mathbf{DB}, \mathbf{DB}^*) \models p(\bar{a}) : \mathbf{t_a}$, then by considering (1) we can conclude that $p(\bar{a}) \notin \mathbf{DB}$ and $p(\bar{a}) \in \mathbf{DB}^*$, and therefore $p(\bar{a}) \in \Delta(\mathbf{DB}, \mathbf{DB}^*)$. But $\Delta(\mathbf{DB}, \mathbf{DB}^*) \subsetneq \Delta(\mathbf{DB}, \mathbf{DB}_{\mathcal{M}})$, and therefore $p(\bar{a}) \in \mathbf{DB}_{\mathcal{M}}$. Thus, we can conclude that $\mathcal{M} \models p(\bar{a}) : \mathbf{t} \vee p(\bar{a}) : \mathbf{t_a}$. If we suppose that $\mathcal{M} \models p(\bar{a}) : \mathbf{t}$, then $\mathcal{M} \not\models p(\bar{a}) : \mathbf{f_d}$, but we know that $\mathcal{M} \models \mathcal{T}(\mathbf{DB}, \mathbf{IC})$ and $p(\bar{a}) : \mathbf{f_d} \in \mathcal{T}(\mathbf{DB}, \mathbf{IC})$, since $p(\bar{a}) \notin \mathbf{DB}$, a contradiction. Therefore, $\mathcal{M} \models p(\bar{a}) : \mathbf{t_a}$.
   If $\mathcal{M}(\mathbf{DB}, \mathbf{DB}^*) \models p(\bar{a}) : \mathbf{f_a}$, then by considering (1) we can conclude that $p(\bar{a}) \in \mathbf{DB}$ and $p(\bar{a}) \notin \mathbf{DB}^*$, and therefore $p(\bar{a}) \in \Delta(\mathbf{DB}, \mathbf{DB}^*)$. But $\Delta(\mathbf{DB}, \mathbf{DB}^*) \subsetneq \Delta(\mathbf{DB}, \mathbf{DB}_{\mathcal{M}})$, and therefore $p(\bar{a}) \notin \mathbf{DB}_{\mathcal{M}}$. Thus, we can conclude that $\mathcal{M} \models p(\bar{a}) : \mathbf{f} \vee p(\bar{a}) : \mathbf{f_a}$. If we suppose that $\mathcal{M} \models p(\bar{a}) : \mathbf{f}$, then $\mathcal{M} \not\models p(\bar{a}) : \mathbf{t_d}$, but we know that $\mathcal{M} \models \mathcal{T}(\mathbf{DB}, \mathbf{IC})$ and $p(\bar{a}) : \mathbf{t_d} \in \mathcal{T}(\mathbf{DB}, \mathbf{IC})$, since $p(\bar{a}) \in \mathbf{DB}$, a contradiction. Therefore, $\mathcal{M} \models p(\bar{a}) : \mathbf{f_a}$. Thus, we can deduce that $\mathcal{M}(\mathbf{DB}, \mathbf{DB}^*) \leq_{\Delta} \mathcal{M}$.
   Finally, we know that there exists $p(\bar{a})$ such that it is not in $\Delta(\mathbf{DB}, \mathbf{DB}^*)$ and it is in $\Delta(\mathbf{DB}, \mathbf{DB}_{\mathcal{M}})$. Thus, $p(\bar{a}) \in \mathbf{DB}$ and $p(\bar{a}) \in \mathbf{DB}^*$, and therefore $\mathcal{M}(\mathbf{DB}, \mathbf{DB}^*) \models p(\bar{a}) : \mathbf{t}$, or $p(\bar{a}) \notin \mathbf{DB}$ and $p(\bar{a}) \notin \mathbf{DB}^*$, and therefore $\mathcal{M}(\mathbf{DB}, \mathbf{DB}^*) \models p(\bar{a}) : \mathbf{f}$. Then, we have that $\mathcal{M}(\mathbf{DB}, \mathbf{DB}^*) \not\models p(\bar{a}) : \mathbf{t_a}$ and $\mathcal{M}(\mathbf{DB}, \mathbf{DB}^*) \not\models p(\bar{a}) : \mathbf{f_a}$. Additionally, since $p(\bar{a}) \in \Delta(\mathbf{DB}, \mathbf{DB}_{\mathcal{M}})$, we can conclude that $p(\bar{a}) \in \mathbf{DB}$ and $p(\bar{a}) \notin \mathbf{DB}_{\mathcal{M}}$, or $p(\bar{a}) \notin \mathbf{DB}$ and $p(\bar{a}) \in \mathbf{DB}_{\mathcal{M}}$. In the first case we can conclude that $\mathcal{M} \models p(\bar{a}) : \mathbf{f_a}$, since $\mathcal{M}$ must be satisfied $p(\bar{a}) : \mathbf{f} \vee p(\bar{a}) : \mathbf{f_a}$, and if we suppose that $\mathcal{M} \models p(\bar{a}) : \mathbf{f}$, then $\mathcal{M} \not\models p(\bar{a}) : \mathbf{t_d}$, but $p(\bar{a}) : \mathbf{t_d} \in \mathcal{T}(\mathbf{DB}, \mathbf{IC})$ in this case, a contradiction. In the second case we can conclude that $\mathcal{M} \models p(\bar{a}) : \mathbf{t_a}$, since $\mathcal{M}$ must be satisfied $p(\bar{a}) : \mathbf{t} \vee p(\bar{a}) : \mathbf{t_a}$, and if we suppose that $\mathcal{M} \models p(\bar{a}) : \mathbf{t}$, then $\mathcal{M} \not\models p(\bar{a}) : \mathbf{f_d}$, but $p(\bar{a}) : \mathbf{f_d} \in \mathcal{T}(\mathbf{DB}, \mathbf{IC})$ in this case, a contradiction. Thus, we can conclude that $\mathcal{M} \models p(\bar{a}) : \mathbf{t_a} \vee p(\bar{a}) : \mathbf{f_a}$. Therefore we can deduce that $\mathcal{M} \not\leq_{\Delta} \mathcal{M}(\mathbf{DB}, \mathbf{DB}^*)$.

   Finally, we deduce that $\mathcal{M}$ is not e-consistent maximal in the class of the models of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$, with respect to Δ, a contradiction.

□

Proof of proposition 2:

1. By Lemma 2, we conclude that $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models \mathcal{T}(\mathbf{DB}, \mathbf{IC})$.
2. Let us suppose that $\mathcal{M}(\mathbf{DB}, \mathbf{DB}')$ is not e-consistent maximal in the class of models of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$ with respect to Δ. Then, there exists $\mathcal{M} \models \mathcal{T}(\mathbf{DB}, \mathbf{IC})$, such that $\mathcal{M} <_{\Delta} \mathcal{M}(\mathbf{DB}, \mathbf{DB}')$. By using this it is possible to prove that $\Delta(\mathbf{DB}, \mathbf{DB}_{\mathcal{M}}) \subsetneq \Delta(\mathbf{DB}, \mathbf{DB}')$.
   (a) Let us suppose that $p(\bar{a}) \in \Delta(\mathbf{DB}, \mathbf{DB}_{\mathcal{M}})$. Then $p(\bar{a}) \in \mathbf{DB}$ and $p(\bar{a}) \notin \mathbf{DB}_{\mathcal{M}}$, or $p(\bar{a}) \notin \mathbf{DB}$ and $p(\bar{a}) \in \mathbf{DB}_{\mathcal{M}}$. In the first case we can conclude that $p(\bar{a}) : \mathbf{t_d} \in \mathcal{T}(\mathbf{DB}, \mathbf{IC})$ and $\mathcal{M} \models p(\bar{a}) : \mathbf{f} \vee p(\bar{a}) : \mathbf{f_a}$. If we suppose that $\mathcal{M} \models p(\bar{a}) : \mathbf{f}$, then $\mathcal{M} \not\models p(\bar{a}) : \mathbf{t_d}$, a contradiction. Thus, we have that $\mathcal{M} \models p(\bar{a}) : \mathbf{f_a}$. But $\mathcal{M} <_{\Delta} \mathcal{M}(\mathbf{DB}, \mathbf{DB}')$, and therefore $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models p(\bar{a}) : \mathbf{f_a}$. Then, by considering (1) we conclude that $p(\bar{a}) \notin \mathbf{DB}'$, and therefore in this case it is possible to conclude that $p(\bar{a}) \in \Delta(\mathbf{DB}, \mathbf{DB}')$. In the second case we can conclude that $p(\bar{a}) : \mathbf{f_d} \in \mathcal{T}(\mathbf{DB}, \mathbf{IC})$ and $\mathcal{M} \models p(\bar{a}) : \mathbf{t} \vee p(\bar{a}) : \mathbf{t_a}$. If we suppose that $\mathcal{M} \models p(\bar{a}) : \mathbf{t}$, then $\mathcal{M} \not\models p(\bar{a}) : \mathbf{f_d}$, a contradiction. Thus, we have that $\mathcal{M} \models p(\bar{a}) : \mathbf{t_a}$. But $\mathcal{M} <_{\Delta} \mathcal{M}(\mathbf{DB}, \mathbf{DB}')$, and therefore $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models p(\bar{a}) : \mathbf{t_a}$. Then, by considering (1) we conclude that $p(\bar{a}) \in \mathbf{DB}'$, and therefore in this case it is possible to conclude that $p(\bar{a}) \in \Delta(\mathbf{DB}, \mathbf{DB}')$. Thus, we can conclude that $\Delta(\mathbf{DB}, \mathbf{DB}_{\mathcal{M}}) \subsetneq \Delta(\mathbf{DB}, \mathbf{DB}')$.
   (b) Since $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \not\leq_{\Delta} \mathcal{M}$, there exists $p(\bar{a})$ such that $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models p(\bar{a}) : \mathbf{t_a} \vee p(\bar{a}) : \mathbf{f_a}$ and $\mathcal{M} \models p(\bar{a}) : \mathbf{t} \vee p(\bar{a}) : \mathbf{f}$. By using (1) and the first fact it is possible to conclude that $p(\bar{a}) \in \Delta(\mathbf{DB}, \mathbf{DB}')$. If we suppose that $p(\bar{a}) \in \mathbf{DB}$, then $p(\bar{a}) : \mathbf{t_d} \in \mathcal{T}(\mathbf{DB}, \mathbf{IC})$, and therefore by considering the second fact it is possible to deduce that $\mathcal{M}$ must satisfy $p(\bar{a}) : \mathbf{t}$. Thus, we can conclude that in this case $p(\bar{a}) \in \mathbf{DB}_{\mathcal{M}}$, and therefore $p(\bar{a}) \notin \Delta(\mathbf{DB}, \mathbf{DB}_{\mathcal{M}})$. By the other hand, if we suppose that $p(\bar{a}) \notin \mathbf{DB}$, then $p(\bar{a}) : \mathbf{f_d} \in \mathcal{T}(\mathbf{DB}, \mathbf{IC})$, and therefore by considering the second fact it is possible to deduce that $\mathcal{M}$ must satisfy $p(\bar{a}) : \mathbf{f}$. Thus, we can conclude that in this case $p(\bar{a}) \notin \mathbf{DB}_{\mathcal{M}}$, and therefore $p(\bar{a}) \notin \Delta(\mathbf{DB}, \mathbf{DB}_{\mathcal{M}})$. Finally, we conclude that $\Delta(\mathbf{DB}, \mathbf{DB}_{\mathcal{M}}) \nsupseteq \Delta(\mathbf{DB}, \mathbf{DB}')$.

   We know that $\mathbf{DB}'$ is a database instance, and therefore $\Delta(\mathbf{DB}, \mathbf{DB}')$ must be a finite set. Thus, we can conclude that $\Delta(\mathbf{DB}, \mathbf{DB}_{\mathcal{M}})$ is a finite set, and therefore $\mathbf{DB}_{\mathcal{M}}$ is a database instance. With the help of Lemma 3, we deduce that $\mathbf{DB}_{\mathcal{M}} \models \mathbf{IC}$. But this a contradiction, since $\mathbf{DB}'$ is a repair of $\mathbf{DB}$ with respect to $\mathbf{IC}$ and $\Delta(\mathbf{DB}, \mathbf{DB}_{\mathcal{M}}) \subsetneq \Delta(\mathbf{DB}, \mathbf{DB}')$.

□

Proof of lemma 4: Let us suppose that

$$\mathcal{T}(\mathbf{DB}, \mathbf{IC}) \vdash r_1(\bar{c}_1) : ?_{\mathbf{a}} \vee \cdots \vee \cdots \vee r_k(\bar{c}_k) : ?_{\mathbf{a}}. \tag{5}$$

Because of the form of the clauses in $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$, the above a-clause can be obtained by applying a series of reduction and resolution rules to the clauses in $\mathcal{T}(\mathbf{DB}) \cup \mathcal{T}(\mathcal{B})$ (the database part of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$ plus builtins) and a clause of the form

$$\begin{aligned} r_1(\bar{t}_1) : \mathbf{f_c} \vee \cdots \vee r_j(\bar{t}_j) : \mathbf{f_c} \vee r_{j+1}(\bar{t}_{j+1}) : \mathbf{t_c} \vee \\ \cdots \vee r_k(\bar{t}_k) : \mathbf{t_c}, \end{aligned} \tag{6}$$

where the latter is a clause obtained from $\mathcal{T}(\mathbf{IC})$ (the constraint part of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$) by resolution (and factorization) alone.

Furthermore, it is easy to show that resolution applied to a pair of range-restricted constraints yields a range-restricted constraint. Thus, (6) is range-restricted.

Since (5) is obtained from (6) by resolution and reduction with the clauses in $\mathcal{T}(\mathbf{DB})$, there must be clauses $r_i(\bar{c}_i) : \mathbf{t} \vee r_i(\bar{c}_i) : \top \in \mathcal{T}(\mathbf{DB})$, $1 \leq i \leq j$ (which are resolved with (6)), and clauses $r_{i'}(\bar{c}_{i'}) : \mathbf{f} \vee r_{i'}(\bar{c}_{i'}) : \top \in \mathcal{T}(\mathbf{DB})$, $j < i' \leq k$ (which are reduced with (6)), such that there is a substitution θ for which $\bar{t}_i\theta = \bar{c}_i$ $(1 \leq i \leq k)$.

Therefore, due to the range-restrictedness of (6), every constant in $\bar{c}_{i'}$ $(j < i' \leq k)$ occurs in some $\bar{c}_i$ $(1 \leq i \leq j)$. Since every constant in $\bar{c}_i$ is in the active domain of $\mathbf{DB}$, we conclude that every constant mentioned in (5) belongs to the active domain of $\mathbf{DB}$. □

Proof of corollary 1: By Lemma 4, the clauses in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB},\mathbf{IC})$ can mention only the constants that occur in the active domain of $\mathbf{DB}$, which is a finite set. □

Proof of theorem 3: At the end of section 6 we showed that the decision problem is equivalent to the problem of deciding, given a finite collection of sets, and a subset of the union of the family, whether the subset can be extended to a minimal hitting set of the family. In the following lemmas we prove that this is *NP-complete.*

Lemma 5. *Given a finite collection of sets S and a hitting set of it H, H is a minimal hitting set of S if and only if for each $h \in H$ there exists an $A \in S$ such that $A \cap H = \{h\}$.*

Proof
($\Rightarrow$) Let us suppose that the lemma is not true. Then there exists $h \in H$ such that for every $A \in S$, $A \cap H \neq \{h\}$. We are going to prove $H' = H - \{h\}$ is also a hitting set. Let us consider $A \in S$. If $h \in A$, then there exists another $h' \in H$ such that $h' \in A$, since $A \cap H \neq \{h\}$, and therefore $A \cap H' \neq \emptyset$. If $h \notin A$, then there exist $h' \neq h$ such that $h' \in A \cap H$, and therefore $A \cap H' \neq \emptyset$. Thus, we obtain a contradiction.

($\Leftarrow$) If $H' \subsetneqq H$, then there exists $h \in H$ such that $h \notin H'$. But we know that there is a set $A \in S$ such that $A \cap H = \{h\}$, and therefore $A \cap H' = \emptyset$. Thus, H' is not a hitting set of S.

Lemma 6. *Given a finite collection of sets S and a set $H \subseteqq \cup S$, the problem of deciding if there exists a minimal hitting set H' of S such that $H \subseteqq H'$ is NP*

Proof We are going to reduce our problem to SAT. For each $x \in \cup S$ we introduce a propositional letter x, and we define:

$$f(S,H) = (\bigwedge_{h \in H} \bigvee_{\{A \in S \ \mid \ h \in A\}} \bigwedge_{\{a \in A \ \mid \ a \neq h\}} \neg a) \wedge$$
$$\bigwedge_{h \in H} h \wedge (\bigwedge_{\{A \in S \ \mid \ A \cap H = \emptyset\}} \bigvee_{a \in A} a).$$

There exists a minimal hitting set H' of S which contains H if and only if $f(H,S)$ is a satisfied formula.
($\Rightarrow$) For every proposition letter x in $f(H,S)$ we define $\sigma(x) = 1$ if and only if $x \in H'$.

1. If $h \in H$, then $h \in H'$, and therefore by lemma 5 we conclude that there exists $A \in S$ such that $A \cap H' = \{h\}$. Thus, for every $a \in A - \{h\}$ we have that $a \notin H'$, and then $\sigma(a) = 0$. We conclude that $\sigma(\bigvee_{\{A \in S \ \mid \ h \in A\}} \bigwedge_{\{a \in A \ \mid \ a \neq h\}} \neg a) = 1$.
2. $\sigma(\bigwedge_{h \in H} h) = 1$, since $H \subseteqq H'$.
3. If $A \in S$ and $A \cap H = \emptyset$, then $A \cap (H' - H) \neq \emptyset$, since H' is a hitting set of S. Thus, there exists $a \in H'$ such that $a \in A$, and therefore $\sigma(a) = 1$. We conclude that $\sigma(\bigvee_{a \in A} a) = 1$.

($\Leftarrow$) Let σ such that $\sigma(f(H,S)) = 1$. We construct $H'' = \{x \ \mid \ \sigma(x) = 1\}$. $H \subseteqq H''$, since $\sigma(\bigwedge_{h \in H} h) = 1$. H'' is a hitting set of S. Let us consider $A \in S$. If $A \cap H \neq \emptyset$, then $A \cap H'' \neq \emptyset$. If $A \cap H = \emptyset$, then $\sigma(\bigvee_{a \in A} a) = 1$, and therefore $A \cap (H'' - H) \neq \emptyset$.

H'' is a finite set. Then there exists a minimal hitting set of S such that $H' \subseteqq H''$. We are going to prove that $H \subseteqq H'$. By contradiction, let us suppose that there exists $h \in H$ such that $h \notin H'$. We know that $\sigma(\bigvee_{\{A \in S \ \mid \ h \in A\}} \bigwedge_{\{a \in A \ \mid \ a \neq h\}} \neg a) = 1$. Then there exists $A \in S$ such that $\sigma(\bigwedge_{\{a \in A \ \mid \ a \neq h\}} \neg a) = 1$, and therefore $A \cap H' = \emptyset$, by definition of H' and given that $h \notin H'$. Thus, we conclude a contradiction.

Lemma 7. *Given a finite collection of sets S and a set $H \subseteqq \cup S$, the problem of deciding if there exists a minimal hitting set H' of S such that $H \subseteqq H'$ is NP-hard*

Proof. We are going to reduce SAT(3) to our problem. Given a formula $\varphi = C_1 \wedge \dots \wedge C_k$, where every C_i is a clause, we define $PL(\varphi)$ as the set of propositional letters mentioned in it. Additionally, for each clause C_i, of the form $p_1 \vee \dots \vee p_n \vee \neg q_1 \vee \dots \vee \neg q_m$, we define

$$CH(C_i) = \{p_1\_1, ..., p_n\_1, q_1\_0, ..., q_m\_0\}.$$

After that, we define $f(\varphi) = (S,H)$, where

$$S = \{\{v\_p, p\_0\} \ \mid \ p \in PL(\varphi)\} \cup \{\{v\_p, p\_1\} \ \mid \ p \in PL(\varphi)\} \cup \{CH(C_i) \ \mid \ 1 \leq i \leq k\}$$
$$H = \{v\_p \ \mid \ p \in PL(\varphi)\}$$

We are going to prove that φ is consistent if and only if there exists a minimal hitting set H' of S such that $H \subseteqq H'$.

($\Rightarrow$) Let σ that satisfies φ. We define

$$H'' = H \cup \{p\_0 \ \mid \ p \in PL(\varphi) \text{ and } \sigma(p) = 0\} \cup \{p\_1 \ \mid \ p \in PL(\varphi) \text{ and } \sigma(p) = 1\}$$

H'' is a hitting set of S, and therefore there exists H' minimal hitting set of S such that $H' \subseteqq H''$, since H'' is a finite set. If we suppose that there is $v\_p \in H$ such that $v\_p \notin H'$, then $H' \cap \{v\_p, p\_0\} = \emptyset$ or $H' \cap \{v\_p, p\_1\} = \emptyset$, given that $\sigma(p) = 1$ or $\sigma(p) = 0$. Thus, we conclude a contradiction.

($\Leftarrow$) Let us suppose that there exists H' minimal hitting set of S such that $H \subseteqq H'$. Notice that for every $p \in PL(\varphi)$ we have that $p\_0 \notin H'$ or $p\_1 \notin H'$, since if both elements would be in H', then $H' - \{v\_p\}$ will be a hitting set, a contradiction given that H' is minimal. Thus, we can define a function $\sigma : PL(\varphi) \to \{0,1\}$ by means of the rule $\sigma(p) = 1$ if and only if $p\_1 \in H'$. We have that $\sigma(\varphi) = 1$, given that for every clause C_i, $H' \cap CH(C_i) \neq \emptyset$. □

Repairing Databases with Annotated Predicate Logic

Pablo Barceló
P. Universidad Católica de Chile
Depto. Ciencia de Computación
Santiago, Chile.
pbarcelo@ing.puc.cl

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

Abstract

Consistent answers from a relational database that violates a given set of integrity constraints are characterized [Arenas et al. 1999] as ordinary answers that can be obtained from *every* repaired version of the database. In this paper we address the problem of specifying the repairs of a database as the minimal models of a theory written in *Annotated Predicate Logic* [Kifer et al. 1992a]. The specification is then transformed into a disjunctive logic program with annotation arguments and a stable model semantics. From the program, consistent answers to first order queries are obtained.

1 Introduction

Integrity constraints (ICs) are important in the design and use of a relational database. They embody the semantics of the application domain and help maintain the correspondence between that application domain and its model provided by the database. Nevertheless, it is not strange for a database instance to become inconsistent with respect to a given, expected set of ICs. This could happen due to different factors, being one of them the integration of several data sources. The integration of consistent databases may easily lead to an inconsistent integrated database.

An important problem in databases consists in retrieving answers to queries that are "consistent" with the given ICs, even when the database as a whole does not satisfy those ICs. Very likely "most" of the data is still consistent. The notion of consistent answers to a first order (FO) query was defined in [Arenas et al. 1999], where also a computational mechanism for obtaining them was presented. Intuitively speaking, an ground tuple $\bar{t}$ is a consistent answer to a first order query $Q(\bar{x})$ in a, possibly inconsistent, relational database instance DB if it is an (ordinary) answer to $Q(\bar{x})$ in every minimal repair of DB, that is in every database instance over the same schema that differs from DB by a minimal (under set inclusion) set of inserted or deleted tuples.

That mechanism presented in [Arenas et al. 1999] has some limitations in terms of the ICs and queries that can be handled. In [Arenas et al. 2000b], a more general methodology based on logic programs with an stable model semantics was introduced. More general queries could be considered, but ICs were restricted to be "binary", i.e. universal with at most two database literals (plus built-in formulas).

For consistent query answering we need to deal with *all* the repairs of a database. In consequence, a natural approach consists in providing a manageable logical specification of the class of database repairs. The specification must include information about (from) the database and the information contained in the ICs. Since these two pieces of information are mutually inconsistent, we need a logic that does not collapse in the presence of contradictions. A logic like *Annotated Predicate Logic* (APC) [Kifer et al. 1992a], for which a classically inconsistent set of premises can still have a model, is a natural candidate.

In [Arenas et al. 2000a], a new declarative semantic framework was introduced for studying the problem of query answering in databases that are inconsistent with integrity constraints. This was done by embedding both the database instance and the integrity constraints into a single theory written in APC, with an appropriate non classical truth-values lattice *Latt*. It was shown that, for universal ICs, there is a one to one correspondence between some minimal models of the annotated theory and the repairs of the inconsistent database. In this way, a logical specification of the database repairs was achieved. The annotated theory was used to obtain some algorithms for obtaining consistent answers to some simple first order queries.

This paper extends the results presented in [Arenas et al. 2000a] in several ways. First, in section 3, we show how to annotate and integrate referential ICs (that contain existential quantifiers) in addition to universal ICs into the annotated theory. The correspondence between minimal models of the theory and the database repairs is established. Next, in section 4, we show how to annotate queries and the formulation of the problem of consistent query answering as a problem of non-monotonic (minimal) entailment from the annotated theory. Then, in section 5.1, on the basis of the generated annotated theory, disjunctive logic programs with annotation arguments are derived in such a way that they specify the database repairs. After that, in section 5.2, we show how to use those programs to obtain consistent answers to first order queries. In section 5.3 the logic programs are transformed into classical disjunctive normal programs with a stable model semantics; and the *coherent* stable models become the database repairs. In section 2 we present the basic framework, and in section 6 we draw some conclusions, mention ongoing work, and consider related work.

The methodology presented here works for arbitrary first order queries and arbitrary universal ICs, what considerable extends the cases that could be handled in [Arenas et al. 1999, Arenas et al. 2000b, Arenas et al. 2000a].

2 Preliminaries

2.1 Database repairs and consistent answers

In the context of relational databases, we will consider a fixed relational schema $\Sigma = (D, P \cup B)$ that determines a first order language. It consists of a fixed, possibly infinite, database domain $D = \{c_1, c_2, ...\}$, a fixed set of database predicates $P = \{p_1, \ldots, p_n\}$, and a fixed set of built-in predicates $B = \{e_1, \ldots, e_m\}$.

A database instance over Σ is a finite collection DB of facts of the form $p(c_1, ..., c_n)$, where p is a predicate in P and $c_1, ..., c_n$ are constants in D. Built-in predicates have a fixed and same extension in every database instance, not subject to changes.

An *integrity constraint* (IC) is an implicitly quantified clause of the form

$$q_1(\bar{t}_1) \vee \cdots \vee q_n(\bar{t}_n) \vee \neg p_1(\bar{s}_1) \vee \cdots \vee \neg p_m(\bar{s}_m) \quad (1)$$

in the FO language of Σ, where each p_i, q_j is a predicate in $P \cup B$ and the $\bar{t}_i, \bar{s}_j$ are tuples containing constants and variables. We assume we have a fixed set IC of ICs.

We will assume that DB and IC, separately, are consistent theories. Nevertheless, it may be the case that $DB \cup IC$ is inconsistent. Equivalently, if we associate to DB a first order structure, also denoted with DB, in the natural way, i.e. by applying the closed world assumption that makes false any ground atom not explicitly appearing in the set of atoms DB, it may happen that DB, as a structure, does not satisfy the IC. We denote with $DB \models_\Sigma IC$ the fact that the database satisfies IC. In this case we say that DB is consistent wrt IC; otherwise we say DB is inconsistent.

As in [Arenas et al. 1999], we define the *distance* between two database instances DB_1 and DB_2 as their symmetric difference $\Delta(DB_1, DB_2) = (DB_1 - DB_2) \cup (DB_2 - DB_1)$.

Now, given a database instance DB, possibly inconsistent wrt IC, we say that the instance DB' is a *repair* of DB iff $DB' \models_\Sigma IC$ and $\Delta(DB, DB')$ is minimal under set inclusion in the class of instances that satisfy IC and conform to schema Σ [Arenas et al. 1999].

Example 1. Consider the relational schema *Book*(*author*, *name*, *publYear*), a database instance $DB = \{Book(kafka, metamorph, 1915), Book(kafka, metamorph, 1919)\}$; and the functional dependency FD : *author*, *name* → *publYear*, that can be expressed by IC : $\neg Book(x, y, z) \vee \neg Book(x, y, w) \vee z = w$. Instance DB is inconsistent with respect to IC. The original instance has two possible repairs: $DB_1 = \{Book(kafka, metamorph, 1915)\}$, and $DB_2 = \{Book(kafka, metamorph, 1919)\}$. □

Let DB be a database instance, possibly not satisfying a set IC of integrity constraints. Given a query $Q(\bar{x})$ to DB, we say that a tuple of constants $\bar{t}$ is a *consistent answer* to $Q(\bar{x})$ in DB [Arenas et al. 1999], denoted $DB \models_c Q(\bar{t})$, if for every repair DB' of DB, $DB' \models_\Sigma Q(\bar{t})$. If Q is a closed formula, *i.e.* a sentence, then *true* is a *consistent answer* to Q, denoted $DB \models_c Q$, if for every repair DB' of DB, $DB' \models_\Sigma Q$.

Example 2. (example 1 continued) The query Q_1 : $Book(kafka, metamorph, 1915)$ does not have *true* as a consistent answer, because it is not true in every repair. Query $Q_2(y)$: $\exists x \exists z Book(x, y, z)$ has $y = metamorph$ as a consistent answer. Query $Q_3(x)$: $\exists z Book(x, metamorph, z)$ has $x = kafka$ as a consistent answer. □

2.2 Annotating DBs and ICs

Annotated Predicate Calculus was introduced in [Kifer et al. 1992a] and also studied in [Blair et al. 1989] and [Kifer et al. 1992b]. It constitutes a non classical logic, where classically inconsistent information does not unravel logical inference, reasoning about causes of inconsistency becomes possible, making one of its goals to study the differences in the contribution to the inconsistency made by the different literals in a theory, what is related to the the problem of consistent query answers.

The syntax of APC is similar to that of classical logic, except for the fact that the atoms are annotated with values drawn from a *truth-values lattice.* The lattice *Latt* we will use throughout this paper is shown in Figure 1, first introduced in [Arenas et al. 2000a].

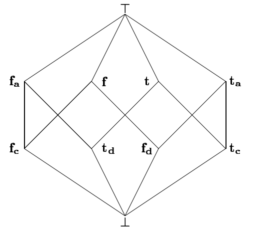


Figure 1: *Latt* with *constraints values*, *database values* and *advisory values*

Intuitively, we can think of values $\mathbf{t_c}$ and $\mathbf{f_c}$ as specifying what is needed for constraint satisfaction. The values $\mathbf{t_d}$ and $\mathbf{f_d}$ represent the truth values according to the original database. Finally, $\mathbf{t_a}$ and $\mathbf{f_a}$ are considered *advisory* truth values. These are intended to solve conflicts between the original database and what is needed for the satisfaction of the integrity constraints. Notice that $lub(\mathbf{t_d}, \mathbf{f_c}) = \mathbf{f_a}$ and $lub(\mathbf{f_d}, \mathbf{t_c}) = \mathbf{t_a}$. The intuition behind is that, in case of a conflict between the constraints and the database, we should obey the constraints, because the database instance only can be changed to restore consistency. This lack of symmetry between data and ICs is captured by the lattice. Advisory value $\mathbf{t_a}$ is an indication that the atom annotated with it must be inserted into the DB; and deleted from the DB when annotated with $\mathbf{f_a}$.

Herbrand interpretations are now sets of annotated ground atoms. The notion of formula satisfaction in an *Herbrand interpretation* I is defined classically, except for atomic formulas p, where we say that $I \models p\!:\!\mathbf{s}$, with $\mathbf{s} \in Latt$, iff for some $\mathbf{s}'$ such that $\mathbf{s} \leq \mathbf{s}'$ we have that $p\!:\!\mathbf{s}' \in I$ [Kifer et al. 1992a].

Given an APC theory $\mathcal{T}$, we say that an Herbrand interpretation I is a Δ-minimal model of $\mathcal{T}$, with $\Delta = \{\mathbf{t_a}, \mathbf{f_a}\}$, if I is a model of $\mathcal{T}$ and no other model of $\mathcal{T}$ has a proper subset of atoms annotated with elements in Δ, *i.e.* the set of atoms annotated with $\mathbf{t_a}$ or $\mathbf{f_a}$ in I is minimal under set inclusion.

Given a database instance DB and a set of integrity constraints IC of the form (1), an embedding $\mathcal{T}(DB, IC)$ of DB and IC into a new APC theory can be defined [Arenas et al. 2000a] in order to restore consistency using the annotations in the lattice. It was shown in [Arenas et al. 2000a] that there is a one-to-one correspondence between the Δ-minimal models (we will simply say "minimal" in the rest of the paper) of theory $\mathcal{T}(DB, IC)$ and the repairs of the original database instance. Repairs can be obtained from minimal models as follows:

Definition 1. Given a minimal model $\mathcal{M}$ of $\mathcal{T}(DB, IC)$, the corresponding DB instance is defined by $DB_{\mathcal{M}} = \{p(\bar{a}) \mid \mathcal{M} \models p(\bar{a})\!:\!\mathbf{t} \vee p(\bar{a})\!:\!\mathbf{t_a}\}$. □

Example 3. (example 1 cont.) The embedding $\mathcal{T}(DB)$ of DB into APC is given by the following formulas:

1. $Book(kafka, metamorph, 1915)\!:\!\mathbf{t_d}$
   $Book(kafka, metamorph, 1919)\!:\!\mathbf{t_d}$.
2. Predicate closure axioms:
   $((x = kafka)\!:\!\mathbf{t_d} \ \wedge \ (y = metamorph)\!:\!\mathbf{t_d} \wedge$
   $(z = 1915)\!:\!\mathbf{t_d}) \vee$
   $((x = kafka)\!:\!\mathbf{t_d} \ \wedge \ (y = metamorph)\!:\!\mathbf{t_d} \wedge$
   $(z = 1919)\!:\!\mathbf{t_d}) \vee$
   $Book(x, y, z)\!:\!\mathbf{f_d}$.

   Every ground atom that is not in DB is (possibly implicitly) annotated with $\mathbf{f_d}$.

The embedding $\mathcal{T}(IC)$ of IC into APC is given by:

3. $Book(x, y, z)\!:\!\mathbf{f_c} \vee Book(x, y, w)\!:\!\mathbf{f_c} \vee (z = w)\!:\!\mathbf{t_c}$.
4. $Book(x, y, z)\!:\!\mathbf{f_c} \vee Book(x, y, z)\!:\!\mathbf{t_c}$
   $\neg Book(x, y, z)\!:\!\mathbf{f_c} \vee \neg Book(x, y, z)\!:\!\mathbf{t_c}$.[1]

   These formulas specify that every fact must have one and just one constraint value.

Furthermore

5. For every true *built-in* atom φ we include $\varphi\!:\!\mathbf{t}$ in $\mathcal{T}(B)$, and $\varphi\!:\!\mathbf{f}$ for every false built-in atom, e.g. $(1915 = 1915)\!:\!\mathbf{t}$, but $(1915 = 1919)\!:\!\mathbf{f}$.

The Δ-minimal models of $\mathcal{T}(DB, IC) = \mathcal{T}(DB) \cup \mathcal{T}(IC) \cup \mathcal{T}(B)$ are:

$$\mathcal{M}_1 = \{Book(kafka, metamorph, 1915)\!:\!\mathbf{t}, \; Book(kafka, metamorph, 1919)\!:\!\mathbf{f_a}\},$$

$$\mathcal{M}_2 = \{Book(kafka, metamorph, 1915)\!:\!\mathbf{f_a}, \; Book(kafka, metamorph, 1919)\!:\!\mathbf{t}\},$$

plus annotated false DB atoms and built-ins in both cases . The correspondent database instances, $DB_{\mathcal{M}_1}, DB_{\mathcal{M}_2}$ are the repairs of DB shown in example 1. □

From the definition of the lattice and the fact that no atom from the database is annotated with both $\mathbf{t_d}$ and $\mathbf{f_d}$, it is possible to show that, in the minimal models of the annotated theory, a DB atom may get the annotations either $\mathbf{t}$ or $\mathbf{f_a}$ if the atom was annotated with $\mathbf{t_d}$, and either $\mathbf{f}$ or $\mathbf{t_a}$ if the atom was annotated with $\mathbf{f_d}$. In the transition from the annotated theory to its minimal models, the annotations $\mathbf{t_d}, \mathbf{f_d}$ "disappear", as we wished the atoms to be annotated in the highest possible layer in the lattice, except for ⊤ if possible. Actually, in the minimal models ⊤ can always be avoided.

[1] Since only atomic formulas are annotated, the non atomic formula $\neg p(\bar{x})\!:\!\mathbf{s}$ is to be read as $\neg(p(\bar{x})\!:\!\mathbf{s})$. We will omit the parenthesis though.

3 Annotating Referential ICs

Referential integrity constraints (RICs) like

$$p(\bar{x}) \rightarrow \exists y q(\bar{x}', y), \quad (2)$$

where the variables in $\bar{x}'$ are a subset of the variables in $\bar{x}$, cannot be expressed as an equivalent clause of the form (1). RICs are important and common in databases. For that reason, we need to extend our embedding methodology. Actually, we embed (2) into APC by means of

$$p(\bar{x})\!:\!\mathbf{f_c} \vee \exists y (q(\bar{x}', y)\!:\!\mathbf{t_c}). \quad (3)$$

Now we allow the given set of ICs to contain, in addition to ICs of the form (1), RICs like (2). The one-to-one correspondence between minimal models of the new theory $\mathcal{T}(DB, IC)$ and the repairs of DB still holds. Most important for us is to obtain repairs from minimal models.

Proposition 1. Let $\mathcal{M}$ be a model of $\mathcal{T}(DB, IC)$. If $\mathcal{M}$ is minimal and $DB_{\mathcal{M}}$ is finite, then $DB_{\mathcal{M}}$ is a repair of DB with respect to IC. □

Example 4. Consider the relational schema of Example 1 extended with table *Author*(*name*, *citizenship*). Now, IC also contains the RIC: $Book(x, y, z) \rightarrow \exists w Author(x, w)$, expressing that every writer of a book in the database instance must be registered as an author. The theory $\mathcal{T}(IC)$ now also contains:

$$Book(x, y, z)\!:\!\mathbf{f_c} \vee \exists w (Author(x, w)\!:\!\mathbf{t_c}),$$

$$Author(x, w)\!:\!\mathbf{f_c} \vee Author(x, w)\!:\!\mathbf{t_c},$$

$$\neg Author(x, w)\!:\!\mathbf{f_c} \vee \neg Author(x, w)\!:\!\mathbf{t_c}.$$

We might also have the functional dependency FD : *name* → *citizenship*, that in conjunction with the RIC, produces a foreign key constraint. The database instance $\{Book(neruda, 20lovepoems, 1924)\}$ is inconsistent wrt the given RIC. If we have the following subdomain $D(Author.citizenship) = \{chilean, canadian\}$ for the attribute "citizenship", we obtain the following database theory:

$$\mathcal{T}(DB) = \{Book(neruda, 20lovepoems, 1924)\!:\!\mathbf{t_d}, \; Author(neruda, chilean)\!:\!\mathbf{f_d}, \; Author(neruda, canadian)\!:\!\mathbf{f_d}, \ldots\}.$$

The minimal models of $\mathcal{T}(DB, IC)$ are:

$$\mathcal{M}_1 = \{Book(neruda, 20lovepoems, 1924)\!:\!\mathbf{f_a}, \; Author(neruda, chilean)\!:\!\mathbf{f},$$

$$\begin{aligned}
&\quad Author(neruda, canadian){:}\mathbf{f}, \ldots\}\\
\mathcal{M}_2 = \{&Book(neruda, \mathit{20lovepoems}, \mathit{1924}){:}\mathbf{t},\\
&Author(neruda, chilean){:}\mathbf{t_a},\\
&Author(neruda, canadian){:}\mathbf{f}, \ldots\}\\
\mathcal{M}_3 = \{&Book(neruda, \mathit{20lovepoems}, \mathit{1924}){:}\mathbf{t},\\
&Author(neruda, chilean){:}\mathbf{f},\\
&Author(neruda, canadian){:}\mathbf{t_a}, \ldots\}.
\end{aligned}$$

We obtain $DB_{\mathcal{M}_1} = \emptyset$, $DB_{\mathcal{M}_2} = \{Book(neruda, \mathit{20lovepoems}, \mathit{1924}), Author(neruda, chilean)\}$ and $DB_{\mathcal{M}_3}$ similar to $DB_{\mathcal{M}_2}$, but with a Canadian Neruda. According to proposition 1, these are repairs of the original database instance, actually the only ones. □

As in [Arenas et al. 2000a], it can be proved that when the original instance is consistent, then it is its only repair and it corresponds to a unique minimal model of the APC theory.

4 Annotation of Queries

According to proposition 1, a ground tuple $\bar{t}$ is a consistent answer to a FO query $Q(\bar{x})$ iff $Q(\bar{t})$ is true of every minimal model of $\mathcal{T}(DB, IC)$. However, if we want to pose the query directly to the theory, it is necessary to reformulate it as an annotated formula.

Definition 2. Given a FO query $Q(\bar{x})$ in language Σ, we denote by $Q^{an}(\bar{x})$ the annotated formula obtained from Q by simultaneously replacing, for $p \in P$, the negative literal $\neg p(\bar{s})$ by the APC formula $p(\bar{s}){:}\mathbf{f} \vee p(\bar{s}){:}\mathbf{f_a}$, and the positive literal $p(\bar{s})$ by the APC formula $p(\bar{s}){:}\mathbf{t} \vee p(\bar{s}){:}\mathbf{t_a}$. For $p \in B$, the literal $p(\bar{s})$ is replaced by the APC formula $p(\bar{s}){:}\mathbf{t}$. □

According to this definition, logically equivalent versions of a query could have different annotated versions, but it can be shown (proposition 2), that they retrieve the same consistent answers.

Example 5. (example 1 cont.) If we want the consistent answers to the query $Q(x) : \neg\exists y\exists z\exists w\exists t(Book(x, y, z) \wedge Book(x, w, t) \wedge y \neq w)$, asking for those authors that have at most one book in DB, we generate the annotated query $Q^{an}(\bar{x}) : \neg\exists y\exists z\exists w\exists t((Book(x, y, z) : \mathbf{t} \vee Book(x, y, z) : \mathbf{t_a}) \wedge (Book(x, w, t) : \mathbf{t} \vee Book(x, w, t) : \mathbf{t_a}) \wedge (y \neq w) : \mathbf{t})$, to be posed to the annotated theory with its minimal model semantics. □

Definition 3. If φ is a sentence in the language of $\mathcal{T}(DB, IC)$, we say that $\mathcal{T}(DB, IC)$ Δ-minimally entails φ, written $\mathcal{T}(DB, IC) \models_\Delta \varphi$, iff every Δ-minimal model $\mathcal{M}$ of $\mathcal{T}(DB, IC)$, such that $DB_{\mathcal{M}}$ is finite, satisfies φ, *i.e.* $\mathcal{M} \models \varphi$. □

Now we characterize consistent query answers wrt the annotated theory.

Proposition 2. Let DB be a database instance, IC a set of integrity constraints and $Q(\bar{x})$ a query in FO language Σ. It holds:

$$DB \models_c Q(\bar{t}) \ \text{ iff } \ \mathcal{T}(DB, IC) \models_\Delta Q^{an}(\bar{t}). \quad □$$

Example 6. (example 5 continued) For consistently answering the query $Q(x)$, we pose the query $Q^{an}(x)$ to the minimal models of $\mathcal{T}(DB, IC)$. The answer we obtain from *every* minimal model is $x = kafka$. □

According to this proposition, in order to consistently answer queries, we are left with the problem of evaluating minimal entailment wrt the annotated theory. In [Arenas et al. 2000a] some limited FO queries were evaluated, but no annotated queries were generated. The original query were answered using ad hoc algorithms that were extracted from theory $\mathcal{T}(DB, IC)$. No advantage was taken from a characterization of consistent answers in terms of minimal entailment from $\mathcal{T}(DB, IC)$. In the next section we will address this issue by taking the original DB instance with the ICs into a logic program that is generated taking advantage of the annotations provided by $\mathcal{T}(DB, IC)$. The query to be posed to the logic program will be built from Q^{an}.

5 Query Answering

In this section we will consider ICs of the form (1), more precisely of the form

$$\bigvee_{i=1}^{n} \neg p_i(\bar{t_i}) \vee \bigvee_{j=1}^{m} q_j(\bar{s_j}) \vee \varphi, \tag{4}$$

where, for every i and j, p_i and q_j are predicates in P, and φ is a formula containing predicates in B only.

5.1 Logic programming specification of repairs

In order to generate a first order logic program that gives an account of annotations, for each predicate $p(\bar{x}) \in P$, we introduce a new, predicate $p(\bar{x}, \cdot)$, with an extra argument for annotations. This defines a new FO language, Σ^{ext}, for extended Σ. The repair logic program, $\Pi(DB, IC)$, for DB and IC, is written with predicates from Σ^{ext} and contains the following clauses:

1. For every atom $p(\bar{a}) \in DB$, $\Pi(DB, IC)$ contains the fact $p(\bar{a}, \mathbf{t_d}) \leftarrow$.

2. For every predicate $p \in P$, $\Pi(DB, IC)$ contains the clauses:

   $p(\bar{x}, \mathbf{t}^\star) \leftarrow p(\bar{x}, \mathbf{t_d})$

   $p(\bar{x}, \mathbf{t}^\star) \leftarrow p(\bar{x}, \mathbf{t_a})$

   $p(\bar{x}, \mathbf{f}^\star) \leftarrow p(\bar{x}, \mathbf{f_a})$,

   where $\mathbf{t}^\star, \mathbf{f}^\star$ are new, auxiliary elements in the domain of annotations.

3. For every constraint of the form (4), $\Pi(DB, IC)$ contains the clause:

   $$\bigvee_{i=1}^n p_i(\bar{t_i}, \mathbf{f_a}) \ \vee \ \bigvee_{j=1}^m q_j(\bar{s_j}, \mathbf{t_a}) \ \longleftarrow \ \bigwedge_{i=1}^n p_i(\bar{t_i}, \mathbf{t}^\star) \ \wedge \ \bigwedge_{j=1}^m q_j(\bar{s_j}, \mathbf{f}^\star) \ \wedge \ \bar{\varphi},$$

   where $\bar{\varphi}$ represents the negation of φ.

Intuitively, the clauses in 3. say that when the IC is violated (the body), then the DB has to be repaired according to one of the alternatives shown in the head. Since there may be interactions between constraints, these single repairing steps may not be enough to restore the consistency of the DB. We have to make sure that the repairing process continues and stabilizes in a state where all the ICs hold[2]. This is the role of the clauses in 2. containing the new annotations $\mathbf{t}^\star$, that groups together those atoms annotated with $\mathbf{t_d}$ and $\mathbf{t_a}$, and $\mathbf{f}^\star$, that does the same with $\mathbf{f_d}$ and $\mathbf{f_a}$ (with the help of Definition 4 below).

The following example shows the interaction of a FD and an inclusion dependency. When atoms are deleted in order to satisfy the FD, the inclusion dependency could be violated, and in a second step it should be repaired. At that second step, the annotations $\mathbf{t}^\star$ and $\mathbf{f}^\star$, computed at the first step where the FD was repaired, will detect the violation of the inclusion dependency and perform the corresponding repairing process.

Example 7. (example 1 cont.) We extend the schema with table $Eurbook(author, name, publYear)$, for European books. Now, DB also contains the literal $Eurbook(\ kafka, metamorph, \mathit{1919})\}$. If in addition to the ICs we had before, we include in IC the set inclusion dependency $\forall xyz\ (Eurbook(x, y, z) \rightarrow Book(x, y, z))$, we obtain the following program $\Pi(DB, IC)$:

1. $EurBook(kafka, metamorph, \mathit{1919}, \mathbf{t_d}) \leftarrow$

$Book(kafka, metamorph, \mathit{1919}, \mathbf{t_d}) \leftarrow$

$Book(kafka, metamorph, \mathit{1915}, \mathbf{t_d}) \leftarrow$.

2. $Book(x, y, z, \mathbf{t}^\star) \leftarrow Book(x, y, z, \mathbf{t_d})$

$Book(x, y, z, \mathbf{t}^\star) \leftarrow Book(x, y, z, \mathbf{t_a})$

$Book(x, y, z, \mathbf{f}^\star) \leftarrow Book(x, y, z, \mathbf{f_a})$

$Eurbook(x, y, z, \mathbf{t}^\star) \leftarrow Eurbook(x, y, z, \mathbf{t_d})$

$Eurbook(x, y, z, \mathbf{t}^\star) \leftarrow Eurbook(x, y, z, \mathbf{t_a})$

$Eurbook(x, y, z, \mathbf{f}^\star) \leftarrow Eurbook(x, y, z, \mathbf{f_a})$.

3. $Book(x, y, z, \mathbf{f_a}) \vee Book(x, y, w, \mathbf{f_a}) \ \leftarrow$
$\quad Book(x, y, z, \mathbf{t}^\star) \wedge Book(x, y, w, \mathbf{t}^\star) \wedge z \neq w$

$Eurbook(x, y, z, \mathbf{f_a}) \vee Book(x, y, z, \mathbf{t_a}) \ \leftarrow$
$\quad Eurbook(x, y, z, \mathbf{t}^\star) \wedge Book(x, y, z, \mathbf{f}^\star)$. □

In order to have a semantics for our repair programs, we define their models. Since the negative information in a database instance is only implicitly available and we want to avoid explicitly representing it, we need to specify when negative information of the form $p(\bar{t}, \mathbf{f}^\star)$ is true of a model.

Definition 4. (a) Let I be an Herbrand interpretation for Σ^{ext} and φ a FO formula in Σ^{ext}. The definition of $\star$-satisfaction of φ by I, denoted $I \models_\star \varphi$, is as usual, except that for a ground atomic formula $p(\bar{a}, \mathbf{f}^\star)$ it holds: $I \models_\star p(\bar{a}, \mathbf{f}^\star)$ iff $p(\bar{a}, \mathbf{f}^\star) \in I$ or $p(\bar{a}, \mathbf{t_d}) \notin I$.
(b) An Herbrand interpretation $\mathcal{M}$ is a $\star$-*model* of $\Pi(DB, IC)$ if for every (ground instantiation of a) clause $(\bigvee_{i=1}^n a_i \leftarrow \bigwedge_{j=1}^m b_j) \in \Pi(DB, IC)$, $\mathcal{M} \not\models_\star \bigwedge_{j=1}^m b_j$ or $\mathcal{M} \models_\star \bigvee_{i=1}^n a_i$. □

Definition 5. (a) An atom $p(\bar{a})$ in a model $\mathcal{M}$ of a program is *plausible* if it belongs to the head of a clause in $\Pi(DB, IC)$ such that $\mathcal{M}$ $\star$-satisfies the body of the clause. A model of a program is plausible if every atom in it is plausible.
(b) A model is *coherent* if it does not contain both $p(\bar{a}, \mathbf{t_a})$ and $p(\bar{a}, \mathbf{f_a})$. □

[2] In [Arenas et al. 2000b] a direct specification of database repairs by means of disjunctive logic programs with a stable model semantics was presented. Those programs contained both repair triggering rules and "stabilizing" rules.

We will be interested only in the Herbrand $\star$-models of the program that are minimal wrt set inclusion[3] and plausible and *coherent*. Notice that in a coherent model we may still find both atoms $p(\bar{a}, \mathbf{t}^\star)$ and $p(\bar{a}, \mathbf{f}^\star)$. Notice also that a plausible atom may belong to a supported disjunctive head [Lobo 1998], without the other disjuncts being forced to be false.

It is possible to prove that every minimal, plausible and coherent $\star$-model of $\Pi(DB, IC)$ is a model of $\Pi(DB, IC)$ (in the usual sense). Furthermore, it is easy to see from the definition of a $\star$-model of a program that we could keep the classical notion of satisfaction by including in the program the additional clauses $p(\bar{x}, \mathbf{f}^\star) \leftarrow not\ p(\bar{x}, \mathbf{t_d})$, that would include in the models all the negative information we usually keep implicit via the closed world assumption. Moreover, we would be left with a normal disjunctive program, for which a stable model semantics could be used [Gelfond et al. 1988] (see section 5.3 below).

Example 8. (example 7 cont.) The coherent plausible minimal $\star$-models of the program presented in example 7 are:

$$\begin{aligned}
\mathcal{M}_1 = \{&Eurbook(kafka, metamorph, \mathit{1919}, \mathbf{t_d}),\\
&Eurbook(kafka, metamorph, \mathit{1919}, \mathbf{t}^\star),\\
&Book(kafka, metamorph, \mathit{1919}, \mathbf{t_d}),\\
&Book(kafka, metamorph, \mathit{1919}, \mathbf{t}^\star),\\
&Book(kafka, metamorph, \mathit{1915}, \mathbf{t_d}),\\
&Book(kafka, metamorph, \mathit{1915}, \mathbf{t}^\star),\\
&Book(kafka, metamorph, \mathit{1915}, \mathbf{f_a}),\\
&Book(kafka, metamorph, \mathit{1915}, \mathbf{f}^\star)\}.
\end{aligned}$$

$$\begin{aligned}
\mathcal{M}_2 = \{&Eurbook(kafka, metamorph, \mathit{1919}, \mathbf{t_d}),\\
&Eurbook(kafka, metamorph, \mathit{1919}, \mathbf{t}^\star),\\
&Eurbook(kafka, metamorph, \mathit{1919}, \mathbf{f_a}),\\
&Eurbook(kafka, metamorph, \mathit{1919}, \mathbf{f}^\star),\\
&Book(kafka, metamorph, \mathit{1919}, \mathbf{t_d}),\\
&Book(kafka, metamorph, \mathit{1919}, \mathbf{t}^\star),\\
&Book(kafka, metamorph, \mathit{1919}, \mathbf{f_a}),\\
&Book(kafka, metamorph, \mathit{1919}, \mathbf{f}^\star),\\
&Book(kafka, metamorph, \mathit{1915}, \mathbf{t_d}),\\
&Book(kafka, metamorph, \mathit{1915}, \mathbf{t}^\star)\}. \quad □
\end{aligned}$$

Notice that, in contrast to the minimal models of the annotated theory $\mathcal{T}(DB, IC)$, the $\star$-models of the program will include the database contents with its original annotations ($\mathbf{t_d}$). Every time there is an atom in a model annotated with $\mathbf{t_d}$ or $\mathbf{t_a}$, it will appear annotated with $\mathbf{t}^\star$. From these models we should be able to "read" database repairs. Every $\star$-model of the logic program has to be interpreted.

Definition 6. Given a coherent plausible $\star$-model $\mathcal{M}$ of $\Pi(DB, IC)$, its *interpretation*, $i(\mathcal{M})$, is a new Herbrand interpretation obtained from $\mathcal{M}$ as follows:

1. If $p(\bar{a}, \mathbf{f_a})$ belongs to $\mathcal{M}$, then $p(\bar{a}, \mathbf{f}^{\star\star})$ belongs to $i(\mathcal{M})$.

2. If neither $p(\bar{a}, \mathbf{t_d})$ nor $p(\bar{a}, \mathbf{t_a})$ belongs to $\mathcal{M}$, then $p(\bar{a}, \mathbf{f}^{\star\star})$ belongs to $i(\mathcal{M})$.

3. If $p(\bar{a}, \mathbf{t_d})$ belongs to $\mathcal{M}$ and $p(\bar{a}, \mathbf{f_a})$ does not belong to $\mathcal{M}$, then $p(\bar{a}, \mathbf{t}^{\star\star})$ belongs to $i(\mathcal{M})$.

4. If $p(\bar{a}, \mathbf{t_a})$ belongs to $\mathcal{M}$, then $p(\bar{a}, \mathbf{t}^{\star\star})$ belongs to $i(\mathcal{M})$. □

Notice that the interpreted models contain two new annotations, $\mathbf{t}^{\star\star}, \mathbf{f}^{\star\star}$, in the last arguments. The first one groups together those atoms annotated either with $\mathbf{t_a}$ or with $\mathbf{t_d}$ but not $\mathbf{f_a}$. Intuitively, the latter correspond to those annotated with $\mathbf{t}$ in the models of $\mathcal{T}(DB, IC)$. A similar role plays the other new annotation wrt the "false" annotations. These new annotations will simplify the expression of the queries to be posed to the program (see section 5.2). Without them, instead of simply asking $p(\bar{x}, \mathbf{t}^{\star\star})$ (for the tuples in a repair), we would have to ask for $p(\bar{x}, \mathbf{t_a}) \vee (p(\bar{x}, \mathbf{t_d}) \wedge \neg p(\bar{x}, \mathbf{f_a}))$.

Example 9. (example 8 cont.) The interpreted models are:

$$\begin{aligned}
i(\mathcal{M}_1) = \{&Eurbook(kafka, metamorph, \mathit{1919}, \mathbf{t}^{\star\star}),\\
&Book(kafka, metamorph, \mathit{1919}, \mathbf{t}^{\star\star}),\\
&Book(kafka, metamorph, \mathit{1915}, \mathbf{f}^{\star\star})\}
\end{aligned}$$

$$\begin{aligned}
i(\mathcal{M}_2) = \{&Eurbook(kafka, metamorph, \mathit{1919}, \mathbf{f}^{\star\star}),\\
&Book(kafka, metamorph, \mathit{1919}, \mathbf{f}^{\star\star}),\\
&Book(kafka, metamorph, \mathit{1915}, \mathbf{t}^{\star\star})\}. \quad □
\end{aligned}$$

The interpreted models could be easily obtained by adding new rules to the program $\Pi(DB, IC)$. This will be shown in section 5.2. From an interpreted model of the program we can obtain a database instance:

Definition 7.

$$DB^{\Pi}_{i(\mathcal{M})} \ = \ \{p(\bar{a}) \mid i(\mathcal{M}) \models p(\bar{a}, \mathbf{t}^{\star\star})\}. \quad □$$

Example 10. (example 9 cont.) The following database instances obtained from definition 7 are the repairs of DB:

[3] To distinguish them from the Δ-minimal model of the annotated theory.

$$\begin{aligned}
DB^{\Pi}_{i(\mathcal{M}_1)} = \{&Eurbook(kafka, metamorph, \mathit{1919}),\\
&Book(kafka, metamorph, \mathit{1919})\},\\
DB^{\Pi}_{i(\mathcal{M}_2)} = \{&Book(kafka, metamorph, \mathit{1915})\}. \quad □
\end{aligned}$$

Theorem 1. If $\mathcal{M}$ is a coherent minimal plausible $\star$-model of $\Pi(DB, IC)$, and $DB^{\Pi}_{i(\mathcal{M})}$ is finite, then $DB^{\Pi}_{i(\mathcal{M})}$ is a repair of DB with respect to IC. Furthermore, the repairs obtained in this way are all the repairs of DB. □

5.2 The query program

Given a first order query Q, we want the consistent answers from DB. In consequence, we need those atoms that are simultaneously true in every interpreted coherent minimal plausible $\star$-model of the program $\Pi(DB, IC)$. They are obtained through the query $Q^{\star\star}$, obtained from Q by replacing, for $p \in P$, every positive literal $p(\bar{s})$ by $p(\bar{s}, \mathbf{t}^{\star\star})$ and every negative literal $\neg p(\bar{s})$ by $p(\bar{s}, \mathbf{f}^{\star\star})$. This query corresponds to the annotated version Q^{an} of Q (see section 4). Now $Q^{\star\star}$ can be transformed into a query program $\Pi(Q^{\star\star})$ by a standard transformation [Lloyd 1987, Abiteboul et al. 1995]. This query program will be run in combination with a program, Π^{int}, that specifies the interpreted models. This program can be obtained extending $\Pi(DB, IC)$ with the following rules:

$p(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow p(\bar{x}, \mathbf{t_a})$,
$p(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow p(\bar{x}, \mathbf{t_d}),\ not\ p(\bar{x}, \mathbf{f_a})$,
$p(\bar{x}, \mathbf{f}^{\star\star}) \leftarrow p(\bar{x}, \mathbf{f_a})$,
$p(\bar{x}, \mathbf{f}^{\star\star}) \leftarrow\ not\ p(\bar{x}, \mathbf{t_d}),\ not\ p(\bar{x}, \mathbf{t_a})$.

The extended program, that now contains weak negation, is basically stratified due to the different ground annotations in the predicates. Thus, its models can be computed by extending the models of the original program in a uniform manner.

Example 11. (example 7 cont.) The program Π^{int} is obtained adding to the program $\Pi(DB, IC)$ of example 7 the following clauses:

$Eurbook(x, y, z, \mathbf{t}^{\star\star}) \leftarrow Eurbook(x, y, z, \mathbf{t_a})$

$Eurbook(x, y, z, \mathbf{f}^{\star\star}) \leftarrow Eurbook(x, y, z, \mathbf{f_a})$

$Eurbook(x, y, z, \mathbf{t}^{\star\star}) \leftarrow Eurbook(x, y, z, \mathbf{t_d}),$
$\quad not\ Eurbook(x, y, z, \mathbf{f_a})$

$Eurbook(x, y, z, \mathbf{f}^{\star\star}) \leftarrow\ not\ Eurbook(x, y, z, \mathbf{t_d}),$
$\quad not\ Eurbook(x, y, z, \mathbf{t_a})$

$Book(x, y, z, \mathbf{t}^{\star\star}) \leftarrow Book(x, y, z, \mathbf{t_a})$

$Book(x, y, z, \mathbf{f}^{\star\star}) \leftarrow Book(x, y, z, \mathbf{f_a})$

$Book(x, y, z, \mathbf{t}^{\star\star}) \leftarrow Book(x, y, z, \mathbf{t_d}),$
$\quad not\ Book(x, y, z, \mathbf{f_a})$

$Book(x, y, z, \mathbf{f}^{\star\star}) \leftarrow\ not\ Book(x, y, z, \mathbf{t_d}),$
$\quad not\ Book(x, y, z, \mathbf{t_a})$.

For the query $Q(y) : \exists z Book(kafka, y, z)$, we generate $Q^{\star\star}(y) : \exists z Book(kafka, y,\ z, \mathbf{t}^{\star\star})$, that is transformed into the query program $\Pi(Q^{\star\star})$: $Answer(y) \leftarrow Book(kafka, y, z, \mathbf{t}^{\star\star})$.

The coherent minimal plausible $\star$-models of $\Pi^{int} \cup \Pi(Q^\star)$ are $\mathfrak{M}_1 = \mathcal{M}_1 \cup i(\mathcal{M}_1) \cup \{Answer(metamorph)\}$ and $\mathfrak{M}_2 = \mathcal{M}_2 \cup i(\mathcal{M}_2) \cup \{Answer(metamorph)\}$, where $\mathcal{M}_1, \mathcal{M}_2$ and $i(\mathcal{M}_1), i(\mathcal{M}_2)$ are given in examples 8 and 9, resp. We can see that $y = metamorph$ is a consistent answer. □

5.3 Computing from the program

The repair programs $\Pi(DB, IC)$ introduced in section 5.1 are based on a non classical notion of satisfaction (definition 4). In order to compute from the program using a stable model semantics for disjunctive programs, we build a new program $\Pi^\neg(DB, IC)$, obtained from the original one by adding the clause $p(\bar{x}, \mathbf{f}^\star) \leftarrow\ not\ p(\bar{x}, \mathbf{t_d})$ that gives an account of the closed world assumption. It holds:

Proposition 3. If $\mathcal{M}$ is a coherent stable model of $\Pi^\neg(DB, IC)$, then $DB^{\Pi}_{\mathcal{M}}$ is a repair of DB with respect to IC; and every repair can be obtained in this way. □

In consequence, the database repairs can be specified using disjunctive logic programs with a stable model semantics [Gelfond et al. 1988, Gelfond et al. 1991], and an implementation of this semantics, like DLV [Eiter et al. 2000], can be used to compute both repairs and consistent answers. Notice that DLV implements denial constraints [Buccafurri et al. 2000], which can be used to keep the coherent stable models only, by pruning those models that do not satisfy $\leftarrow p(\bar{x}, \mathbf{t_a}), p(\bar{x}, \mathbf{f_a})$.

Example 12. Consider the database instance $\{p(a)\}$ that is inconsistent wrt the set inclusion dependency $\forall x\ (p(x) \rightarrow q(x))$. The program $\Pi^\neg(DB, IC)$ contains the following clauses:

1. The following rules do not depend on ICs

   $p(x, \mathbf{f}^\star) \leftarrow p(x, \mathbf{f_a})$

   $p(x, \mathbf{t}^\star) \leftarrow p(x, \mathbf{t_a})$

$p(x, \mathbf{t}^\star) \leftarrow p(x, \mathbf{t_d})$

$q(x, \mathbf{f}^\star) \leftarrow q(x, \mathbf{f_a})$

$q(x, \mathbf{t}^\star) \leftarrow q(x, \mathbf{t_a})$

$q(x, \mathbf{t}^\star) \leftarrow q(x, \mathbf{t_d})$

2. A single rule capturing the IC

   $p(x, \mathbf{f_a}) \vee q(x, \mathbf{t_a}) \leftarrow p(x, \mathbf{t}^\star), q(x, \mathbf{f}^\star).$

3. Database contents

   $p(a, \mathbf{t_d}) \leftarrow$

4. The new rules for the closed world assumption

   $p(x, \mathbf{f}^\star) \leftarrow \; \mathit{not}\ p(x, \mathbf{t_d})$

   $q(x, \mathbf{f}^\star) \leftarrow \; \mathit{not}\ q(x, \mathbf{t_d}).$

5. Denial constraints for coherence

   $\leftarrow p(\bar{x}, \mathbf{t_a}), p(\bar{x}, \mathbf{f_a})$

   $\leftarrow q(\bar{x}, \mathbf{t_a}), q(\bar{x}, \mathbf{f_a}).$

6. Rules for interpreting the models

   $p(x, \mathbf{t}^{\star\star}) \leftarrow p(x, \mathbf{t_a})$

   $p(x, \mathbf{t}^{\star\star}) \leftarrow p(x, \mathbf{t_d}),\ \mathit{not}\ p(x, \mathbf{f_a})$

   $p(x, \mathbf{f}^{\star\star}) \leftarrow p(x, \mathbf{f_a})$

   $p(x, \mathbf{f}^{\star\star}) \leftarrow \; \mathit{not}\ p(x, \mathbf{t_d}),\ \mathit{not}\ p(x, \mathbf{t_a})$

   $q(x, \mathbf{t}^{\star\star}) \leftarrow q(x, \mathbf{t_a})$

   $q(x, \mathbf{t}^{\star\star}) \leftarrow q(x, \mathbf{t_d}),\ \mathit{not}\ q(x, \mathbf{f_a})$

   $q(x, \mathbf{f}^{\star\star}) \leftarrow q(x, \mathbf{f_a})$

   $q(x, \mathbf{f}^{\star\star}) \leftarrow \; \mathit{not}\ q(x, \mathbf{t_d}),\ \mathit{not}\ q(x, \mathbf{t_a}).$ □

It can be seen that the programs with annotations we have obtained are very simple in terms of their dependency on the ICs.

As mentioned before, consistent answers can be obtained "running" the query program introduced in section 5.2 in combination with the repair program $\Pi^\neg(DB, IC)$, under the skeptical stable model semantics, that sanctions as true what is true of all stable models.

6 Conclusions

Extending work presented in [Arenas et al. 2000a], we have shown how to annotate referential ICs in order to obtain a specification in annotated predicate logic of the class of repairs of a relational database. The correspondence between the minimal models of the annotated first order specification and the database repairs is established. Ongoing work considers the extension of the annotated embedding methodology to the class of all ICs found in DB praxis [Abiteboul et al. 1995, chap. 10].

We formulated the problem of consistent query answering as a problem of non monotonic entailment of a modified query from the annotated theory.

We have presented a general treatment of consistent query answering for first order queries and universal ICs. This is done by means of disjunctive logic programs with a stable model semantics, where annotations are now arguments, that specify the database repairs in the case of universal ICs. In consequence, consistent query answers can be obtained by "running" the program. Ongoing work considers the extension of the logic programs to include referential (and more general) ICs.

In [Greco et al. 2001], a general methodology for specifying database repairs wrt universal ICs is presented. There, disjunctive logic programs with stable model semantics are used. They also consider the problem of specifying preferences between possible repairs. Independently, [Arenas et al. 2000b] also presents a specification of database repairs by means of disjunctive logic programs with a stable model semantics. The programs capture the repairs for binary universal ICs. The programs presented here also work for the whole class of universal ICs, but they are much simpler than those presented in [Greco et al. 2001, Arenas et al. 2000b]; this is due to the simplicity of the stabilizing rules, that take full advantage of the relationship between the annotations. The simplicity is expressed in a much smaller number of rules and the syntactic properties of the programs.

In [Blair et al. 1989, Kifer et al. 1992b, Leach et al. 1996] paraconsistent and annotated logic programs are introduced. In particular, in [Subrahmanian 1994] those programs are used to integrate databases, a problem closely related to inconsistency handling. It is not clear how to use those definitive non disjunctive programs to capture database repairs. Furthermore, notice that the programs presented in this paper have a completely classical semantics.

In [Gaasterland 1994], annotated logic (programs) were used to specify and obtain answers matching user needs and preferences. Deeper relationships to our work deserve to be explored.

Acknowledgments: Work supported by DIPUC, FONDECYT Grant 1000593, and Carleton University Start-Up Grant 9364-01. We are grateful to Alberto Mendelzon and Marcelo Arenas for stimulating and useful conversations, and comments.

References

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

[Arenas et al. 1999] Arenas, M.; Bertossi, L. and Chomicki, J. "Consistent Query Answers in Inconsistent Databases". In *Proc. ACM Symposium on Principles of Database Systems (ACM PODS'99), Philadelphia)*, pages 68-79, 1999.

[Arenas et al. 2000a] Arenas, M.; Bertossi, L. and Kifer, M. "Applications of Annotated Predicate Calculus to Querying Inconsistent Databases". In *'Computational Logic - CL2000' Stream: 6th International Conference on Rules and Objects in Databases (DOOD'2000)*. Springer Lecture Notes in Artificial Intelligence 1861, pages 926-941.

[Arenas et al. 2000b] 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.L. Larsen, J. Kacprzyk, S. Zadrozny, H. Christiansen (eds.), Springer, 2000, pp. 27–41.

[Blair et al. 1989] Blair, H.A. and Subrahmanian, V.S. "Paraconsistent Logic Programming". *Theoretical Computer Science,* 68:135-154, 1989.

[Buccafurri et al. 2000] Buccafurri, F.; Leone, N. and Rullo, P. "Enhancing Disjunctive Datalog by Constraints". *IEEE Transactions on Knowledge and Data Engineering*, 2000, 12(5) : 845-860.

[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.

[Gaasterland 1994] Gaasterland, T. and Lobo, J. "Qualified Answers That Reflect User Needs and Preferences". In Proc. 20th International Conference of Very Large Databases (VLDB'94). Morgan Kaufmann Publishers, 1994, pages 309–320.

[Gelfond et al. 1988] Gelfond, M. and Lifschitz, V. "The Stable Model Semantics for Logic Programming". In *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, R. A. Kowalski and K. A. Bowen (eds.), MIT Press 1988, pp. 1070–1080.

[Gelfond et al. 1991] Gelfond, M. and Lifschitz, V. "Classical Negation in Logic Programs and Disjunctive Databases". *New Generation Computing*, 1991, 9:365–385.

[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. 17th International Conference on Logic Programming, ICLP'01, Ph. Codognet (ed.), LNCS 2237, Springer, 2001, pp. 348–364.

[Kifer et al. 1992a] Kifer, M. and Lozinskii, E.L. "A Logic for Reasoning with Inconsistency". *Journal of Automated reasoning*, 9(2):179-215, November 1992.

[Kifer et al. 1992b] Kifer, M. and Subrahmanian, V.S. "Theory of Generalized Annotated Logic Programming and its Applications". *Journal of Logic Programming*, 12(4):335-368, April 1992.

[Leach et al. 1996] Leach, S.M. and Lu, J.J. "Query Processing in Annotated Logic Programming: Theory and Implementation". *Journal of Intelligent Information Systems*, 6, January 1996, pp. 33–58.

[Lloyd 1987] Lloyd, J.W. "Foundations of Logic Programming". Springer Verlag, 1987.

[Lobo 1998] Lobo, J.; Minker, J. and Rajasekar, A. "Semantics for Disjunctive and Normal Disjunctive Logic Programs". In *Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 5*, D. Gabbay et al. (eds.). Oxford University Press, 1998.

[Subrahmanian 1994] Subrahmanian, V.S. "Amalgamating Knowledge Bases". *ACM Transactions on Database Systems*, 1994, 19(2):291–331.

Database Repairs and Analytic Tableaux

Leopoldo Bertossi (bertossi@scs.carleton.ca)
Carleton University
School of Computer Science
Ottawa, Canada K1S 5B6

Camilla Schwind (schwind@map.archi.fr)
MAP-CNRS, Ecole d'architecture de Marseille
183, Avenue de Luminy,
13288 Marseille, Cedex 9, France

Abstract. In this article, we characterize in terms of analytic tableaux the repairs of inconsistent relational databases, that is databases that do not satisfy a given set of integrity constraints. For this purpose we provide closing and opening criteria for branches in tableaux that are built for database instances and their integrity constraints. We use the tableaux based characterization as a basis for consistent query answering, that is for retrieving from the database answers to queries that are consistent wrt the integrity constraints.

Keywords: Databases, Queries, Integrity Constraints, Analytic Tableaux

1. Introduction

The notion of consistent answer to a query posed to an inconsistent database was defined in (Arenas et al., 1999): A tuple is a consistent answer if it is an answer, in the usual sense, in every possible repair of the inconsistent database. A repair is a new database instance that satisfies the integrity constraints and differs from the original instance by a minimal set of changes wrt set inclusion.

A computational methodology to obtain such consistent answers was also presented in (Arenas et al., 1999). Nevertheless, it has some limitations in terms of the syntactical form of integrity constraints and queries it can handle. In particular, it does not cover the case of existential queries and constraints.

In classical logic, analytic tableaux (Beth, 1959) are used as a formal deductive system for propositional and predicate logic. Similar in spirit to resolution, but with some important methodological and practical differences (Fitting, 1988), they are mainly used for producing formal refutations from a contradictory set of formulas. Starting from a set of formulas, the system produces a tree with formulas in its nodes. The set of formulas is inconsistent whenever all the branches in the tableau can be closed. A branch closes when it contains a formula and its negation.

In this paper we extend the tableaux methodology to deal with a relational database instance plus a set of integrity constraints that the first fails to satisfy. Consequently, both inputs together can be considered as building an inconsistent set of sentences. In this situation, we give criteria for closing branches in a tableau for a relational database instance.

The technique of "opening tableaux" was introduced in (Lafon et al., 1988) for a solution to the frame problem, and in (Schwind, 1990; Schwind et al., 1994) for applying tableaux methods to default logic. In this paper we show how to open tableaux for database instances plus their constraints, and this notion of opening is applied to characterize and represent by means of a tree structure all the repairs of the original database. Finally, we sketch how this representation could be used to retrieve consistent query answers. At least at the theoretical level, the methodology introduced in this paper could be applied to any kind of first order (FO) queries and constraints.

This paper is organized as follows. In section 2, we define our notion of repair of an inconsistent database. Section 3 recalls the definition of analytic tableaux and shows how databases and their repairs can be characterized as openings of closed tableaux. In section 4 we show the relationship between consistent query answering and Winslett's approach to knowledge base update; this allows us to obtain some complexity results for our methodology. Section 5 shows how consistent answers to queries posed to an inconsistent database can be obtained using the analytic tableaux. In section 6 we show the relationship of consistent query answering with minimal entailment, more specifically, in section 6.1, with circumscriptive reasoning. This yields a method for implementing the approach, which is studied in section 6.2. Preliminary versions of this paper appeared in (Bertossi et al., 2001; Bertossi et al., 2002).

2. Inconsistent Databases and Repairs

In this paper a database instance is given by a finite set of finite relations on a database schema. A database schema can be represented in logic by a typed first-order language, $\mathcal{L}$, containing a finite set of sorted database predicates and a fixed infinite set of constants D. The language contains a predicate for each database relation and the constants in D correspond to the elements in the database domain, that will be also denoted by D. That is every database instance has an infinite domain D. We also have a set of integrity constraints IC expressed in language $\mathcal{L}$. These are first-order formulas which the database instances

are expected to satisfy. In spite of this, there are realistic situations where a database may not satisfy its integrity constraints (Arenas et al., 1999). If a database instance satisfies IC, we say that it is consistent (wrt IC), otherwise we say it is inconsistent. In any case, we will assume from now on that IC is a consistent set of first order sentences.

A database instance r can be represented by a finite set of ground atoms in the database language, or alternatively, as a Herbrand structure over this language, with Herbrand domain D (Lloyd, 1987). In consequence, we can say that a database instance r is consistent, wrt IC, when its corresponding Herbrand structure is a model of IC, and we write $r \models IC$.

The active domain of a database instance r is the set of those elements of D that explicitly appear (in the extensions of the database predicates) in r. The active domain is always finite and we denote it by $Act(r)$. We may also have a set of built-in (or evaluable) predicates, like equality, arithmetical relations, etc. In this case, we have the language $\mathcal{L}$ possibly extended with these predicates. In all database instances each of these predicates has a fixed and possibly infinite extension. Of course, since we defined database instances as finite sets of ground atoms, we are not considering these built-in atoms as members of database instances.

In database applications, it is usually the case that an inconsistent database[1] has "most" of its data contents still consistent wrt IC and can still provide "consistent answers" to queries posed to it. The notion of consistent answer was defined and analyzed in (Arenas et al., 1999). This was done on the basis of considering all possible changes to r, in such a way that it becomes a consistent database instance. A consistent answer is an answer that can be retrieved from all those repairs that differ from the original instance in a minimal way.

The notion of minimal change, defined in (Arenas et al., 1999), is based on the notion of minimal distance between models using symmetric set difference Δ of sets of database tuples.

Definition 1. *(*Arenas et al., 1999*) Given databases instances*[2] *r, r' and r'', we say that r' is closer to r than r'' iff $r\Delta r' \subseteq r\Delta r''$. This is denoted by $r' \leq_r r''$.* □

It is easy to see that $\leq_r$ is an order relation. Only database predicates are taken into account for the notion of distance. This is because built-in predicates are not subject to change; and then they have the same

[1] Sometimes we will simply say "database" instead of "database instance".

[2] We are assuming here and everywhere in the paper that all database instances have the same predicates and domain.

extension in all database instances. Now we can define the "repairs" of an inconsistent database instance.

Definition 2. *(*Arenas et al., 1999*)*
(a) Given database instances r and r', r' is a repair *of r, if $r' \models IC$ and r' is a minimal element in the set of instances wrt the order $\leq_r$.*
(b) Given a database instance r, a set IC and a first order query $Q(\bar{x})$, we say that a ground tuple $\bar{t}$ is a consistent answer to Q in r wrt IC iff $r' \models Q[\bar{t}]$ for every repair r' of r (wrt IC). □

Example 1. Consider the integrity constraint

$$IC: \ \forall x, y, z(Supply(x,y,z) \ \wedge \ Class(z, T_4) \ \rightarrow \ x = C),$$

stating that C is the only provider of items of class T_4; and the inconsistent database $r = \{Supply(C, D_1, It_1), Supply(D, D_2, It_2), \ Class(It_1, T_4), Class(It_2, T_4)\}$. We have only two possible (minimal) repairs of the original database instance, namely $r_1 = \{Supply(C, D_1, It_1), Class(It_1, T_4), Class(It_2, T_4)\}$ and $r_2 = \{Supply(C, D_1, It_1), Supply(D, D_2, It_2), Class(It_1, T_4)\}$.

Given the query $Q(x,y,z): \ Supply(x,y,z)?$, the tuple (C, D_1, It_1) is a consistent answer because it can be obtained from every repair, but (D, D_2, It_2) is not, because it cannot be retrieved from r_1. □

It is possible to prove (Arenas et al., 1999) that for every database instance r and set IC of integrity constraints, there is always a repair r'. If r is already consistent, then r is the only repair. The following lemma, which is easy to prove, will be useful.

Lemma 1.

1. *If $r' \leq_r r''$, then $r \cap r'' \subseteq r \cap r'$.*
2. *If $r' \subseteq r$, then $r \Delta r' = r \setminus r'$.* □

We have given a semantic definition of consistent answer to a query in an inconsistent database. We would like to compute consistent answers, but **not** via computing all possible repairs and checking answers in common in **all** of them. Actually there may be an exponential number of repairs in the size of the database (Arenas et al., 2001).

In (Arenas et al., 1999; Celle et al., 2000) a mechanism for computing and checking consistent query answers was considered. It does not produce/use the repairs, but it queries the only explicitly available inconsistent database instance. Given a FO query Q, to obtain the consistent answers wrt a finite set of FO ICs IC, Q is qualified with appropriate information derived from the interaction between Q and

IC. More precisely, if we want the consistent answers to $Q(\bar{x})$ in r, the query is rewritten into a new query $\mathcal{T}(Q(\bar{x}))$; and then the (ordinary) answers to $\mathcal{T}(Q(\bar{x}))$ are retrieved from r.

Example 2. (example 1 continued) Consider the query Q : $Supply(x, y, z)$? about the items supplied together with their associated information. In order to obtain the consistent answers, the query $\mathcal{T}(Q)$: $Supply(x, y, z) \wedge (Class(z, T_4) \rightarrow x = C)$ is generated and posed to the original database. The extra conjunct in it is the "residue" obtained from the interaction between the query and the constraint. Residues can be obtained automatically (Arenas et al., 1999). □

In general, $\mathcal{T}$ is an iterative operator. There are sufficient conditions on queries and ICs for soundness, completeness and termination of operator $\mathcal{T}$; and natural and useful syntactical classes satisfy those conditions. There are some limitations though: $\mathcal{T}$ can not be applied to existential queries like $Q(X) : \exists Y\ Supplies(X, Y, It_1)$?. However, this query does have consistent answers at the semantic level. Furthermore, the methodology presented in (Arenas et al., 1999) assumes that the ICs are (universal) constraints written in clausal form.

There are fundamental reasons for the limitations of the query rewriting approach. If a FO query can be always rewritten into a new FO query, then the problem of *consistent query answering* (CQA) would have polynomial time data complexity. However, CQA is likely to have a higher computational complexity (see sections 4 and 6.1 for a discussion).

Notice that $\mathcal{T}$ is based on the interaction between the queries and the ICs. It does not consider the interaction between the ICs and the database instance. In this paper we concentrate mostly on this second form of interaction. In particular, we wonder if we can obtain an implicit and compact representation of the database repairs.

Furthermore, the database, seen as a set of logical formulas, plus IC is an inconsistent first order theory; and we know that such an inconsistency can be detected and represented by means of an analytic tableau.

An analytic tableau is a syntactically generated tree-like structure that, starting from a set of formulas placed at the root, has all its branches "closed" when the initial set of formulas is inconsistent. This tableaux can show us how to repair inconsistencies, because closed branches can be opened by removing literals.

In the next sections, we show how to generate, close and open tableaux for database instances with their constraints; and we apply the notion of opening to characterize and represent by means of a tree

structure all the repairs of the original database. We also sketch how this representation could be used to retrieve consistent query answers. At least at the theoretical level, the methodology introduced here could be applied to any kind of first order queries and constraints.

3. Database Repairs and Analytic Tableaux

In order to use analytic tableaux to represent database repairs and characterize consistent query answers, we need a special form of tableaux, suitable for representing database instances and their integrity constraints.

Given a database instance r and a finite set of integrity constraints IC, we first compute the tableau, $TP(IC \cup r)$, for IC and r. This tableau has as root node the set of formulas $IC \cup r$. This tableau should be closed, that is the tableau has only closed branches, if and only if r is inconsistent. By removing database literals in every closed branch we can transform r into a consistent database instance and thus obtain a repair of the database. For all this to work, we must take into account, when computing the tableau, that r represents a database instance and not just a set of formulas, in particular, that the absence of positive information means negative information, etc. (see section 3.2). First, we give a brief review of classical first order analytic tableaux (Beth, 1959; Smullyan, 1968; Fitting, 1996).

3.1. Analytic tableaux

The tableau of a set of formulas is obtained by recursively *breaking down* the formulas into subformulas, obtaining sets of sets of formulas. These are the usual Smullyan's classes of formulas:

α	$\alpha_1 \quad \alpha_2$	β	$\beta_1 \quad \beta_2$
$f \wedge g$	$f \quad g$	$f \vee g$	$f \quad g$
$\neg(f \vee g)$	$\neg f \quad \neg g$	$\neg(f \wedge g)$	$\neg f \quad \neg g$
$\neg(f \rightarrow g)$	$f \quad \neg g$	$f \rightarrow g$	$\neg f \quad g$

γ	$\gamma(p), p$ any constant	δ	$\delta(p), p$ a fresh constant
$(\forall x)f$	$f[x/p]$	$(\exists x)f$	$f[x/p]$
$\neg(\exists x)f$	$\neg f[x/p]$	$\neg(\forall x)f$	$\neg f[x/p]$

A tableaux prover produces a formula tree. An α-rule adds new formulas to branches, a β-rule splits the tableau and adds a new branch. Given a formula φ, we denote by $TP(\varphi)$ the tree produced by the

tableaux system. We can think of this tree as the set of its branches, that we usually denote with $X, Y, \ldots$.

Notice that the original set of constants in the language, in our case, D, is extended with a set of new constants, P, the so-called Skolem functions or parameters. These parameters, that we will denote by $p, p_1, \ldots$, have to be new at the point of their introduction in the tree, in the sense that they have not appeared so far in the (same branch of the) tableau. When applying the γ-rule, the parameter can be any of the old or new constants.

A tableau branch is *closed* if it contains a formula and its negation, otherwise it is *open*. Every open branch corresponds to a model of the formula: If a branch $B \in TP(\varphi)$ is open and finished, then the set of ground atoms on B is a model of φ. If the set of initial formulas is inconsistent, it does not have models, and then all branches (and thus the tableau) have to be closed. Actually, the completeness theorem for tableaux theorem proving (Smullyan, 1968) states that: F is a theorem iff $TP(\{\neg F\})$ is closed.

The intuitive idea of finished branch, of one to which no tableaux rule can be applied obtaining something new and relevant, is captured by means of the notion of *saturated branch*: this is a branch where all possible rules have been applied.

Definition 3. *A branch B is* saturated *iff it satisfies*

1. *If $\neg\neg\varphi \in B$, then $\varphi \in B$*
2. *If $(\varphi \vee \psi) \in B$, then $\varphi \in B$ or $\psi \in B$*
3. *If $(\varphi \wedge \psi) \in B$, then $\varphi \in B$ and $\psi \in B$*
4. *If $\exists x\varphi \in B$, then $\varphi[c] \in B$ for some constant c*
5. *If $\forall x\varphi \in B$, then $\varphi[c] \in B$ for any constant c.*[3] □

A branch is called *Hintikka* if it is saturated and not closed (Fitting, 1996). It is easy to see that a saturated branch is Hintikka iff it does not contain any atomic formula A and its negation $\neg A$. From now on, tableaux branches will be assumed to be saturated. Nevertheless, sometimes we talk about branches even when they are only partially developed.

We consider TP not only as a theorem prover (or consistency checker) for formulae, but also as an application from (sets of) formulas to trees which has some useful properties. Thus, operations on tableaux can

[3] If the language had function symbols, we would replace constants by ground terms in this definition.

be defined on the basis of the logical connectives occurring inside the formulas involved.

Lemma 2. *Let φ and ψ be any formulae. Then TP has the following properties.*

1. *$TP(\{\varphi \vee \psi\}) = TP(\{\varphi\}) \cup TP(\{\psi\})$*
2. *$TP(\{\varphi \wedge \psi\}) = \{X \cup Y : X \in TP(\{\varphi\})$ and $Y \in TP(\{\psi\})\}$*
3. *If $B \in TP(\varphi \wedge \psi)$, then $B = B' \cup B''$ with $B' \in TP(\varphi)$ and $B'' \in TP(\psi)$.* □

Property 3. follows directly from properties 1. and 2. The properties in the lemma motivate the following definition.

Definition 4. *Given tableaux T and T', each of them identified with the set of its branches, the combined tableaux is $T \otimes T' = \{X \cup Y : X \in T$ and $Y \in T'\}$.* □

Remark 1. The properties in lemma 2 can be used to check whether a formula φ derives from a theory A. $A \models \varphi$ iff $(A \rightarrow \varphi)$ is a theorem, what will be proved if we derive a contradiction from assuming $\neg(A \rightarrow \varphi)$. Therefore we will have to compute $TP(\{\neg(A \rightarrow \varphi)\})$ and check for closure. Using the second property, we will check $TP(\{A\}) \otimes TP(\{\neg\varphi\})$ for closure, allowing us to compute $TP(A)$ only once for any number of requests. □

The following relationship between the open branches of the tableaux for a formula and its models has been shown, among others by (Bellleannee et al., 1995; Schwind et al., 1994).

Theorem 1. *Let $B \in TP(\{\varphi\})$ be an open branch of the tableau for φ. Then there is a model M of φ, which satisfies B, i.e. $B \subseteq M$. More precisely, there is Herbrand model of φ such that the ground atoms in B belong to M.* □

3.2. Representing database instances by tableaux

In database theory, we usually make the following assumptions[4]: (a) Unique Names Assumption (UNA): If a and b are different constants in D, then $a \neq b$ holds in r. (b) Closed World Assumption (CWA): If r is a database instance, then for any ground database atom $P(c)$,

[4] Actually, it is possible to make all these assumptions explicit and transform the database instance into a first-order theory (Reiter, 1984).

if $P(c) \notin r$, then $\neg P(c)$ holds for r, more precisely, implicitly $\neg P(c)$ belongs to r.

In consequence, if we see the relational database as the set of its explicit atoms plus its implicit negative atoms, we can always repair the database by removing ground database literals.

When computing a tableau for a database instance r, we do not add explicitly the formulas corresponding to the UNA and CWA, rather we keep them implicit, but taking them into account when computing the tableau. This means, for example, that the presence on a tableau branch of a formula $a = b$, for different constants a, b in D, closes the branch.

Given a database r and integrity constraints IC, we will generate the tableau $TP(IC \cup r)$. Notice that every branch B of this tableau will be of the form $I \cup r$, where $I \in TP(IC)$ (see lemma 2). I is the "IC-part" of the branch.

Notice also that a tableau for IC only will never be closed, because IC is consistent. The same happens with any tableau for r. Only the combination of r and IC may produce a closed tableau.

$TP(IC \cup r)$ is defined as in section 3.1, but we still have to define the closure conditions for tableaux associated to database instances. Before, we present some motivating examples.

Example 3. (example 1 continued) In this case, $TP(IC \cup r)$ is the tree in figure 1. The last branch is closed because $D = C$ is false in the database (alternatively, because $D \neq C$ is implicitly in the database). We can see that $TP(IC \cup r)$ is closed. r is inconsistent wrt IC. The nodes $(Supply(C, D_1, It_1) \wedge Class(It_1, T_4) \rightarrow C = C)$ and $(Supply(D, D_2, It_2) \wedge Class(It_2, T_4) \rightarrow D = C)$ are obtained by applying the γ-rule to $\forall x, y, z(Supply(x, y, z) \wedge Class(z, T_4) \rightarrow x = C)$. Application of the β-rule to $(Supply(D, D_2, It_2) \wedge Class(It_2, T_4) \rightarrow D = C)$ produces the same subtree for all three leaves: $\neg Supply(C, D_1, It_1)$, $\neg Class(It_1, T_4)$ and $C = C$. In the figure, we indicate this subtree by "...". We will see later (see section 3.3) that, in some cases, we can omit the development of subtrees that should develop under branches that are already closed. Here we can omit the explicit further development of the subtree from the first two leftmost branches, because these branches are already closed. □

In tableaux with equality, we need extra rules. We will assume that we can always introduce equalities of the form $t = t$, for a term t, and that we can replace a term t in a predicate P by t' whenever $t = t'$ belongs to the same tableau branch (that is, we use a form of paramodulation (Fitting, 1996)). It will be simpler to define the closure

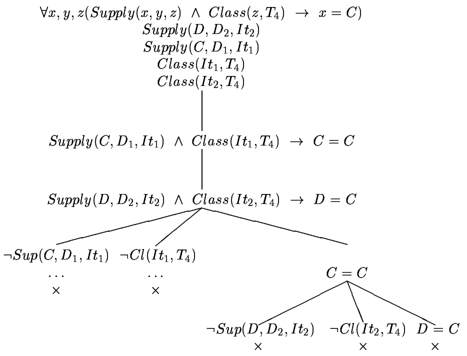

$\forall x, y, z(Supply(x, y, z) \wedge Class(z, T_4) \rightarrow x = C)$
$Supply(D, D_2, It_2)$
$Supply(C, D_1, It_1)$
$Class(It_1, T_4)$
$Class(It_2, T_4)$
$Supply(C, D_1, It_1) \wedge Class(It_1, T_4) \rightarrow C = C$
$Supply(D, D_2, It_2) \wedge Class(It_2, T_4) \rightarrow D = C$
$\neg Sup(C, D_1, It_1)$
$\neg Cl(It_1, T_4)$
...
...
$C = C$
$\neg Sup(D, D_2, It_2)$
$\neg Cl(It_2, T_4)$
$D = C$
×
×
×
×
×

Figure 1. Tableau for Example 3

rules for database tableaux, if we skolemize existential formulas before developing the tableau (Fitting, 1988). We assume from now on that all integrity constraints are skolemized by means of a set of Skolem constants (the parameters in P) and new function symbols.

Example 4. Consider the referential IC : $\forall x\ (P(x) \rightarrow \exists y\ Q(x, y))$, and the inconsistent database instance $r = \{P(a), Q(b, d)\}$, for $a, b, c \in D$. With an initial skolemization, we can develop the tableau $TP(IC \cup r)$ in figure 2. In this tableau, the second branch closes because $Q(a, f(a))$ does not belong to the database instance. There is no x in the active database domain, such that r contains $Q(a, x)$. Implicitly, by the CWA, r contains then $\neg Q(a, x)$ for any x. Hence the branch containing $Q(a, f(a))$ closes and r is inconsistent for IC.

Example 5. Consider the inconsistent database $r_1 = \{Q(a), Q(b)\}$ wrt the IC: $\exists x\ P(x)$. After having skolemized $\exists x\ P(x)$ into $P(p)$, a tableau proof for the inconsistency is the following

$$\begin{array}{c} P(p) \\ Q(a), Q(b) \\ \times \end{array}$$

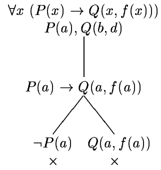

$\forall x\ (P(x) \rightarrow Q(x, f(x)))$
$P(a), Q(b, d)$
$P(a) \rightarrow Q(a, f(a))$
$\neg P(a)$
$Q(a, f(a))$
×
×

Figure 2. Tableau for Example 4

This branch closes because there is no x in D such that $P(x) \in r$ and therefore $\neg P(x)$ belongs to r for any x in D. $P(p)$ cannot belong to this database. □

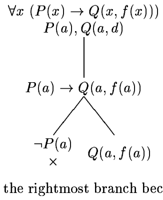
Example 6. Let us now change the database instance in example 5 to $r_2 = \{P(a), P(b)\}$, keeping the integrity constraint. Now, the database is consistent, and we have the following tableau $TP(IC \cup r_2)$:

$$\begin{array}{c} P(p) \\ P(a), P(b) \end{array}$$

This time we do not want the tableau to close, and thus sanctioning the inconsistency of the database. The reason is that we could make p take any of the values in the active domain $\{a, b\} \subseteq D$ of the database. □

A similar situation can be found in a modified version of example 4.

Example 7. Change the database instance in example 4 to $\{P(a), Q(a, d)\}$. Now it is consistent wrt the same IC. We obtain


$\forall x\ (P(x) \rightarrow Q(x, f(x)))$
$P(a), Q(a, d)$
$P(a) \rightarrow Q(a, f(a))$
$\neg P(a)$
×
$Q(a, f(a))$

Now we do not close the rightmost branch because we may define f as a function from the active domain into itself that makes $Q(a, f(a))$ become a member of the database, actually by defining $f(a) = d$. □

Example 8. Consider IC : $\exists x\ \neg P(x)$ and the consistent database instance $r = \{P(a)\}$. The tableau $TP(IC \cup r)$ after skolemization of IC is:

$$\begin{array}{c} \neg P(p) \\ P(a) \end{array}$$

This tableau cannot be closed, because p must be a new parameter, not occurring in the same branch of the tableau and it is not the case that $P(p) \in r$ (alternatively, we may think of p as a constant that can be defined as any element in $D \setminus \{a\}$, that is in the complement of the active domain of the database). □

In general, a tableau branch closes whenever it contains a formula and its negation. However, in our case, it is necessary to take into account that, due to the UNA and CWA, not all literals are explicit on the branches. The following definition of closed branch modifies the standard definition, and considers those assumptions.

Definition 5. *Let B be a tableau branch for a database instance r with integrity constraints IC, say $B = I \cup r$. B is closed iff one of the following conditions holds:*

1. *$a = b \in B$ for different constants a, b in D.*
2. *a) $P(\bar{c}) \in I$ and $P(\bar{c}) \notin r$, for a ground tuple $\bar{c}$ containing elements of D only.*

   b) $P(\bar{c}) \in I$ and there is no substitution σ for the parameters in $\bar{c}$ such that $P(\bar{c})\sigma \in r$.[5]
3. *$\neg P(\bar{c}) \in I$ and $P(\bar{c}) \in r$ for a ground tuple $\bar{c}$ containing elements of D only.*
4. *$\varphi \in B$ and $\neg\varphi \in B$, for an arbitrary formula φ.*
5. *$\neg t = t\ \in B$ for any term t.* □

Condition 1. takes UNA into account. Notice that it is restricted to database constants, so that it does not apply to new parameters[6]. Condition 2(a) takes CWA into account. Alternative condition 2(b) (actually it subsumes 2(a)) gives an account of examples 4, 5, 6, and 7.

[5] A substitution is given as a pair $\sigma = (p, t)$, where p is a variable (parameter) and t is a term. The result of applying σ to formula F, noted $F\sigma$, is the formula obtained by replacing every occurrence of p in F by t.

[6] That is, elements of P are treated as null values in Reiter's logical reconstruction of relational databases (Reiter, 1984).

In condition 3. one might miss a second alternative as in condition 2., something like "$\neg P(\bar{c}) \in I$ for a ground tuple containing Skolem symbols, when there is no way to define them considering elements of $D \setminus Act(r)$ in such a way that $P(\bar{c}) \notin r$". This condition can be never satisfied because we have an infinite database domain D, but a finite active domain $Act(r)$. So, it will never apply. This gives an account of example 8. Conditions 4. and 5. are the usual closure conditions. Conditions 2(a) and 3. are special cases of 4.

Now we can state the main properties of tableaux for database instances and their integrity constraints.

Proposition 1. *For a database instance r and integrity constraints IC, it holds:*

1. *r is inconsistent wrt to IC iff the tableau $TP(IC \cup r)$ is closed (i.e. each of its branches is closed).*
2. *$TP(IC \cup r)$ is closed iff r does not satisfy IC (i.e. $r \not\models IC$).* □

3.3. Opening tableaux

The inconsistency of a database r wrt IC is characterized by a tableau $TP(IC \cup r)$ which has only closed branches. In order to obtain a repair of r, we may remove the literals in the branches which are "responsible" for the inconsistencies, even implicit literals corresponding to the CWA. Every branch which can be "opened" in this way will possibly yield a repair. We can only repair inconsistencies due to literals in r. We cannot remove literals in I because, according to our approach, integrity constraints are rigid, we are not willing to give them up; we only allow changes in the database instances. We cannot suppress equalities $a = b$ neither built-in predicates.

Remark 2. According to Definition 5, we can repair inconsistencies due only to cases 2. and 3. More precisely, given a closed branch B in $TP(IC \cup r)$:

1. If B is closed because of the CWA, it can be opened by inserting $P(\bar{c})\sigma$ into r, or, equivalently, by removing the implicit literal $\neg P(\bar{c})\sigma$ from r for any substitution σ from the parameters into D (case 2(b) in definition 5).
2. If B is closed because of contradictory literals $\neg P(\bar{c}) \in I$ and $P(\bar{c}) \in r$, then it can be opened by removing $P(\bar{c})$ from r (case 3 in definition 5) . □

Example 9. (example 3 continued) The tableau has 9 closed branches: (we display the literals within the branches only)

B_1	B_2	B_3
$Supply(C, D_1, It_1)$	$Supply(C, D_1, It_1)$	$Supply(C, D_1, It_1)$
$Supply(D, D_2, It_2)$	$Supply(D, D_2, It_2)$	$Supply(D, D_2, It_2)$
$Class(It_1, T_4)$	$Class(It_1, T_4)$	$Class(It_1, T_4)$
$Class(It_2, T_4)$	$Class(It_2, T_4)$	$Class(It_2, T_4)$
$\neg Supply(C, D_1, It_1)$	$\neg Supply(C, D_1, It_1)$	$\neg Supply(C, D_1, It_1)$
$\neg Supply(D, D_2, It_2)$	$\neg Class(It_2, T_4)$	$D = C$

B_4	B_5	B_6
$Supply(C, D_1, It_1)$	$Supply(C, D_1, It_1)$	$Supply(C, D_1, It_1)$
$Supply(D, D_2, It_2)$	$Supply(D, D_2, It_2)$	$Supply(D, D_2, It_2)$
$Class(It_1, T_4)$	$Class(It_1, T_4)$	$Class(It_1, T_4)$
$Class(It_2, T_4)$	$Class(It_2, T_4)$	$Class(It_2, T_4)$
$\neg Class(It_1, T_4)$	$\neg Class(It_1, T_4)$	$\neg Class(It_1, T_4)$
$\neg Supply(D, D_2, It_2)$	$\neg Class(It_2, T_4)$	$D = C$

B_7	B_8	B_9
$Supply(C, D_1, It_1)$	$Supply(C, D_1, It_1)$	$Supply(C, D_1, It_1)$
$Supply(D, D_2, It_2)$	$Supply(D, D_2, It_2)$	$Supply(D, D_2, It_2)$
$Class(It_1, T_4)$	$Class(It_1, T_4)$	$Class(It_1, T_4)$
$Class(It_2, T_4)$	$Class(It_2, T_4)$	$Class(It_2, T_4)$
$C = C$	$C = C$	$C = C$
$\neg Supply(D, D_2, It_2)$	$\neg Class(It_2, T_4)$	$D = C$

The first four tuples in every branch correspond to the initial instance r. Each branch B_i consists of an I-part and the r-part, say $B_i = r \cup I_i$. And we have

I_1	I_2	I_3
$\neg Supply(C, D_1, It_1)$	$\neg Supply(C, D_1, It_1)$	$\neg Supply(C, D_1, It_1)$
$\neg Supply(D, D_2, It_2)$	$\neg Class(It_2, T_4)$	$D = C$

I_4	I_5	I_6
$\neg Class(It_1, T_4)$	$\neg Class(It_1, T_4)$	$\neg Class(It_1, T_4)$
$\neg Supply(D, D_2, It_2)$	$\neg Class(It_2, T_4)$	$D = C$

I_7	I_8	I_9
$C = C$	$C = C$	$C = C$
$\neg Supply(D, D_2, It_2)$	$\neg Class(It_2, T_4)$	$D = C$

In order to open this closed tableau, we can remove literals in the closed branches. Since a tableau is open whenever it has an open branch, each opened branch of the closed tableau might produce one possible transformed open tableau. Since we want to modify the database r, which should become consistent, we should try to remove a minimal set of literals in the r-part of the branches in order to open the tableau. This automatically excludes branches B_3, B_6 and B_9, because they close due to the literals $D = C$, which do not correspond to database literals, but come from the constraints.

In this example we observe that the sets of database literals of some of the I_j are included in others. Let us denote by I'_j the set of literals in I_j that are database literals (i.e. not built-in literals), e.g. $I'_1 = I_1, I'_7 = \{\neg Supply(D, D_2,\ It_2)\}$. We have then $I'_1 \supset I'_7$, $I'_2 \supset I'_8$, $I'_3 \supset I'_9$, $I'_4 \supset I'_7$, $I'_5 \supset I'_8$, $I'_6 \supset I'_9$. This shows, for example, that in order to open B_1, we have to remove from r a superset of the set of literals that have to be removed from r for opening B_7. Hence, we can decide that the branches whose database part contains the database part of another branch can be ignored because they will not produce any (minimal) repairs. This allows us not to consider B_1 through B_6 in our example, and B_7 and B_8 are the only branches that can lead us to repairs. □

The following lemma tells us that we can ignore branches with subsumed I-parts, because those branches cannot become repairs.

Lemma 3. *If $r'' \subseteq r' \subseteq r$, then $r' \leq_r r''$.* □

Moreover, as illustrated in example 3, where the tableau tree is shown, sometimes we can detect possible subsuming branches without fully developing the tableau. There, the first formula has been split by a tableau rule and we have already closed two branches. When we apply another rule, we know then, that the branch $C = C$, which is not closed yet, will not be closed or will be closed by a subset of the database literals appearing in the first two branches.

Definition 6. *Let $B = I \cup r$ be a closed branch of the tableau $TP(IC \cup r)$.*

(a) If I is not closed, i.e the branch is closed due to database literals only, we say that B is data closed.

(b) Let $B = I \cup r$ be a data closed branch in the tableau $TP(IC \cup r)$, we define $op(B) := (r \setminus L(B)) \cup K(B)$, where

1. *$L(B) = \{l \mid l \in r \text{ and } \neg l \in I\}$*

2. *$K(B) = \{l \mid l$ is a ground atom in I and there is no substitution σ such that $l\sigma \in r\}\tau$, where τ is any substitution of the parameters into D.*

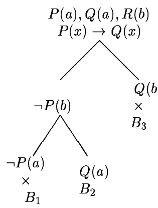
(c) An instance r' is called an opening *of r iff $r' = op(B)$ for a data closed branch B in $TP(IC \cup r)$.* □

If the branch B is clear from the context, we simply write $r' = (r \setminus L) \cup K$. If no parameters have been introduced in the branch, then we do not need to consider the substitutions above. In this case, for an opening $I \cup r'$ of a branch $I \cup r$ it holds: (a) If $P(\bar{c}) \in I$ and $P(\bar{c}) \notin r$, then $P(\bar{c}) \in r'$. (b) If $\neg P(\bar{c}) \in I$ and $P(\bar{c}) \in r$, then $P(\bar{c}) \notin r'$. Notice that we only open branches which are closed because of conflicting database literals.

When $r \models IC$, then $TP(IC \cup r)$ will have (finished) open branches B. For any of those branches $op(B)$ can be defined exactly as in definition 6. It is easy to verify that in this case $op(B)$ coincides with the original instance r.

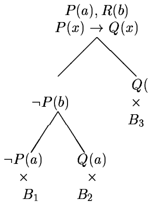
Proposition 2. *Let r' be an opening of r. Then r' is consistent with IC, i.e. $r' \models IC$.* □

Example 10. Consider $r = \{P(a), Q(a), R(b)\}$ and $IC = \{\forall x(P(x) \rightarrow Q(x)\}$. Here $r \models IC$ and $TP(IC \cup r)$ is



The first branch, B_1, is closed and $op(B_1) = \{Q(a), R(b)\}$ that satisfies IC. The second branch, B_2, is open and $op(B_2) = r$. The third branch, B_3, is closed and $op(B_3) = \{P(a), Q(a), Q(b), R(b)\}$ that satisfies IC. Notice that we could further develop the last node there, obtaining the same tree that is hanging from $\neg P(b)$ in the tree on the LHS. If we do this, we obtain closed branches B_4, B_5, with $op(B_4) = \{Q(a), Q(b), R(b)\}$, and $op(B_5) = \{P(a), Q(a), Q(b), R(b)\}$. With these last two openings we do not get any closer to r than with $op(B_3)$, that is still not as close to r as the only repair, r, obtained with branch B_2. □

Example 11. Consider IC as in example 10, but now $r = \{P(a), R(b)\}$, that does not satisfy IC. $TP(IC \cup r)$ is



For the first branch B_1, we obtain $op(B_1) = \{R(b)\}$, that is a repair. Branch B_2 gives $op(B_2) = \{P(a), R(b), Q(a)\}$, the other repair.

For the closed branch B_3 we have $op(B_3) = \{P(a), Q(b), R(b)\}$. This is not a model of IC, apparently contradicting proposition 2, in particular, it is not a repair of r. If we keep developing node $Q(b)$ exactly as $\neg P(b)$ on the LHS, we obtain extended (closed) branches, with associated instances $\{Q(b), R(b)\}$ and $\{P(a), Q(a), Q(b), R(b)\}$. Both of them satisfy IC, but are non minimal; and then they are not repairs of r. This example shows the importance of having the open and closed branches (maybe not explicitly) saturated (see definition 3). □

We can see that every opening is related to a possibly non minimal repair of the original database instance[7]. For repairs, we are only interested in "minimally" opened branches, i.e. in open branches which are as close as possible to r. In consequence, we may define a *minimal opening* r' as an opening such that $r\Delta r'$ is minimal under set inclusion.

Openings of r are obtained by deletion of literals from r, or, equivalently, by deletion/insertion of atoms from/into r. In order to obtain minimal repairs, we have to make a minimal set of changes, therefore we do not keep openings associated to an r'', such that $r'\Delta r \subsetneq r''\Delta r$,

[7] Strictly speaking, we should not say "non minimal repair", because repairs are minimal by definition. Instead, we should talk of database instances that differ from the original one and satisfy the ICs. In any case, we think there should be no confusion if we relax the language in this sense.

where r' is associated to another opening. We will show subsequently that these are the openings where L and K are minimal in the sense of set inclusion wrt all other openings in the same tree.

The following theorem establishes a relationship between the order of repairs defined in definition 1 and the set inclusion of the database atoms that have been inserted or deleted when opening a database instance.

Lemma 4. *For any opening* $r' = (r \setminus L) \cup K$, *we have* $r\Delta r' = L \cup K$. □

Proposition 3. *Let* $r_1 = (r \setminus L_1) \cup K_1$ *and* $r_2 = (r \setminus L_2) \cup K_2$. *Then* r_1 *is closer to* r *than* r_2, *i.e.* $r_1 \leq_r r_2$, *iff* $L_1 \subseteq L_2$ *and* $K_1 \subseteq K_2$. □

Theorem 2. *Let* r *be an inconsistent database wrt* IC. *Then* r' *is a repair of* r *iff there is an open branch* I *of* $TP(IC)$, *such that* $I \cup r$ *is closed and* $I \cup r'$ *is a minimal opening of* $I \cup r$ *in* $TP(IC \cup r)$. □

Example 12. (example 9 continued) $TP(IC \cup r)$ has two minimal openings:

r'_7	r'_8
$Supply(C, D_1, It_1)$	$Supply(C, D_1, It_1)$
$Class(It_1, T_4)$	$Class(It_1, T_4)$
$Class(It_2, T_4)$	$Supply(D, D_2, It_2)$

The rightmost closed branch cannot be opened because it is closed by the atom $D = C$ which is not a database predicate. □

4. Repairs, Knowledge Base Updates and Complexity

Our definition of repairs is based on a minimal distance function as used by Winslett for knowledge base update (Winslett, 1988). More precisely, Winslett in her "possible models approach" defines the knowledge base change operator ∘ for the update of a propositional knowledge base K by a propositional formula p by

$$Mod(K \circ p) = \bigcup_{m \in Mod(K)} \{m' \in Mod(p) : m\Delta m' \in Min_{\subseteq}(\{m\Delta m' : m' \in Mod(p)\})\}.$$

In (Eiter et al., 1992), Eiter and Gottlob present complexity results for propositional knowledge base revision and update. According to

these results, Winslett's update operator is on the second level of the polynomial hierarchy in the general case (i.e. without any syntactic restriction on the propositional formulas): the problem of deciding whether a formula q is a logical consequence of the update by p of a knowledge base T is Π_2^P−complete.

Update	General case	General case	Horn	Horn
	arbitrary p	$\parallel p \parallel \leq k$	arbitrary p	$\parallel p \parallel \leq k$
$T \circ p \rightarrow q$	Π_2^P−complete	co-NP-complete	co-NP-complete	$O(\parallel T \parallel \cdot \parallel q \parallel)$

In the above table, we summarize the results reported in (Eiter et al., 1992). The table contains five columns. In the general case (columns two and three), T is a general propositional knowledge base. In the Horn-case (columns four and five), it is assumed that p and q and all formulas in T are conjunctions of Horn-clauses. Columns two and four account for cases where no bound is imposed on the length of the update formula p, while columns three and five describe the case where the length of p is bounded by a constant k. The table illustrates that the general problem in the worst case (arbitrary propositional formulas without bound on the size) is intractable, whereas it becomes very well tractable (linear in the size of T and query q) in the case of Horn formulas with bounded size.

How are these results related to CQA? If r is a database which is inconsistent with respect to the set of integrity constraints IC, the derivation of a consistent answer to a query Q from r corresponds to the derivation of Q from the database r updated by the integrity constraints IC. Hence, the (inconsistent) knowledge base instance r, which is just a conjunction of literals, corresponds to the propositional knowledge base T. The integrity constraints IC correspond to the update formula p; and deriving an answer to query Q from r (and IC) corresponds to the derivation of Q from r updated by IC.

Update is defined above for propositional formulas. Update is defined by means of models of the knowledge base r and the update formula IC. In our case, r is a finite conjunction of grounded literals, i.e. r is a propositional Horn formula. The update formulas however (integrity constraints IC) are FO formulas. Since the Herbrand universe of the database is a finite set of constants, we can consider instead of IC the finite set of instantiations of the formulas in IC by database constants. Let us denote the conjunction of these instantiations by ic. Note that ic is Horn whenever all formulas in IC are Horn, what is common for database ICs.

It is then easy to see that the following relationship holds between update and repairs and CQA. It follows straightforwardly from the definitions of repairs and update.

Theorem 3. *Given a database instance* r *and a set of integrity constraints* IC *with their propositional database representation* ic:
(a) r' *is a repair of* r *wrt* IC *iff* $r' \in Mod(r \circ ic)$.
(b) If Q *is a query,* $\bar{t}$ *is a consistent answer to* Q *wrt* IC *iff every model of* $r \circ ic$ *is a model of* $Q(\bar{t})$, *i.e.* $Mod(r \circ ic) \subseteq Mod(Q(\bar{t}))$. □

In consequence, the results given by Eiter and Gottlob apply directly to CQA.

The number of branches of a fully developed tableaux is very high: in the worst case, it contains $o(2^n)$ branches, where n is the length of the formula. Moreover, we have to find minimal elements within this exponential set, what increases the complexity. Theorem 3 tells us that we do not need to compare the entire branches but only parts of them, namely the literals which have been removed in order to open the tableau. This reduces the size of the sets we have to compare, but not their number. Let us reconsider in example 3 the point just before applying the tableaux rule which develops formula $Supply(D, D_2, I_2) \wedge Class(I_2, T_4) \rightarrow D = C$. As we pointed out in the discussion of example 3, it is possible, under some conditions, to avoid the development of closed branches, because we know in advance, without developing them, that they will not be minimal.

Example 13. (example 3 continued) In this case, $TP(IC \cup r)$ is the tree in Figure 3. This tree has two closed branches, B_1 and B_2, and one open branch B_3. Each of these branches will receive an identical subtree due to the application of the tableaux rules to the formulas not yet developed on the tree, namely $(Supply(D, D_2, It_2) \wedge Class(It_2, T_4) \rightarrow D = C)$. We know at this stage of the development that B_1 is closed due to $\neg Supply(C, D_1, It_1)$ and B_2 is closed due to $\neg Class(It_1, T_4)$; B_3 is not closed. □

In this example, we can see that if we further develop the tree, every B_i will have the same sets of sub-branches, say L_1, L_2, …, where L_i is a set of literals. The final fully developed tableau will then consist of the branches $B_1 \cup L_1$, $B_1 \cup L_2$, …, $B_2 \cup L_1$, $B_2 \cup L_2$, … $B_3 \cup L_1$, $B_3 \cup L_2$, …, …. If the final tableau is closed, since B_3 is not closed, every $B_3 \cup L_j$ will be closed due to literals within L_j, say K_j.

We have then two cases: either the literals in K_j close due to literals in r (which is the original inconsistent database instance) or they close due to literals in the part of B_3 not in r. In the first case, these literals from K_j will close every branch of the tree (also B_1 and B_2). Since B_1 and B_2 were already closed, they will be closed due to a set of literals that is strictly bigger than before, and therefore they will not produce minimally closed branches (and no repairs). In this situation, those branches can immediately be ignored and not further developed. This can considerably reduce the size of the tableau. In this example, at the end of the development, only B_3 will produce repairs (see example 3).

In the second case, the literals in K_j close due to literals in the part of B_3 that are not in r. If these literals are not database literals (we have called them built-in predicates), the branch cannot be opened, because we cannot repair inconsistencies that are not due to database instances. Then, we only have to consider the case of database literals that are not in r. Since B_3 is open, those literals are negative literals (in the other case, B_3 would not have been open, due to condition 2. in definition 5). This is the only situation where the sub-branches which are closed at a previous point of development may still become minimal. In consequence, a reasonable heuristics will be to suspend the explicit development of already closed branches unless we are sure that this case will not occur.

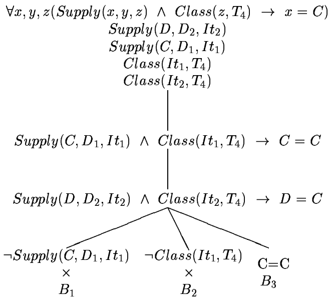

$\forall x,y,z(Supply(x,y,z) \wedge Class(z,T_4) \rightarrow x = C)$
$Supply(D,D_2,It_2)$
$Supply(C,D_1,It_1)$
$Class(It_1,T_4)$
$Class(It_2,T_4)$
$Supply(C,D_1,It_1) \wedge Class(It_1,T_4) \rightarrow C = C$
$Supply(D,D_2,It_2) \wedge Class(It_2,T_4) \rightarrow D = C$
$\neg Supply(C,D_1,It_1)$
×
B_1
$\neg Class(It_1,T_4)$
×
B_2
C=C
B_3

Figure 3. Tableau for Example 13

5. Consistent Query Answering

In order to determine consistent answers to queries, we can also use, at least at the theoretical level, a tableaux theorem prover to produce $TP(IC \cup r)$ and its openings. Let us denote by $op(TP(IC \cup r))$ the tableau $TP(IC \cup r)$, with its minimal openings: All branches which cannot be opened or which cannot be minimally opened are pruned and all branches which can be minimally opened are kept (and opened). (We reconsider this pruning process in section 6.2.)

According to definition 2 and theorem 2, $\bar{t}$ is a consistent answer to the open query $Q(\bar{x})$ when the combined tableau $op(TP(IC \cup r)) \otimes TP(\neg Q(\bar{t}))$ (c.f. definition 4) is, again, a closed tableau. In consequence, we might use the tableau $op(TP(IC \cup r)) \otimes TP(\neg Q(\bar{x}))$ in order to retrieve those values for $\bar{x}$ that restore the closure of all the opened branches in the tableau.

Example 14. Consider the functional dependency

$$IC\colon \forall(x,y,z,u,v)(Student(x,y,z) \wedge Student(x,u,v) \rightarrow y = u \wedge z = v);$$

and the inconsistent students database instance

$$r = \{Student(S_1,N_1,D_1), Student(S_1,N_2,D_1), Course(S_1,C_1,G_1),$$
$$Course(S_1,C_2,G_2)\},$$

which has the two repairs, namely

$$r_1 = \{Student(S_1,N_1,D_1), Course(S_1,C_1,G_1), Course(S_1,C_2,G_2)\},$$

$$r_2 = \{Student(S_1,N_2,D_1), Course(S_1,C_1,G_1), Course(S_1,C_2,G_2)\}.$$

We can distinguish two kinds of queries. The first one corresponds to a first order formula containing free variables (not quantified), and then expects a (set of database) tuple(s) as answer. For example, we want the consistent answers to the query "$Course(x,y,z)$?". Here we have that $op(TP(IC \cup r)) \otimes TP(\neg Course(x,y,z))$ is closed for the tuples (S_1,C_1,G_1) and (S_1,C_2,G_2).

A second kind of queries corresponds to queries without free variables, i.e. to sentences. They should get the answer "yes" or "no". For example, consider the query "$Course(S_1,C_2,G_2)$?". Here $op(TP(IC \cup r)) \otimes TP(\neg Course(S_1,C_2,G_2))$ is closed. The answer is "yes", meaning that the sentence is true in all repairs.

Now, consider the query "$Student(S_1,N_2,D_1)$?". The tableau $op(TP(IC \cup r)) \otimes TP(\neg Student(S_1,N_2,D_1))$ is not closed, and $Student(S_1, N_2,D_1)$ is not a member of both repairs. The answer is "no", meaning that the query is not true in all repairs. □

The following example shows that, as opposed to (Arenas et al., 1999), we are able to treat existential queries in a proper way.

Example 15. Consider the query "$\exists x Course(x,C_2,G_2)$?" for the database in example 14. Here we have that $op(TP(IC \cup r)) \otimes TP(\neg \exists x\, Course(x,C_2,G_2))$ is closed. The second tableau introduces the formulas $\neg Course(c,C_2,G_2)$, for every $c \in D \cup P$ in every branch. The answer is "yes". This answer has been obtained by taking c as the same constant S_1 in both branches. This does not need to be always the case. For example, with the query "$\exists x\ Student(S_1,x,D_1)$?", that introduces the formulas $\neg Student(S_1,c,D_1)$ in every branch of $op(TP(IC \cup r)) \otimes TP(\neg \exists x\ Student(S_1,x,D_1))$, the tableau closes, the answer is "yes", but one repair has been closed for $c = N_1$ and the other repair has been closed for $c = N_2$.

We can also handle open existential queries. Consider now the query with y as the free variable "$\exists z Course(S_1,y,z)$?". The tableaux for $op(TP(IC \cup r)) \otimes TP(\neg \exists z Course(S_1,y,z))$, which introduces the formulas $\neg Course(S_1,y,c)$ in every branch, is closed, actually by $y = C_1$, and also by $y = C_2$, but for two different values for c, namely G_1 and G_2, resp. □

Theorem 4. *Let r be an inconsistent database wrt the set of integrity constraints IC.*

1. *Let $Q(\bar{x})$ be an open query with the free variables $\bar{x}$. A ground tuple $\bar{t}$ is a consistent answer to $Q(\bar{x})$ iff $op(TP(IC \cup r)) \otimes TP(\neg Q(\bar{x}))$ is closed for the substitution $\bar{x} \mapsto \bar{t}$.*

2. *Let Q be query without free variables. The answer is "yes", meaning that the query is true in all repairs, iff $op(TP(IC \cup r)) \otimes TP(\neg Q)$ is closed.*

6. CQA, Minimal Entailment and Tableaux

As the following example shows, CQA is a form of *non-monotonic entailment*, i.e. given a relational database instance r, a set of ICs IC, and a consistent answer $\bar{a}$ to a query $Q(\bar{x})$ wrt IC, denoted $r \models_c Q(\bar{a})$, it may be the case that $r' \not\models_c Q(\bar{a})$, for an instance r' that extends r.

Example 16. The database containing the table

Employee	*Name*	*Salary*
	J.Page	5000
	V.Smith	3000
	M.Stowe	7000

is consistent wrt the FD: *Name* → *Salary*. In consequence, the set of consistent answers to the query $Q(x,y) : Employee(x,y)$ is $\{(J.Page, 5000), (V.Smith, 3000), (M.Stowe, 7000)\}$. If we add the tuple $(J.Page, 8000)$ to the database, the set of consistent answers to the same query is reduced to $\{(V.Smith, 3000), (M.Stowe, 7000)\}$. □

We may be interested in having a logical specification $Spec_r$ of the repairs of the database instance r. In this case, we could consistently answer a query $Q(\bar{x})$, by asking for those $\bar{t}$ such that

$$Spec_r \mathrel{|\!\!\approx} Q(\bar{t}) \quad \equiv \quad r \models_c Q(\bar{t}), \tag{1}$$

where $\mathrel{|\!\!\approx}$ is a new, suitable consequence relation, that, as the example shows, has to be non-monotonic.

6.1. A circumscriptive characterization of CQA

Notice that with CQA we have a minimal entailment relation in the sense that consistent answers are true of certain minimal models, those that minimally differ from the original instance. This is a more general reason for obtaining a nonmonotonic consequence relation. Actually, the database repairs can be specified by means of a circumscription axiom (McCarthy, 1986; Lifschitz, 1994) that has the effect of minimizing the set of changes to the original database performed in order to satisfy the ICs.

Let $P_1,\ldots,P_n$ be the database predicates in $\mathcal{L}$. In the original instance r, each P_i has a finite extension that we also denote by P_i. Let $R_1,\ldots,R_n$ be new copies of $P_1,\ldots,P_i$, standing for the corresponding tables in the database repairs. Define, for $i = 1,\ldots,n$,

$$\forall\bar{x}[P_i^{in}(\bar{x}) \ {}_{def}\!\longleftrightarrow \ (R_i(\bar{x}) \wedge \neg P_i(\bar{x}))], \tag{2}$$

$$\forall\bar{x}[P_i^{out}(\bar{x}) \ {}_{def}\!\longleftrightarrow \ (P_i(\bar{x}) \wedge \neg R_i(\bar{x}))]. \tag{3}$$

Consider now the theory Σ consisting of axioms (2), (3) plus r, i.e. the (finite) conjunction of the atoms in the database, plus $IC(P_1/R_1,\cdots, P_n/R_n)$, i.e. the set of ICs, but with the original database predicates replaced by the new predicates; and possibly, axioms for the built-in predicates, e.g. equality.

In order to minimize the set of changes, we circumscribe in parallel the predicates P_i^{in}, P_i^{out} in the theory Σ, with variable predicates $R_1, \ldots, R_n$, and fixed predicates $P_1, \ldots, P_n$ (Lifschitz, 1985), that is, we consider the following circumscription

$$\mathit{Circum}(\Sigma; P_1^{in}, \ldots P_n^{out}; R_1, \ldots, R_n; P_1, \ldots, P_n). \tag{4}$$

The semi-colons separate the theory, the predicates minimized in parallel, the variable predicates and the fixed predicate, in that order.

We want to minimize the differences between a database repair and the original database instance. For this reason we need the R_i to be flexible in the minimization process. The original predicates P_is are not subject to changes, because the changes can be read from the R_i (or from their differences with the P_i).

Example 17. Consider $r = \{P(a)\}$ and $IC = \{\forall x(P(x) \rightarrow Q(x))\}$. In this case, Σ consists of the following sentences: $P(a), \forall x(R_P(x) \rightarrow R_Q(x))$, $\forall x(P^{in}(x) \leftrightarrow R_P(x) \wedge \neg P(x))$, $\forall x(P^{out}(x) \leftrightarrow P(x) \wedge \neg R_P(x))$, $\forall x(Q^{in}(x) \leftrightarrow R_Q(x) \wedge \neg Q(x))$, $\forall x(Q^{out}(x) \leftrightarrow Q(x) \wedge \neg R_Q(x))$. Here the new database predicates are R_P and R_Q. They vary when P^{in}, P^{out}, Q^{in}, Q^{out} are minimized.

The models of th circumscription are the minimal (classical) models of the theory Σ. A model $\mathfrak{M} =< M, (P^{in})^M, (P^{out})^M, (Q^{in})^M, (Q^{out})^M, R_P^M, R_Q^M, P^M, Q^M, a^M >$ is minimal if there is no other model with the same domain M that interprets P, Q, a in the same way as $\mathfrak{M}$ and has at least one of the interpretations of P^{in}, P^{out}, Q^{in}, Q^{out} strictly included in the corresponding in $\mathfrak{M}$ and the others (not necessarily strictly) included in the corresponding interpretations in $\mathfrak{M}$. □

Circumscription (4) can be specified by means of a second-order axiom

$$\Sigma(P_1^{in}, \ldots, P_n^{in}, P_1^{out}, \ldots, P_n^{out}, R_1, \ldots, R_n) \ \wedge \tag{5}$$
$$\forall X_1 \cdot\cdot \forall X_n \forall Y_1 \cdot\cdot \forall Y_n \forall Z_1 \cdot\cdot \forall Z_n (\Sigma(X_1, .., X_n, Y_1, .., Y_n, Z_1, .., Z_n) \ \wedge$$
$$\textstyle\bigwedge_1^n X_i \subseteq P_i^{in} \ \wedge \ \bigwedge_1^n Y_i \subseteq P_i^{out} \ \longrightarrow \ \bigwedge_1^n P_i^{in} \subseteq X_i \ \wedge \ \bigwedge_1^n P_i^{out} \subseteq Y_i).$$

The first conjunct emphasizes the fact that the theory is expressed in terms of the predicates shown there. Those predicates are replaced by second-order variables in the Σ in the quantified part of the formula. The circumscription axiom says that the change predicates R_i^{in}, R_i^{out} have the minimal extension under set inclusion among those that satisfy the ICs. It is straightforward to prove that the database repairs are in one to one correspondence with the restrictions to $R_1, \ldots, R_n$ of those Herbrand models of the circumscription that have domain D and the extensions of the predicates $P_1, \ldots, P_n$ as in the original instance r.

An alternative to externally fixing the domain D consists in minimizing the finite active domain, that is a subset of D. This can be achieved by means of a circumscription as well, and then that domain can be extended to the whole of D. Notice that in order to capture the unique names assumption, the equality predicate could be minimized. Furthermore, if we want the minimal models to have the extensions for the P_i as in r, we can either include in Σ predicate closure axioms of the form $\forall \bar{x}(P_i(\bar{x}) \leftrightarrow \bigvee_1^{k_i} \bar{x}_j = \bar{a}_j)$ if P_i's extension is non-empty and $\forall \bar{x}(P_i(\bar{x})) \leftrightarrow \bar{x} \neq \bar{x})$ if it is empty; or apply to those predicates the closed world assumption, that can also be captured by means of circumscription. See (Lifschitz, 1994) for details. Another alternative is to fix the domain D and replace everywhere r in Σ by the first-order sentence, $\sigma(r)$, corresponding to Reiter's logical reconstruction of database instance r (Reiter, 1984). We do not specify any of these alternatives explicitly, but leave all this as something to be captured at the implementation level.

Example 18. (example 17 continued) The minimal models of the circumscription of the theory are $\langle D, \emptyset, \{a\}, \emptyset, \emptyset, \emptyset, \emptyset, \{a\}, \emptyset\rangle$ and $\langle D, \emptyset, \emptyset, \{a\}, \emptyset, \{a\}, \{a\}, \{a\}, \emptyset\rangle$, that show first the domain and next the extensions of P^{in}, P^{out}, Q^{in}, Q^{out}, R_P, R_Q, P, Q, in this order. The first model corresponds to repairing the database by deleting $P(a)$; the second, to inserting $Q(a)$. □

By playing with different kinds of circumscription, e.g. introducing priorities (Lifschitz, 1985), or considering only some change predicates, e.g. only P_i^{out}'s (only deletions), preferences for some particular kinds of database repairs could be captured. We do not explore here this direction any further.

The original theory Σ can be written as $\Sigma' \wedge r$, where Σ' is formed by all the conjunctions in Σ, except for r. It is easy to see that the circumscription $\mathit{Circum}(\Sigma; R_1^{in}, \ldots R_n^{out}; R_1, \ldots, R_n; P_1, \ldots, P_n)$ is logically equivalent to $r \wedge \mathit{Circum}(\Sigma'; R_1^{in}, \ldots R_n^{out}; R_1, \ldots, R_n; P_1, \ldots, P_n)$. In consequence, we can replace (1) by

$$r \wedge \mathit{Circum}(\Sigma'; R_1^{in}, ..R_n^{out}; R_1, .., R_n; P_1, .., P_n) \models Q(\bar{t}) \quad \equiv \quad r \models_c Q(\bar{t}). \tag{6}$$

We can see that in this case the nonmonotonic consequence relation $\approx$ corresponds to classical logical consequence, but with the original data put in conjunction with a second-order theory.

Some work has been done on detecting conditions and developing algorithms for the collapse of a (second-order) circumscription to a first-order theory (Lifschitz, 1985; Doherty et al., 1997). The same for collapsing circumscription to logic programs (Gelfond et al., 1989). In

our case, this should not be surprising. In (Arenas et al., 1999), for some classes of queries and ICs, CQA can be reduced to first-order query evaluation. In (Arenas et al., 2000; Greco et al., 2001; Barcelo et al., 2002; Barcelo *et al.*, 2003), direct specifications of database repairs by means of logic programs are presented.

However, there is not much hope in having the circumscription always collapsing to a first-order sentence, $\varphi_{_{Circ}}$. If this were the case, CQA would be feasible in polynomial time in the size of the database, because then for a query Q, the query $(\varphi_{_{Circ}} \rightarrow Q)$ could be posed to the original instance r. As shown in (Chomicki et al., 2002), CQA can be coNP-complete, even with simple functional dependencies and (existentially quantified) conjunctive queries.

Under those circumstances, it seems a natural idea to explore to what extent our modified semantic tableaux can be used for CQA. Actually, some implementations to nomonotonic reasoning, more precisely to minimal entailment, based on semantic tableaux have been proposed in (Olivetti, 1992; Niemela, 1996a; Niemela, 1996b; Olivetti, 1999; Bry et al., 2000).

6.2. Towards implementation

The most interesting proposal for implementing first order circumscriptive reasoning with semantic tableaux is offered by Niemela in (Niemela, 1996b), where optimized techniques for developing tableaux branches and checking their minimality are introduced. The techniques presented there, that allow minimized, variable and fixed predicates, could be applied in our context, either directly, appealing to the circumscriptive characterization of CQA we gave before, or adapting Niemela's techniques to the particular kind of process we have at hand, in terms of minimal opening of branches in the tableau $TP(IC \cup r)$.[8] We will briefly explore this second alternative.

As in (Niemela, 1996b), we assume in this section that (a) the semantic tableaux are applied to formulas in clausal form, and (b) only Herbrand models are considered, what in our case represents no limitation, because our openings, repairs, etc. are all Herbrand structures. Furthermore, if IC contains *safe* formulas (Ullman, 1988), what is commonly required in database applications, we can restrict the Herbrand domain to be the finite active domain of the database.

As seen in section 5, consistently answering query Q from instance r wrt IC, can be based on the combination of $op(TP(IC \cup r))$ and $TP(\neg Q(\bar{x}))$. Nevertheless, explicitly having the first, pruned, tableau

[8] Notice that the input theory in this case differs from the theory to which the circumscription is applied in the previous section.

amounts to having also explicitly all possible repairs of the original database. Moreover, this requires having verified the property of minimality in the data closed branches, possibly comparing different branches wrt to inclusion. It is more appealing to check minimality as the tableau $TP(IC \cup r)$ is developed.

Notice that if a finished branch $B \in TP(IC \cup r)$, opened after a preliminary data closure was reached, remains open for $\bar{x} = \bar{t}$ when combined with $TP(\neg Q(\bar{x}))$, then $op(B)$ is a model of IC and $\neg Q(\bar{t})$, and in consequence $op(B)$ provides a counterexample to $IC \models Q(\bar{t})$. However, this is classical entailment, and we are interested in those models of IC that minimally differ from r, in consequence, $op(B)$ may not be a counterexample for our problem of CQA, because it may not correspond to a repair of the original instance. Such branches that would lead to a non minimal opening in $TP(IC \cup r)$ should be closed, and left closed exactly as those branches that were closed due to built-ins.

As we can see, what is needed is a methodology for developing the tableaux in such a way that: (a) Each potential counterexample is explored, and hopefully at most once. (b) Being a non minimal opening is treated as a closure condition (because, as we just saw, they do not provide appropriate counterexamples). (c) The minimality condition is checked locally, without comparison with other branches, what is much more efficient in terms of space.

Such methodology is proposed in (Niemela, 1996b), with two classical rules for generating tableaux, a kind of hyper-type rule, and a kind of cut rule. The closure conditions are as in the classical case, but a new closure condition is added, to close branches that do not lead to minimal models. This is achieved by means of a "local" minimality test, that can also be found in (Niemela, 1996a; Eiter et al., 1993). We can adapt and adopt such a test in our framework on the basis of the definition of *grounded* model given in (Niemela, 1996b) and our circumscriptive characterization of CQA given above.

Let B be a data closed branch in $TP(IC \cup r)$, with $op(B) = (r \setminus L) \cup K$. We associate to B a Herbrand structure $M(B)$ over the first order language $\mathcal{L}(\bar{K}, \bar{L}, \bar{P}, \bar{R})$, where $\bar{R} = \langle R_1, \ldots, R_n\rangle$ is the list of original database predicates, $\bar{P} = \langle P_1, \ldots, P_n\rangle$ is the list of predicates for the repaired versions of the R_is, $\bar{L} = \langle L_1, \ldots, L_n\rangle, \bar{K} = \langle K_1, \ldots, K_n\rangle$ are predicates for $R_i \setminus P_i$ and $P_i \setminus R_i$, resp. (Then it makes sense to identify the list of predicates $\bar{L}$ and $\bar{K}$ with the sets of differences K and L in the branch B). $M(B) = \langle \mathit{Act}(r), \bar{L}^B, \bar{K}^B, \bar{P}^B, \bar{R}^B\rangle$ is defined through (and can be identified with) the subset $\Lambda := \bigcup_1^n L_i^B \cup \bigcup_1^n K_i^B \cup \bigcup_1^n P_i^B \cup \bigcup_i^n R_i^B$ of the Herbrand base $\mathcal{B}$, where $\bigcup_1^n R_i^B$ coincides with the database contents r, and the elements in $\bigcup_1^n P_i^B$ are taken from $op(B)$.

Now we can reformulate for our context the notion of grounded Herbrand structure given in (Niemela, 1996b).

Definition 7. *(adapted from (*Niemela, 1996b*)) An opening $op(B)$ is* grounded *iff for all $p \in \bar{K} \cup \bar{L}$ with $p(\bar{t}) \in \Lambda$ it holds*

$$IC(P_1/R_1, \ldots, P_n/R_n) \;\cup\; \{\bigwedge_i^n (L_i = R_i \setminus P_i), \bigwedge_i^n K_i = P_i \setminus R_i)\} \quad (7)$$
$$\cup\; N^{<\bar{L},\bar{K};\bar{R}>}(\Lambda) \;\models\; p(\bar{t}),$$

where $N^{<\bar{L},\bar{K};\bar{R}>}(\Lambda) := \{\neg q(\bar{t}) \mid q \in \bar{L} \cup \bar{K} \cup \bar{R} \text{ and } q(\bar{t}) \in \mathcal{B} \setminus \Lambda\} \cup \{q(\bar{t}) \mid q \in \bar{R} \text{ and } q(\bar{t}) \in \Lambda\}$. □

Notice that the first set in the union that defines $N^{<\bar{L},\bar{K};\bar{R}>}(\Lambda)$ corresponds to the CWA applied to the minimized predicates. i.e. those in $\bar{L}, \bar{K}$, and the fixed predicates, i.e. those in $\bar{R}$. The second set coincides with the original database contents r. From the results in (Niemela, 1996b) and our circumscriptive characterization of CQA, we obtain the following theorem.

Theorem 5. *An opening $op(B)$ corresponds to a database repair iff $M(B)$ is a grounded model of* (7). □

Ungrounded models can be discarded, and then ungroundedness can be used as an additional closure condition on branches. Notice that the test is local to a branch and can be applied at any stage of the development of a branch, even when it is not finished yet. The test is based on classical logical consequence, and then not on any kind of minimal entailment.

Example 19. (example 11 continued) We need some extra predicates. P_P, P_Q, P_R stand for the repaired versions of P, Q, R, resp. L_P, L_Q, L_R, K_P, K_Q, K_R stand for $P \setminus P_P, \ldots, P_R \setminus R$, resp. Here $\bar{L} = \langle L_P, L_Q, L_R \rangle$, $\bar{K} = \langle K_P, K_Q, K_R \rangle$, $\bar{P} = \langle P_P, P_Q, P_R \rangle$, $\bar{R} = \langle P, Q, R \rangle$.

In order to check groundedness for branches, we have the underlying theory $\Sigma = \{\forall x(P_P(x) \rightarrow P_Q(x)), \forall x(L_P(x) \leftrightarrow (P(x) \wedge \neg P_P(x))), \ldots, \forall x(K_R(x) \leftrightarrow (P_R(x) \wedge \neg R(x)))\}$, corresponding to the LHS of (7).

In order to check the minimality of branch B_1, we consider $M(B_1)$, that is determined by the set of ground atoms $\Lambda(B_1) = \{P(a), R(b), L_P(a), R_R(b)\}$. First, this structure satisfies Σ. Now, for this branch

$$\begin{aligned} N^{<\bar{L},\bar{K};\bar{R}>}(\Lambda(B_1)) \;=\; & \{\neg L_P(b), \neg L_Q(a), \neg L_Q(b), \neg L_R(a), \neg L_R(b), \\ & \neg K_P(a), \neg K_P(b), \neg K_Q(a), \neg K_Q(b), \neg K_R(a), \\ & \neg K_R(b), \neg P(b), \neg Q(a), \neg Q(b), \neg R(a)\} \;\cup \\ & \{P(a), R(b)\}. \end{aligned}$$

For groundedness, we have to check if $L_P(a)$ is a classical logical consequence of $\Sigma \cup N^{<\bar{L},\bar{K};\bar{R}>}(\Lambda(B_1))$. This is true, because, from $\neg K_Q(a)$, we obtain $\neg P_Q(a)$. Using the contrapositive of the IC in Σ, we obtain, $\neg P_P(a)$. In consequence, the opening corresponding to branch B_1 is a repair of the original database.

Consider now the unfinished branch B_3, for which $\Lambda(B_3) = \{P(a), R(b), K_Q(b), R_P(a), R_Q(b), R_R(b)\}$, and

$$\begin{aligned} N^{<\bar{L},\bar{K};\bar{R}>}(\Lambda(B_3)) \;=\; & \{\neg L_P(a), \neg L_P(b), \neg L_Q(a), \neg L_Q(b), \neg L_R(a), \\ & \neg L_R(b), \neg K_P(a), \neg K_P(b), \neg K_Q(a), \neg K_R(a), \\ & \neg K_R(b), \neg P(b), \neg Q(a), \neg Q(b), \neg R(a)\} \;\cup \\ & \{P(a), R(b)\}. \end{aligned}$$

We have to apply the groundedness test to $K_Q(b)$. In this case it is not possible to derive this atom from $\Sigma \cup N^{<\bar{L},\bar{K};\bar{R}>}(\Lambda(B_3))$, meaning that the set of literal is not grounded. If we keep developing that branch, the set N can only shrink. In consequence, we will not derive the atom in the extensions. We can stop developing branch B_3 because we will not get a minimal opening. □

7. Conclusions

We have presented the theoretical basis for a treatment of consistent query answering in relational databases by means of analytic tableaux. We have mainly concentrated on the interaction of the database instance and the integrity constraints; and on the problem of representing database repairs by means of opened tableaux. However, we also showed how the analytic tableaux methodology could we also used for consistent query answering.

We established the connections between the problem of consistent query answering and knowledge base update, on one side, and circumscriptive reasoning, on the other. The relationship between knowledge base update and circumscription has already been studied by Winslett (Winslett, 1989; Winslett, 1991) (see also (Liberatore et al., 1997)).

The connection of CQA to updates and minimal entailment allowed us to apply known complexity results to our scenario. Furthermore, we have seen that the reformulation of the problem of CQA as one of computing circumscription opens the possibility of applying established semantic tableaux based methodologies for circumscriptive reasoning.

As we have seen, there are several similarities between our approach to consistency handling and those followed by the belief revision/update community. Database repairs coincide with revised models

defined by Winslett in (Winslett, 1988). The treatment in (Winslett, 1988) is mainly propositional, but a preliminary extension to first order knowledge bases can be found in (Chou et al., 1994). Those papers concentrate on the computation of the models of the revised theory, i.e., the repairs in our case, but not on query answering. Comparing our framework with that of belief revision, we have an empty domain theory, one model: the database instance, and a revision by a set of ICs. The revision of a database instance by the ICs produces new database instances, the repairs of the original database.

Nevertheless, our motivation and starting point are quite different from those of belief revision. We are not interested in computing the repairs *per se*, but in answering queries, hopefully using the original database as much as possible, possibly posing a modified query. If this is not possible, we look for methodologies for representing and querying simultaneously and implicitly all the repairs of the database. Furthermore, we work in a fully first-order framework. Other connections to belief revision/update can be found in (Arenas et al., 1999).

We should emphasize that in this paper we are not addressing the issues of database maintenance, integrity enforcement (Schewe et al., 1999) or integrity checking (Nicolas, 1982). Techniques in those directions are intended to keep the database consistent; whereas we accept inconsistent databases, and the database repairs are just an auxiliary concept used to characterize those answers to queries that are consistent with (possibly globally) violated integrity constraints.

To the best of our knowledge, the first treatment of CQA in databases goes back to (Bry, 1997). The approach is based on a purely proof-theoretic notion of consistent query answer. This notion, described only in the propositional case, is more restricted than the one we used in this paper. In (Cholvy, 1998), Cholvy presents a general logic framework for reasoning about contradictory information which is based on an axiomatization in modal propositional logic. Instead, our approach is based on classical first order logic.

Other approaches to consistent query answering based on logic programs with stable model semantics were presented in (Arenas et al., 2000; Barcelo et al., 2002; Greco et al., 2001). They can handle general first order queries with universal ICs.

There are many open issues. One of them has to do with the possibility of obtaining from the tableaux for instances and ICs the right "residues" that can be used to rewrite a query as in (Arenas et al., 1999). The theoretical basis of CQA proposed in (Arenas et al., 1999) were refined and implemented in (Celle et al., 2000). Comparisons of the tableaux based methodology for CQA and the "rewriting based approach" presented in those papers is an open issue. However, query rewriting, as presented in (Arenas et al., 1999), can not be applied to existential queries like the one in example 15, whereas the tableaux methodology can be used. Perhaps, an appropriate use of tableaux could make possible an extension of the rewriting approach to syntactically richer queries and ICs.

Another interesting open issue has to do with the fact that we have treated Skolem parameters as null values. It would be interesting to study the applicability in our scenario of methodologies for query evaluation in databases in the presence of null values like the one presented in (Reiter, 1986).

In this paper we have concentrated mostly on the theoretical foundations of a methodology based on semantic tableaux for querying inconsistent databases. Nevertheless, the methodology for CQA requires further investigation. In this context, the most interesting open problems have to do with implementation issues. More specifically, the main challenge consists in developing heuristics and mechanisms for using a tableaux theorem prover to generate/store/represent $TP(IC \cup r)$ in a compact form with the purpose of: (a) applying the database assumptions, (b) interacting with a DBMS on request, in particular, without replicating the whole database instance at the tableau level, (c) detecting and producing the minimal openings (only), (d) using a theorem prover (in combination with a DBMS) in order to consistently answer queries.

From our experience with logic programming based CQA (Arenas et al., 2000; Barcelo et al., 2002), we know it is possible to optimize the representations in general, obtaining simpler logic programs with lower evaluation complexity (Barcelo *et al.*, 2003). Similar investigations should be carried out in the tableau based approach.

An important issue in database applications is that usually queries have free variables and then answer sets have to be retrieved as a result of the automated reasoning process. Notice that once we have $op(TP(IC \cup r))$, we need to be able to: (a) use it for different queries Q, (b) process the combined tableau $op(TP(IC \cup r)) \otimes TP(\neg Q)$ in an "reasonable and practical" way. We have seen that existing methodologies and algorithms like the one presented in (Niemela, 1996b), can be used in this direction. However, producing a working implementation, considering all kinds of optimizations with respect to representation and development of the tableaux, grounding techniques, database/theorem-prover interaction, etc. is a major task that deserves separate investigation.

Acknowledgements

Work supported by FONDECYT Grant 1000593; ECOS /CONICYT Grant C97E05, Carleton University Start-Up Grant 9364-01, NSERC Grant 250279-02. L. Bertossi is a Faculty Fellow of the IBM Center for Advanced Studies, Toronto Lab. We are grateful to anonymous referees for their remarks.

References

Arenas, A., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. Proc. ACM Symposium on Principles of Database Systems (ACM PODS'99). ACM Press, 1999, pp. 68–79.

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.L. Larsen, J. Kacprzyk, S. Zadrozny, H. Christiansen (eds.). Springer-Verlag, 2000, pp. 27–41.

Arenas, A., Bertossi, L. and Chomicki, J. Scalar Aggregation in FD-Inconsistent Databases. In Database Theory - ICDT 2001 (Proc. International Conference on Database Theory, ICDT'2001). Springer LNCS 1973, 2001, pp. 39 – 53.

Bertossi, L. and Schwind, C. B. An Analytic Tableaux based Characterization of Database Repairs for Consistent Query Answering (preliminary report). In Working Notes of the IJCAI'01 Workshop on Inconsistency in Data and Knowledge. AAAI Press, 2001, pp. 95 – 106.

Bertossi, L. and Schwind, C. B. Analytic Tableaux and Database Repairs: Foundations. In *Foundations of Information and Knowledge Systems (Proc. FoIKS 2002)*, Eiter, T. and Schewe, K.-D. (eds.). Springer LNCS 2284, 2002, pp. 32-48.

Barcelo, P. and Bertossi, L. Repairing Databases with Annotated Predicate Logic. In *Proc. Ninth International Workshop on Non-Monotonic Reasoning (NMR'2002). Special session on Changing and Integrating Information: From Theory to Practice.* S. Benferhat and E. Giunchiglia (eds.). Morgan Kaufmann Publishers, 2002, pp. 160 – 170.

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.

Belleannée, C., Kuhna, P., Lamarre, P., Schwind, C., Thiébaux, S., Vialard, V. and Vorc'h, R. Méthodes Sémantiques de Démonstration pour Logiques Nonstandards. In *PRC GDR Intelligence artificielle, Actes des 5èmes Journées Nationales*, Nancy 2-5, février 1995.

Beth, E. W. *The Foundations of Mathematics.* North Holland, 1959.

Bry, F. Query Answering in Information Systems with Integrity Constraints. In Proc. IFIP WG 11.5 Working Conference on Integrity and Control in Information Systems, Chapman & Hall, 1997.

Bry, F. and Yahya, A.H. Positive Unit Hyperresolution Tableaux and Their Application to Minimal Model Generation. *Journal of Automated Reasoning*, 25(1) (2000) 35–82.

Celle, A. and Bertossi, L. Querying Inconsistent Databases: Algorithms and Implementation. In 'Computational Logic - CL 2000', J. Lloyd et al. (eds.). Stream: 6th International Conference on Rules and Objects in Databases (DOOD'2000). Springer LNAI 1861, 2000, pp. 942 – 956.

Cholvy, L. A General Framework for Reasoning about Contradictory Information and some of its Applications. In Proceedings of ECAI Workshop "Conflicts among Agents", Brighton, England, August 1998.

Chomicki, J. and Marcinkowski, J. Minimal-Change Integrity Maintenance Using Tuple Deletions. Submitted in 2002 (CoRR paper cs.DB/0212004).

Chou, T. and Winslett, M. A Model-Based Belief Revision System. *J. Automated Reasoning*, 12 (1994) 157–208.

Doherty, P., Lukaszewicz, W. and Szalas, A. Computing Circumscription Revisited: A Reduction Algorithm. *Journal of Automated Reasoning*, 18(3) (1997) 297–336.

Eiter, T. and Gottlob, G. On the Complexity of Propositional Knowledge Base Revision, Updates, and Counterfactuals. *Artificial Intelligence*, 57 (1992) 227-270.

Eiter, T. and Gottlob, G. Propositional Circumscription and Extended Closed World Assumption are Π_2^p-complete. *Theoretical Computer Science*, 114 (1993) 231-245.

Fitting, M. First Order Modal Tableaux. *Journal of Automated Reasoning*, 4(2) (1988) 191–213.

Fitting, M. *First Order Logic and Automated Theorem Proving.* Texts and Monographs in Computer Science. Springer-Verlag, 2nd Edition, 1996.

Gelfond, G. and Lifschitz, V. Compiling Circumscriptive Theories into Logic Programs. In *Non–Monotonic Reasoning.* Springer LNAI 346, 1989, pp. 74–99.

Gottlob, G. Complexity Results for Nonmonotonic Logics. *Journal of Logic and Computation*, 2(3) (1992).

Greco, G.; Greco, S. and Zumpano, E. A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. In *Proc. 17th International Conference on Logic Programming (ICLP'01)*, Ph. Codognet (ed.). Springer LNCS 2237, 2001, pp. 348–364.

Lafon, E. and Schwind, C. B. A Theorem Prover for Action Performance. In Y. Kodratoff, editor, *Proceedings of the 8th European Conference on Artificial Intelligence*, Pitman Publishing, 1988, pp. 541–546.

Liberatore, P. and Schaerf, M. Reducing Belief Revision to Circumscription (and vice versa). *Artificial Intelligence*, 93 (1997) 261–296.

Lifschitz, V. Computing Cirscumscription. In Proc. IJCAI'85, 1985, pp. 121-127.

Lifschitz, V. Circumscription. In Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 3. Oxford University Press, 1994, pp. 297–352.

Lloyd, J.W. *Foundations of Logic Programming.* Springer-Verlag, 1987.

McCarthy, J. Applications of Circumscription to Formalizing Common Sense Knowledge. *Artificial Intelligence*, 26(3) (1986) 89–118.

Nicolas, J-M. Logic for Improving Integrity Checking in Relational Databases. *Acta Informatica*, 18 (1982) 227–253.

Niemela, I. A Tableau Calculus for Minimal Model Reasoning. In Proc. Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods. Springer LNCS 1071, 1996, pp. 278-294.

Niemela, I. Implementing Circumscription Using a Tableau Method. Proc. ECAI 1996, pp. 80–84.

Olivetti, N. Tableaux and Sequent Calculus for Minimal Entailment. *Journal of Automated Reasoning*, 9(1) (1992) 99–139.

Olivetti, N. Tableaux for Nonmonotonic Logics. In Handbook of Tableaux Methods. Kluwer Publishers, 1999, pp. 469–528.

Reiter, R. Towards a Logical Reconstruction of Relational Database Theory. In 'On Conceptual Modeling', Brodie, M. L. and Mylopoulos, J. and Schmidt, J. W. (eds.). Springer-Verlag, 1984, pp. 191–233.

Reiter, R. A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values. *Journal of the ACM*, 33(2) (1986) 349–370.

Schewe, K-D. and Thalheim, B. Towards a Theory of Consistency Enforcement. *Acta Informatica*, 36(2) (1999) 97–141.

Schwind, C. B. A Tableau-based Theorem Prover for a Decidable Subset of Default Logic. In M. E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction.* Springer LNAI 449, 1990, pp. 541–546.

Schwind, C. B. and Risch, V. Tableau-based Characterisation and Theorem Proving for Default Logic. *Journal of Automated Reasoning*, 13(4) (1994) 223–242.

Smullyan, R. M. *First Order Logic.* Springer-Verlag, 1968.

Ullman, J. *Principles of Database and Knowledge-Base Systems, Vol. I.* Computer Science Press, 1988.

Winslett, M. Reasoning about Action with a Possible Models Approach. In *Proceedings of the 8th National Conference on Artificial Intelligence*, 1988, pp. 89–93.

Winslett, M. Sometimes Updates are Circumscription. *Proceeding of the International Joint Conference on Artificial Intelligence (IJCAI'89)*, 1989, pp. 859-863.

Winslett, M. Cirscumscriptive Semantics for Updating Knowledge Bases. *Annals of Mathematics and Artificial Intelligence*, 3(2-4) (1991) 429–450.

Appendix: Proofs

Proof of Lemma 3

We have by Lemma 1 $r'\Delta r = r \setminus r'$ and $r''\Delta r = r \setminus r''$. Then $l \in r'\Delta r$ iff $l \in r \setminus r'$, i.e. $l \in r$ and $l \notin r'$ from which it follows that $l \in r$ and $l \notin r''$. Hence $l \in r \setminus r'' = r''\Delta r$.

Proof of Lemma 4

Let r' be an opening of r. Then $r' = (r \setminus L) \cup K$, where $L = \{l : l \in r \text{ and } \neg l \in I\}$ and $K = \{l : l \in I$ and there is no substitution σ such that $l\sigma \in r\}\tau$. Let us first observe that $L \cap K = \emptyset$ since $L \subseteq r$ and for $l \in K$, $l \notin r$. We show that $r\Delta r' = L \cup K$. Let be $x \in r\Delta r'$.
1. Case $x \in r$ and $x \notin r'$. Then $x \notin K$ and $x \notin (r \setminus L)$. But from this, we get $x \in L$, hence $x \in L \cup K$
2. Case $x \notin r$ and $x \in r'$, iff $x \notin r$ and $((x \in r$ and $x \notin L)$ or $x \in K)$, iff $x \notin r$ or $x \in K$ from which it follows $x \in K \cup L$.

On the other hand, let be $x \in L \cup K$. Again, we consider two cases:
1. Case $x \in L$, then by definition, $\neg x \in I$. Then, $x \notin r \setminus L$ and, since

I is open, $x \notin I$. From this, we get $x \notin K$ and, since $r' = (r \setminus L) \cup K$, $x \notin r'$, from which it follows that $x \in r\Delta r'$.
2. Case $x \in K$, then $x \in I$ and $x \notin r$. But then $x \in r'$ and therefore $x \in r\Delta r'$.

Proof of Proposition 3

By Lemma 4, we have $r\Delta r_1 = L_1 \cup K_1$ and $r\Delta r_2 = L_2 \cup K_2$. From $r_1 \leq_r r_2$ we get then $L_1 \cup K_1 \subseteq L_2 \cup K_2$. Since $L_i \cap K_i = \emptyset$, we have $L_1 \subseteq L_2$ and $K_1 \subseteq K_2$.

Proof of Theorem 2

Let r' be a repair of r. Then $r' \models IC$ and $r' \in Min_{leq_r}(ic)$. Since r' is a model of IC, by Theorem 1, r' contains an open branch I of the tableau $TP(IC)$ for IC. We have $r' = (r \setminus L) \cup K$ and since r' is minimal wrt $\leq_r$, there is no r'' closer to r than r', i.e. there is no $r'' = (r \setminus L') \cup K'$ such that $L' \subset L$ and $K' \subset K$. Hence $r' \cup I$ is a minimal opening of $r \cup I$.

On the other hand, let $I \cup r'$ be a minimal opening of $I \cup r$ in $TP(IC \cup r)$ where I is an open branch of $TP(IC)$. Then, by Definition 6, $r' = (r \setminus L) \cup K$ where $L = \{l : l \in r$ and $\neg l \in I$ and $K = \{l : l \in I$ and there is no substitution σ such that $l\sigma \in r\}$. By Lemma 4, we have $r\Delta r' = L \cup K$. Since $I \cup r$ is a minimal opening of $I \cup r'$, we have by Theorem 3, that there is no r'', L'' and K'' such that r'' is an opening of r and $r'' = (r \setminus L'') \cup K''$ and $L'' \subset L$ and $K'' \subset K$. By Lemma 4, this means that there is no r'' such that $r\Delta r'' \subset r\Delta r'$, i. e. r' is a minimal element of $Mod(IC)$ wrt the order $\leq_r$.

Under consideration for publication in Theory and Practice of Logic Programming

Answer sets for consistent query answering in inconsistent databases

MARCELO ARENAS∗

*Pontificia Universidad Catolica de Chile, Departamento de Ciencia de Computacion,
Santiago, Chile*
(*e-mail:* marenas@ing.puc.cl)

LEOPOLDO BERTOSSI

School of Computer Science, Carleton University, Ottawa, Canada
(*e-mail:* bertossi@scs.carleton.ca)

JAN CHOMICKI

*Department of Computer Science and Engineering, State University of New York at Buffalo,
Buffalo, NY, USA*
(*e-mail:* chomicki@cse.buffalo.edu)

Abstract

A relational database is *inconsistent* if it does not satisfy a given set of integrity constraints. Nevertheless, it is likely that most of the data in it is consistent with the constraints. In this paper we apply logic programming based on answer sets to the problem of retrieving consistent information from a possibly inconsistent database. Since consistent information persists from the original database to every of its minimal repairs, the approach is based on a specification of database repairs using *disjunctive logic programs with exceptions*, whose answer set semantics can be represented and computed by systems that implement stable model semantics. These programs allow us to declare persistence by default of data from the original instance to the repairs; and changes to restore consistency, by exceptions. We concentrate mainly on logic programs for binary integrity constraints, among which we find most of the integrity constraints found in practice.

KEYWORDS: databases, answer set programming, integrity constraints, consistency

1 Introduction

Integrity Constraints (IC) capture an important normative aspect of every database application, whose aim is to guarantee the consistency of its data. However, it is very difficult, if not impossible, to always have a consistent database instance. Databases may become inconsistent with respect to a given set of integrity constraints. This may happen due, among others, to the following factors: (1) Certain ICs cannot be expressed/maintained by existing DBMSs; (2) transient inconsistencies caused by the inherent non-atomicity of database transactions, (3) delayed updates of a datawarehouse, (4) integration of heterogeneous databases, in particular with duplicated information; (5) inconsistency with respect to *soft* integrity constraints, where transactions in violation of their conditions are not prevented from executing; (6) legacy data on which one wants to impose semantic constraints; (7) the consistency of the database will be restored by executing further transactions; and (8) user constraints than cannot be checked or maintained.

Independently of the cause of inconsistency, the inconsistent database may be the only source of data available, and we may still want or need to use it for a number of reasons. Restoring the consistency of the database may not be an option since that may require permissions we don't have, lead to the loss of useful information, or be a complex and non-deterministic process. Under such circumstances, one faces the natural problem of characterizing and retrieving the *consistent* information from the database. Most likely, most of the information in the database is still consistent, and the database can still provide us with correct answers to certain queries, making the problem of determining what kinds of queries and query answers are consistent with the integrity constraints a worthwhile effort.

The problem of defining and retrieving consistent information from an inconsistent relational database has been studied in the context of relational databases. The basic approach is based on the intuition that the information that is consistent, despite the inconsistency of the database as a whole, is the one that is invariant under all sensible ways in which the consistency of the database is restored. More precisely, an answer to a query is consistent if it is obtained as an answer every time the query is posed to a minimally repaired version of the original database (Arenas *et al.* 1999).

Example 1
Assume we have the following database instance *Salary*:

Salary	*Name*	*Amount*
	V.Smith	5000
	V.Smith	8000
	P.Jones	3000
	M.Stone	7000

and *FD* is the functional dependency *Name* → *Amount*, meaning that *Name* functionally determines *Amount*, that is violated by the table *Salary*. Actually the tuples participating in this violation are those with the value *V.Smith* in attribute *Name*.

When we ask about the tuples that are consistent wrt the FD, we should retrieve only $(P.Jones, 3000)$ and $(M.Stone, 7000)$, because those tuples should stay in any reasonable way in which we restore consistency.

If we want to consider only repaired versions of the original instance that minimally differ from the original instance, in the sense that the set of inserted or deleted tuples (to restore inconsistency) is minimal under set inclusion, the possible *repairs* of the inconsistent database are

$Salary_1$	*Name*	*Amount*
	V.Smith	5000
	P.Jones	3000
	M.Stone	7000

$Salary_2$	*Name*	*Amount*
	V.Smith	8000
	P.Jones	3000
	M.Stone	7000

We can see that only tuples $(P.Jones, 3000)$ and $(M.Stone, 7000)$ can be found in both repairs. □

In this paper, we address the problem of retrieving consistent information when general first order queries are posed to an inconsistent relational database. Since the consistent information in the database is the one that persists across all repairs, we solve this problem by using logic programs with answer sets semantics to specify in a compact manner the class of repairs of the inconsistent instance.

Although the consistent answers are defined in terms of minimally repaired version of the database, we are not interested in restoring consistency, in particular, in computing the repairs of the database: Repairs are used as an auxiliary notion in order to give a model-theoretic characterization of the consistent answers to queries. Actually, it is easy to find situations where exponentially many repairs of an inconsistent database exist (Arenas *et al.* 2001).

A possible computational mechanism for retrieving consistent answers, first introduced in Arenas *et al.* (1999) and Celle and Bertossi (2000), is as follows: Given a first-order query Q and an inconsistent database instance r, instead of explicitly computing all the repairs of r and querying all of them, a new query $T(Q)$ is computed and posed to r, the only available database. The answers to the new query are expected to be the consistent answers to Q. Such an iterative operator for query transformation was introduced and analyzed with respect to soundness, completeness and termination in Arenas *et al.* (1999) and Celle and Bertossi (2000).

Nevertheless, the query rewriting approach has some limitations. The iterative operator introduced in Arenas *et al.* (1999) and Celle and Bertossi (2000) works for some particular classes of queries and constraints, e.g. for queries that are conjunctions of literals and universal integrity constraints, but completeness is lost when it is applied to disjunctive or existential queries. The methodology for obtaining consistent answers that we present in this paper can applied to any first-order query instead.

Furthermore, the notion of consistent answer introduced in Arenas *et al.* (1999) is a model theoretic notion, that is complemented by a computational mechanism. Nevertheless, that approach is not based on or accompanied by a *logical specification* of the class of all the repairs of a given database instance relative to a fixed set of ICs. Such a specification is another contribution of this paper, namely a specification expressed as a disjunctive logic program with answer set semantics. The database repairs correspond to the intended models or answer sets of the program.

In this paper, we are motivated mainly by the possibility of retrieving consistent answers to general first-order queries, extending the possibilities we developed in Arenas *et al.* (1999). However, the logical specification could be also used to (1) Reason about all database repairs, in particular about consistent query answers, (2) derive specialized algorithms for consistent query answering, (3) analyze complexity issues related to consistent query answering, and (4) obtain the intended models of the specification, i.e. the database repairs, allowing us to analyze different ways to restore the consistency of the database. That is, a mechanism for computing database repairs could be used for conflict resolution.

Notice that consistent answers are non-monotonic in the sense that adding information to the original database, may cause loosing previous consistent answers. In consequence, a non-monotonic semantics for the specification (or its consequences) should be expected.

A preliminary version of this paper appeared in Arenas *et al.* (2000a), where extended disjunctive logic programs with exceptions where introduced and applied to the specification of database repairs and to retrieve consistent answers to general first-order queries. This paper extends Arenas *et al.* (2000a), addressing several new issues, among which we find (1) a detailed analysis of the correspondence between e-answer sets and database repairs for binary integrity constraints, (2) application of the *DLV* system (Eiter *et al.* 1998) to obtain database repairs and consistent answers, (3) extensions of the methodology to more general universal constraints and to referential integrity constraints, (4) an analysis of the applicability of the disjunctive well-founded semantics to consistent query answering, and (5) the use of weak constraints to capture database repairs based on minimal *number* of changes.

This paper is structured as follows. In section 2 we introduce the notions of database repair and consistent answer to a query, and the query language. Section 3 introduces extended disjunctive logic programs with exceptions. In section, 4, the main section of the paper, we present the repair programs for binary integrity constraints, and show how to consistently evaluate queries. In section 5 we show some examples using the *DLV* system to obtain database repairs and consistent answers. In section 6 we illustrate how to handle referential integrity constraints. In section 7 we analyze the well-founded interpretation as an approximation to the set of consistent answers, and we identify cases where it provides the exact solution. In section 8 we show how database repairs based on minimal number of changes can be specified by introducing weak constraints in the repair programs. In section 9, we draw conclusions, we sketch some extensions, e.g. to the case of general universal ICs, we also mention open issues, and discuss related work.

2 Consistent query answers

A database schema can be represented by a typed language $\mathcal{L}$ of first-order predicate logic, that contains a finite set of predicates and a fixed infinite set of constants D. A relational database instance r can be seen as an interpretation or a first order structure for $\mathcal{L}$, whose domain is also D (the interpretation of each constant is the constant itself), and the predicates have finite extensions. In what follows, a database instance r will be represented, in a natural way, as a finite set of ground atoms (the atoms true in r).

The active domain of a database instance r is the set of those elements of D that explicitly appear in r. The active domain is always finite and we denote it by $Act(r)$. We may also have a set of built-in (or evaluable) predicates, like equality, order

∗ Current address: University of Toronto, Department of Computer Science, Toronto, Canada. E-mail: marenas@cs.toronto.edu.

relations, arithmetical relations, etc. In this case, we have the language $\mathcal{L}$ possibly extended with those predicates. In all database instances, for a given schema, each of these predicates has a fixed and possibly infinite extension. Since we defined database instances as finite sets of ground atoms, we are not considering those built-in atoms as members of database instances.

In addition to the database schema and instances, we may also have a set of integrity constraints IC expressed in language $\mathcal{L}$. These are first-order formulas which the database instances are expected to satisfy. If a database instance r satisfies IC in the standard model-theoretic sense, what is denoted by $r \models IC$, we say that it is consistent (wrt IC), otherwise we say it is inconsistent. In any case, we will assume from now on that IC is logically consistent set of first-order sentences.

The original motivation in Arenas *et al.* (1999) was to consistently answer first-order queries. We shall call them *basic queries* and define them by the grammar

$$B \ ::= \ Atom \mid B \wedge B \mid \neg B \mid \exists x \ B.$$

One way of explicitly asking at the object level about the consistent answers to a first-order query consists in introducing a new logical operator $\mathcal{K}$, in such a way that $\mathcal{K}\varphi(\bar{x})$, where $\varphi(\bar{x})$ is a basic query, asks for the values of $\bar{x}$ that are consistent answers to $\varphi(\bar{x})$ (or whether φ is consistently true, i.e. true in all repairs, when φ is a sentence). The *$\mathcal{K}$-queries* are similarly defined:

$$A \ ::= \ \mathcal{K}B \mid A \wedge A \mid \neg A \mid \exists x. \ A.$$

In this paper, we concentrate mostly on answering *basic $\mathcal{K}$-queries* of the form $\mathcal{K}B$, where B is a basic query.

Definition 1

(a) (Arenas *et al.* 1999) Given a database instance r and a set of integrity constraints, IC, a *repair* of r wrt IC is a database instance r', over the same schema, that satisfies IC and such that $r\Delta r' = (r \setminus r') \cup (r' \setminus r)$, the symmetric difference of r and r', is minimal under set inclusion.

(b) (Arenas *et al.* 1999) A tuple $\bar{t}$ is a *consistent answer* to a first-order query $Q(\bar{x})$, or equivalently, an answer to the query $\mathcal{K}Q(\bar{x})$, in a database instance r iff $\bar{t}$ is an answer to query $Q(\bar{x})$ in every repair r' of r wrt IC. In symbols:

$$r \models \mathcal{K}Q[\bar{t}] \quad \Longleftrightarrow \quad r' \models Q[\bar{t}] \quad \text{for every repair } \ r' \ \text{ of } \ r.$$

(c) If Q is a general $\mathcal{K}$-query, then $r \models Q$ is defined inductively as usual, (b) being the base case.

Example 2

(Example 1 continued.) For the inconsistent database and the given FD, $\bar{t}_3 = (P.Jones, 3000)$ is a consistent answer to the query $Salary(\bar{x})$, i.e. $r \models \mathcal{K}\ Salary(x, y)[(P.Jones, 3000)]$, but $r \not\models \mathcal{K}\ Salary(x,\ y)[(V.Smith,\ 8000)]$. It also holds $r \models \mathcal{K}\ (Salary(V.Smith, 5000) \ \vee \ Salary(V.Smith, 8000))$, and $r \models \mathcal{K}\ \exists X\,(Salary(V.Smith, X) \wedge X > 4000)$.

Computing consistent answer through generation of all possible repairs is not a

natural and feasible alternative (Arenas *et al.* (2001)). Instead, an approach based on querying the available, although inconsistent, database is much more natural. This rewriting approach introduced in Arenas *et al.* (1999) is not complete for disjunctive or existential queries, like $\exists Y\ Salary(V.Smith, Y)$ in Example 2. We would like to be able to obtain consistent answers to basic $\mathcal{K}$-queries at least.

Notice that the definition of consistent query answer depends on our definition of repair. In section 8.1 we will consider an alternative definition of repair based on minimal *number* of changes instead of minimal *set* of changes.

3 Logic programs with exceptions

Logic Programs with Exceptions (LPEs) (Kowalski and Sadri 1991) have default rules whose consequences can be overridden by the consequences of exception rules. They turn out to be the right formalism for specifying the database repairs: by default everything persists from the original database instance to any of its repairs, except for the changes that are necessary to restore the consistency.

LPEs as introduced in Kowalski and Sadri (1991) consist of definite clauses, whose head and body contain literals of the form $A, \neg A$, where A is an atom and $\neg$ is classical negation. In the bodies, literals may be affected by weak negation, *not* (negation as failure). In a LPE there are *default* rules, which are clauses with positive heads, and *exception* rules, which are clauses with negative heads. To capture the intuition that exceptions have priority over defaults, in Kowalski and Sadri (1991) a new semantics was introduced based on *e-answer sets.* It is defined as follows.

First, instantiate the program Π in the database domain, making it ground. Now, let S be a set of ground literals $S = \{L, \ldots\}$. This S is a candidate to be a model, a guess to be verified, and accepted if properly justified.

Next, generate a new set of ground rules ${}^{S}\Pi$ according to the following steps:

(1) Delete every rule in Π containing *not* L in the body, with $L \in S$.
(2) Delete from the clauses every condition *not* L in the body, when $L \notin S$.
(3) Delete every default rule having a positive conclusion A with $\neg A \in S$.

The result is a ground extended logic program without *not*. Now, S is an *e-answer set* of the original program if S is the smallest set of ground literals, such that: (a) For any clause $L_0 \longleftarrow L_1, \cdots, L_m$ in ${}^{S}\Pi$, if $L_1, \cdots, L_m \in S$, then $L_0 \in S$; (b) if S contains two complementary literals, then S is the set of all literals.

The e-answer sets are the intended models of the original program. Above, (1), (2) are as in the *answer sets semantics* for extended logic programs (Gelfond and Lifschitz 1991), but now (3) gives an account of exceptions.

To specify database repairs, we need to extend the LPEs and their semantics as presented in Kowalski and Sadri (1991), considering *Disjunctive Logic Programs with Exceptions* (DLPEs), that contain also negative defaults, i.e. defaults with negative conclusions that can be overridden by positive exceptions, and extended disjunctive exceptions, i.e. rules of the form

$$L_1 \vee \cdots \vee L_k \ \longleftarrow \ L_{k+1}, \ldots, L_r, \mathit{not}\ L_{r+1}, \ldots, \mathit{not}\ L_m,$$

where the L_is are literals.[1] The e-answer semantics is extended as follows. The ground program is pruned according to a modified rule (3) above:

(3') Delete every (positive) default having a positive conclusion A, with $\neg A \in S$; and every (negative) default having a negative conclusion $\neg A$, with $A \in S$.

Applying (1), (2) and (3') to the ground program, we are left with a ground disjunctive logic program without *not*. If the candidate set of literals S belongs to $\alpha({}^{S}\Pi)$, the set of minimal models of program ${}^{S}\Pi$, then we say that S is an *e-answer set.*

This semantics does not capture priorities between defaults, and, in principle, there could be conflicting defaults. In this case, the semantics seems to allow that defaults override other defaults, without preferences for any of them. For example, the program containing only the defaults $p \leftarrow \mathit{not}\ q$ and $\neg p \leftarrow \mathit{not}\ r$ (without exception rules), has two e-answer sets, namely $\{p\}$ and $\{\neg p\}$. In any case, in our applications of logic programs with exceptions, due to the kind of defaults we will use (see Definition 6), such a situation will never appear, because the potentially conflicting defaults apply to mutually exclusive cases.

Finally, we take advantage of the existence of a one to one correspondence between the e-answer sets of a DLPE and the answer sets of an extended disjunctive logic program (Gelfond and Lifschitz 1991 (see section 4.1, Remark 1).

4 Logic programs for CQA

We shall use DLPEs for specifying database repairs and answering basic $\mathcal{K}$-queries. Given a set of ICs and an inconsistent database instance r, the first step consists of writing a *repair program*, $\Pi(r)$, having as the e-answer sets the repairs of the original database instance. $\Pi(r)$ captures the fact that when a database instance r is repaired most of the data persists, except for some tuples. In consequence, default rules are introduced: everything persists from the instance r to the repairs. It is also necessary to introduce exception rules: everything persists, as stated by the defaults, unless the ICs are violated and have to be satisfied.

Next, if a first-order query is posed with the intention of retrieving all and only its consistent answers, then a *query program*, that expresses the query, is run together with the repair program.

In this section we introduce the DLPEs for specifying database repairs, and give a careful analysis of those programs for consistent query answering wrt Binary Integrity Constraints (BICs), i.e. they are universally quantified sentences of the form $\overline{\forall}(L_1 \vee L_2 \vee \varphi)$, where $\overline{\forall}$ denotes the universal closure, L_1, L_2 are literals associated to the database schema; φ is a first-order formula containing only built-in predicates[2] and free variables appearing in L_1, L_2.

[1] In our application scenario we will need disjunctive exceptions rules, but not disjunctive defaults.

[2] Built-in predicates have a fixed extension in every database, in particular, in every repair; so they are not subject to changes.

We have three possibilities for BICs in terms of the sign of literals in them, namely the universal closures of:

$$p_1(\bar{x_1}) \vee p_2(\bar{x_2}) \vee \varphi; \quad p_1(\bar{x_1}) \vee \neg q_1(\bar{y_1}) \vee \varphi; \quad \neg q_1(\bar{y_1}) \vee \neg q_2(\bar{y_2}) \vee \varphi, \qquad (1)$$

where the $p_i(\bar{x_i}), q_j(\bar{y_j})$ are database atoms. BICs with one database literal plus possibly a formula containing built-ins are called *unary ICs.*

Several interesting classes of ICs (Abiteboul *et al.* 1995) used in database praxis can be represented by BICs: (a) Range constraints, e.g. $P(x,y) \rightarrow x > 5$; (b) Full inclusion dependencies; (b) functional dependencies (see Example 1), etc. Nevertheless, for referential ICs, like in $P(x,y) \rightarrow \exists z Q(x,z)$, we need existential quantifiers or Skolem functions (Fitting 1996). They are considered in section 6.

4.1 Finite domain databases

In this section we will momentarily depart from our assumption that databases have an infinite domain D (see section 1), and will analyze the case of finite domain databases. The reason is that in the general case, we will be interested in *domain independent* BICs, for which only the active domain is relevant (and finite).

4.1.1 The change program

To introduce the repair programs $\Pi(r)$ and analyze their behavior, we will concentrate first on the sub-program that does not contain defaults rules. This program, denoted by $\Pi_{\Delta}(r)$, is responsible for the changes (but not for persistence).

Splitting the program in this way makes the analysis easier. Furthermore, keeping $\Pi_{\Delta}(r)$, but using different form of defaults, we can capture different kinds of repairs. In section 4.1.2, we will introduce defaults leading to our notion of repair based on minimal set of changes (Definition 1). In section 8.1, we will use other defaults that lead to repairs based on minimal number of changes.

Definition 2

Given a set of BICs IC and an instance r, the *change program*, $\Pi_{\Delta}(r)$, contains the following rules:

1. Facts: (a) For every ground database atom $p(\bar{a}) \in r$, the fact $p(\bar{a})$.
   (b) For every a in D, the fact $dom(a)$.
2. For each IC of the forms in (1), respectively, the triggering rule

$$\begin{aligned} p_1'(\bar{X}_1) \vee p_2'(\bar{X}_2) \ &\longleftarrow \quad dom(\bar{X}_1, \bar{X}_2), \ \mathit{not}\ p_1(\bar{X}_1), \ \mathit{not}\ p_2(\bar{X}_2), \ \overline{\varphi} \\ p_1'(\bar{X}_1) \vee \neg q_1'(\bar{Y}_1) \ &\longleftarrow \quad dom(\bar{X}_1), \ \mathit{not}\ p_1(\bar{X}_1), \ q_1(\bar{Y}_1), \ \overline{\varphi} \\ \neg q_1'(\bar{Y}_1) \vee \neg q_2'(\bar{Y}_2) \ &\longleftarrow \quad q_1(\bar{Y}_1), \ q_2(\bar{Y}_2), \ \overline{\varphi} \end{aligned}$$

3. For an IC of the first form in (1), the *pair* of stabilizing rules

$$\begin{aligned} p_1'(\bar{X}_1) \ &\longleftarrow \quad dom(\bar{X}_1), \ \neg p_2'(\bar{X}_2), \ \overline{\varphi} \\ p_2'(\bar{X}_2) \ &\longleftarrow \quad dom(\bar{X}_2), \ \neg p_1'(\bar{X}_1), \ \overline{\varphi} \end{aligned}$$

For an IC of the second form in (1), the pair of stabilizing rules

$$\begin{array}{rcl} p'_1(\bar{X}_1) & \longleftarrow & dom(\bar{X}_1),\ q'_1(\bar{Y}_1),\ \overline{\varphi} \\ \neg q'_1(\bar{Y}_1) & \longleftarrow & dom(\bar{Y}_1),\ \neg p'_1(\bar{X}_1),\ \overline{\varphi} \end{array}$$

For an IC of the third form in (1), the pair of stabilizing rules

$$\begin{array}{rcl} \neg q'_1(\bar{Y}_1) & \longleftarrow & dom(\bar{Y}_1),\ q'_2(\bar{Y}_2),\ \overline{\varphi} \\ \neg q'_2(\bar{Y}_2) & \longleftarrow & dom(\bar{Y}_2),\ q'_1(\bar{Y}_1),\ \overline{\varphi}. \end{array}$$

The primed versions $(p', \ldots)$ of the original database predicates $(p, \ldots)$ stand for the database predicates in the repairs. In these rules, $dom(.,.)$ is an abbreviation for the conjunction of memberships to *dom* of all the individual variables; and $\overline{\varphi}$, an abbreviation for a representation of the negation of φ. Depending on its syntax, it may be necessary to unfold the formula $\overline{\varphi}$ into additional program rules, but $\overline{\varphi}$ will usually be a conjunction of literals. □

Example 3
Consider the inclusion dependencies $IC : \{\forall xy\,(P(x,y) \rightarrow Q(x,y)), \forall xy\,(Q(x,y) \rightarrow R(x,y))\}$ and the inconsistent database instance $r = \{P(a,b), Q(a,b)\}$. The program $\Pi_\Delta(r)$ contains the following clauses:

1. *Facts*: $P(a,b),\ Q(a,b)$.
2. *Triggering exceptions*: $\neg P'(X,Y) \vee Q'(X,Y) \ \leftarrow\ P(X,Y),\ not\ Q(X,Y).$
   $\neg Q'(X,Y) \vee R'(X,Y) \ \leftarrow\ Q(X,Y),\ not\ R(X,Y).$
   Each of these rules represent the two possible ways to repair the corresponding IC, separately: The first rule says that in order to "locally" repair the first IC, either eliminate (X,Y) from P or insert (X,Y) into Q. The semantics of these DLPEs gives the disjunction an exclusive interpretation. In this example, due to the form of the ICs, we do not need domain predicates.
3. *Stabilizing exceptions*: $Q'(X,Y) \leftarrow P'(X,Y); \quad \neg P'(X,Y) \leftarrow \neg Q'(X,Y).$
   $R'(X,Y) \leftarrow Q'(X,Y); \quad \neg Q'(X,Y) \leftarrow \neg R'(X,Y).$
   These rules state that eventually the ICs have to be satisfied in the repairs. They are necessary if, like in this example, there are interacting ICs and local repairs alone are not sufficient. Propagation of changes are required beyond the first triggering step. Since the ICs can be repaired by either deleting or inserting a tuple, the contrapositive versions of the ICs are needed.

Notice that for BICs, the stabilizing rules in $\Pi_\Delta(r)$ do not contain disjunctions in the heads.

Definition 3
A *model* of a DLPE, Π, is a set of ground literals, S, that does not contain complementary literals and satisfies Π in the usual logical sense, but with weak negation interpreted as not being an element of S.

Definition 4
Given a model S of $\Pi_\Delta(r)$, we define the database instance corresponding to S by

$$I(S) = \{p(\bar{a}) \mid p'(\bar{a}) \in S\} \ \cup\ \{p(\bar{a}) \mid p(\bar{a}) \in S \text{ and } \neg p'(\bar{a}) \notin S)\}.$$

Notice that, for a given model S of the change program, $I(S)$ merges in one new instance all the positive primed tuples with all the old, non primed tuples that persisted, i.e. that their negative primed version do not belong to the model. Since there are no persistence defaults in $\Pi_\Delta(r)$, persistence is captured explicitly in $I(S)$.

Proposition 1
Given a database instance r and a set of BICs IC, if S is a model of $\Pi_\Delta(r)$, then $I(S)$ satisfies IC.

Definition 5
Given database instances r and r' over the same schema and domain, we define

$$\begin{array}{rcl} S(r,r') & = & \{p(\bar{a}) \mid r \models p(\bar{a})\} \ \cup\ \{p'(\bar{a}) \mid r' \models p(\bar{a})\} \ \cup\ \{\neg p'(\bar{a}) \mid r' \not\models p(\bar{a})\} \\ & & \cup\ \{dom(a) \mid a \in D\}. \end{array}$$
□

$S(r,r')$ collects the maximal consistent set of literals that can be obtained from two database instances, e.g. the original instance and a repair. The atoms corresponding to the second argument are primed. Negative literals corresponding to the first argument are not considered, because weak negation will be applied.

Proposition 2
Given a database instance r and a set of BICs IC, if r' satisfies IC, then $S(r,r')$ is a model of $\Pi_\Delta(r)$.

This result tells us that subsets of $S(r,r')$ could be potential models of the change program. $S(r,r')$ can be a large model, in the sense that the difference between r and r' may not be minimal.

Proposition 3
For BICs, the change program $\Pi_\Delta(r)$ has an answer set; and all the answer sets are consistent, i.e. they do not contain complementary literals.[3]

4.1.2 The repair program

Program $\Pi_\Delta(r)$ gives an account of changes only. The fact that repairs contain data that persists from the original instance is captured with persistence defaults.

Definition 6
The repair program $\Pi(r)$ consists of the rules in program $\Pi_\Delta(r)$ (Definition 2) plus the following two rules for each predicate p in the original database:

4. Persistence *defaults*:

$$p'(\bar{X}) \longleftarrow p(\bar{X}); \qquad \neg p'(\bar{X}) \longleftarrow dom(\bar{X}),\ not\ p(\bar{X}).$$
□

Example 4
(example 3 continued) We have the following persistence defaults:

[3] In $\Pi_\Delta(r)$ there are no defaults. In consequence, we can talk about answer sets as in Gelfond and Lifschitz (1991) instead of e-answer sets (Kowalski and Sadri 1991).

4. $P'(X,Y) \leftarrow P(X,Y); \qquad \neg P'(X,Y) \leftarrow dom(X,Y),\ not\ P(X,Y)$
   $Q'(X,Y) \leftarrow Q(X,Y); \qquad \neg Q'(X,Y) \leftarrow dom(X,Y),\ not\ Q(X,Y).$
   $R'(X,Y) \leftarrow R(X,Y); \qquad \neg R'(X,Y) \leftarrow dom(X,Y),\ not\ R(X,Y).$

   This means that, by default, everything from r is put into a repair r' and nothing else.

In this program rules 2 and 3 have priority over rule 4. It is possible to verify that the e-answer sets of the program are the expected database repairs: $\{P'(a,b), Q'(a,b), \underline{R'(a,b)}, P(a,b), Q(a,b), \ldots\}$, $\{\underline{\neg P'(a,b)}, \underline{\neg Q'(a,b)}, P(a,b), Q(a,b), \ldots\}$. The underlined literals represent the insertion of $R(a,b)$ in one repair and the deletion of of both $P(a,b)$ and $Q(a,b)$, in the other one, respectively. The original atoms remain, because there are no rules that can change them. The literals not shown explicitly in these e-answer sets are the negative literals, e.g. $\neg P'(a,a), \neg Q'(b,a)$, inherited from the original instance with the negative defaults.

Remark 1
As shown in Kowalski and Sadri (1991), the program $\Pi(r)$, which has an e-answer semantics, can be transformed into a disjunctive extended logic program with answer set semantics, by transforming the persistence defaults in Definition 6, respectively, into

4'. Persistence *rules*:

$$p'(\bar{X}) \longleftarrow p(\bar{X}),\ not\ \neg p'(\bar{X}); \qquad \neg p'(\bar{X}) \longleftarrow dom(\bar{X}),\ not\ p(\bar{X}),\ not\ p'(\bar{X}).$$

As shown in Gelfond and Lifschitz (1991), the resulting program can be further transformed into a disjunctive normal program with a stable model semantics. For the one to one correspondence between answer sets and stable models, we can interchangeably talk about (e-)answer sets and stable models.

Proposition 4
Given a database instance r over a finite domain, and a set of BICs IC, if S_M is an answer set of $\Pi_\Delta(r)$, then $S = S_M \cup \{p'(\bar{a}) \mid p(\bar{a}) \in S_M \text{ and } \neg p'(\bar{a}) \notin S_M\} \ \cup\ \{\neg p'(\bar{a}) \mid p(\bar{a}) \notin S_M \text{ and } p'(\bar{a}) \notin S_M\}$ is an answer set of $\Pi(r)$.

The following lemma says that whenever we build an answer set S with literals taken from $S(r,r')$, and r' satisfies the ICs and is already as close as possible to r, then in S we recover r' only. The condition that S is contained in $S(r,r')$ makes sure that its literals are taken from the right, maximal set of literals.

Lemma 1
Let r and r' be database instances over the same schema and domain, and IC, a set of BICs. Assume that $r' \models IC$ and the symmetric difference $\Delta(r,r')$ is a minimal element under set inclusion in the set $\{\Delta(r,r^*) \mid r^* \models IC\}$. Then, for every answer set S of $\Pi_\Delta(r)$ contained in $S(r,r')$, it holds $r' = I(S)$.

Theorem 1
If $\Pi(r)$ is the program $\Pi_\Delta(r)$ plus rules 4'., for a finite domain database instance r and a set of BICs IC, it holds:

1. For every repair r' of r wrt IC, there exists an answer set S of $\Pi(r)$ such that $r' = \{p(a) \mid p'(a) \in S\}$.
2. For every answer set S of $\Pi(r)$, there exists a repair r' of r wrt IC such that $r' = \{p(a) \mid p'(a) \in S\}$.

In the case of finite domain databases, the domain can be and has been declared. In this situation, we can handle any set of binary ICs, without caring about their safeness or domain independence (Ullman 1988).

Example 5
Consider $D = \{a,b,c\}$, $IC = \{\forall x p(x)\}$ and the inconsistent instance $r = \{p(a)\}$. $\Pi(r)$ contains the default rules $p'(X) \longleftarrow p(X), not\ \neg p'(X)$; $\neg p'(X) \longleftarrow dom(X)$, $not\ p(X), not\ p'(X)$; the triggering exception $p'(X) \longleftarrow dom(X), not\ p(X)$, the stabilizing exception $p'(X) \longleftarrow dom(X)$; and the facts $dom(a), dom(b), dom(c)$, $p(a)$. The only answer set is $\{dom(a), dom(b), dom(c), p(a), p'(a), p'(b), p'(c)\}$, that corresponds to the only repair $r' = \{p(a), p(b), p(c)\}$.

The IC requires that every element in the finite domain D belongs to table p; and this can be achieved. However, with an infinite domain D, we could not obtain a finite program nor an instance with a table p containing finitely many tuples.

4.2 Infinite domain databases

Now we consider ICs that are *domain independent*, for which checking their satisfaction in an instance r can be done considering the elements of the finite active domain $Act(r)$ only (Ullman 1988). The IC in Example 5 is not domain independent.

For domain independent BICs all previous lemmas and theorems still hold if we have an infinite domain D. To obtain them, all we need to do is to use a predicate $act_r(x)$, standing for the active domain $Act(r)$ of instance r, instead of predicate $dom(x)$. This is because, for domain independent BICs, the database domain can be considered to be $Act(r)$. Furthermore, in this case we can omit the *dom* facts and goals from $\Pi(r)$. In consequence, we have the following theorem.

Theorem 2
For a set of domain independent binary integrity constraints and a database instance r, there is a one to one correspondence between the answers sets of the repair program $\Pi(r)$ and the repairs of r.

4.3 Evaluating basic $\mathcal{K}$-queries

The specification of database repairs we have obtained provides the underpinning of a general method of evaluating a basic $\mathcal{K}$-query of the form $\beta \equiv \mathcal{K}\alpha$, where α is a basic query.

First, from α, that is expressed in terms of the database predicates in $\mathcal{L}$, we obtain a stratified logic program $\Pi(\alpha)$ (this is a standard construction (Lloyd 1987; Abiteboul *et al.* 1995)) in terms of the new, primed predicates introduced in $\Pi(r)$.

One of the predicate symbols, $Answer_\alpha$, of $\Pi(\alpha)$ is designated as the query answer predicate. Second, determine all the answers sets $S_1, \ldots, S_k$ of the logic program $\Pi = \Pi(\alpha) \cup \Pi(r)$. Third, compute the intersection $r_\beta = \bigcap_{1 \leq i \leq k} S_i/Answer_\alpha$, where $S_i/Answer_\alpha$ is the extension of $Answer_\alpha$ in S_i. The set of tuples r_β is the set of answers to β, or equivalently, the set of consistent answers to α, in r.

Example 6
(example 4 continued) Consider the query for the consistent answers to $\alpha_1(x)$: $(P(x,a) \vee Q(a,x))$, in the database instance. This query can be transformed into the query program $\Pi(\alpha_1)$ containing the rules $Answer_{\alpha_1}(X) \longleftarrow P'(X,a)$, and $Answer_{\alpha_1}(X) \longleftarrow Q'(a,X)$.

To obtain consistent answers it is necessary to evaluate the query goal $Answer_{\alpha_1}(X)$ wrt the program obtained by combining $\Pi(r)$, already obtained in Examples 3 and 4, and program $\Pi(\alpha_1)$. Each of the answer sets of the combined program will contain a set of ground $Answer_{\alpha_1}$-atoms. The arguments of the $Answer_{\alpha_1}$-atoms that are present simultaneously in all the answer sets will be the consistent answers to the original query.

As a second example, consider the query $\alpha_2(y):\ \exists x Q(x,y)$. In order to obtain the consistent answers, we keep $\Pi(r)$ as before, but we run it in combination with the new query program $\Pi(\alpha_2)$: $Answer_{\alpha_2}(Y) \longleftarrow Q'(X,Y)$.

Notice that consistent answers to a query are those that can be obtained from the repair program plus the query program under the *cautious* or *skeptical* answer set semantics for the combined logic program: what is true of the program is what is true of all its answer sets. In section 5 we give computational examples.

The program $\Pi = \Pi(\alpha) \cup \Pi(r)$, where α is a first order query, is naturally split into $\Pi(\alpha)$ and $\Pi(r)$, but also split in the precise sense introduced in Lifschitz and Turner (1994) as follows: the set U of literals consisting of all the primed database literals, $(\neg)p'(\bar{t})$ plus and all the non primed database literals, $(\neg)p(\bar{t})$ appearing in $\Pi(r)$, form a *splitting set* for Π, because whenever a literal in U appears in a head of a rule in Π, all the literals in the body of that rule also appear in U. U splits Π precisely into the two expected parts, $\Pi(r)$ and $\Pi(Q)$, because the literals in U do not appear in heads of rules of $\Pi(\alpha)$ (for $\Pi(\alpha)$ the literals in U act as extensional literals).

As a consequence of this splitting, we know from Lifschitz and Turner (1994), that every answer set of Π can be represented as the union of an answer set of $\Pi(r)$ and an answer set of $\Pi(\alpha)$, where each answer set for $\Pi(r)$ acts as an extensional database for the computation of the answer sets of $\Pi(\alpha)$. Since program $\Pi(\alpha)$ is stratified, for each answer set of $\Pi(r)$, there will only one answer set for $\Pi(\alpha)$.

5 Computational examples

In this section we will assume that, according to Remark 1, the repair programs are given as extended disjunctive logic programs with answer set semantics. In consequence, we can use any implementation for that semantics. In particular, we will give examples of the application of the *DLV* system (Eiter *et al.* 1998) to the computation of database repairs and consistent query answers.

5.1 Computing database repairs with DLV

Example 7
Consider the schema $Emp(Name, SSN)$, and the functional dependencies $Name \rightarrow SSN$, $SSN \rightarrow Name$, stating that each person should have just one SSN and different persons should have different SSNs. The following is an inconsistent instance:

Emp	*Name*	*SSN*
	Irwin Koper	677-223-112
	Irwin Koper	952-223-564
	Mike Baneman	334-454-991

The following *DLV* program corresponds to the repair program. In it, the repaired, primed version of table *Emp* is now denoted by *emp_p*:

```
% domains of the database
dom_name("Irwin Koper"). dom_name("Mike Baneman"). dom_number("677-223-112").
dom_number("952-223-564"). dom_number("334-454-991").

% initial database
emp("Irwin Koper","677-223-112"). emp("Irwin Koper","952-223-564").
emp("Mike Baneman","334-454-991").

% default rules
emp_p(X,Y)  :- emp(X,Y), not -emp_p(X,Y).
-emp_p(X,Y) :- dom_name(X), dom_number(Y), not emp(X,Y), not emp_p(X,Y).

% triggering rules
-emp_p(X,Y) v -emp_p(X,Z) :- emp(X,Y), emp(X,Z), Y!=Z.
-emp_p(Y,X) v -emp_p(Z,X) :- emp(Y,X), emp(Z,X), Y!=Z.

% stabilizing rules.
-emp_p(X,Y) :- emp_p(X,Z), dom_number(Y), Y!=Z.
-emp_p(Y,X) :- emp_p(Z,X), dom_name(Y), Y!=Z.
```

If *DLV* is asked to compute the answer sets, we obtain two of them, corresponding to the two possible repairs:

Emp	*Name*	*SSN*
	Irwin Koper	952-223-564
	Mike Baneman	334-454-991

Emp	*Name*	*SSN*
	Irwin Koper	677-223-112
	Mike Baneman	334-454-991

To pose the query $Emp(X,Y)$?, asking for the consistent tuples in table *Employee*, we add a new query rule to the program: `answer(X,Y) :- emp_p(X,Y).`
Now, the two answer sets contain `answer`-literals, namely

```
{..,answer("Irwin Koper","952-223-564"),answer("Mike Baneman","334-454-991")}
{..,answer("Irwin Koper","677-223-112"),answer("Mike Baneman","334-454-991")}
```

There is only one ground `answer`-atom in the intersection of the answer sets of the new program. Then, the only consistent answer is the tuple: `X="Mike Baneman"`, `Y="334-454-991"`.

6 Referential integrity constraints

In this section, we show how to extend the specifications of repairs given for binary integrity constraints to Referential Integrity Constraints (RICs). This can be done via an appropriate representation of existential quantifiers as program rules.

Consider the *RIC*: $\forall \bar{x}\ (P(\bar{x}) \rightarrow \exists \bar{y}\ R(\bar{x}, \bar{y}))$, and the inconsistent database instance $r = \{P(\bar{a}), P(\bar{b}), R(\bar{b}, \bar{a})\}$. We assume that there is an underlying database domain D. The repair program has the persistence default rules

$$P'(\bar{X}) \leftarrow P(\bar{X}); \quad \neg P'(\bar{X}) \leftarrow dom(\bar{X}),\ not\ P(\bar{X});$$

$$R'(\bar{X},\bar{Y}) \leftarrow R(\bar{X},\bar{Y}); \quad \neg R'(\bar{X},\bar{Y}) \leftarrow dom(\bar{X},\bar{Y}),\ not\ R(\bar{X},\bar{Y}).$$

In addition, it has the triggering exception rule

$$\neg P'(\bar{X}) \vee R'(\bar{X}, null) \leftarrow P(\bar{X}), not\ aux(\bar{X}), \tag{2}$$

with $aux(\bar{X}) \leftarrow R(\bar{X},\bar{Y})$; $null \notin D$; and the stabilizing exception rules

$$\neg P'(\bar{X}) \quad \leftarrow \quad \neg R'(\bar{X}, null), not\ aux'(\bar{X}), \tag{3}$$

$$R'(\bar{X}, null) \quad \leftarrow \quad P'(\bar{X}), not\ aux'(\bar{X}); \tag{4}$$

with $aux'(\bar{X}) \leftarrow R'(\bar{X},\bar{Y})$.

The variables in this program range over D, that is, they do not take the value *null*. This is the reason for the first literal in clause (3). The last literal in clause (4) is necessary to insert a null value only when it is needed; this clause relies on the fact that variables range over D only. Instantiating variables on D only[4], the only two answer sets are the expected ones, namely delete $P(\bar{a})$ or insert $R(\bar{a}, null)$.

It would be natural to include here the functional dependency $\bar{X} \rightarrow \bar{Y}$ on R, expressing that $\bar{X}$ is a primary key in R and a foreign key in P. This can be done without problems, actually the two constraints would not interact, that is, repairing one of them will not cause violations of the other one.

Finally, if only elimination of tuples were considered admissible changes, but not introduction of null values, then the triggering exception (2) would have to be changed into $\neg P'(\bar{X}) \leftarrow P(\bar{X}), not\ aux(\bar{X})$.

[4] A simple way to enforce this at the object level is to introduce the predicate D in the clauses, to force variables to take values in D only, excluding the null value. Alternatively, conditions of the form $X \neq null$ can be placed in the bodies.

6.1 Referential ICs and strong constraints

It is possible to use *DLV* to impose preferences on repairs via an appropriate representation of constraints. For RICs, for example, preference for introduction of null values or for a cascade policy can be captured.

Example 8
(Example 7 continued.) Consider the same schema and *FDs* as before, but now we have the following instance:

Emp	*Name*	*SSN*
	Irwin Koper	677-223-112
	Irwin Koper	952-223-564
	Mike Baneman	952-223-564

The *DLV* repair program is as in Example 7, but with the facts:

```
dom_number("677-223-112").  dom_number("952-223-564").
emp("Irwin Koper","677-223-112"). emp("Irwin Koper","952-223-564").
emp("Mike Baneman","952-223-564").
```

If *DLV* is run with this program as input, we obtain two answer sets:

```
{..,emp_p("Irwin Koper","677-223-112"),-emp_p("Irwin Koper","952-223-564"),
emp_p("Mike Baneman","952-223-564"),-emp_p("Mike Baneman","677-223-112")}
{..,-emp_p("Irwin Koper","677-223-112"),emp_p("Irwin Koper","952-223-564"),
-emp_p("Mike Baneman","952-223-564"),-emp_p("Mike Baneman","677-223-112")}
```

corresponding to the database repairs:

Emp	*Name*	*SSN*
	Irwin Koper	677-223-112
	Mike Baneman	952-223-564

Emp	*Name*	*SSN*
	Irwin Koper	952-223-564

Adding the query rule `answer(X):- emp_p(X,Y).`, we can ask for those persons who have a SSN. Two answer sets are obtained:

```
{..,answer("Irwin Koper"),answer("Mike Baneman")}, {..,answer("Irwin Koper")}
```

From them, we can -consistently- say that only Irwin Koper has a SSN.

Let us now extend the schema with a unary table $Person(Name)$, whose contents, together with the original contents of table *Emp*, is

Person	*Name*
	Irwin Koper
	Mike Baneman

If we want every person to have a SSN, we may impose the RIC $\forall\, x(\mathit{Person}(x) \rightarrow\ \exists\, y\mathit{Emp}(x,y))$, stating that every person must have a SSN, that we saw how to repair at the beginning of this section, either by introducing null values or by cascading deletions.

We may not want any of these two options (we do not want null values in the key *SSN*) or we do not want to delete any employees (in this case, M. Baneman from *Person*). An alternative is to use *DLV*'s possibility of specifying *strong constraints*, that have the effect of pruning those answer sets that do not satisfy them. This can be done in *DLV* by introducing the denial `:- dom_name(X), not has_ssn(X).`, with `has_ssn(X) :- emp_p(X,Y).`. The answer sets of the original program that do not satisfy the ICs are filtered out; and now, only one repair is obtained:

```
{..,emp_p("Irwin Koper","677-223-112"),-emp_p("Irwin Koper","952-223-564"),
emp_p("Mike Baneman","952-223-564"),-emp_p("Mike Baneman","677-223-112"),
has_ssn("Irwin Koper"), has_ssn("Mike Baneman"), answer("Irwin Koper"),
answer("Mike Baneman")}
```

In it, every person has a SSN (according to the `has_ssa` predicate). As expected, the answers to the original query are `X="Irwin Koper"` and `X="Mike Baneman"`. Notice that strong constraints differ from the database integrity constraints in that they are not used in the generation of repairs, but only at a final step where some repairs are discarded. Furthermore, strong constraints are constraints on the answer sets, but not directly on the semantics of the database.

7 Well-founded consistent answers

The intersection of all answer sets of a extended disjunctive logic program contains the *well-founded interpretation* for such programs (Leone *et al.* 1997), which can be computed in polynomial time in the size of the ground program. This interpretation may be partial and not necessarily a model of the program. Actually, it is a total interpretation if and only if it is the only answer set.

In Leone *et al.* (1997) it is shown how to compute the answer sets of a program starting from the well-founded interpretation. This is what *DLV* basically does, but instead of starting from the well-founded interpretation, it starts from the also efficiently computable set of *deterministic consequences* of the program, that is still contained in the intersection of all answer sets, and in its turn, contains the well-founded interpretation (Calimieri *et al.* 2002). Actually, *DLV* can be explicitly asked to return the set of deterministic consequences of the program[5], and it can be also used as an approximation from below to the intersection of all answer sets.

On the other side, in the general case, computing the stable model semantics for disjunctive programs is Π_2^P-complete in the size of the ground program[6].

The well-founded interpretation, $W_{\Pi(r)} = \langle W^+, W^-, W^u\rangle$, of program $\Pi(r)$, where W^+, W^-, W^u are the sets of true positive, true negative, and unknown literals, resp., is given by the least fixpoint $\mathcal{W}^{\omega}_{\Pi(r)}(\emptyset)$ of operator $\mathcal{W}_{\Pi(r)}$, that maps interpretations to interpretations Leone *et al.* 1997). More precisely, assuming that we have the ground instantiation of the repair program $\Pi(r)$, $\mathcal{W}_{\Pi(r)}(I)$ is defined on interpretations I that are sets of ground literals (without pairs of complementary literals) by: $\mathcal{W}_{\Pi(r)}(I) := T_{\Pi(r)}(I) \cup \neg.\mathit{GUS}_{\Pi(r)}(I)$.

Intuitively, $T_{\Pi(r)}$ is the immediate consequence operator that declares a literal true whenever there is ground rule containing it in the head, the body is true in I and the other literals in the (disjunctive) head are false in I. $\neg.\mathit{GUS}_{\Pi(r)}(I)$ denotes the set of complements of the literals in $\mathit{GUS}_{\Pi(r)}(I)$, being the latter the largest set of unfounded literals, those that definitely cannot be derived from the program and the set I of assumptions; in consequence their complements are declared true.

The intersection of all answer sets of $\Pi(r)$ is

$$\mathit{Core}(\Pi(r)) := \bigcap\{S \mid S \text{ is an answer set of } \Pi(r)\}.$$

Interpretation $W_{\Pi(r)}$, being a subset of $\mathit{Core}(\Pi(r))$, can be used as an approximation from below to the core, but can be computed more efficiently than all database repairs, or their intersection, in the general case. However, it is possible to identify classes of ICs for which $W_{\Pi(r)}$ coincides with $\mathit{Core}(\Pi(r))$.

Proposition 5
For a database instance r, and a set of ICs containing functional dependencies and unary ICs only, the $\mathit{Core}(\Pi(r))$ of program $\Pi(r)$ coincides with the set of true ground literals in $W_{\Pi(r)}$, the well-founded interpretation of program $\Pi(r)$.

Results like the previous one can be established using the repair programs introduced in section 4.1, for finite database domains D. Then, the results are known to still hold for infinite domain databases, but domain independent integrity constraints, like the ones in Proposition 5.

As corollary of Proposition 5, we obtain that, for FDs and unary constraints, $\mathit{Core}(\Pi(r))$ can be computed in polynomial time in the size of the ground instantiation of $\Pi(r)$, a result first established in Arenas *et al.*(2001) for FDs. The core alone can be used to consistently answer non-existential conjunctive queries. Furthermore, in Arenas *et al.*(2001), for the case of functional dependencies, conditions on queries are identified under which one can take advantage of computations on the core to answer aggregate queries more efficiently.

As the following example shows, for other BICs, the core of the repair program may not coincide with the well-founded interpretation.

Example 9
Consider the BICs $IC = \{q \vee r,\ s \vee \neg q,\ s \vee \neg r\}$ and the empty database instance. The program $\Pi(r)$ contains

Triggering rules: $q' \vee r' \longleftarrow \mathit{not}\ q, \mathit{not}\ r;\quad s' \vee \neg q' \longleftarrow q, \mathit{not}\ s;\quad s' \vee \neg r' \longleftarrow r, \mathit{not}\ s.$

Stabilizing rules: $q' \longleftarrow \neg r';\quad r' \longleftarrow \neg q';\quad s' \longleftarrow q';\quad \neg q' \longleftarrow \neg s;$
$s' \longleftarrow r';\quad \neg r' \longleftarrow \neg s'.$

Persistence rules: $q' \longleftarrow q, \mathit{not}\ \neg q';\quad s' \longleftarrow s, \mathit{not}\ \neg s';\quad r' \longleftarrow r, \mathit{not}\ \neg r';$
$\neg q' \longleftarrow \mathit{not}\ q, \mathit{not}\ q';\quad \neg s' \longleftarrow \mathit{not}\ s, \mathit{not}\ s';\quad \neg r' \longleftarrow \mathit{not}\ r, \mathit{not}\ r'.$

The answer sets are: $\{q', s', \neg r'\}$ and $\{\neg q', s', r'\}$. Then $\mathit{Core}(\Pi(r)) = \{s'\}$, but for $W_{\Pi(r)}$, one has $W^+ \cup W^- = \emptyset$.

The results obtained so far in this section apply to the repair program $\Pi(r)$. Nevertheless, when we add an arbitrary query program $\Pi(\alpha)$ to $\Pi(r)$, then it is possible that the new core properly extends the well-founded interpretation of the extended program, even for FDs.

Example 10
Consider $r = \{P(a,b), P(a,c)\}$, with the FD, $P(X,Y)\colon\ X \rightarrow Y$, and the query $\alpha(x)\colon\ \exists y\ P(x,y)$. The combined Π program is:

$\mathit{dom}(a).\quad \mathit{dom}(b).\quad \mathit{dom}(c).\quad P(a,b).\quad P(a,c).$
$\mathit{Answer}(X) \leftarrow P'(X,Y)$
$P'(X,Y) \leftarrow P(X,Y), \mathit{not}\ \neg P'(X,Y).$
$\neg P'(X,Y) \leftarrow \mathit{dom}(X), \mathit{dom}(Y), \mathit{not}\ P(X,Y), \mathit{not}\ P'(X,Y).$
$\neg P'(X,Y) \vee \neg P'(X,Z) \leftarrow P(X,Y), P(X,Z), Y \neq Z.$
$\neg P'(X,Y) \leftarrow \mathit{dom}(Y), P'(X,Z), Y \neq Z.$

The answer sets are $S_1 = \{\mathit{Answer}(a), P'(a,b), P(a,b), P(a,c), \ldots\}$ and $S_2 = \{\mathit{Answer}(a), P'(a,c), P(a,b), P(a,c), \ldots\}$. The well-founded interpretation is $W_\Pi = \langle W^+, W^-, W^u\rangle$, with $W^+ = \{P(a,b), P(a,c), \mathit{dom}(a), \ldots\}$, $W^- = \{\neg P'(a,a), \ldots\}$, and $W^u = \{P'(a,b), P'(a,c), \mathit{Answer}(a)\}$. In particular, $\mathit{Answer}(a) \in \mathit{Core}(\Pi)$, but $\mathit{Answer}(a) \notin W^+$.

We know, by complexity results presented in Arenas *et al.*(2001) for functional dependencies that, unless $P = NP$, consistent answers to first-order queries cannot be computed in polynomial time. In consequence, we cannot expect to compute $\mathit{Core}(\Pi)$ of the program that includes the query program by means of the well-founded interpretation of Π alone.

8 An alternative semantics

As discussed in Arenas *et al.*(1999), our database repairs can be obtained as the revision models corresponding to the "possible model approach" introduced in Winslett (1988) and Chou and Winslett (1994) in the context of belief update. When the database instance (a model) is updated by the set of ICs, a new set of models is generated, the database repairs. Winslett's revision models, as our repairs, are based on minimal *set* of changes wrt the original model.

8.1 Cardinality-based repairs and weak constraints

In Dalal (1988), again in the context of belief revision/update, an alternative notion of revision model based on minimal *number* of changes is introduced.

Definition 7
Given a database instance r, an instance r' is a *Dalal repair* of r wrt IC iff $r' \models IC$ and $|\Delta(r,r')|$ is a minimal element of $\{|\Delta(r,r^*)| \mid r^* \models IC\}$. □

We could give a definition of *Dalal consistent answer* exactly in the terms of Definition 1, but replacing "repair" by "Dalal repair". We can also specify Dalal repairs using the same repair programs we had in section 4, but with the persistence defaults replaced by *weak constraints* (Buccafurri *et al.* 2000). The latter will not be imposed on the original database, but rather on the answer sets of the change program, $\Pi_\Delta(r)$, that is responsible for the changes, and was introduced in section 4.1.1.

Weak constraints are of the form $\Leftarrow L_1, \ldots, L_k, \mathit{not}\ L_{k+1}, \ldots, \mathit{not}\ L_n$, where the L_i's are literals. These constraints are added to an extended disjunctive program, with the effect that only those answer sets that minimize the *number* of violated ground instantiations of the weak constraints are kept.

In order to capture Dalal repairs, we need very simple weak constraint. The program $\Pi^D(r)$ that specifies the Dalal repairs of a database instance r wrt a set of BICs consists of program $\Pi_\Delta(r)$ of section 4.1.1 (rules 1–3) plus

4". For every database predicate p, the weak constraints

$$\begin{aligned} &\Leftarrow\quad p'(\bar{X}),\ \mathit{not}\ p(\bar{X}), \\ &\Leftarrow\quad \neg p'(\bar{X}),\ p(\bar{X}). \end{aligned} \tag{5}$$

These constraints say that the original database and a repair are expected to coincide. Since they are weak constraints, they allow violations, but only a minimum *n*umber of tuples that belong to the repair and not to the original instance, or the other way around, are be accepted.

The results for the change program $\Pi_\Delta(r)$ still hold here. In consequence, the program $\Pi^D(r)$ will have answers sets that correspond to repairs that are minimal both under set inclusion and number of changes, i.e. only answer sets corresponding to Dalal repairs.

Example 11
Let $D = \{a\}$, $r = \{p(a)\}$ and $IC = \{\neg p(x) \vee q(x), \neg q(x) \vee r(x)\}$. $\Pi^D(r)$ contains

Facts: $\mathit{dom}(a).\quad p(a).$

Triggering exceptions: $\neg p'(X) \vee q'(X) \longleftarrow p(X),\ \mathit{not}\ q(X)$
$\neg q'(X) \vee r'(X) \longleftarrow q(X),\ \mathit{not}\ r(X)$

Stabilizing exceptions: $q'(X) \longleftarrow p'(X);\quad \neg p'(X) \longleftarrow \neg q'(X)$
$r'(X) \longleftarrow q'(X);\quad \neg q'(X) \longleftarrow \neg r'(X)$

Weak constraints: $\Leftarrow p'(X),\ \mathit{not}\ p(X);\quad \Leftarrow q'(X),\ \mathit{not}\ q(X);\quad \Leftarrow r'(X),\ \mathit{not}\ r(X);$
$\Leftarrow \neg p'(X), p(X);\quad \Leftarrow \neg q'(X), q(X);\quad \Leftarrow \neg r'(X), r(X).$

Weak constraints are implemented in *DLV*[7], that run on this program returns the answer set $\{\mathit{dom}(a), p(a), \neg p'(a)\}$, corresponding to the empty database repair, but not the other Winslett's repair $\{\mathit{dom}(a), p(a), q'(a), r'(a)\}$, whose set of changes wrt r has two elements, whereas the first repair differs from r by one change only.

[5] By means of its option `-det`.
[6] See Dantsin *et al.* (2001) for a review of complexity results in logic programming.
[7] They are specified by `:~` *Conj*., where *Conj* is a conjunction of (possibly negated) literals. See *DLV*'s user manual in http://www.dbai.tuwien.ac.at/proj/dlv/man.

Notice that in Example 11, from the change program $\Pi_\Delta(r)$, without the weak constraints, we obtain the eventually discarded answer set $\{dom(a), p(a), q'(a), r'(a)\}$, that only implicitly contains $p'(a)$. The reason is that now we do not have the persistence rules that cause $p(a)$ to persist in the database as $p'(a)$. In consequence, we have to interpret these answer sets to establish the correspondence between them and the repairs. This is done in Theorem 3 via the interpretation I of Definition 4. In consequence, for BICs and finite domain databases we have

Theorem 3
Given a (finite domain) database instance r and a set of BICs IC:

1. For every Dalal repair r' of r wrt IC, there exists an answer set S of $\Pi^D(r)$ such that $I(S) = r'$.
2. For every answer set S of $\Pi^D(r)$, there exists a Dalal repair r' of r wrt IC such that $I(S) = r'$. □

As with Winslett's repairs, the theorem still holds for infinite domain databases when the BICs are domain independent.

Instead of interpreting the answer sets due to the only implicit presence of primed literals caused by the lack of persistence defaults, when we pose queries expecting consistent answers, we may transform the original query according to the following table:

original query	query in the program
$p(\bar{x})$	$\mathtt{query_p}(\bar{\mathtt{X}}) \leftarrow \mathtt{p}'(\bar{\mathtt{X}})$. $\mathtt{query_p}(\bar{\mathtt{X}}) \leftarrow \mathtt{p}(\bar{\mathtt{X}}),\ \mathtt{not}\ \neg\mathtt{p}'(\bar{\mathtt{X}})$.
$\neg p(\bar{x})$	$\mathtt{query_{\neg p}}(\bar{\mathtt{X}}) \leftarrow \neg\mathtt{p}'(\bar{\mathtt{X}})$. $\mathtt{query_{\neg p}}(\bar{\mathtt{X}}) \leftarrow dom(\bar{\mathtt{X}}),\ \mathtt{not\ p}(\bar{\mathtt{X}}),\ \mathtt{not\ p}'(\bar{\mathtt{X}})$.

That is, everywhere in the original query, we replace p and $\neg p$ by $query_p$, and $query_{\neg p}$, respectively, and we add the rules on the right-hand side of the table.

Finally, as an alternative, we could avoid interpreting answer sets or transforming queries, and explicitly obtain the Dalal repairs, by imposing the weak constraints on the repair program $\Pi(r)$, that contains the default rules.

9 Conclusions

We have presented a general methodology to consistently answer first order queries posed to relational databases that violate given ICs. We have restricted ourselves mainly to the case of binary integrity contraints, i.e. universal ICs containing at most two database literals. However the methodology can be extended to universal ICs with a larger number of database literals. Facts, persistence and triggering exceptions rules are as before, but the number of stabilizing rules grows according to the number of subsets of database literals in each IC. We sketch the solution by means of an example.

Example 12

Consider $r = \{P(a), Q(a), R(a)\}$ and the ternary integrity constraints $IC = \{\neg P(x) \vee \neg Q(x) \vee R(x), \neg P(x) \vee \neg Q(x) \vee \neg R(x), \neg P(x) \vee Q(x) \vee \neg R(x), P(x) \vee \neg Q(x) \vee \neg R(x), \neg P(x) \vee Q(x) \vee R(x), P(x) \vee \neg Q(x) \vee R(x), P(x) \vee Q(x) \vee \neg R(x)\}$. The repair program $\Pi(r)$ contains the usual persistence defaults for P, Q, R, and triggering exception rules, e.g. for the first IC in IC:

$$\neg P'(x) \vee \neg Q'(x) \vee R'(x) \quad \leftarrow \quad P(x), Q(x), not\ R(x).$$

We also need the stabilizing rules, e.g. for the first IC

$$\begin{aligned} \neg P'(x) \vee \neg Q'(x) &\quad \leftarrow \quad \neg R'(x), \\ \neg P'(x) \vee R'(x) &\quad \leftarrow \quad Q'(x), \\ \neg Q'(x) \vee R'(x) &\quad \leftarrow \quad P'(x); \end{aligned} \tag{6}$$

but also for the first IC:

$$\begin{aligned} \neg P'(x) &\quad \leftarrow \quad Q'(x), \neg R'(x), \\ R'(x) &\quad \leftarrow \quad P'(x), Q'(x), \\ \neg Q'(x) &\quad \leftarrow \quad P'(x), \neg R'(x). \end{aligned} \tag{7}$$

In this case we obtain as answer set the only repair, namely the empty instance, represented by $\{P(a), Q(a), R(a), \neg P'(a), \neg Q'(a), \neg R'(a)\}$. Using rules (7) as the only stabilizing rules, without using the disjunctive stabilizing rules (6), the empty repair cannot be obtained.

Extending the current methodology to relational databases with view definitions should be straightforward.

9.1 Ongoing and future work

There are several open issues that deserve further investigation, among them: (a) Analyze conditions under which simpler and optimized programs can be obtained; (b) a more detailed treatment of referential ICs (and other existential ICs); (c) identification of other classes of ICs for which the well-founded interpretation and the intersection of all database repairs coincide; and (d) representation of preferences for certain kinds of repair actions. In principle, the preferences could be captured by choosing the right disjuncts in the triggering rules.

The approach to CQA is based on the specification of all repairs, where each of them completely restores the consistency of the database, independently from the query that is posed and from the fact that it might have nothing to do with some of the violated ICs. This approach work well if the repairs are stored and different queries are posed after that. However, it would be useful to specify and compute "repairs" that partially restore the consistency of the database, only wrt the ICs that are relevant to the query. Possibly appropriate grounding techniques could be used in this case.

The repair programs we presented materialize the closed-world assumption by explicitly producing the negative primed literals. This is due to the persistence

default rules. In practical applications this should and could be avoided by restoring, via the program, the *implicit* closed world assumption applied to the repairs.

We have not addressed the problem of obtaining query answers to general $\mathcal{K}$-queries. The method we presented for basic $\mathcal{K}$-queries needs to be combined with some method of evaluating first-order queries. For example, safe-range first-order queries (Abiteboul *et al.* 1995) can be translated to relational algebra. The same approach can be used for $\mathcal{K}$ queries with the subqueries of the form $\mathcal{K}\alpha$ replaced by new relation symbols. Then when the resulting relational algebra query is evaluated and the need arises to materialize one of the new relations, the above method can be used to accomplish that goal.

There are several interesting open issues related to computational implementation of the methodology we have presented.

The existing implementations of stable models semantics are based on grounding the rules, what, in database applications, may lead to huge ground programs. Some "intelligent" grounding techniques have been implemented in *DLV*. Furthermore, those implementations are geared to computing stable models, possibly only one or some of them, whereas consistent query answering requires, at least implicitly, having all stable models, or the "relevant parts" of all of them. In particular, this opens the interesting issue of having the construction of (the relevant parts of) the stable models guided by the query, because query answering is our primary goal, but not the computation of repairs. Current query evaluation methodologies under the stable model semantics, specially for disjunctive programs, are completely insensitive to the query at hand. The goal is to avoid irrelevant computations.

In database applications, posing and answering open queries (with variables) is more natural and common that answering ground queries. However, existing implementations of stable model semantics are better designed to do the latter.

It would be useful to implement a consistent query answering system based on the interaction of our repairs logic programs with relational DBMS. For this purpose, some functionalities and front-ends included in *DLV*'s architecture (Eiter *et al.* 2000) could be used. Trying to push most of the computation to the DBMS seems to be the right way to proceed.

9.2 Related work

Work on inconsistency handling has been done for long time and by different communities, e.g. philosophical and non-classical logic, knowledge representation, logic programming, databases, software specification, etc. We mention only some related work that has, or may have, some relation to our notions of repair and consistent answer, or are based on some form of logic programming.

There are several similarities between our approach to consistency handling and those followed by the belief revision/update community. As already mentioned in section 8, database repairs coincide with the revised models defined in Winslett (1988). The treatment there is mainly propositional, but a preliminary extension to first order knowledge bases can be found in Chou and Winslett (1994). Those papers concentrate on the computation of the models of the revised theory, i.e. the

repairs in our case, but not on query answering. The revision of a database instance by the ICs produces new database instances, the repairs of the original database.

Nevertheless, our motivation and starting point are quite different from those of belief revision. We are not interested in computing the repairs *per se*, but in answering queries, hopefully using the original database as much as possible. If this is not possible, we look for methodologies, as our logic programming approach, for representing and querying simultaneously and implicitly all the repairs of the database.

Bry (1997) was, to our knowledge, the first author to consider the notion of consistent query answer in inconsistent databases. He defined consistent query answers using provability in minimal logic. The proposed inference method is nonmonotonic but fails to capture minimal change (thus Bry's notion of consistent query answer is weaker than ours). Moreover, Bry's approach is entirely proof-theoretic and does not provide a computational mechanism to obtain consistent answers to first-order queries.

Several papers studied the problem of making inferences from a possibly inconsistent, propositional or first-order, knowledge base. The basic idea is to infer the classical consequences of all maximal consistent subsets of the knowledge base (Lozinskii 1994; Baral *et al.* 1991), or all *most consistent* models of the knowledge base (Kifer and Lozinskii 1992; Arieli and avron 1999) (where the order on models is defined on the base of atom annotations drawing values from a lattice or a bilattice). This provides a non-monotonic consequence relation but the special role of the integrity constraints (whose truth cannot be given up) is not captured. Also, the issue of processing general first-order queries is not considered.

Now we briefly review specification and logic programming based approaches to consistency handling in databases. In this direction, the closest approach to ours was presented, independently, in Greco *et al.* (2001) (see also Greco and Zumpano 2000, 2001). There, disjunctive programs are used to specify the minimal sets of changes, under set inclusion, that lead to database repairs in the sense of Arenas *et al.* (1999). The authors present a compact schema for generating repair programs for general universal integrity constraints. The application of such a schema leads to programs that involve essentially all possible disjunctions of database literals in the heads, ending up with programs like the one in Example 12. They concentrate mainly on producing the set of changes, rather than the repaired databases explicitly. In particular, they do no have persistence rules in the program. In consequence, the program cannot be used directly to obtain consistent answers. An interpretation of the results, possibly like the one introduced in section 8 would be necessary. They also introduce "repair constraints" to specify preferences for certain kinds of repairs.

The annotated predicate logic introduced in Kiffer and Lozinskii (1992) was applied in Arenas *et al.* (2000) to the task of computing consistent query answers via a specification of the database repairs. The specification was used to derive algorithms for consistently answering some restricted forms of first order queries and to obtain some complexity results. As expected, the database repairs correspond to certain minimal models of the specification. This approach is based on a nonclassical logic, and computing consistent answers from it is not straightforward. The specification methodology was extended from universal ICs to referential ICs in Barcelo and Bertossi (2002).

There are several proposals for language constructs extending stratified Datalog programs with the purpose of specifying nondeterministic queries. Essentially, the idea is to construct a maximal subset of a given relation that satisfies a given set of functional dependencies. Since there is usually more than one such subset, the approach yields nondeterministic queries in a natural way. Clearly, maximal consistent subsets, choice models in Giannotti *et al.* (1997), correspond to our repairs in the case of functional dependencies. Stratified Datalog with choice (Giannotti *et al.* 1997) combines enforcing functional dependencies with inference using stratified Datalog programs. Answering queries in all choice models ($\forall G$-queries (Greco *et al.* 1995)) corresponds to our notion of computation of consistent query answers for first-order queries.

The *revision programs* (Marek and Truszczynski 1998) are logic programs for updating databases, and could be used to restore consistency, and then to compute database repairs. The rules in those programs allow explicitly declaring how to enforce the satisfaction of an integrity constraint, rather than explicitly stating the ICs, e.g. $in(a) \leftarrow in(a_1), \ldots, in(a_k), out(b_1), \ldots, out(b_m)$ has the intended procedural meaning of inserting the database atom a whenever $a_1, \ldots, a_k$ are in the database, but not $b_1, \ldots, b_m$. They also give a declarative, stable model semantics to revision programs. Preferences for certain kinds of repair actions can be captured by declaring the corresponding rules in program and omitting rules that could lead to other forms of repairs. Revision programs could be used, as the programs in Greco *et al.* (2001), to obtain consistent answers, but not directly, because they give an account of changes only.

Blair and Subrahmanian (1989) introduced *paraconsistent logic programs.* They have a non-classical semantics, inspired by paraconsistent first-order semantics. In Kifer and Subrahmanian (1992), general annotated logic programs are presented. Their lattice-based semantics is also non-classical. Atoms in clauses have annotations, as in Kifer and Lozinskii (1992), but now annotations may also contain variables and functions, providing a stronger representation formalism. Implementation of annotated logic programs and query answering mechanisms are discussed in Leach and Lu (1996). In Subrahmanian (1994), annotated programs are further generalized, in order to be used for amalgamating databases, resolving potential conflicts between integrated data. For this purpose the product of the lattices underlying each database is constructed as the semantic basis for the integrated database. Conflict resolutions and preferences are captured by means of function-based annotations. Other approaches to paraconsistent logic programming are discussed in Damasio and Moniz-Pereira (1998).

In Barcelo and Bertossi (2002, 2003), starting from the lattice and specification introduced in Arenas *et al.* (2000b), logic programs containing annotations as arguments (as opposed to annotated programs that contain annotated atoms) were used to specify database repairs and compute consistent answers to queries. The approach works for general universal ICs and referential ICs. The logic programs

have stable model semantics. The cost of using annotations as extra arguments in the program is balanced by the fact that the program contains only a linear number of rules, what is not the case if the number of literals per IC grows beyond two (see Example 12). In consequence, for ICs that are non binary, the approach in Barcelo and Bertossi (2003) should be more convenient.

Acknowledgments

Work supported by FONDECYT Grant 1000593, NSF Grant INT-9901877/CONICYT Grant 1998-02-083, NSF Grant IIS-0119186, Carleton University Start-Up Grant 9364-01, NSERC Grant 250279-02. L. Bertossi holds a Faculty Fellowship of the Center for Advanced Studies, IBM Toronto Lab. We are grateful to Francisco Orchard for informative paper presentations, discussions, and experiments with *DLV*. We are grateful to Nicola Leone for kindly answering all our questions about *DLV*. We appreciate comments received from anonymous reviewers, that have helped us to substantially improve the presentation.

Appendix: Proofs

Proof of Proposition 1

Consider an arbitrary element in IC. Assume that this element is of the form $p(\bar{x}) \vee \neg q(\bar{y}) \vee \varphi$ (the proof is analogous for binary constraints containing either two positive literals or two negative literals). We have to prove that $I(S)$ satisfies any instantiation of this formula, say $p(\bar{a}) \vee \neg q(\bar{b}) \vee \varphi$. We consider two cases.

(I) If r does not satisfy this ground constraint, then S satisfies the body of the ground triggering rule: $p'(\bar{a}) \vee \neg q'(\bar{b}) \leftarrow dom(\bar{a}),\ not\, p(\bar{a}),\ q(\bar{b}),\ \overline{\varphi}$. Thus, $p'(\bar{a}) \in S$ or $\neg q'(\bar{b}) \in S$. If $p'(\bar{a}) \in S$, then $I(S) \models p(\bar{a})$, and if $\neg q'(\bar{b}) \in S$, then $q'(\bar{b}) \notin S$ and, therefore, $I(S) \models \neg q(\bar{b})$. In any case, $I(S) \models p(\bar{a}) \vee \neg q(\bar{b}) \vee \varphi$.

(II) If r satisfies the ground constraint, then r satisfies φ, $p(\bar{a})$ or $\neg q(\bar{b})$. In the first case, $I(S) \models \varphi$ and, therefore, $I(S) \models p(\bar{a}) \vee \neg q(\bar{b}) \vee \varphi$. Thus, assume that $r \not\models \varphi$ and $r \models p(\bar{a})$ or $r \models \neg q(\bar{b})$.
By contradiction, assume that $I(S) \not\models p(\bar{a}) \vee \neg q(\bar{b})$. If $r \models p(\bar{a})$, then $p(\bar{a}) \in S$ and, therefore, $\neg p'(\bar{a}) \in S$, by definition of $I(S)$. But in this case S satisfies the body of the ground stabilizing rule: $\neg q'(\bar{b}) \leftarrow dom(\bar{b}),\ \neg p'(\bar{a}),\ \overline{\varphi}.$ and, therefore, $\neg q'(\bar{b}) \in S$. We conclude that $I(S) \models \neg q(\bar{b})$, a contradiction. If $r \models \neg q(\bar{b})$, then $\neg q(\bar{b}) \notin S$ and, therefore, $q'(\bar{b}) \in S$, by definition of $I(S)$. But in this case S satisfies the body of the ground stabilizing rule: $p'(\bar{a}) \leftarrow dom(\bar{a}),\ q'(\bar{b}),\ \overline{\varphi}.$ and, hence, $p'(\bar{a}) \in S$. We conclude that $I(S) \models p(\bar{a})$, again a contradiction.

Proof of Proposition 2

To prove that $S(r, r')$ satisfies $\Pi_\Delta(r)$, we need to consider only the four different types of ground stabilizing rules (the satisfaction of the other rules follows from the fact that r' satisfies IC).

If $S(r, r')$ satisfies the body of the rule $q'(\bar{b}) \leftarrow dom(\bar{b}),\ p'(\bar{a}),\ \overline{\varphi}$, then r' must satisfy $dom(\bar{b})$, $p(\bar{a})$ and $\neg\varphi$. But $r' \models q(\bar{b}) \vee \neg p(\bar{a}) \vee \varphi$, since $\forall\bar{x}\forall\bar{y}(q(\bar{x}) \vee \neg p(\bar{y}) \vee \varphi) \in IC$, and, therefore, $r' \models q(\bar{b})$. Thus, $q'(\bar{b}) \in S(r, r')$.

Analogously, it is possible to prove that $S(r, r')$ satisfies the remaining types of ground stabilizing rules. □

Proof of Proposition 3

From the previous proposition, we know that the change program has models; so it is a consistent program. If the program has a consistent (i.e. non trivial) answer set, they are all consistent (Lifschitz and Turner 1994). Now we show how to obtain such consistent answer sets. The program can be split into two subprograms (Lifschitz and Turner 1994). The first one contains the domain and database facts plus the rules $p^\star(\bar{X}) \leftarrow not\ p(\bar{X})$. The second one containing the stabilizing rules and the triggering rules modified by replacing the literals of the form $not\ p$ in the bodies by by $p^\star$.

The first program is stratified and has one (consistent) answer set. The second subprogram does not contain weak negation, it is a positive program in that sense, and its minimal models coincide with its answer sets. By a result in Lifschitz and Turner (1994), the original program has as answer sets the unions of the answer sets of the first program and the answer sets of the second one, where the atoms $p^\star$ are treated as extensional database predicates for the computation of the answer sets of the second subprogram. □

Proof of Proposition 4

Let S'_M be the set added to S_M. It is easy to verify that ${}^S\Pi(r) \supseteq^{S_M} \Pi_\Delta(r)$. Then, since S_M is an answer set of $\Pi_\Delta(r)$, to prove that S is an answer set of $\Pi(r)$, it suffices to prove (I) and (II) below.

(I) $S'_M \subseteq \cap\alpha({}^S\Pi(r))$. Let $l(\bar{a})$ be an element of S'_M. If $l(\bar{a}) = p'(\bar{a})$, then $p(\bar{a}) \in S_M$ and $\neg p'(\bar{a}) \notin S_M$, and, therefore, $p(\bar{a})$ and $p'(\bar{a}) \leftarrow p(\bar{a})$ are rules in ${}^S\Pi(r)$. Thus, $p'(\bar{a})$ is in $\cap\alpha({}^S\Pi(r))$. If $l(\bar{a}) = \neg p'(\bar{a})$, then $p(\bar{a}) \notin S_M$ and $p'(\bar{a}) \notin S_M$, and, therefore, $\neg p'(\bar{a}) \leftarrow dom(\bar{a})$ is a reduced ground persistence rule in ${}^S\Pi(r)$. Thus, $\neg p'(\bar{a})$ is in $\cap\alpha({}^S\Pi(r))$.

(II) From S'_M is not possible to deduce an element that is not in S by using the stabilizing rules.
Assume that $q'(\bar{Y}) \leftarrow dom(\bar{Y}),\ p'(\bar{X}),\ \overline{\varphi}$ is a rule in $\Pi_\Delta(r)$, and $q'(\bar{b}) \leftarrow dom(\bar{b}),\ p'(\bar{a})$ is a rule in ${}^S\Pi(r)$. If $p'(\bar{a}) \in S'_M$, we need to show that $q'(\bar{b}) \in S$. By contradiction, suppose that $q'(\bar{b}) \notin S$. Then $q'(\bar{b}) \notin S_M$ and $q'(\bar{b}) \notin S'_M$, and, therefore, $q(\bar{b}) \notin S_M$ or $\neg q'(\bar{b}) \in S_M$, by definition of S'_M. If $q(\bar{b})$ is not in S_M, then given that $p'(\bar{a}) \in S'_M$, S_M satisfies the body of the rule: $q'(\bar{b}) \vee \neg p'(\bar{a}) \leftarrow dom(\bar{b}),\ p(\bar{a}),\ not\ q(\bar{b}),\ \overline{\varphi}$. But, this implies that $q'(\bar{b}) \in S_M$, a contradiction, or $\neg p'(\bar{a}) \in S_M$, also a contradiction (since $p'(\bar{a}) \in S'_M$). Otherwise, if $\neg q'(\bar{b}) \in S_M$, then by using the rule $\neg p'(\bar{a}) \leftarrow dom(\bar{a}),\ \neg q'(\bar{b})$, we can conclude that $\neg p'(\bar{a})$ is in S_M, a contradiction.

Analogously, it is possible to prove the same property for any other type of stabilizing rule.

Proof of Lemma 1

Let S be a answer set of $\Pi_\Delta(r)$ such that S is a subset of $S(r, r')$. First, we prove that $\Delta(r, I(S)) \subseteq \Delta(r, r')$. If $p(\bar{a}) \in \Delta(r, I(S))$, then one of the following cases holds.

(I) $r \models p(\bar{a})$ and $I(S) \not\models p(\bar{a})$. In this case, $p(\bar{a}) \in S$ and $p'(\bar{a}) \notin S$. Thus, by definition of $I(S)$ we conclude that $\neg p'(\bar{a}) \in S$ and, therefore, $\neg p'(\bar{a}) \in S(r, r')$. But this implies that $r' \not\models p(\bar{a})$. Thus, $p(\bar{a}) \in \Delta(r, r')$.

(II) $r \not\models p(\bar{a})$ and $I(S) \models p(\bar{a})$. In this case, $p(\bar{a}) \notin S$ (S is a minimal model and $p(a)$ does not need to be in S if it was not in r). Thus, by definition of $I(S)$ we conclude that $p'(\bar{a}) \in S$ and, therefore, $p'(\bar{a}) \in S(r, r')$. But this implies that $r' \models p(\bar{a})$. Thus, $p(\bar{a}) \in \Delta(r, r')$.

Hence, $\Delta(r, I(S)) \subseteq \Delta(r, r')$. But, by Proposition 1, $I(S)$ satisfies IC, and therefore, $\Delta(r, I(S))$ must be equal to $\Delta(r, r')$, since $\Delta(r, r')$ is minimal under set inclusion in $\{\Delta(r, r^*) \mid r^* \models IC\}$. Then, we conclude that $I(S) = r'$. □

Proof of Theorem 1

We shall prove the first part of this theorem. The second one can be proved analogously.

Given a repair r' of r, by Lemma 1, $r' = I(S_M)$, where S_M is an answer set of $\Pi_\Delta(r)$, with $S_M \subseteq S(r, r')$. Define S from S_M as in Proposition 4. Then, S is an answer set of $\Pi(r)$. By construction of S, $I(S) = I(S_M)$. Furthermore, $I(S) = \{p(a) \mid p'(a) \in S\}$. □

Proof of Proposition 5

Since it is always the case that $W_{\Pi(r)} \subsetneqq Core(\Pi(r))$ (Leone *et al.* 1997), we only need to show that $Core(\Pi(r)) \subsetneqq W_{\Pi(r)}$. In consequence, it is necessary to check that whenever a literal $(\neg)p'(a)$ belongs to $Core(\Pi(r))$, where a is tuple of elements in the domain D and p is a database predicate, $(\neg)p'(a)$ can be fetched into $\mathcal{W}^n_{\Pi(r)}(\emptyset)$ for some finite integer n.

For each literal L in the original database r, and its primed version L' and each answer set S, either L' or its complement $\overline{L'} \in S$.[8] We will do the proof by cases, considering for a literal $L' : (\neg)p'(a)$ contained in $Core(\Pi(r))$ all the possible transitions from the original instance to the core: (a) negative to positive, i.e. $\neg p(a) \in r$, and $p'(a) \in Core(\Pi(r))$, (b) positive to positive. (c) negative to negative. (d) positive to negative. We will prove only the first two cases, the other two are similar. For each case, again several cases have to be verified according to the different ground program rules that could have made $p'(a)$ get into $Core(\Pi(r))$.

(a) Assume $p'(a) \in Core(\Pi(r))$, and $p(a) \notin r$. To prove: $p'(a) \in W_{\Pi(r)}$.

[8] Actually only positive literals appear in r, but we are invoking the CWA. All the literals in the original instance will belong to $Core(\Pi(r))$.

Since FDs can only produce deletions $p'(a)$ has to be true due to an unary constraint that was false for $p(a)$: $(p(a) \vee \varphi(a)) \in IC_D$ is false, with $\varphi(a)$ is false, where IC_D is the instantiation of the ICs in the domain D. In the ground program we find the rule $p'(a) \longleftarrow dom(a), \neg\varphi(a)$. The second subgoal becomes true of $\emptyset$. Since $dom(a) \in \mathcal{W}^1_{\Pi(r)}(\emptyset)$, we obtain $p'(a) \in \mathcal{W}^2_{\Pi(r)}(\emptyset)$.

(b) Assume that $p'(a) \in Core(\Pi(r))$ and $p(a) \in r$.
This means that $p(a)$ persisted from the original instance to every answer set.

1. There is $(p(a) \vee \varphi(a)) \in IC_D$ with $\varphi(a)$ false. Then the ground program has a rule $p'(a) \longleftarrow dom(a), \neg\varphi(a)$. The body becomes true, $dom(a)$ gets into WFS after the first step, then, as in case (a), $p'(a) \in \mathcal{W}^2_{\Pi(r)}(\emptyset)$.
2. There is no ground constraint as in item 1, i.e. there is no $(p(a) \vee \varphi(a)) \in IC_D$ or the $\varphi(a)$'s are true. In this case, there is no applicable rule of the form $p'(a) \longleftarrow dom(a), \neg\varphi(a)$ in the ground program.
Since rules associated to FDs delete tuples only, we must have obtained $p'(a)$ via a default rule $p'(a) \longleftarrow dom(a), p(a), not\ \neg p'(a)$ and the unfoundedness of $\neg p'(a)$ in the ground program. If the $\mathcal{W}_{\Pi(r)}$ operator declares $\neg p'(a)$ unfounded, then $p'(a)$ will belong to $W_{\Pi(r)}$. So, we have to concentrate on the unfoundedness of $\neg p'(a)$.
(a) We can never get $\neg p'(a)$ from rules of the form $\neg p'(a) \longleftarrow dom(a), \neg\psi(a)$, obtained from unary ICs. If this were the case, we would have $\neg p'(a) \in Core(\Pi(r))$, what is not possible, since $p'(a) \in Core(\Pi(r))$.
(b) $\neg p'(a)$ cannot be obtained via the default rule

$$\neg p'(a) \longleftarrow dom(a), not\ p(a), not\ p'(a),$$

because it has the second subgoal false.
(c) $\neg p'(a)$ cannot be obtained via a possible unfoundedness of $p'(a)$, because $p'(a)$ belongs to answer sets.
(d) We are left with rules associated to FDs. Assume that

$$(\neg p(a) \vee \neg p(b) \vee c = d) \in IC_D. \quad (8)$$

i If $c = d$, the associated triggering rule $\neg p'(a) \vee \neg p'(b) \longleftarrow p(a), p(b), c \neq d$. cannot be applied.

ii If $c \neq d$, then in principle the triggering rule could be applied, but since $p'(a)$ belongs to all answer sets, without being forced to by a unary ICs, it must be case that $p(b)$ is false (otherwise, some repairs would get $p'(a)$ and others $p'(b)$, but not $p'(a)$).
For the same reason, there is no $(p(b) \vee \chi(b)) \in IC_D$ with $\chi(b)$ false, because this would force $p'(b)$ to be true via the corresponding triggering rule; and this in its turn would force $\neg p'(a)$ to be true (to be in every answer set) due to the FD. This is not possible, because $p'(a)$ is already in $Core(\Pi(r))$.
In consequence, the rule

$$\neg p'(a) \vee \neg p'(b) \longleftarrow p(a), p(b), c \neq d$$

cannot be applied.

Now, we have to analyze the stabilizing rule $\neg p'(a) \longleftarrow p'(b), c \neq d$ associated to (8).

i If $c = d$, the rule does not apply.

ii If $c \neq d$, we have (as above) $p(b) \notin r$. Then, $\neg p(b) \in \mathcal{W}^1_{\Pi(r)}(\emptyset)$.
Furthermore, $p'(b)$ cannot be obtained from the default $p'(b) \longleftarrow dom(b), p(b), not\ \neg p'(b)$, because $p(b)$ is false. We already saw that $p'(b)$ cannot be obtained from a rule $p'(b) \longleftarrow dom(b), \neg\chi(b)$.

In consequence, $p'(b)$ is unfounded, i.e. $\neg p'(b) \in \mathcal{W}^2_{\Pi(r)}(\emptyset)$, then, from the stabilizing rule, $\neg p'(a)$ turns out to be unfounded too: $p'(a) \in \mathcal{W}^3_{\Pi(r)}(\emptyset)$.

The two remaining cases that we will not prove are: (c) $p(a) \notin r$ and $\neg p'(a) \in Core(\Pi(r))$; (d) $p(a) \in r$ and $\neg p'(a) \in Core(\Pi(r))$. It is possible to show that always $Core(\Pi(r)) \subseteq \mathcal{W}^3_{\Pi(r)}(\emptyset)$. □

References

ABITEBOUL, S., HULL, R. AND VIANU, V. 1995. *Foundations of Databases.* Addison-Wesley.

ARENAS, M., BERTOSSI, L. AND CHOMICKI, J. 1999. Consistent query answers in inconsistent databases. *Proceedings ACM Symposium on Principles of Database Systems (ACM PODS'99)*, pp. 68–79. ACM Press.

ARENAS, M., BERTOSSI, L. AND CHOMICKI, J. 2000. Specifying and querying database repairs using logic programs with exceptions. In: H. L. Larsen, J. Kacprzyk, S. Zadrozny and H. Christiansen (Eds.), *Flexible Query Answering Systems. Recent Developments*, pp. 27–41. Springer.

ARENAS, M., BERTOSSI, L. AND KIFER, M. 2000. Applications of annotated predicate calculus to querying inconsistent databases. *'Computational Logic - CL 2000'. Stream: 6th International Conference on Rules and Objects in Databases (DOOD'2000): LNAI 1861*, pp. 926–941. Springer.

ARENAS, M., BERTOSSI, L. AND CHOMICKI, J. 2001. Scalar aggregation in FD-inconsistent databases. *Database Theory – ICDT 2001 (Proceedings International Conference on Database Theory, ICDT'2001): LNCS 1973*, pp. 39 – 53. Springer.

ARIELI, O. AND AVRON, A. 1999. A model-theoretic approach for recovering consistent data from inconsistent knowledge bases. *Journal of Automated Reasoning 22*, 2, 263–309.

BARAL, C., MINKER, J. AND KRAUS, S. 1991. Combining multiple knowledge bases. *IEEE Transactions on Knowledge and Data Engineering 3*, 2, 208–221.

BARCELO, P. AND BERTOSSI, L. 2002. Repairing databases with annotated predicate logic. *Proceedings Ninth International Workshop on Non-Monotonic Reasoning (NMR'2002)*, pp. 160–170. Morgan Kaufmann.

BARCELO, P. AND BERTOSSI, L. 2003. Logic programs for querying inconsistent databases. *Proceedings Fifth International Symposium on Practical Aspects of Declarative Languages (PADL 2003): LNCS 2562*, pp. 208–222. Springer.

BLAIR, H. A. AND SUBRAHMANIAN, V. S. 1989. Paraconsistent logic programming. *Theoretical Computer Science 68*, 135–154.

BRY, F. 1997. Query answering in information systems with integrity constraints. *Proceedings First IFIP TC11 Working Conference on Integrity and Internal Control in Information Systems*, pp. 113-130. Chapman & Hall.

BUCCAFURRI, F., LEONE, N. AND RULLO, P. 2000. Enhancing disjunctive datalog by constraints. *IEEE Transactions on Knowledge and Data Engineering 12*, 5, 845–860.

CALIMERI, F., FABER, W., LEONE, N. AND PFEIFER, G. 2002. Pruning operators for answer set programming systems. In: S. Benferhat and E. Giunchiglia (Eds.), *Proceedings Ninth International Workshop on Non-Monotonic Reasoning (NMR'2002)*, pp. 200–209. Morgan Kaufmann.

CELLE, A. AND BERTOSSI, L. 2000. Querying inconsistent databases: algorithms and implementation. *'Computational Logic – CL 2000'. Stream: 6th International Conference on Rules and Objects in Databases (DOOD'2000): LNAI 1861*, pp. 942–956. Springer.

CHOU, T. AND WINSLETT, M. 1994. A model-based belief revision system. *Journal of Automated Reasoning 12*, 157–208.

DALAL, M. 1988. Investigations into a theory of knowledge base revision: preliminary report. *Proceedings Seventh National Conference on Artificial Intelligence (AAAI'88)*, pp. 475–479.

DAMASIO, C. V. AND MONIZ-PEREIRA, L. 1998. A survey on paraconsistent semantics for extended logic programs. In: D. M. Gabbay and Ph. Smets (Eds.), *Handbook of Defeasible Reasoning and Uncertainty Management Systems, Vol. 2*, pp. 241–320. Kluwer Academic.

DANTSIN, E., EITER, T., GOTTLOB, G. AND VORONKOV, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys 33*, 3, 374–425.

EITER, T., LEONE, N., MATEIS, C., PFEIFER, G. AND SCARCELLO, F. 1998. The knowledge representation system DLV: progress report, comparisons, and benchmarks. *Proceedings International Conference on Principles of Knowledge Representation and Reasoning (KR98)*, Trento, Italy. Morgan Kaufman.

EITER, T., FABER, W., LEONE, N. AND PFEIFER, G. 2000. Declarative problem-solving in DLV. In: J. Minker (Ed.), *Logic-Based Artificial Intelligence*, pp. 79–103. Kluwer.

FITTING, M. 1996. *First Order Logic and Automated Theorem Proving. 2nd ed.* Texts and Monographs in Computer Science, Springer.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In: R. A. Kowalski and K. A. Bowen (Eds.), *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, pp. 1070–1080. MIT Press.

GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 9, 365–385.

GIANNOTTI, F., GRECO, S., SACCA, D. AND ZANIOLO, C. 1997. Programming with non-determinism in deductive databases. *Annals of Mathematics and Artificial Intelligence 19*, 3–4.

GRECO, S., SACCA, D. AND ZANIOLO, C. 1995. Datalog queries with stratified negation and choice: from P to D^P. *Proceedings International Conference on Database Theory*, pp. 82–96. Springer.

GRECO, S. AND ZUMPANO, E. 2000. Querying inconsistent databases. *Proceedings 7th International Conference on Logic for Programming and Automated Reasoning (LPAR'2000): LNCS 1955*, pp. 308–325.

GRECO, S. AND ZUMPANO, E. 2001. Computing repairs for inconsistent databases. *Proceedings Third International Symposium on Cooperative Database Systems for Advanced Applications (CODAS01)*, Beijing, China.

GRECO, G., GRECO, S. AND ZUMPANO, E. 2001. A logic programming approach to the integration, repairing and querying of inconsistent databases. In: Ph. Codognet (Ed.), *Proceedings 17th International Conference on Logic Programming (ICLP'01): LNCS 2237*, pp. 348–364. Springer.

KOWALSKI, R. AND SADRI, F. 1991. Logic programs with exceptions. *New Generation Computing 9*, 387–400.

KIFER, M. AND LOZINSKII, E. L. 1992. A logic for reasoning with inconsistency. *Journal of Automated Reasoning 9*, 2, 179–215.

KIFER, M. AND SUBRAHMANIAN, V. S. 1992. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming 12*, 4, 335–368.

LEACH, S. M. AND LU, J. J. 1996. Query processing in annotated logic programming: theory and implementation. *Journal of Intelligent Information Systems 6*, 33–58.

LEONE, N., RULLO, P. AND SCARCELLO, F. 1997. Disjunctive stable models: unfounded sets, fixpoint semantics, and computation. *Information and Computation 135*, 2, 69–112.

LIFSCHITZ, V. AND TURNER, H. 1994. Splitting a logic program. In: P. van Hentenryck (Ed.), *Proceedings Eleventh International Conference on Logic Programming*, pp. 23-37. MIT Press.

LLOYD, J. W. 1987. *Foundations of Logic Programming.* Springer Verlag.

LOZINSKII, E. L. 1994. Resolving contradictions: a plausible semantics for inconsistent systems. *Journal of Automated Reasoning 12*, 1, 1–32.

MAREK, V. W. AND TRUSZCZYNSKI, M. 1998. Revision programming. *Theoretical Computer Science 190*, 2, 241–277.

SUBRAHMANIAN V. S. 1994. Amalgamating knowledge bases. *ACM Transactions on Database Systems 19*, 2, 291–331.

ULLMAN, J. 1988. *Principles of Database and Knowledge-Base Systems, Vol. I.* Computer Science Press.

WINSLETT, M. 1988. Reasoning about action using a possible model approach. *Proceedings Seventh National Conference on Artificial Intelligence (AAAI'88)*, pp. 89–93.

Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets

Pablo Barceló[1], **Leopoldo Bertossi**[2], and **Loreto Bravo**[3]

[1] University of Toronto, Department of Computer Science, Toronto, Canada.
pablo@cs.toronto.edu
[2] Carleton University, School of Computer Science, Ottawa, Canada.
bertossi@scs.carleton.ca
[3] Pontificia Universidad Catolica de Chile, Departamento de Ciencia de Computación, Santiago, Chile. lbravo@ing.puc.cl

Abstract. A relational database may not satisfy certain integrity constraints (ICs) for several reasons. However most likely most of the information in it is still consistent with the ICs. The answers to queries that are consistent with the ICs can be considered semantically correct answers, and are characterized [2] as ordinary answers that can be obtained from *every* minimally repaired version of the database. In this paper we address the problem of specifying those repaired versions as the minimal models of a theory written in *Annotated Predicate Logic* [27]. It is also shown how to specify database repairs using disjunctive logic program with annotation arguments and a classical stable model semantics. Those programs are then used to compute consistent answers to general first order queries. Both the annotated logic and the logic programming approaches work for any set of universal and referential integrity constraints. Optimizations of the logic programs are also analyzed.

1 Introduction

In databases, integrity constraints (ICs) capture the semantics of the application domain and help maintain the correspondence between that domain and its model provided by the database when updates on the database are performed. However, there are several reasons why a database may be or become inconsistent wrt a given set of integrity constraints (ICs) [2]. This could happen due to the materialized integration of several, possibly consistent data sources. We can also reach such a situation when we need to impose certain, new semantic constraints on legacy data. Another natural scenario is provided by a user who does not have control on the database maintenance mechanisms and wants to query the database through his/her own semantics of the database. Actually such a user could be querying several data sources and needs to impose some semantics on the combined information.

More generally speaking, we could think ICs on a database as constraints on the answers to queries rather than on the information stored in the database [32]. In this case, retrieving answers to queries that are consistent wrt the ICs becomes a central issue in the development of DBMSs.

In consequence, in any of the scenarios above and others, we are in the presence of an inconsistent database, where maybe a small portion of the information is incorrect wrt the intended semantics of the database; and as a an important and natural problem we have to characterize and retrieve data that is still correct wrt the ICs when queries are posed.

The notion of consistent answer to a first order (FO) query was defined in [2], where also a computational mechanism for obtaining consistent answers was presented. Intuitively speaking, a ground tuple $\bar{t}$ to a first order query $Q(\bar{x})$ is *consistent* in a, possibly inconsistent, relational database instance DB, if it is an (ordinary) answer to $Q(\bar{x})$ in every minimal repair of DB, that is in every database instance over the same schema and domain that differs from DB by a minimal (under set inclusion) set of inserted or deleted tuples. In other words, the consistent data in an inconsistent database is invariant under sensible restorations of the consistency of the database.

The mechanism presented in [2] has some limitations in terms of the ICs and queries that can handle. Although most of the ICs found in database praxis are covered by the positive cases in [2], the queries are restricted to conjunctions of literals. In [4, 6], a more general methodology based on logic programs with a stable model semantics was introduced. There is a one to one correspondence between the stable models of the logic programs and the database repairs. More general queries could be considered, but ICs were restricted to be "binary", i.e. universal with at most two database literals (plus built-in formulas). A similar, independent approach to database repair based on logic programs was also presented in [26].

The basic idea behind the logic programming based approach to consistent query answering is that since we need to deal with *all* the repairs of a database, we had better specify the class of the repairs. From a manageable logical specification of this class different reasoning tasks could be performed, in particular, computation of consistent answers to queries.

Notice that a specification of the class of database repairs must include information about (from) the database and the information contained in the ICs. Since these two pieces of information may be mutually inconsistent, we need a logic that does not collapse in the presence of contradictions. A non classical logic, like *Annotated Predicate Calculus* (*APC*) [27], for which a classically inconsistent set of premises can still have a model, is a natural candidate. In [3], a new declarative semantic framework was presented for studying the problem of query answering in databases that are inconsistent with respect to universal integrity constraints. This was done by embedding both the database instance and the integrity constraints into a single theory written in *APC*, with an appropriate, non classical truth-values lattice *Latt*.

In [3] it was shown that there is a one to one correspondence between some minimal models of the annotated theory and the repairs of the inconsistent database for universal ICs. In this way, a non monotonic logical specification of the database repairs was achieved. The annotated theory was used to derived some algorithms for obtaining consistent answers to some simple first order queries.

The results presented here extend those presented in [3] in different ways. First, we show how to annotate other important classes of ICs found in database praxis, e.g. referential integrity constraints [1], and the correspondence results are extended. Next, the problem of consistent query answering is characterized as a problem of non monotonic entailment.

We also show how the the *APC* theory that specifies the database repairs motivates the generation of new logic programs to specify the database repairs. Those programs have a classical stable model semantics and contain the annotations as constants that appear as new arguments of the database predicates. We establish a one to one correspondence between the stable models of the program and the repairs of the original database. The programs obtained in this way are simpler than those presented in in [4, 6, 26] in the sense that only one rule per IC is needed, whereas the latter may lead to an exponential number of rules.

The logic programs obtained can be used to retrieve consistent answers to arbitrary FO queries. Some computational experiments with *DLV* [21] are shown. The methodology for consistent query answering based on logic programs presented here works for arbitrary FO queries and universal ICs, what considerable extends the cases that could be handled in [2, 4, 3].

This paper improves, combines and extends results presented in [8, 9]. The main extensions have to do with the analysis and optimizations of the logic programs for consistent query answering introduced here.

This paper is structured as follows. In Section 2 we give some basic background. In section 3, we show how to annotate referential ICs, taking them, in addition to universal ICs, into a theory written in annotated predicate calculus. The correspondence between minimal models of the theory and database repairs is also established. Next, in Section 4, we show how to annotate queries and formulate the problem of consistent query answering as a problem of non-monotonic (minimal) entailment from the annotated theory. Then, in Section 5, on the basis of the generated annotated theory, disjunctive logic programs with annotation arguments to specify the database repairs are presented. It is also shown how to use them for consistent query answering. Some computational examples are presented in Section 6. Section 7 gives the first full treatment of logic program for computing repairs wrt referential integrity constraints. In Section 8 we introduce some optimizations of the logic programs. Finally, in Section 9 we draw some conclusions and consider related work. Proofs and intermediate results can be found in http://www.scs.carleton.ca/~bertossi/papers/proofsChap.ps.

2 Preliminaries

2.1 Database repairs and consistent answers

In the context of relational databases, we will consider a fixed relational schema $\Sigma = (D, \mathcal{P} \cup \mathcal{B})$ that determines a first order language. It consists of a fixed, possibly infinite, database domain $D = \{c_1, c_2, ...\}$, a fixed set of database predicates $\mathcal{P} = \{p_1, \ldots, p_n\}$, and a fixed set of built-in predicates $\mathcal{B} = \{e_1, \ldots, e_m\}$.

A database instance over Σ is a finite collection DB of facts of the form $p(c_1, ..., c_n)$, where p is a predicate in $\mathcal{P}$ and $c_1, ..., c_n$ are constants in D. Built-in predicates have a fixed and same extension in every database instance, not subject to changes.

A *universal integrity constraint* (IC) is an implicitly universally quantified clause of the form

$$q_1(\bar{t}_1) \vee \cdots \vee q_n(\bar{t}_n) \vee \neg p_1(\bar{s}_1) \vee \cdots \vee \neg p_m(\bar{s}_m) \qquad (1)$$

in the *FO* language $\mathcal{L}(\Sigma)$ based on Σ, where each p_i, q_j is a predicate in $\mathcal{P} \cup \mathcal{B}$ and the $\bar{t}_i, \bar{s}_j$ are tuples containing constants and variables. We assume we have a fixed set IC of ICs that is consistent as a FO theory. The database DB is always logically consistent if considered in isolation from the ICs.

It may be the case that $DB \cup IC$ is inconsistent. Equivalently, if we associate to DB a first order structure, also denoted with DB, in the natural way, i.e. by applying the domain closure and unique names assumptions and the closed world assumption [33] that makes false any ground atom not explicitly appearing in the set of atoms DB, it may happen that DB, as a structure, does not satisfy the IC. We denote with $DB \models_\Sigma IC$ the fact that the database satisfies IC. In this case we say that DB is consistent wrt IC; otherwise we say DB is inconsistent.

The *distance* [2] between two database instances DB_1 and DB_2 is their symmetric difference $\Delta(DB_1, DB_2) = (DB_1 - DB_2) \cup (DB_2 - DB_1)$. Now, given a database instance DB, possibly inconsistent wrt IC, we say that the instance DB' is a *repair* [2] of DB wrt IC iff $DB' \models_\Sigma IC$ and $\Delta(DB, DB')$ is minimal under set inclusion in the class of instances that satisfy IC and are compatible with the schema Σ.

Example 1. Consider the relational schema $Book(author, name, publYear)$, a database instance $DB = \{Book(kafka, metamorph, 1915), Book(kafka, metamorph, 1919)\}$; and the functional dependency $FD : author, name \rightarrow publYear$, that can be expressed by $IC : \neg Book(x, y, z) \vee \neg Book(x, y, w) \vee z = w$. Instance DB is inconsistent with respect to IC. The original instance has two possible repairs: $DB_1 = \{Book(kafka, metamorph, 1915)\}$, and $DB_2 = \{Book(kafka, metamorph, 1919)\}$. □

Let DB be a database instance, possibly not satisfying a set IC of integrity constraints. Given a query $Q(\bar{x}) \in \mathcal{L}(\Sigma)$, we say that a tuple of constants $\bar{t}$ is a *consistent answer* to $Q(\bar{x})$ in DB wrt IC, denoted $DB \models_c Q(\bar{t})$, if for every repair DB' of DB, $DB' \models_\Sigma Q(\bar{t})$ [2].[1] If Q is a closed formula, *i.e.* a sentence, then *true* is a *consistent answer* to Q, denoted $DB \models_c Q$, if for every repair DB' of DB, $DB' \models_\Sigma Q$.

[1] $DB' \models_\Sigma Q(\bar{t})$ means that when the variables in $\bar{x}$ are replaced in Q by the constants in $\bar{t}$ we obtain a sentence that is true in DB'.

Example 2. (example 1 continued) The query $Q_1: \ Book(kafka, metamorph, 1915)$ does not have *true* as a consistent answer, because it is not true in every repair. Query $Q_2(y): \ \exists x \exists z Book(x, y, z)$ has $y = metamorph$ as a consistent answer. Query $Q_3(x): \exists z Book(x, metamorph, z)$ has $x = kafka$ as a consistent answer. □

2.2 Annotating DBs and ICs

Annotated Predicate Calculus (*APC*) was introduced in [27] and also studied in [12] and [28]. It constitutes a non classical logic, where classically inconsistent information does not trivializes logical inference, reasoning about causes of inconsistency becomes possible, making one of its goals to study the differences in the contribution to the inconsistency made by the different literals in a theory, what is related to the problem of consistent query answers.

The syntax of *APC* is similar to that of classical logic, except for the fact that the atoms (and only the atoms) are annotated with values drawn from a *truth-values lattice*. The lattice *Latt* we will use throughout this paper is shown in Figure 1, first introduced in [3].

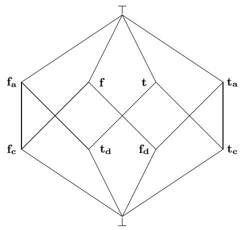


Fig. 1. *Latt* with *constraints values*, *database values* and *advisory values*

The lattice contain the usual truth values $\mathbf{t}, \mathbf{f}, \top, \bot$, for true, false, inconsistent and unknown, resp., but also six new truth values. Intuitively, we can think of values $\mathbf{t_c}$ and $\mathbf{f_c}$ as specifying what is needed for constraint satisfaction and will be used to annotate atoms appearing in ICs. The values $\mathbf{t_d}$ and $\mathbf{f_d}$ represent the truth values according to the original database and will be used to annotate atoms inside, resp. outside, the database. Finally, $\mathbf{t_a}$ and $\mathbf{f_a}$ are considered *advisory* truth values. These are intended to solve conflicts between the original database and the integrity constraints. Notice that $lub(\mathbf{t_d}, \mathbf{f_c}) = \mathbf{f_a}$ and $lub(\mathbf{f_d}, \mathbf{t_c}) = \mathbf{t_a}$. This means that whenever we have an atom, e.g. annotated with both $\mathbf{t_d}$ and $\mathbf{f_c}$, i.e. it is true according to the DB, but false according to the ICs, then it becomes automatically annotated with $\mathbf{f_a}$, meaning that the advise is to make it false. This will be made precise through the notion of formula satisfaction in *APC* below.

The intuition behind is that, in case of a conflict between the constraints and the database, we should obey the constraints, because the database instance only can be changed to restore consistency. This lack of symmetry between data and ICs is precisely captured by the lattice. Advisory value $\mathbf{t_a}$ is an indication that the atom annotated with it must be inserted into the DB; and deleted from the DB when annotated with $\mathbf{f_a}$.

Herbrand interpretations are now sets of annotated ground atoms. The notion of formula satisfaction in an *Herbrand interpretation* I is defined classically, except for atomic formulas p: we say that $I \models p\!:\!\mathbf{s}$, with $\mathbf{s} \in Latt$, iff for some $\mathbf{s}'$ such that $\mathbf{s} \leq \mathbf{s}'$ we have that $p\!:\!\mathbf{s}' \in I$ [27].

Given an *APC* theory $\mathcal{T}$, we say that an Herbrand interpretation I is a Δ-minimal model of $\mathcal{T}$, with $\Delta = \{\mathbf{t_a}, \mathbf{f_a}\}$, if I is a model of $\mathcal{T}$ and no other model of $\mathcal{T}$ has a proper subset of atoms annotated with elements in Δ, *i.e.* the set of atoms annotated with $\mathbf{t_a}$ or $\mathbf{f_a}$ in I is minimal under set inclusion. Considering Δ-minimal models is natural, because they minimize the set of changes, which in their turn are represented by the atoms annotated with $\mathbf{t_a}$ or $\mathbf{f_a}$.[2]

Given a database instance DB and a set of integrity constraints IC of the form (1), an embedding $\mathcal{T}(DB, IC)$ of DB and IC into a new *APC* theory was defined [3]. The new theory reconciles in a non classical setting the conflicts between data and ICs. In [3] it was also shown that there is a one-to-one correspondence between the Δ-minimal models of theory $\mathcal{T}(DB, IC)$ and the repairs of the original database instance. Actually, repairs can be obtained from minimal models as follows:

Definition 1. [3] *Given a minimal model $\mathcal{M}$ of $\mathcal{T}(DB, IC)$, the corresponding DB instance is defined by* $DB_{\mathcal{M}} = \{p(\bar{a}) \ | \ \mathcal{M} \models p(\bar{a})\!:\!\mathbf{t} \ \vee \ p(\bar{a})\!:\!\mathbf{t_a}\}$. □

Example 3. (example 1 cont.) The embedding $\mathcal{T}(DB)$ of DB into *APC* is given by the following formulas:

1. $Book(kafka, metamorph, 1915)\!:\!\mathbf{t_d}, \quad Book(kafka, metamorph, 1919)\!:\!\mathbf{t_d}$.
   Every ground atom that is not in DB is (possibly implicitly) annotated with $\mathbf{f_d}$.
2. Predicate closure axioms:
   $((x = kafka)\!:\!\mathbf{t_d} \ \wedge \ (y = metamorph)\!:\!\mathbf{t_d} \wedge (z = 1915)\!:\!\mathbf{t_d}) \ \vee$
   $((x = kafka)\!:\!\mathbf{t_d} \ \wedge \ (y = metamorph)\!:\!\mathbf{t_d} \wedge (z = 1919)\!:\!\mathbf{t_d}) \vee Book(x, y, z)\!:\!\mathbf{f_d}$.

The embedding $\mathcal{T}(IC)$ of IC into *APC* is given by:

[2] Most of the time we will simply say "minimal" instead of Δ-minimal. In this case there should be no confusion with the other notion of minimality in this paper, namely the one that applies to repairs.

3. $Book(x, y, z)\!:\!\mathbf{f_c} \vee Book(x, y, w)\!:\!\mathbf{f_c} \vee (z = w)\!:\!\mathbf{t_c}$.
4. $Book(x, y, z)\!:\!\mathbf{f_c} \vee Book(x, y, z)\!:\!\mathbf{t_c}, \quad \neg Book(x, y, z)\!:\!\mathbf{f_c} \vee \neg Book(x, y, z)\!:\!\mathbf{t_c}$.[3]
   These formulas specify that every fact must have one and just one constraint value.

Furthermore

5. For every true *built-in* atom φ we include φ:$\mathbf{t}$ in $\mathcal{T}(\mathcal{B})$, and φ:$\mathbf{f}$ for every false built-in atom, e.g. $(1915 = 1915)\!:\!\mathbf{t}$, but $(1915 = 1919)\!:\!\mathbf{f}$.

The Δ-minimal models of $\mathcal{T}(DB, IC) = \mathcal{T}(DB) \cup \mathcal{T}(IC) \cup \mathcal{T}(\mathcal{B})$ are:

$$\mathcal{M}_1 = \{Book(kafka, metamorph, 1915)\!:\!\mathbf{t}, \, Book(kafka, metamorph, 1919)\!:\!\mathbf{f_a}\},$$
$$\mathcal{M}_2 = \{Book(kafka, metamorph, 1915)\!:\!\mathbf{f_a}, \, Book(kafka, metamorph, 1919)\!:\!\mathbf{t}\}.$$

They also contain annotated false DB atoms and built-ins, but we will show only the most relevant data in them. The corresponding database instances, $DB_{\mathcal{M}_1}, DB_{\mathcal{M}_2}$ are the repairs of DB shown in Example 1. □

From the definition of the lattice and the fact that no atom from the database is annotated with both $\mathbf{t_d}$ and $\mathbf{f_d}$, it is possible to show that, in the minimal models of the annotated theory, a DB atom may get as annotation either $\mathbf{t}$ or $\mathbf{f_a}$ if the atom was annotated with $\mathbf{t_d}$; similarly either $\mathbf{f}$ or $\mathbf{t_a}$ if the atom was annotated with $\mathbf{f_d}$. In the transition from the annotated theory to its minimal models, the annotations $\mathbf{t_d}, \mathbf{f_d}$ "disappear", as we want the atoms to be annotated at the highest possible layer in the lattice; except for $\top$, that can always we avoided in the minimal models.

3 Annotating Referential ICs

Referential integrity constraints (RICs) like

$$\forall \bar{x}(p(\bar{x}) \rightarrow \exists y q(\bar{x}', y)), \tag{2}$$

where the variables in $\bar{x}'$ are a subset of the variables in $\bar{x}$, cannot be expressed as an equivalent clause of the form (1). RICs are important and common in databases. For that reason, we need to extend our embedding methodology. Actually, we embed (2) into *APC* by means of

$$p(\bar{x})\!:\!\mathbf{f_c} \vee \exists y (q(\bar{x}', y)\!:\!\mathbf{t_c}). \tag{3}$$

In the rest of this section we allow the given set of ICs to contain, in addition to universal ICs of the form (1), also RICs like (2). The one-to-one correspondence between minimal models of the new theory $\mathcal{T}(DB, IC)$ and the repairs of DB still holds. Most important for us is to obtain repairs from minimal models.

[3] Since only atomic formulas are annotated, the non atomic formula $\neg p(\bar{x})\!:\!\mathbf{s}$ is to be read as $\neg(p(\bar{x})\!:\!\mathbf{s})$. We will omit the parenthesis though.

Given a pair of database instances DB_1 and DB_2 over the same schema (and domain), we construct the Herbrand structure $\mathcal{M}(DB_1, DB_2) = \langle D, I_{\mathcal{P}}, I_{\mathcal{B}} \rangle$, where D is the domain of the database and $I_{\mathcal{P}}$, $I_{\mathcal{B}}$ are the interpretations for the predicates and the built-ins, respectively. $I_{\mathcal{P}}$ is defined as follows:

$$I_{\mathcal{P}}(p(\bar{a})) = \begin{cases} \mathbf{t} & p(\bar{a}) \in DB_1, \ p(\bar{a}) \in DB_2 \\ \mathbf{f} & p(\bar{a}) \notin DB_1, \ p(\bar{a}) \notin DB_2 \\ \mathbf{f_a} & p(\bar{a}) \in DB_1, \ p(\bar{a}) \notin DB_2 \\ \mathbf{t_a} & p(\bar{a}) \notin DB_1, \ p(\bar{a}) \in DB_2 \end{cases}$$

The interpretation $I_{\mathcal{B}}$ is defined as expected: if q is a built-in, then $I_P(q(\bar{a})) = \mathbf{t}$ iff $q(\bar{a})$ is true in classical logic, and $I_P(q(\bar{a})) = \mathbf{f}$ iff $q(\bar{a})$ is false.

Lemma 1. *Given two database instances DB and DB', if $DB' \models_{\Sigma} IC$, then $\mathcal{M}(DB, DB') \models \mathcal{T}(DB, IC)$.* □

Lemma 2. *If $\mathcal{M}$ is a model of $\mathcal{T}(DB, IC)$ such that $DB_{\mathcal{M}}$ is finite*[4]*, then $DB_{\mathcal{M}} \models_{\Sigma} IC$.* □

The following results shows the one-to one correspondence between the minimal models of $\mathcal{T}(DB, IC)$ and the repairs of DB.

Proposition 1. *If DB' is a repair of DB with respect to the set of integrity constraints IC, then $\mathcal{M}(DB, DB')$ is minimal among the models of $\mathcal{T}(DB, IC)$.*
□

Proposition 2. *Let $\mathcal{M}$ be a model of $\mathcal{T}(DB, IC)$. If $\mathcal{M}$ is minimal and $DB_{\mathcal{M}}$ is finite, then $DB_{\mathcal{M}}$ is a repair of DB with respect to IC.* □

Example 4. Consider the relational schema of Example 1 extended with the table $Author(name, citizenship)$. Now, IC also contains the RIC: $Book(x, y, z) \rightarrow \exists w Author(x, w)$, expressing that every writer of a book in the database instance must be registered as an author. The theory $\mathcal{T}(IC)$ now also contains:

$Book(x, y, z)\!:\!\mathbf{f_c} \vee \exists w (Author(x, w)\!:\!\mathbf{t_c}), \quad Author(x, w)\!:\!\mathbf{f_c} \vee Author(x, w)\!:\!\mathbf{t_c}$,
$\neg Author(x, w)\!:\!\mathbf{f_c} \vee \neg Author(x, w)\!:\!\mathbf{t_c}$.

We might also have the functional dependency $FD: \ name \rightarrow citizenship$, that in conjunction with the RIC, produces a foreign key constraint. The database instance $\{Book(neruda, 20lovepoems, 1924)\}$ is inconsistent wrt the given RIC. If we have the following subdomain $D(Author.citizenship) = \{chilean, canadian\}$ for the attribute "citizenship", we obtain the following database theory:

$\mathcal{T}(DB) = \{Book(neruda, 20lovepoems, 1924)\!:\!\mathbf{t_d}, \ Author(neruda, chilean)\!:\!\mathbf{f_d}, \ Author(neruda, canadian)\!:\!\mathbf{f_d}, \ldots\}$.

[4] That is, the extensions of the database predicates are finite. These are the models that may lead to database instances, because the latter have finite database relations.

The minimal models of $\mathcal{T}(DB, IC)$ are:

$$\begin{aligned}\mathcal{M}_1 = \{&Book(neruda, \mathit{20lovepoems}, \mathit{1924}){:}\mathbf{f_a},\ Author(neruda, chilean){:}\mathbf{f},\\ &Author(neruda, canadian){:}\mathbf{f}, \ldots\}\\ \mathcal{M}_2 = \{&Book(neruda, \mathit{20lovepoems}, \mathit{1924}){:}\mathbf{t},\ Author(neruda, chilean){:}\mathbf{t_a},\\ &Author(neruda, canadian){:}\mathbf{f}, \ldots\}\\ \mathcal{M}_3 = \{&Book(neruda, \mathit{20lovepoems}, \mathit{1924}){:}\mathbf{t},\ Author(neruda, chilean){:}\mathbf{f},\\ &Author(neruda, canadian){:}\mathbf{t_a}, \ldots\}.\end{aligned}$$

We obtain $DB_{\mathcal{M}_1} = \emptyset$, $DB_{\mathcal{M}_2} = \{Book(neruda,\ \mathit{20lovepoems}, \mathit{1924}), Author($ $neruda,\ chilean)\}$ and $DB_{\mathcal{M}_3}$ similar to $DB_{\mathcal{M}_2}$, but with a Canadian Neruda. According to Proposition 2, these are repairs of the original database instance, actually the only ones. □

As in [3], it can be proved that when the original instance is consistent, then it is its only repair and it corresponds to a unique minimal model of the APC theory.

3.1 Annotating general ICs

The class of ICs found in database praxis is contained in the class of FO formulas of the form:

$$\forall \bar{x}\ (\varphi(\bar{x}) \rightarrow \exists \bar{z} \psi(\bar{y})) \tag{4}$$

where φ and ψ are (possibly empty) conjunctions of literals, and $\bar{z} = \bar{y} - \bar{x}$. This class [1, chapter 10] includes the ICs of the form (1), in particular, range constraints (e.g. $\forall x\ (p(x) \rightarrow x > 30)$), join dependencies, functional dependencies, full inclusion dependencies; and also referential integrity constraints, and in consequence, also foreign key constraints.

The annotation methodology introduced so far can be extended to the whole class (4). We only sketch this extension here.

If in (4) $\varphi(\bar{x})$ is $\bigwedge_{i=1}^{k} p_i(\bar{x}_i) \wedge \bigwedge_{i=k+1}^{m} \neg p_i(\bar{x}_i)$ and $\psi(\bar{y})$ is $\bigwedge_{j=1}^{l} q_j(\bar{y}_j) \wedge \bigwedge_{j=l+1}^{r} \neg q_j(\bar{y}_j)$, we embed the constraint into APC as follows:

$$\bigvee_{i=1}^{k} p_i(\bar{x}_i){:}\mathbf{f_c} \vee \bigvee_{i=k+1}^{m} p_i(\bar{x}_i){:}\mathbf{t_c}\ \vee\ \exists \bar{z}(\bigwedge_{j=1}^{l} q_j(\bar{y}_j){:}\mathbf{t_c} \wedge \bigwedge_{j=l+1}^{r} q_j(\bar{y}_j){:}\mathbf{f_c}).$$

If we allow now that IC contains ICs of the form (4), it is still possible to establish the one-to-one correspondence between minimal models of $\mathcal{T}(DB, IC)$ and the repairs of DB.

4 Annotation of Queries

According to Proposition 2, a ground tuple $\bar{t}$ is a consistent answer to a FO query $Q(\bar{x})$ iff $Q(\bar{t})$ is true in every minimal model of $\mathcal{T}(DB, IC)$. However, if we want to pose the query directly to the theory, it is necessary to reformulate it as an annotated formula.

Definition 2. *Given a FO query $Q(\bar{x})$ in language $\mathcal{L}(\Sigma)$, we denote by $Q^{an}(\bar{x})$ the APC formula obtained from Q by simultaneously replacing, for $p \in \mathcal{P}$, the negative literal $\neg p(\bar{s})$ by the APC formula $p(\bar{s}){:}\mathbf{f} \vee p(\bar{s}){:}\mathbf{f_a}$, and the positive literal $p(\bar{s})$ by the APC formula $p(\bar{s}){:}\mathbf{t} \vee p(\bar{s}){:}\mathbf{t_a}$. For $p \in \mathcal{B}$, the atom $p(\bar{s})$ is replaced by the APC formula $p(\bar{s}){:}\mathbf{t}$.* □

According to this definition, logically equivalent versions of a query could have different annotated versions, but it can be shown (Proposition 3), that they retrieve the same consistent answers.

Example 5. (example 1 cont.) If we want the consistent answers to the query $Q(x) : \neg\exists y \exists z \exists w \exists t(Book(x, y, z) \wedge Book(x, w, t) \wedge y \neq w)$, asking for those authors that have at most one book, we generate the annotated query $Q^{an}(\bar{x})$: $\neg\exists y \exists z \exists w \exists t((Book(x, y, z){:}\mathbf{t} \vee Book(x, y, z){:}\mathbf{t_a}) \wedge (Book(x, w, t){:}\mathbf{t} \vee Book(x, w, t){:}$ $\mathbf{t_a}) \wedge (y \neq w){:}\mathbf{t})$, to be posed to the annotated theory with its minimal model semantics. □

Definition 3. *If φ is an APC sentence in the language of $\mathcal{T}(DB, IC)$, we say that $\mathcal{T}(DB, IC)$ Δ-minimally entails φ, written $\mathcal{T}(DB, IC) \models_\Delta \varphi$, iff every Δ-minimal model $\mathcal{M}$ of $\mathcal{T}(DB, IC)$, such that $DB_\mathcal{M}$ is finite, satisfies φ,* i.e. *$\mathcal{M} \models_{APC} \varphi$.* □

Now we characterize consistent query answers wrt the annotated theory.

Proposition 3. *Let DB be a database instance, IC a set of integrity constraints and $Q(\bar{x})$ a query in FO language $\mathcal{L}(\Sigma)$. It holds:*

$$DB \models_c Q(\bar{t}) \quad \textit{iff} \quad \mathcal{T}(DB, IC) \models_\Delta Q^{an}(\bar{t}).$$ □

Example 6. (example 5 continued) For consistently answering the query $Q(x)$, we pose the query $Q^{an}(x)$ to the minimal models of $\mathcal{T}(DB, IC)$. The answer we obtain from *every* minimal model is $x = kafka$. □

According to this proposition, in order to consistently answer queries, we are left with the problem of evaluating minimal entailment wrt the annotated theory. In [3] some limited FO queries were evaluated without passing to their annotated versions. The algorithms for consistent query answering were rather ad hoc and were extracted from the theory $\mathcal{T}(DB, IC)$. However, no advantage was taken from a characterization of consistent answers in terms of minimal entailment from $\mathcal{T}(DB, IC)$. In the next section we will address this issue by taking the original DB instance with the ICs into a logic program that is inspired by the annotated theory $\mathcal{T}(DB, IC)$. Furthermore, the query to be posed to the logic program will be built from Q^{an}.

5 Logic Programming Specification of Repairs

In this section we will consider ICs of the form (1). Our aim is to specify database repairs using classical first order logic programs. However, those programs will be suggested by the non classical annotated theory.

In order to accommodate annotations in this classical framework, we will first consider the annotations in the lattice *Latt* as new constants in the language. Next, we will replace each predicate $p(\bar{x}) \in \mathcal{P}$ by a new predicate $p(\bar{x}, \cdot)$, with an extra argument to be occupied by annotation constants. In this way we can simulate the annotations we had before, but in a classical setting. With all this, we have a new FO language, $\mathcal{L}(\Sigma)^{an}$, for annotated $\mathcal{L}(\Sigma)$.

Definition 4. *The repair logic program, $\Pi(DB, IC)$, for DB and IC, is written with predicates from $\mathcal{L}(\Sigma)^{an}$ and contains the following clauses:*

1. *For every atom $p(\bar{a}) \in DB$, $\Pi(DB, IC)$ contains the fact $p(\bar{a}, \mathbf{t_d})$.*
2. *For every predicate $p \in P$, $\Pi(DB, IC)$ contains the clauses:*

   $p(\bar{x}, \mathbf{t}^\star) \leftarrow p(\bar{x}, \mathbf{t_d})$. $\quad p(\bar{x}, \mathbf{t}^\star) \leftarrow p(\bar{x}, \mathbf{t_a})$.
   $p(\bar{x}, \mathbf{f}^\star) \leftarrow p(\bar{x}, \mathbf{f_a})$. $\quad p(\bar{x}, \mathbf{f}^\star) \leftarrow$ *not* $p(\bar{x}, \mathbf{t_d})$.,

   where $\mathbf{t}^\star, \mathbf{f}^\star$ are new, auxiliary elements in the domain of annotations.
3. *For every constraint of the form (1), $\Pi(DB, IC)$ contains the clause:*

   $\bigvee_{i=1}^{n} p_i(\bar{t}_i, \mathbf{f_a}) \vee \bigvee_{j=1}^{m} q_j(\bar{s}_j, \mathbf{t_a}) \quad \longleftarrow \quad \bigwedge_{i=1}^{n} p_i(\bar{t}_i, \mathbf{t}^\star) \wedge \bigwedge_{j=1}^{m} q_j(\bar{s}_j, \mathbf{f}^\star) \wedge \bar{\varphi}$,

   where $\bar{\varphi}$ represents the negation of φ. □

Intuitively, the clauses in 3. say that when the IC is violated (the body), then *DB* has to be repaired according to one of the alternatives shown in the head. Since there may be interactions between constraints, these single repairing steps may not be enough to restore the consistency of *DB*. We have to make sure that the repairing process continues and stabilizes in a state where all the ICs hold. This is the role of the clauses in 2. containing the new annotations $\mathbf{t}^\star$, that groups together those atoms annotated with $\mathbf{t_d}$ and $\mathbf{t_a}$, and $\mathbf{f}^\star$, that does the same with $\mathbf{f_d}$ and $\mathbf{f_a}$. Notice that the annotations $\mathbf{t}^\star$, $\mathbf{f}^\star$, obtained through the combined effect of rules 2. and 3., can be fed back into rules 3. until consistency is restored. This possibility is what allows us to have just one program rule for each IC.

Example 7 shows the interaction of a functional dependency and a full inclusion dependency. When atoms are deleted in order to satisfy the functional dependency, the inclusion dependency could be violated, and in a second step it should be repaired. At that second step, the annotations $\mathbf{t}^\star$ and $\mathbf{f}^\star$, computed at the first step where the functional dependency was repaired, will detect the violation of the inclusion dependency and trigger the corresponding repairing process.

Example 7. (example 1 continued) We extend the schema with the table *Eurbook*(*author*, *name*, *publYear*), for European books. Now, *DB* also contains the literal *Eurbook*(*kafka*, *metamorph*, *1919*)}. If in addition to the ICs we had before, we consider the full inclusion dependency $\forall xyz\ (Eurbook(x, y, z) \rightarrow Book(x, y, z))$, we obtain the following program $\Pi(DB, IC)$:

1. $EurBook(kafka, metamorph, \mathit{1919}, \mathbf{t_d})$. $\quad Book(kafka, metamorph, \mathit{1919}, \mathbf{t_d})$.
$Book(kafka, metamorph, \mathit{1915}, \mathbf{t_d})$.

2. $Book(x, y, z, \mathbf{t}^\star) \leftarrow Book(x, y, z, \mathbf{t_d})$. $\quad Book(x, y, z, \mathbf{t}^\star) \leftarrow Book(x, y, z, \mathbf{t_a})$.
$Book(x, y, z, \mathbf{f}^\star) \leftarrow Book(x, y, z, \mathbf{f_a})$. $\quad Book(x, y, z, \mathbf{f}^\star) \leftarrow$ *not* $Book(x, y, z, \mathbf{t_d})$.
$Eurbook(x, y, z, \mathbf{t}^\star) \leftarrow Eurbook(x, y, z, \mathbf{t_d})$.
$Eurbook(x, y, z, \mathbf{t}^\star) \leftarrow Eurbook(x, y, z, \mathbf{t_a})$.
$Eurbook(x, y, z, \mathbf{f}^\star) \leftarrow Eurbook(x, y, z, \mathbf{f_a})$.
$Eurbook(x, y, z, \mathbf{f}^\star) \leftarrow$ *not* $Eurbook(x, y, z, \mathbf{t_d})$.

3. $Book(x, y, z, \mathbf{f_a}) \vee Book(x, y, w, \mathbf{f_a}) \ \leftarrow\ Book(x, y, z, \mathbf{t}^\star), Book(x, y, w, \mathbf{t}^\star),$
$z \neq w.$
$Eurbook(x, y, z, \mathbf{f_a}) \vee Book(x, y, z, \mathbf{t_a}) \ \leftarrow\ Eurbook(x, y, z, \mathbf{t}^\star), Book(x, y, z, \mathbf{f}^\star).$
□

Our programs are standard logic programs (as opposed to annotated logic programs [28]) and, finding in them negation as failure, we will give them an also standard stable model semantics.

Let Π be the ground logic program obtained by instantiating the disjunctive program $\Pi(DB, IC)$ in its Herbrand universe. A set of ground atoms $\mathcal{M}$ is a *stable model* of $\Pi(DB, IC)$ iff it is a minimal model of $\Pi^\mathcal{M}$, where $\Pi^\mathcal{M} = \{A_1 \vee \cdots \vee A_n \leftarrow B_1, \cdots, B_m \mid A_1 \vee \cdots \vee A_n \leftarrow B_1, \cdots, B_m, not\ C_1, \cdots, not\ C_k\ \in \Pi$ and $C_i \notin \mathcal{M}$ for $1 \leq i \leq k\}$ [23, 24].

Definition 5. *A Herbrand model $\mathcal{M}$ is* coherent *if it does not contain a pair of literals of the form $\{p(\bar{a}, \mathbf{t_a}), p(\bar{a}, \mathbf{f_a})\}$.* □

Example 8. (example 7 continued) The coherent stable models of the program presented in Example 7 are:

$\mathcal{M}_1 = \{Book(kafka, metamorph, \mathit{1919}, \mathbf{t_d}),\ Book(kafka, metamorph, \mathit{1919}, \mathbf{t}^\star),$
$Book(kafka, metamorph, \mathit{1915}, \mathbf{t_d}),\ Book(kafka, metamorph, \mathit{1915}, \mathbf{t}^\star),$
$Book(kafka, metamorph, \mathit{1915}, \mathbf{f_a}),\ Book(kafka, metamorph, \mathit{1915}, \mathbf{f}^\star),$
$Eurbook(kafka, metamorph, \mathit{1919}, \mathbf{t_d}),\ Eurbook(kafka, metamorph, \mathit{1919}, \mathbf{t}^\star)\};$

$\mathcal{M}_2 = \{Book(kafka, metamorph, \mathit{1919}, \mathbf{t_d}),\ Book(kafka, metamorph, \mathit{1919}, \mathbf{t}^\star),$
$Book(kafka, metamorph, \mathit{1919}, \mathbf{f_a}),\ Book(kafka, metamorph, \mathit{1919}, \mathbf{f}^\star),$
$Book(kafka, metamorph, \mathit{1915}, \mathbf{t_d}),\ Book(kafka, metamorph, \mathit{1915}, \mathbf{t}^\star),$
$Eurbook(kafka, metamorph, \mathit{1919}, \mathbf{t_d}),\ Eurbook(kafka, metamorph, \mathit{1919}, \mathbf{t}^\star),$
$Eurbook(kafka, metamorph, \mathit{1919}, \mathbf{f_a}),\ Eurbook(kafka, metamorph, \mathit{1919}, \mathbf{f}^\star)\}.$ □

The stable models of the program will include the database contents with its original annotations ($\mathbf{t_d}$). Every time there is an atom in a model annotated with $\mathbf{t_d}$ or $\mathbf{t_a}$, it will appear annotated with $\mathbf{t}^\star$. From these models we should be able to "read" database repairs. Every stable model of the logic program has to be interpreted. In order to do this, we introduce two new annotations, $\mathbf{t}^{\star\star}, \mathbf{f}^{\star\star}$, in the last arguments. The first one groups together those atoms annotated with $\mathbf{t_a}$ and those annotated with $\mathbf{t_d}$, but not $\mathbf{f_a}$. Intuitively, they correspond to those annotated with $\mathbf{t}$ in the models of $\mathcal{T}(DB, IC)$. A similar role plays the other new annotation wrt the "false" annotations. These new annotations will simplify the expression of the queries to be posed to the program. Without them, instead of simply asking $p(\bar{x}, \mathbf{t}^{\star\star})$ (for the tuples in p in a repair), we would have to ask for $p(\bar{x}, \mathbf{t_a}) \vee (p(\bar{x}, \mathbf{t_d}) \wedge \neg p(\bar{x}, \mathbf{f_a}))$. The interpreted models can be easily obtained by adding new rules.

Definition 6. *The interpretation program $\Pi^\star(DB, IC)$ extends $\Pi(DB, IC)$ with the following rules:*

$$\begin{array}{llll} p(\bar{a}, \mathbf{f}^{\star\star}) \leftarrow p(\bar{a}, \mathbf{f_a}). & & p(\bar{a}, \mathbf{f}^{\star\star}) \leftarrow \mathit{not}\ p(\bar{a}, \mathbf{t_d}),\ \mathit{not}\ p(\bar{a}, \mathbf{t_a}). \\ p(\bar{a}, \mathbf{t}^{\star\star}) \leftarrow p(\bar{a}, \mathbf{t_a}). & & p(\bar{a}, \mathbf{t}^{\star\star}) \leftarrow p(\bar{a}, \mathbf{t_d}),\ \mathit{not}\ p(\bar{a}, \mathbf{f_a}). \end{array}$$ □

Example 9. (example 8 continued) The coherent stable models of the interpretation program extend

$\mathcal{M}_1$ with $\{Eurbook(kafka, metamorph, 1919, \mathbf{t}^{\star\star}),$
$Book(kafka, metamorph, 1919, \mathbf{t}^{\star\star}), Book(kafka, metamorph, 1915, \mathbf{f}^{\star\star})\}$;

$\mathcal{M}_2$ with $\{Eurbook(kafka, metamorph, 1919, \mathbf{f}^{\star\star}),$
$Book(kafka, metamorph, 1919, \mathbf{f}^{\star\star}), Book(kafka, metamorph, 1915, \mathbf{t}^{\star\star})\}$. □

From an interpretation model we can obtain a database instance.

Definition 7. *Let $\mathcal{M}$ be a coherent stable model of program $\Pi^\star(DB, IC)$. The database associated to $\mathcal{M}$ is $DB_{\mathcal{M}} = \{p(\bar{a}) \mid p(\bar{a}, \mathbf{t}^{\star\star}) \in \mathcal{M}\}$.* □

The following theorem establishes the one-to-one correspondence between coherent stable models of the program and the repairs of the original instance.

Theorem 1. *If $\mathcal{M}$ is a coherent stable model of $\Pi^\star(DB, IC)$, and $DB_{\mathcal{M}}$ is finite, then $DB_{\mathcal{M}}$ is a repair of DB with respect to IC. Furthermore, the repairs obtained in this way are all the repairs of DB.* □

Example 10. (example 9 continued) The following database instances obtained from Definition 7 are the repairs of DB:

$$DB_{\mathcal{M}_1} = \{Eurbook(kafka, metamorph, 1919),\ Book(kafka, metamorph, 1919)\},$$
$$DB_{\mathcal{M}_2} = \{Book(kafka, metamorph, 1915)\}.$$ □

5.1 The query program

Given a first order query Q, we want the consistent answers from DB. In consequence, we need those atoms that are simultaneously true of Q in every stable model of the program $\Pi(DB, IC)$. They are obtained through the query $Q^{\star\star}$, obtained from Q by replacing, for $p \in \mathcal{P}$, every positive literal $p(\bar{s})$ by $p(\bar{s}, \mathbf{t}^{\star\star})$ and every negative literal $\neg p(\bar{s})$ by $p(\bar{s}, \mathbf{f}^{\star\star})$. Now $Q^{\star\star}$ can be transformed into a query program $\Pi(Q^{\star\star})$ by a standard transformation [30, 1]. This query program will be run in combination with $\Pi^\star(DB, IC)$.

Example 11. For the query $Q(y) : \exists z Book(kafka, y, z)$, we generate $Q^{\star\star}(y) : \exists z Book(kafka, y, z, \mathbf{t}^{\star\star})$, that is transformed into the query program clause $Answer(y) \leftarrow Book(kafka, y, z, \mathbf{t}^{\star\star})$. □

6 Computing from the Program

The database repairs could be computed using an implementation of the disjunctive stable models semantics like *DLV* [21], that also supports denial constraints as studied in [13]. In this way we are able to prune out the models that are not coherent, imposing for every predicate p the constraint $\leftarrow p(\bar{x}, \mathbf{t_a}), p(\bar{x}, \mathbf{f_a})$.

Example 12. Consider the database instance $\{p(a)\}$ that is inconsistent wrt the full inclusion dependency $\forall x(p(x) \rightarrow q(x))$. The program $\Pi^\star(DB, IC)$ contains the following clauses:

1. Database contents: $p(a, \mathbf{t_d})$.
2. Rules for the closed world assumption:
   $p(x, \mathbf{f}^\star) \leftarrow \mathit{not}\ p(x, \mathbf{t_d})$. $\quad q(x, \mathbf{f}^\star) \leftarrow \mathit{not}\ q(x, \mathbf{t_d})$.
3. Annotation rules:
   $$\begin{array}{lll} p(x, \mathbf{f}^\star) \leftarrow p(x, \mathbf{f_a}). & p(x, \mathbf{t}^\star) \leftarrow p(x, \mathbf{t_a}). & p(x, \mathbf{t}^\star) \leftarrow p(x, \mathbf{t_d}). \\ q(x, \mathbf{f}^\star) \leftarrow q(x, \mathbf{f_a}). & q(x, \mathbf{t}^\star) \leftarrow q(x, \mathbf{t_a}). & q(x, \mathbf{t}^\star) \leftarrow q(x, \mathbf{t_d}). \end{array}$$
4. Rule for the IC: $p(x, \mathbf{f_a}) \vee q(x, \mathbf{t_a}) \leftarrow p(x, \mathbf{t}^\star), q(x, \mathbf{f}^\star)$.
5. Denial constraints for coherence
   $\leftarrow p(\bar{x}, \mathbf{t_a}), p(\bar{x}, \mathbf{f_a})$. $\quad \leftarrow q(\bar{x}, \mathbf{t_a}), q(\bar{x}, \mathbf{f_a})$.
6. Interpretation rules:
   $$\begin{array}{ll} p(x, \mathbf{t}^{\star\star}) \leftarrow p(x, \mathbf{t_a}). & p(x, \mathbf{t}^{\star\star}) \leftarrow p(x, \mathbf{t_d}),\ \mathit{not}\ p(x, \mathbf{f_a}). \\ p(x, \mathbf{f}^{\star\star}) \leftarrow p(x, \mathbf{f_a}). & p(x, \mathbf{f}^{\star\star}) \leftarrow \mathit{not}\ p(x, \mathbf{t_d}),\ \mathit{not}\ p(x, \mathbf{t_a}). \\ q(x, \mathbf{t}^{\star\star}) \leftarrow q(x, \mathbf{t_a}). & q(x, \mathbf{t}^{\star\star}) \leftarrow q(x, \mathbf{t_d}),\ \mathit{not}\ q(x, \mathbf{f_a}). \\ q(x, \mathbf{f}^{\star\star}) \leftarrow q(x, \mathbf{f_a}). & q(x, \mathbf{f}^{\star\star}) \leftarrow \mathit{not}\ q(x, \mathbf{t_d}),\ \mathit{not}\ q(x, \mathbf{t_a}). \end{array}$$

Running program $\Pi^\star(DB, IC)$ with *DLV* we obtain two stable models:

$$\mathcal{M}_1 = \{p(a, \mathbf{t_d}), p(a, \mathbf{t}^\star), q(a, \mathbf{f}^\star), q(a, \mathbf{t_a}), p(a, \mathbf{t}^{\star\star}), q(a, \mathbf{t}^\star), q(a, \mathbf{t}^{\star\star})\},$$
$$\mathcal{M}_2 = \{p(a, \mathbf{t_d}), p(a, \mathbf{t}^\star), p(a, \mathbf{f}^\star)), q(a, \mathbf{f}^\star), p(a, \mathbf{f}^{\star\star}), q(a, \mathbf{f}^{\star\star}), p(a, \mathbf{f_a})\}.$$

The first model says, through its atom $q(a, \mathbf{t}^{\star\star})$, that $q(a)$ has to be inserted in the database. The second one, through its atom $p(a, \mathbf{f}^{\star\star})$, that $p(a)$ has to be deleted. □

The coherence denial constraints did not play any role in the previous example, we obtain exactly the same model with or without them. The reason is that we have only one IC; in consequence, only one step is needed to obtain a repair of the database. There is no way to obtain an incoherent stable model due to the application of the rules 1. and 2. in Example 12 in a second repair step.

Example 13. (example 12 continued) Let us now add an extra full inclusion dependency, $\forall x(q(x) \rightarrow r(x))$, keeping the same instance. One repair is obtained by inserting $q(a)$, what causes the insertion of $r(a)$. The program is as before, but with the additional rules

$$\begin{array}{lll} r(x, \mathbf{f}^\star) \leftarrow \mathit{not}\ r(x, \mathbf{t_d}). & r(x, \mathbf{f}^\star) \leftarrow r(x, \mathbf{f_a}). & r(x, \mathbf{t}^\star) \leftarrow r(x, \mathbf{t_a}). \\ r(X, \mathbf{t}^\star) \leftarrow r(X, \mathbf{t_d}). & r(x, \mathbf{t}^{\star\star}) \leftarrow r(x, \mathbf{t_a}). & r(x, \mathbf{t}^{\star\star}) \leftarrow r(x, \mathbf{t_d}), \mathit{not}\ r(x, \mathbf{f_a}). \\ r(x, \mathbf{f}^{\star\star}) \leftarrow r(x, \mathbf{f_a}). & r(x, \mathbf{f}^{\star\star}) \leftarrow \mathit{not}\ r(x, \mathbf{t_d}), \mathit{not}\ r(x, \mathbf{t_a}). & \\ q(x, \mathbf{f_a}) \vee r(x, \mathbf{t_a}) \leftarrow q(x, \mathbf{t}^\star), r(x, \mathbf{f}^\star). & \leftarrow r(x, \mathbf{t_a}), r(x, \mathbf{f_a}). & \end{array}$$

If we run the program we obtain the expected models, one that deletes $p(a)$, and a second one that inserts both $q(a)$ and $r(a)$. However, if we omit the coherence denial constraints, more precisely the one for table q, we obtain a third model, namely $\{p(a, \mathbf{t_d}), p(a, \mathbf{t}^\star), q(a, \mathbf{f}^\star), r(a, \mathbf{f}^\star), q(a, \mathbf{f_a}), q(a, \mathbf{t_a}), p(a, \mathbf{t}^{\star\star}), q(a, \mathbf{t}^\star), q(a, \mathbf{t}^{\star\star}), q(a, \mathbf{f}^{\star\star}), r(a, \mathbf{f}^{\star\star})\}$, that is not coherent, because it contains both $q(a, \mathbf{f_a})$ and $q(a, \mathbf{t_a})$, and cannot be interpreted as a repair of the original database. □

Notice that the programs with annotations obtained are very simple in terms of their dependency on the ICs. As mentioned before, consistent answers can be obtained "running" a query program together with the repair program $\Pi^\star(DB, IC)$, under the skeptical stable model semantics, that sanctions as true what is true of all stable models.

Example 14. (example 12 continued) Assume now that the original database is $\{p(a), p(b), q(b)\}$, and we want the consistent answers to the query $p(x)$. In this case we need to add the facts $p(b, \mathbf{t_d}), q(b, \mathbf{t_d})$, and the query rule $ans(x) \leftarrow p(x, \mathbf{t}^{\star\star})$ to the program.

Now the stable models we had before are extended with ground query atoms. In $\mathcal{M}_1$ we find $ans(a), ans(b)$. In $\mathcal{M}_2$ we find $ans(b)$ only. In consequence, the tuple b is the only consistent answer to the query. □

7 Repair Programs for Referential ICs

So far we have presented repair programs for universal ICs. Now we also want to consider referential ICs (RICs) of the form (2). We assume that the variables range over the underlying database domain D that now may may contain the *null* value (a new constant). A RIC can be repaired by cascaded deletion, but also by insertion of this null value, i.e. through insertion of the atom $q(\bar{a}, null)$. If this second case, it is expected that this change will not propagate through other ICs like a full inclusion dependency of the form $\forall \bar{x}(q(\bar{x}, y) \rightarrow r(\bar{x}, y))$. The program should not detect such inconsistency wrt this IC. This can be easily avoided at the program level by appropriately qualifying the values of variables in the disjunctive repair clause for the other ICs, like the full inclusion dependency above.

The program $\Pi^\star(DB, IC)$ we presented in previous sections is, therefore, extended with the following formulas:

$$p(\bar{x}, \mathbf{f_a}) \vee q(\bar{x}', null, \mathbf{t_a}) \leftarrow p(\bar{x}, \mathbf{t}^\star),\ \mathit{not}\ aux(\bar{x}'),\ \mathit{not}\ q(\bar{x}', null, \mathbf{t_d}). \quad (5)$$
$$aux(\bar{x}') \leftarrow q(\bar{x}', y, \mathbf{t_d}),\ \mathit{not}\ q(\bar{x}', y, \mathbf{f_a}). \quad (6)$$
$$aux(\bar{x}') \leftarrow q(\bar{x}', y, \mathbf{t_a}). \quad (7)$$

Intuitively, clauses (6) and (7) detect if the formula $\exists y(q(\bar{a}', y){:}\mathbf{t} \vee q(\bar{a}', y){:}\mathbf{t_a}))$ is satisfied by the model. If this is not the case and $p(\bar{a}, \mathbf{t}^\star)$ belongs to the model (in which case (2) is violated by $\bar{a}$), and $q(\bar{a}', null)$ is not in the database, then, according to rule (5), the repair is done either by deleting $p(\bar{a})$ or inserting $q(\bar{a}', null)$.

Notice that in this section we have been departing from the definition of repair given in Section 2, in the sense that repairs are obtained now by deletion of tuples or insertion of null values only, the usual ways in which RICs are maintained. In particular, if the instance is $\{p(\bar{a})\}$ and IC contains only $p(\bar{x}) \rightarrow \exists y q(\bar{x}, y)$, then $\{p(\bar{a}), q(\bar{a}, b)\}$, with $b \in D$, will not be obtained as a repair (although it is according to the initial definition), because it will not be captured by the program. This makes sense, because allowing such repairs would produce infinitely many of them, all of which are not natural from the perspective of usual database praxis.

If we want to establish a correspondence between stable models of the new repair program and the database repairs, we need first a precise definition of a repair in the new sense, according to which repairs can be achieved by insertion of null values that do not propagate through other ICs. We proceed by first redefining when a database instance, possibly containing null values, satisfies a set of ICs.

Definition 8. *For a database instance DB, whose domain D may contain the constant null and a set of integrity constraints $IC = IC_U \cup IC_R$, where IC_U is a set of universal integrity constraints of the form $\forall \bar{x}\varphi$, with φ quantifier free, and IC_R is a set of referential integrity constraints of the form $\forall \bar{x}(p(\bar{x}) \rightarrow \exists y q(\bar{x}', y))$, with $\bar{x}' \subseteq \bar{x}$, we say that r satisfies IC, written $DB \models_\Sigma IC$ iff:*

1. *For each $\forall \bar{x}\varphi \in IC_U$, $DB \models_\Sigma \varphi[\bar{a}]$ for all $\bar{a} \in D - \{null\}$, and*
2. *For each $\forall \bar{x}(p(\bar{x}) \rightarrow \exists y q(\bar{x}', y)) \in IC_R$, if $DB \models_\Sigma p(\bar{a})$, with $\bar{a} \in D - \{null\}$, then $DB \models_\Sigma \exists y q(\bar{a}, y)$.* □

Definition 9. *Let DB, DB_1, DB_2 be database instances over the same schema and domain D (that may now contain null). It holds $DB_1 \leq_{DB} DB_2$ iff:*

1. *for every atom $p(\bar{a}) \in \Delta(DB, DB_1)$, with $\bar{a} \in D - \{null\}$, it holds $p(\bar{a}) \in \Delta(DB, DB_2)$, and*
2. *for every atom $p(\bar{a}, null) \in \Delta(DB, DB_1)$, it holds $p(\bar{a}, null) \in \Delta(DB, DB_2)$ or $p(\bar{a}, \bar{b}) \in \Delta(DB, DB_2)$ with $\bar{b} \in D - \{null\}$.* □

Definition 10. *Given a database instance DB and a set of universal and referential integrity constraints IC, a repair of DB wrt IC is a database instance DB' over the same schema and domain (plus possibly null if it was not in the domain of DB), such that $DB' \models_{\Sigma} IC$ (in the sense of Definition 8) and DB' is $\leq_{DB}$-minimal in the class of database instances that satisfy IC.* □

Example 15. Consider the universal integrity constraint $\forall xy(q(x,y) \rightarrow r(x,y))$ together with the referential integrity constraints $\forall x(p(x) \rightarrow \exists y q(x,y))$ and $\forall x(s(x) \rightarrow \exists y r(x,y))$ and the inconsistent database instance $DB = \{q(a,b), p(c), s(a)\}$. The repairs for the latter are:

i	DB_i	$\Delta(DB, DB_i)$
1	$\{q(a,b), r(a,b), p(c), q(c, null), s(a)\}$	$\{r(a,b), q(c, null)\}$
2	$\{q(a,b), r(a,b), s(a)\}$	$\{p(c), r(a,b)\}$
3	$\{p(c), q(c, null), s(a), r(a, null)\}$	$\{q(a,b), q(c, null), r(a, null)\}$
4	$\{p(c), q(c, null)\}$	$\{q(a,b), q(c, null), s(a)\}$
5	$\{s(a), r(a, null)\}$	$\{q(a,b), p(c), r(a, null)\}$
6	$\emptyset$	$\{q(a,b), p(c), s(a)\}$

In the first repair it can be seen that the atom $q(c, null)$ does not propagate through the universal constraint to $r(c, null)$. For example, the instance $DB_7 = \{q(a,b), r(a,b),\ p(c), q(c,a), r(c,a), s(a))\}$, where we have introduced $r(c,a)$ in order to satisfy the second RIC, does satisfy IC, but is not a repair because $\Delta(DB, DB_1) \leq_{DB} \Delta(DB, DB_7) = \{r(a,b), q(c,a),\ r(c,a)\}$.

If $r(a,b)$ was inserted due to the universal constraint, we do want $r(a, null)$ to be inserted in order to satisfy the second referential constraint. This fact is captured by both the definition of repair and the repair program. Actually, the instance $DB_8 = \{q(a,b), r(a,b), s(a), r(a, null))\}$ is not a repair, because $\Delta(DB, DB_2) \subseteq \Delta(DB, DB_8) = \{p(c), r(a,b), r(a, null)\}$ and, in consequence, $\Delta(DB, DB_2) \leq_{DB} \Delta(DB, DB_8)$. The program also does not consider DB_8 as a repair, because the clauses (6) and (7) detect that $r(a,b)$ is already in the repair. □

If the set of IC contains both universal ICs and referential ICs, then the repair program $\Pi^{\star}(DB, IC)$ contains now the extra rules we introduced at the beginning of this section. As before, for a stable model $\mathcal{M}$ of the program, $DB_{\mathcal{M}}$ denotes the corresponding database as in Definition 7. With the class of repairs introduced in Definition 10 it holds as before

Theorem 2. *If $\mathcal{M}$ is a coherent stable model of $\Pi^{\star}(DB, IC)$, and $DB_{\mathcal{M}}$ is finite, then $DB_{\mathcal{M}}$ is a repair of DB with respect to IC. Furthermore, the repairs obtained in this way are all the repairs of DB.* □

Example 16. Consider the database instance $\{p(\bar{a})\}$ and the following set of ICs: $p(x) \rightarrow \exists y q(x,y)$, $q(x,y) \rightarrow r(x,y)$. The program $\Pi^{\star}(DB, IC)$ is written in *DLV* as follows (`ts, tss, ta`, etc. stand for $\mathbf{t}^{\star}, \mathbf{t}^{\star\star}, \mathbf{t_a}$, etc.):[5]

[5] The domain predicate used in the program should contain all the constants different from *null* that appear in the active domain of the database.

Database contents

```
    domd(a).
    p(a,td).
```

Rules for CWA

```
    p(X,fs)   :- domd(X), not p(X,td).
    q(X,Y,fs) :- domd(X), domd(Y), not q(X,Y,td).
    r(X,Y,fs) :-  not r(X,Y,td), domd(X), domd(Y).
```

Annotation rules

```
    p(X,fs)   :- p(X,fa), domd(X).
    p(X,ts)   :- p(X,ta),domd(X).
    p(X,ts)   :- p(X,td), domd(X).
    q(X,Y,fs) :- q(X,Y,fa),domd(X),domd(Y).
    q(X,Y,ts) :- q(X,Y,ta), domd(X), domd(Y).
    q(X,Y,ts) :- q(X,Y,td), domd(X), domd(Y).
    r(X,Y,fs) :- r(X,Y,fa), domd(X), domd(Y).
    r(X,Y,ts) :- r(X,Y,ta), domd(X), domd(Y).
    r(X,Y,ts) :- r(X,Y,td), domd(X), domd(Y).
```

Rules for the ICs

```
    p(X,fa) v q(X,null,ta) :- p(X,ts), not aux(x), not q(X,null,td),domd(X).
                    aux(X) :- q(X,Y,td), not q(X,Y,fa), domd(X), domd(Y).
                    aux(X) :- q(X,Y,ta), domd(X), domd(Y).
    q(X,Y,fa) v r(X,Y,ta)  :- q(X,Y,ts), r(X,Y,fs), domd(X), domd(Y).
```

Interpretation rules

```
    p(X,tss)   :- p(X,ta).
    p(X,tss)   :- p(X,td), not p(X,fa).
    p(X,fss)   :- p(X,fa).
    p(X,fss)   :- domd(X), not p(X,td), not p(X,ta).
    q(X,Y,tss) :- q(X,Y,ta).
    q(X,Y,tss) :- q(X,Y,td), not q(X,Y,fa).
    q(X,Y,fss) :- q(X,Y,fa).
    q(X,Y,fss) :- domd(X), domd(Y),not q(X,Y,td), not q(X,Y,ta).
    r(X,Y,tss) :- r(X,Y,ta).
    r(X,Y,tss) :- r(X,Y,td), not q(X,Y,fa).
    r(X,Y,fss) :- r(X,Y,fa).
    r(X,Y,fss) :- domd(X), domd(Y), not r(X,Y,td), not r(X,Y,ta).
```

Denial constraints

```
    :- p(X,ta), p(X,fa).
    :- q(X,Y,ta), q(X,Y,fa).
    :- r(X,Y,ta),r(X,Y,fa).
```

The stable models of the program are:

```
{domd(a), p(a,td), p(a,ts), p(a,fs), p(a,fss), p(a,fa), q(a,a,fs),
r(a,a,fs), q(a,a,fss), r(a,a,fss)}

{domd(a), p(a,td), p(a,ts), p(a,tss), q(a,null,ta), q(a,a,fs),
r(a,a,fs), q(a,a,fss), r(a,a,fss), q(a,null,tss)},
```

corresponding to the database instances $\emptyset$ and $\{p(a),\ q(a, null)\}$.

If the fact $q(a, null)$ is added to the original instance, the fact `q(a,null,td)` becomes a part of the program. In this case, the program considers that the new instance $\{p(a),\ q(a, null)\}$ satisfies the RIC. It also considers that the full inclusion dependency $q(x,y) \rightarrow r(x,y)$ is satisfied, because we do not want null values to be propagated. All this is reflected in the only model of the program, namely

```
{domd(a), p(a,td), p(a,ts), q(a,null,td), p(a,tss), q(a,a,fs),
r(a,a,fs), q(a,a,fss), r(a,a,fss), q(a,null,tss)}.
```

□

If we want to impose the policy of repairing the violation of a RIC just by deleting tuples, then, rule (5) should be changed by

$$p(\bar{x}, \mathbf{f_a}) \leftarrow p(\bar{x}, \mathbf{t}^{\star}),\ not\ aux(\bar{x}'),\ not\ q(\bar{x}', null, \mathbf{t_d}),$$

that says that if the RIC is violated, then the fact $p(\bar{a})$ that produces such violation must be deleted.

If we insist in keeping the original definition of repair (Section 2), i.e. allowing $\{p(\bar{a}), q(\bar{a}, b)\}$ to be a repair for every element $b \in D$, clause (5) could be replaced by:

$$p(\bar{x}, \mathbf{f_a}) \vee q(\bar{x}', y, \mathbf{t_a}) \leftarrow p(\bar{x}, \mathbf{t}^{\star}),\ not\ aux(\bar{x}'),\ not\ q(\bar{x}', null, \mathbf{t_d}), choice(\bar{x}', y). \tag{8}$$

where $choice(\bar{X}, \bar{Y})$ is the static non-deterministic choice operator [25] that selects one value for attribute tuple $\bar{Y}$ for each value of the attribute tuple $\bar{X}$. In equation (8), $choice(\bar{x}', y)$ would select one value for y from the domain for each combination of values $\bar{x}'$. Then, this rule forces the one to one correspondence between stable models of the program and the repairs as introduced in Section 2.

8 Optimization of Repair Programs

The logic programs used to specify database repairs can be optimized in several ways. In Section 8.1 we examine certain program transformations that can lead to programs with a lower computational complexity. In Section 8.2, we address the issue of avoiding the explicit computation of negative information or of materialization of absent data, what in database applications can be a serious problem from the point of view of space and time complexity.

Other possible optimizations, that are not further discussed here, have to do with avoiding the complete computation of all stable models (the repairs) whenever a query is to be answered. The query rewriting methodology introduced in [2] had this advantage: inconsistencies were solved locally, without having to restore the consistency of the complete database. In contrast, the logic programming base methodology, at least if implemented in a straightforward manner, computes all stable models. This issue is related to finding methodologies for minimizing the number of rules to be instantiated, the way ground instantiations are done, avoiding evaluation of irrelevant subgoals, etc. Further implementation issues are discussed in Section 9.

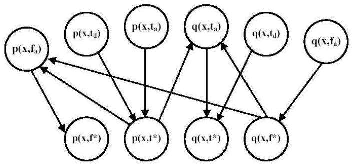
8.1 Head cycle free programs

In some cases, the repair programs we have introduced may be transformed into equivalent non disjunctive programs. This is the case of head-cycle-free programs [10] introduced below. These programs have better computational complexity than general disjunctive programs in the sense that the complexity is reduced from Π_2^P-complete to coNP-complete [18, 29].

The *dependency graph* of a ground disjunctive program Π is defined as a directed graph where each literal is a node and there is an arch from L to L' iff there is a rule in which L appears positive in the body and L' appears in the head. Π is *headcycle free* (HCF) iff its dependency graph does not contain directed cycles that go through two literals that belong to the head of the same rule.

A disjunctive program Π is HCF if its ground version is HCF. If this is the case, Π can be transformed into a non disjunctive normal program $sh(\Pi)$ with the same stable models that is obtained by replacing every disjunctive rule of the form: $\bigvee_{i=1}^{n} p_i(\bar{x}_i) \leftarrow \bigwedge_{j=1}^{m} q_j(\bar{y}_j)$ by the n following rules $p_i(\bar{x}_i) \leftarrow \bigwedge_{j=1}^{m} q_j(\bar{y}_j) \wedge \bigwedge_{k \neq i}\ not\ p_k(\bar{x}_k),\ i = 1, ..., n$. Such transformations can be justified or discarded on the basis of a careful analysis of the intrinsic complexity of consistent query answering [15]. If the original program can be transformed into a normal program, then also other efficient implementations could be used for query evaluation, e.g. *XSB* [34], that has been already successfully applied in the context of consistent query answering via query transformation, with non-existentially quantified conjunctive queries [14].

Example 17. (example 12 continued)



The repair program is HCF because, as it can be seen from the (relevant part of the) dependency graph, there is no cycle involving both $p(x, \mathbf{f_a})$ and $q(x, \mathbf{t_a})$, the atoms that appear in the only disjunctive head.

The non disjunctive version of the program has the disjunctive clause replaced by the two definite clauses $p(x, \mathbf{f_a}) \leftarrow p(x, \mathbf{t}^{\star}), q(x, \mathbf{f}^{\star}),\ not\ q(x, \mathbf{t_a})$, and $q(x, \mathbf{t_a}) \leftarrow p(x, \mathbf{t}^{\star}), q(x, \mathbf{f}^{\star}),\ not\ p(x, \mathbf{f_a})$. The two programs have the same stable models. □

In the rest of this section we will consider a set IC of universal ICs of the form

$$q_1(\bar{t}_1) \vee \cdots \vee q_n(\bar{t}_n) \leftarrow p_1(\bar{s}_1) \wedge \cdots \wedge p_m(\bar{s}_m). \qquad (9)$$

(the rule version of (1)). We denote with $ground(IC)$ the set of ground instantiations of the clauses in IC in D. A ground literal l is *bilateral* with respect to $ground(IC)$ if appears in the head of a rule in $ground(IC)$ and in the body of a possibly different rule in $ground(IC)$.

Example 18. In $ground(IC) = \{s(a,b) \rightarrow s(a,b) \vee r(a,b),\ r(a,b) \rightarrow r(b,a)\}$, the literals $s(a,b)$ and $r(a,b)$ are bilateral, because they appear in a head of a rule and in a body of a rule. Instead, $r(b,a)$ is not bilateral. □

The following theorem tells us how to check if the repair program is HCF by analyzing just the set of ICs.

Theorem 3. *The program $\Pi^\star(DB,IC)$ is HCF iff $ground(IC)$ does not have a pair of bilateral literals in the same rule.* □

Example 19. If $IC = \{s(x,y) \rightarrow r(x),\ r(x) \rightarrow p(x)\}$ and the domain is $D = \{a,b\}$, we have $ground(IC) = \{s(a,a) \rightarrow r(a),\ s(a,b) \rightarrow r(a),\ s(b,a) \rightarrow r(b),\ s(b,b) \rightarrow r(b),\ r(a) \rightarrow p(a),\ r(b) \rightarrow p(b)\}$. The bilateral literals are $r(a)$ and $r(b)$. The program $\Pi^\star(DB,IC)$ is HCF because $r(a)$ and $r(b)$ do not appear in a same rule in $ground(IC)$. As a consequence, the clause $s(x,y,\mathbf{f_a}) \vee r(x,\mathbf{t_a}) \leftarrow s(x,y,\mathbf{f}^\star), r(x,\mathbf{t}^\star)$ of $\Pi^\star(DB,IC)$, for example, can be replaced in $sh(\Pi^\star(DB,IC))$ by the two clauses $s(x,y,\mathbf{f_a}) \leftarrow s(x,y,\mathbf{f}^\star), r(x,\mathbf{t}^\star), not\ r(x,\mathbf{t_a})$ and $r(x,\mathbf{t_a}) \leftarrow s(x,y,\mathbf{f}^\star), r(x,\mathbf{t}^\star), not\ s(x,y,\mathbf{f_a})$. □

Example 20. If $IC = \{s(x) \rightarrow r(x),\ p(x) \rightarrow s(x),\ u(x,y) \rightarrow p(x)\}$ and the domain is $D = \{a,b\}$, we have $ground(IC) = \{s(a) \rightarrow r(a),\ p(a) \rightarrow s(a),\ u(a,a) \rightarrow p(a),\ s(b) \rightarrow r(b),\ p(b) \rightarrow s(b),\ u(b,b) \rightarrow p(b),\ u(a,b) \rightarrow p(a),\ u(b,a) \rightarrow p(b)\}$. The bilateral literals in $ground(IC)$ are $s(a), s(b), p(a), p(b)$. The program $\Pi^\star(DB,IC)$ is not HCF, because there are pairs of bilateral literals appearing in the same rule in $ground(IC)$, e.g. $\{s(a),p(a)\}$ and $\{s(b),p(b)\}$. □

Corollary 1. *If IC contains only* denial constraints*, i.e. formulas of the form $\bigvee_{i=1}^n p_i(\bar{t}_i) \rightarrow \varphi$, where $p_i(\bar{t}_i)$ is an atom and φ is a formula containing built-in predicates only, then $\Pi^\star(DB,IC)$ is HCF.* □

Example 21. For $IC = \{\forall xyzuv(p(x,y,z) \wedge p(x,u,v) \rightarrow y = u),\ \forall xyzuv(p(x,y,z) \wedge p(x,u,v) \rightarrow z = v),\ \forall xyzv(q(x,y,z) \wedge p(x,y,v) \rightarrow z = v)\}$, and any ground instantiation, there are no bilateral literals. In consequence, the program $\Pi^\star(DB,IC)$ will be always HCF. □

This corollary includes important classes of *IC*s as key constraints, functional dependencies and range constraints. In [15] it was shown that, for this kind of ICs, the intrinsic lower bound complexity of consistent query answering is coNP-complete. The corollary shows that by means of the transformed program this lower bound is achieved.

8.2 Avoiding materialization of the CWA

The repair programs introduced in Section 5 contain clauses of the form $p(\bar{x}, f^\star) \leftarrow\ not\ p(\bar{x}, t_d)$, that have the consequence of materializing negative information in the stable models of the programs. The repairs programs can be optimized, making them compute only that negative data that is needed to obtain the database repairs.

First, by unfolding, atoms of the form $p(\bar{x}, \mathbf{f}^\star)$ that appear as subgoals in the bodies are replaced by their definitions. More precisely, replace every rule that contains an atom of the form $p(\bar{x}, \mathbf{f}^\star)$ in the body, by two rules, one replacing the atom by $p(\bar{x}, \mathbf{f_a})$, and another replacing the atom by $not\ p(\bar{x}, \mathbf{t_d})$. Next, eliminate from the repair program those rules that have atoms annotated with $\mathbf{f}^{\star\star}$ or $\mathbf{f}^\star$ in their heads, because they compute data that should not be explicitly contained in the repairs. If $\Pi^{\star opt}(DB,IC)$ denotes the program obtained after applying these two transformations, it is easy to see that the following holds

Proposition 4. *$\Pi^{\star opt}(DB,IC)$ and $\Pi^\star(DB,IC)$ produce the same database repairs, more precisely, they compute exactly the same database instances in the sense of Definition 7.* □

Example 22. (example 16 continued) The optimized program $\Pi^{\star opt}(DB,IC)$ is as below and determines the same repairs as the original program. Notice that the second disjunctive rule in the original program was replaced by two new rules in the new program.

```
domd(a).
p(a,td).

p(X,ts)   :- p(X,ta),domd(X).
p(X,ts)   :- p(X,td), domd(X).

q(X,Y,ts) :- q(X,Y,ta), domd(X), domd(Y).
q(X,Y,ts) :- q(X,Y,td), domd(X), domd(Y).

r(X,Y,ts) :- r(X,Y,ta), domd(X), domd(Y).
r(X,Y,ts) :- r(X,Y,td), domd(X), domd(Y).

p(X,fa) v q(X,null,ta) :- p(X,ts), not aux(x), not q(X,null,td),domd(X).
                aux(X) :- q(X,Y,td), not q(X,Y,fa), domd(X), domd(Y).
                aux(X) :- q(X,Y,ta), domd(X), domd(Y).
q(X,Y,fa) v r(X,Y,ta)  :- q(X,Y,ts), r(X,Y,fa), domd(X), domd(Y).
q(X,Y,fa) v r(X,Y,ta)  :- q(X,Y,ts), not r(X,Y,td), domd(X), domd(Y).

p(X,tss)   :- p(X,ta).
p(X,tss)   :- p(X,td), not p(X,fa).

q(X,Y,tss) :- q(X,Y,ta).
q(X,Y,tss) :- q(X,Y,td), not q(X,Y,fa).

r(X,Y,tss) :- r(X,Y,ta).
r(X,Y,tss) :- r(X,Y,td), not q(X,Y,fa).

:- p(X,ta), p(X,fa).
:- q(X,Y,ta), q(X,Y,fa).
:- r(X,Y,ta),r(X,Y,fa).
```

□

The optimization for HCF programs of Section 8.1 and the one that avoids the materialization of unnecessary negative data can be combined.

Theorem 4. *If $\Pi^\star(DB,IC)$ is HCF, then $sh(\Pi^\star(DB,IC))^{opt}$ and $\Pi^\star(DB,IC)$ compute the same database repairs in the sense of Definition 7.* □

9 Conclusions

We have presented a general treatment of consistent query answering for first order queries and universal and referential ICs that is based on annotated predicate calculus (APC). Integrity constraints and database information are translated into a theory written in APC in such a way that there is a correspondence between the minimal models of the new theory and the repairs of the original database.

We have also shown how to specify database repairs by means of classical disjunctive logic programs with stable model semantics. Those programs have annotations as new arguments, and are inspired by the APC theory mentioned above. In consequence, consistent query answers can be obtained by "running" a query program together with the specification program. We illustrated their use by means of the *DLV* system. Finally, some optimizations of the repair programs were introduced.

The problem of consistent query answering was explicitly presented in [2], where also the notions of repair and consistent answer were formally defined. In addition, a methodology for consistent query answering based on a rewriting of the original query was developed (and further investigated and implemented in [14]). Basically, if we want the consistent answers to a FO query expressed in, say SQL2, a new query in SQL2 can be computed, such that its usual answers from the database are the consistent answers to the original query. That methodology has a polynomial data complexity, and that is the reason why it works for some restricted classes of FO ICs and queries, basically for non existentially quantified conjunctive queries [1]. Actually, in [15] it is shown that the problem of CQA is coNP-complete for simple functional dependencies and existential queries.

In this paper, we have formulated the problem of CQA as a problem of nonmonotonic reasoning, more precisely of minimal entailment, whose complexity, even in the propositional case, can be at least Π_2^P-complete [19]. Having a problem of nonmonotonic reasoning with such complexity, it is not strange to try to use disjunctive logic programs with negation with a stable or answer set semantics to solve the problem of CQA, because such programs have nonmonotonic consequences and a Π_2^P-complete complexity [18]. Answer set programming has been successfully used in formalizing and implementing complex nommonotonic reasoning tasks [7].

Under those circumstances, the problem then is to come up with the best logic programming specifications and the best way to use them, so that the computational complexity involved does not go beyond the intrinsic, theoretical lower bound complexity of consistent query answering.

Implementation and applications are important directions of research. The logic programming environment will interact with a DBMS, where the inconsistent DB will be stored. As much of the computation as possible should be pushed into the DBMS instead of doing it at the logic programming level.

The problem of developing query evaluation mechanisms from disjunctive logic programs that are guided by the query, most likely containing free variables and then expecting a set of answers, like magic sets [1], deserves more attention from the logic programming and database communities. The current alternative relies on finding those ground query atoms that belong to all the stable models once they have been computed via a ground instantiation of the original program (see Example 11). In [20] intelligent grounding strategies for pruning in advance the instantiated program have been explored and incorporated into *DLV*. It would be interesting to explore to what extent the program can be further pruned from irrelevant rules and subgoals using information obtained by querying the original database.

As shown in [6], there are classes of ICs for which the intersection of the stable models of the repair program coincides with the well-founded semantics, which can be computed more efficiently than the stable model semantics. It could be possible to take advantage of this efficient "core" computation for consistent query answering if ways of modularizing or splitting the whole computation into a core part and a query specific part are found. Such cases were identified in [5] for FDs and aggregation queries.

In [26], a general methodology based on disjunctive logic programs with stable model semantics is used for specifying database repairs wrt universal ICs. In their approach, preferences between repairs can be specified. The program is given through a schema for rule generation.

Independently, in [4] a specification of database repairs for binary universal ICs by means of disjunctive logic programs with a stable model semantics was presented. Those programs contained both "triggering" rules and "stabilizing" rules. The former trigger local, one-step changes, and the latter stabilize the chain of local changes in a state where all the ICs hold. The same rules, among others, are generated by the rule generation schema introduced in [26].

The programs presented here also work for the whole class of universal ICs, but they are much simpler and shorter than those presented in [26, 4]. Actually, the schema presented in [26] and the extended methodology sketched in [4], both generate an exponential number of rules in terms of the number of ICs and literals in them. Instead, in the present work, due to the simplicity of the program, that takes full advantage of the relationship between the annotations, a linear number of rules is generated. Our treatment of referential ICs considerably extends what has been sketched in [4, 26].

There are several similarities between our approach to consistency handling and those followed by the belief revision/update community. Database repairs coincide with revised models defined by Winslett in [35]. The treatment in [35] is mainly propositional, but a preliminary extension to first order knowledge bases can be found in [16]. Those papers concentrate on the computation of the models of the revised theory, i.e., the repairs in our case, but not on query answering. Comparing our framework with that of belief revision, we have an empty domain theory, one model: the database instance, and a revision by a set of ICs. The revision of a database instance by the ICs produces new database instances, the repairs of the original database.

Nevertheless, our motivation and starting point are quite different from those of belief revision. We are not interested in computing the repairs *per se*, but in answering queries, hopefully using the original database as much as possible, possibly posing a modified query. If this is not possible, we look for methodologies for representing and querying simultaneously and implicitly all the repairs of the database. Furthermore, we work in a fully first-order framework.

The semantics of database updates is treated in [22], a treatment that is close to belief revision. That paper represents databases as collections of theories, in such a way that under updates a new collection of theories is generated that minimally differ from the original ones. So, there is some similarity to our database repairs. However, that paper does not consider inconsistencies, nor query answering in any sense.

Another approach to database repairs based on a logic programming semantics consists of the *revision programs* [31]. The rules in those programs explicitly declare how to enforce the satisfaction of an integrity constraint, rather than explicitly stating the ICs, e.g. $in(a) \leftarrow in(a_1), \ldots, in(a_k), out(b_1), \ldots, out(b_m)$ has the intended procedural meaning of inserting the database atom a whenever $a_1, \ldots, a_k$ are in the database, but not $b_1, \ldots, b_m$. Also a declarative, stable model semantics is given to revision programs. Preferences for certain kinds of repair actions can be captured by declaring the corresponding rules in program and omitting rules that could lead to other forms of repairs.

In [12, 28] paraconsistent and annotated logic programs, with non classical semantics, are introduced. However, in [17] some transformation methodologies for paraconsistent logic programs [12] are shown that allow assigning to them extensions of classical semantics. Our programs have a fully standard stable model semantics.

Acknowledgments: Work funded by DIPUC, MECESUP, FONDECYT Grant 1000593, CONICYT, Carleton University Start-Up Grant 9364-01, NSERC Grant 250279-02. L. Bertossi holds a Faculty Fellowship of the Center for Advanced Studies, IBM Toronto Lab. We are grateful to Marcelo Arenas, Alvaro Campos, Alberto Mendelzon, and Nicola Leone for useful conversations.

References

1. Abiteboul, S., Hull, R. and Vianu, V. *Foundations of Databases.* Addison-Wesley, 1995.
2. Arenas, M., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. In *Proc. ACM Symposium on Principles of Database Systems (ACM PODS'99)*, 1999, pp. 68–79.
3. Arenas, M., Bertossi, L. and Kifer, M. Applications of Annotated Predicate Calculus to Querying Inconsistent Databases. In *'Computational Logic - CL2000' Stream: 6th International Conference on Rules and Objects in Databases (DOOD'2000).* Springer Lecture Notes in Artificial Intelligence 1861, 2000, pp. 926–941.
4. 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.L. Larsen, J. Kacprzyk, S. Zadrozny, H. Christiansen (eds.), Springer, 2000, pp. 27–41.
5. Arenas, M., Bertossi, L. and Chomicki, J. Scalar Aggregation in FD-Inconsistent Databases. In *Database Theory - ICDT 2001*, Springer, LNCS 1973, 2001, pp. 39–53.
6. Arenas, M., Bertossi, L. and Chomicki, J. Answer Sets for Consistent Query Answers. To appear in Theory and Practice of Logic Programming.
7. Baral, C. *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press, 2003.
8. Barcelo, P. and Bertossi, L. Repairing Databases with Annotated Predicate Logic. In *Proc. Nineth International Workshop on Non-Monotonic Reasoning (NMR'2002), Special session: Changing and Integrating Information: From Theory to Practice*, S. Benferhat and E. Giunchiglia (eds.), 2002, pp. 160–170.
9. Barcelo, P. and Bertossi, L. Logic Programs for Querying Inconsistent Databases. Proc. Practical Aspects of Declarative Languages (PADL03), Springer LNCS 2562, 2003, pp. 208-222.
10. Ben-Eliyahu, R. and Dechter, R. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics in Artificial Intelligence*, 1994, 12:53-87.
11. Bertossi, L., Chomicki, J., Cortes, A. and Gutierrez, C. Consistent Answers from Integrated Data Sources. In 'Flexible Query Answering Systems', Proc. of the 5th International Conference, FQAS 2002. T. Andreasen, A. Motro, H. Christiansen, H. L. Larsen (eds.). Springer LNAI 2522, 2002, pp. 71–85.
12. Blair, H.A. and Subrahmanian, V.S. Paraconsistent Logic Programming. *Theoretical Computer Science,* 1989, 68:135-154.
13. Buccafurri, F., Leone, N. and Rullo, P. Enhancing Disjunctive Datalog by Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 2000, 12(5):845-860.
14. Celle, A. and Bertossi, L. Querying Inconsistent Databases: Algorithms and Implementation. In 'Computational Logic - CL 2000', J. Lloyd et al. (eds.). Stream: 6th International Conference on Rules and Objects in Databases (DOOD'2000). Springer Lecture Notes in Artificial Intelligence 1861, 2000, pp. 942–956.
15. Chomicki, J. and Marcinkowski, J. On the Computational Complexity of Consistent Query Answers. Submitted in 2002 (CoRR paper cs.DB/0204010).
16. Chou, T. and Winslett, M. A Model-Based Belief Revision System. *Journal of Automated Reasoning*, 1994, 12:157–208.
17. Damasio, C. V. and Pereira, L.M. A Survey on Paraconsistent Semantics for Extended Logic Programas. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, Vol. 2, D.M. Gabbay and Ph. Smets (eds.), Kluwer Academic Publishers, 1998, pp. 241–320.
18. Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 2001, 33(3): 374–425.
19. Eiter, T. and Gottlob, G. Propositional Circumscription and Extended Closed World Assumption are Π_2^p-complete. Theoretical Computer Science, 1993, 114, pp. 231-245.
20. Eiter, T., Leone, N., Mateis, C., Pfeifer, G. and Scarcello, F. A Deductive System for Non-Monotonic Reasoning. Proc. LPNMR'97, Springer LNAI 1265, 1997, pp. 364–375.
21. 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.
22. Fagin, R., Kuper, G., Ullman, J. and Vardi, M. Updating Logical Databases. In *Advances in Computing Research*, JAI Press, 1986, Vol. 3, pp. 1-18.
23. Gelfond, M. and Lifschitz, V. The Stable Model Semantics for Logic Programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, R. A. Kowalski and K. A. Bowen (eds.), MIT Press, 1988, pp. 1070–1080.
24. Gelfond, M. and Lifschitz, V. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 1991, 9:365–385.
25. Giannotti, F., Greco, S.; Sacca, D. and Zaniolo, C. Programming with Nondeterminism in Deductive Databases. *Annals of Mathematics and Artificial Intelligence*, 1997, 19(3-4).
26. Greco, G., Greco, S. and Zumpano, E. A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. In *Proc. 17th International Conference on Logic Programming, ICLP'01*, Ph. Codognet (ed.), LNCS 2237, Springer, 2001, pp. 348–364.
27. Kifer, M. and Lozinskii, E.L. A Logic for Reasoning with Inconsistency. *Journal of Automated reasoning*, 1992, 9(2):179-215.
28. Kifer, M. and Subrahmanian, V.S. Theory of Generalized Annotated Logic Programming and its Applications. *Journal of Logic Programming*, 1992, 12(4):335-368.
29. Leone, N., Rullo, P. and Scarcello, F. Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics, and Computation. *Information and Computation*, 1997, 135(2):69-112.
30. Lloyd, J.W. *Foundations of Logic Programming.* Springer Verlag, 1987.
31. Marek, V.W. and Truszczynski, M. Revision Programming. *Theoretical Computer Science*, 1998, 190(2):241–277.
32. Pradhan, S. Reasoning with Conflicting Information in Artificial Intelligence and Database Theory. PhD thesis, Department of Computer Science, University of Maryland, 2001.
33. Reiter, R. Towards a Logical Reconstruction of Relational Database Theory. In *On Conceptual Modelling*, M.L. Brodie, J. Mylopoulos, J.W. Schmidt (eds.), Springer, 1984.
34. Sagonas, K.F., Swift, T. and Warren, D.S. XSB as an Efficient Deductive Database Engine. In *Proc. of the 1994 ACM SIGMOD International Conference on Management of Data*, ACM Press, 1994, pp. 442-453.
35. Winslett, M. Reasoning about Action using a Possible Models Approach. In *Proc. Seventh National Conference on Artificial Intelligence (AAAI'88)*, 1988, pp. 89–93.

Appendix: Intermediate Results and Proofs

Proofs for Section 3

Proof of Lemma 1: This result extends a similar result in [3]. We concentrate on the cases not covered there. We have to show $\mathcal{M}(DB, DB') \models p(\bar{x})\text{:}\mathbf{f_c} \vee \exists y(q(\bar{x}', y)\text{:}\mathbf{t_d} \wedge q(\bar{x}', y)\text{:}\mathbf{t_c}) \vee \exists y(q(\bar{x}', y)\text{:}\mathbf{f_d} \wedge q(\bar{x}', y)\text{:}\mathbf{t_c})$. We have that IC contains the formula $p(\bar{x}) \rightarrow \exists y q(\bar{x}', y)$. As $DB' \models_\Sigma IC$ we must analyze two cases. The first one is $DB' \models_\Sigma \neg p(\bar{a})$. Then $I_P(p(\bar{a})) = \mathbf{f}$ or $I_P(p(\bar{a})) = \mathbf{f_a}$, so $\mathcal{M}(DB, DB') \models p(\bar{a})\text{:}\mathbf{f_c}$. The second case is $DB' \models_\Sigma q(\bar{a}', b_1), \ldots, q(\bar{a}', b_n)$ for elements $b_1, \ldots, b_n$ in the domain $(n \geq 1)$. Hence, $I_P(q(\bar{a}', b_i)) = \mathbf{t}$ or $I_P(q(\bar{a}', b_i)) = \mathbf{t_a}$, for every $1 \leq i \leq n$. Then, $\mathcal{M}(DB, DB') \models \exists y(q(\bar{a}', y)\text{:}\mathbf{t_d} \wedge q(\bar{a}', y)\text{:}\mathbf{t_c})$ or $\mathcal{M}(DB, DB') \models \exists y(q(\bar{a}', y)\text{:}\mathbf{f_d} \wedge q(\bar{a}', y)\text{:}\mathbf{t_c})$. As the analysis was done for an arbitrary value $\bar{a}$, we have that $\mathcal{M}(DB, DB') \models \mathcal{T}(DB, IC)$. □

Proof of Lemma 2: This result extends a similar result in [3]. We concentrate on the cases not covered there. We have to show $DB_\mathcal{M} \models_\Sigma p(\bar{x}) \rightarrow \exists y q(\bar{x}', y)$. Let us suppose first $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{f_c}$. Then, we either have $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{f}$ or $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{f_a}$. Hence, $DB_\mathcal{M} \models_\Sigma \neg p(\bar{a})$, and from there $DB_\mathcal{M} \models_\Sigma p(\bar{a}) \rightarrow \exists y q(\bar{a}', y)$. Let us suppose now $\mathcal{M} \models \exists y(q(\bar{a}', y)\text{:}\mathbf{t_d} \wedge q(\bar{a}', y)\text{:}\mathbf{t_c})$. Therefore, $\mathcal{M} \models q(\bar{a}', b)\text{:}\mathbf{t}$ for some element b in the domain. Hence $DB_\mathcal{M} \models_\Sigma q(\bar{a}', b)$, and from there $DB_\mathcal{M} \models_\Sigma p(\bar{a}) \rightarrow \exists y q(\bar{a}', y)$. Finally, we will assume $\mathcal{M} \models \exists y(q(\bar{a}', y)\text{:}\mathbf{f_d} \wedge q(\bar{a}', y)\text{:}\mathbf{t_c})$. Then, $\mathcal{M} \models q(\bar{a}', b)\text{:}\mathbf{t_a}$ for some element b in the domain. Hence, $DB_\mathcal{M} \models_\Sigma q(\bar{a}', b)$, and from there $DB_\mathcal{M} \models_\Sigma p(\bar{a}) \rightarrow \exists y q(\bar{a}', y)$. As this is valid for any value $\bar{a}$, we have that $DB_\mathcal{M} \models_\Sigma p(\bar{x}) \rightarrow \exists y q(\bar{x}', y)$. □

Proof of Proposition 1: By Lemma 1, we conclude that $\mathcal{M}(DB, DB') \models \mathcal{T}(DB, IC)$. Let us suppose that $\mathcal{M}(DB, DB')$ is not Δ-minimal in the class of models of $\mathcal{T}(DB, IC)$. Then, there exists $\mathcal{M} \models \mathcal{T}(DB, IC)$, such that $\mathcal{M} <_\Delta \mathcal{M}(DB, DB')$. By using this it is possible to prove that $\Delta(DB, DB_\mathcal{M}) \subsetneqq \Delta(DB, DB')$.

1. Let us suppose that $p(\bar{a}) \in \Delta(DB, DB_\mathcal{M})$. Then $p(\bar{a}) \in DB$ and $p(\bar{a}) \notin DB_\mathcal{M}$, or $p(\bar{a}) \notin DB$ and $p(\bar{a}) \in DB_\mathcal{M}$. In the first case we can conclude that $p(\bar{a})\text{:}\mathbf{t_d} \in \mathcal{T}(DB, IC)$ and $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{f} \vee p(\bar{a})\text{:}\mathbf{f_a}$. If we suppose that $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{f}$, then $\mathcal{M} \not\models p(\bar{a})\text{:}\mathbf{t_d}$, a contradiction. Thus, we have that $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{f_a}$. But $\mathcal{M} <_\Delta \mathcal{M}(DB, DB')$, and therefore $\mathcal{M}(DB, DB') \models p(\bar{a})\text{:}\mathbf{f_a}$. Then, we conclude that $p(\bar{a}) \notin DB'$, and therefore in this case it is possible to conclude that $p(\bar{a}) \in \Delta(DB, DB')$. In the second case we can conclude that $p(\bar{a})\text{:}\mathbf{f_d} \in \mathcal{T}(DB, IC)$ and $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{t} \vee p(\bar{a})\text{:}\mathbf{t_a}$. If we suppose that $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{t}$, then $\mathcal{M} \not\models p(\bar{a})\text{:}\mathbf{f_d}$, a contradiction. Thus, we have that $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{t_a}$. But $\mathcal{M} <_\Delta \mathcal{M}(DB, DB')$, and therefore $\mathcal{M}(DB, DB') \models p(\bar{a})\text{:}\mathbf{t_a}$. Then, we conclude that $p(\bar{a}) \in DB'$, and therefore in this case it is possible to conclude that $p(\bar{a}) \in \Delta(DB, DB')$. Thus, we can conclude that $\Delta(DB, DB_\mathcal{M}) \subseteq \Delta(DB, DB')$.
2. Since $\mathcal{M}(DB, DB') \not\leq_\Delta \mathcal{M}$, there exists $p(\bar{a})$ such that $\mathcal{M}(DB, DB') \models p(\bar{a})\text{:}\mathbf{t_a} \vee p(\bar{a})\text{:}\mathbf{f_a}$ and $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{t} \vee p(\bar{a})\text{:}\mathbf{f}$. By using the first fact it is possible to conclude that $p(\bar{a}) \in \Delta(DB, DB')$. If we suppose that $p(\bar{a}) \in DB$, then $p(\bar{a})\text{:}\mathbf{t_d} \in \mathcal{T}(DB, IC)$, and therefore by considering the second fact it is possible to deduce that $\mathcal{M}$ must satisfy $p(\bar{a})\text{:}\mathbf{t}$. Thus, we can conclude that in this case $p(\bar{a}) \in DB_\mathcal{M}$, and therefore $p(\bar{a}) \notin \Delta(DB, DB_\mathcal{M})$. By the other hand, if we suppose that $p(\bar{a}) \notin DB$, then $p(\bar{a})\text{:}\mathbf{f_d} \in \mathcal{T}(DB, IC)$, and therefore by considering the second fact it is possible to deduce that $\mathcal{M}$ must satisfy $p(\bar{a})\text{:}\mathbf{f}$. Thus, we can conclude that in this case $p(\bar{a}) \notin DB_\mathcal{M}$, and therefore $p(\bar{a}) \notin \Delta(DB, DB_\mathcal{M})$. Finally, we conclude that $\Delta(DB, DB') \nsubseteq \Delta(DB, DB_\mathcal{M})$.

We know that DB' is a database instance, and therefore $\Delta(DB, DB')$ must be a finite set. Thus, we can conclude that $\Delta(DB, DB_\mathcal{M})$ is a finite set, and therefore $DB_\mathcal{M}$ is a database instance. With the help of Lemma 2, we deduce that $DB_\mathcal{M} \models IC$. But this is a contradiction, since DB' is a repair of DB with respect to IC and $\Delta(DB, DB_\mathcal{M}) \subsetneqq \Delta(DB, DB')$. □

Proof of Proposition 2: By Lemma 2, we conclude that $DB_\mathcal{M} \models_\Sigma IC$. Now, we need to prove that $DB_\mathcal{M}$ is minimal. Let us suppose this is not true. Then, there is a database instance DB^* such that $DB^* \models_\Sigma IC$ and $\Delta(DB, DB^*) \subsetneqq \Delta(DB, DB_\mathcal{M})$.

1. From Lemma 1, we conclude that $\mathcal{M}(DB, DB^*) \models \mathcal{T}(DB, IC)$.
2. Now, we are going to prove that $\mathcal{M}(DB, DB^*) <_\Delta \mathcal{M}$.
   If $\mathcal{M}(DB, DB^*) \models p(\bar{a})\text{:}\mathbf{t_a}$, then we can conclude that $p(\bar{a}) \notin DB$ and $p(\bar{a}) \in DB^*$, and therefore $p(\bar{a}) \in \Delta(DB, DB^*)$. But $\Delta(DB, DB^*) \subsetneqq \Delta(DB, DB_\mathcal{M})$, and therefore $p(\bar{a}) \in DB_\mathcal{M}$. Thus, we can conclude that $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{t} \vee p(\bar{a})\text{:}\mathbf{t_a}$. If we suppose that $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{t}$, then $\mathcal{M} \not\models p(\bar{a})\text{:}\mathbf{f_d}$, but we know that $\mathcal{M} \models \mathcal{T}(DB, IC)$ and $p(\bar{a})\text{:}\mathbf{f_d} \in \mathcal{T}(DB, IC)$, since $p(\bar{a}) \notin DB$, a contradiction. Therefore, $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{t_a}$.
   If $\mathcal{M}(DB, DB^*) \models p(\bar{a})\text{:}\mathbf{f_a}$, then we can conclude that $p(\bar{a}) \in DB$ and $p(\bar{a}) \notin DB^*$, and therefore $p(\bar{a}) \in \Delta(DB, DB^*)$. But $\Delta(DB, DB^*) \subsetneqq \Delta(DB, DB_\mathcal{M})$, and therefore $p(\bar{a}) \notin DB_\mathcal{M}$. Thus, we can conclude that $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{f} \vee p(\bar{a})\text{:}\mathbf{f_a}$. If we suppose that $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{f}$, then $\mathcal{M} \not\models p(\bar{a})\text{:}\mathbf{t_d}$, but we know that $\mathcal{M} \models \mathcal{T}(DB, IC)$ and $p(\bar{a})\text{:}\mathbf{t_d} \in \mathcal{T}(DB, IC)$, since $p(\bar{a}) \in DB$, a contradiction. Therefore, $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{f_a}$. Thus, we can deduce that $\mathcal{M}(DB, DB^*) \leq_\Delta \mathcal{M}$.
   Finally, we know that there exists $p(\bar{a})$ such that it is not in $\Delta(DB, DB^*)$ and it is in $\Delta(DB, DB_\mathcal{M})$. Thus, $p(\bar{a}) \in DB$ and $p(\bar{a}) \in DB^*$, and therefore $\mathcal{M}(DB, DB^*) \models p(\bar{a})\text{:}\mathbf{t}$, or $p(\bar{a}) \notin DB$ and $p(\bar{a}) \notin DB^*$, and therefore $\mathcal{M}(DB, DB^*) \models p(\bar{a})\text{:}\mathbf{f}$. Then, we have that $\mathcal{M}(DB, DB^*) \not\models p(\bar{a})\text{:}\mathbf{t_a}$ and $\mathcal{M}(DB, DB^*) \not\models p(\bar{a})\text{:}\mathbf{f_a}$. Additionally, since $p(\bar{a}) \in \Delta(DB, DB_\mathcal{M})$, we can conclude that $p(\bar{a}) \in DB$ and $p(\bar{a}) \notin DB_\mathcal{M}$, or $p(\bar{a}) \notin DB$ and $p(\bar{a}) \in DB_\mathcal{M}$. In the first case we can conclude that $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{f_a}$. In the second case we can conclude that $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{t_a}$. Thus, we can conclude that $\mathcal{M} \models p(\bar{a})\text{:}\mathbf{t_a} \vee p(\bar{a})\text{:}\mathbf{f_a}$. Therefore we can deduce that $\mathcal{M} \not\leq_\Delta \mathcal{M}(DB, DB^*)$.

Finally, we deduce that $\mathcal{M}$ is not minimal in the class of the models of $\mathcal{T}(DB, IC)$, with respect to Δ, a contradiction. □

Proofs for Section 4

Lemma 3. *For a minimal model $\mathcal{M}$ of $\mathcal{T}(DB, IC)$ and APC formula $\varphi(\bar{x})$, $\mathcal{M} \models_{APC} (\neg\varphi)^{an}(\bar{t})$ iff $\mathcal{M} \models_{APC} \neg\varphi^{an}(\bar{t})$.*

Proof: By induction on φ.
Initial step: $\varphi(\bar{t}) = p(\bar{t})$. Trivial, by the fact that every model of $\mathcal{T}(DB, IC)$ annotates atoms either with $\mathbf{t}$, $\mathbf{f}$, $\mathbf{t_a}$ or $\mathbf{f_a}$.
Inductive step:

- $\varphi(\bar{t}) = \neg\alpha(\bar{t})$. $\mathcal{M} \models (\neg\neg\alpha)^{an}(\bar{t})$ iff $\mathcal{M} \models (\alpha)^{an}(\bar{t})$ iff $\mathcal{M} \not\models \neg(\alpha)^{an}(\bar{t})$ iff $\mathcal{M} \not\models (\neg\alpha)^{an}(\bar{t})$ (by induction hypothesis) iff $\mathcal{M} \models \neg(\neg\alpha)^{an}(\bar{t})$.
- $\varphi(\bar{t}) = \alpha(\bar{t_1}) \vee \beta(\bar{t_2}) = (\alpha \vee \beta)(\bar{t})$, where $\bar{t_1}$ is the restriction of $\bar{t}$ to α (the same for $\bar{t_2}$ and β). Now, $\mathcal{M} \models (\neg(\alpha \vee \beta))^{an}(\bar{t})$ iff $\mathcal{M} \models (\neg\alpha)^{an}(\bar{t_1})$ and $\mathcal{M} \models (\neg\beta)^{an}(\bar{t_2})$ iff $\mathcal{M} \models \neg(\alpha)^{an}(\bar{t_1})$ and $\mathcal{M} \models \neg(\beta)^{an}(\bar{t_2})$ (by induction hypothesis) iff $\mathcal{M} \models \neg(\alpha \vee \ \beta)^{an}(\bar{t})$. □

Proof of of Proposition 3: We will prove it by induction on φ.
Initial step: $\varphi(\bar{x}) = p(\bar{x})$. $DB \models_c p(\bar{t})$ iff for every repair DB' of DB, $DB' \models_\Sigma p(\bar{t})$ iff for every minimal model $\mathcal{M}$ of $\mathcal{T}(DB, IC)$, $\mathcal{M} \models p(\bar{t})\text{:}\mathbf{t} \vee p(\bar{t})\text{:}\mathbf{t_a}$ iff $\mathcal{T}(DB, IC) \models_\Delta p(\bar{t})\text{:}\mathbf{t} \vee p(\bar{t})\text{:}\mathbf{t_a}$.

Inductive step:

- $\varphi(\bar{x}) = \neg\alpha(\bar{x})$. $DB \models_c \neg\alpha(\bar{t})$ iff for every repair DB' of DB we have that $DB' \not\models_\Sigma \alpha(\bar{t})$ iff for every minimal model $\mathcal{M}$ of $\mathcal{T}(DB, IC)$, $\mathcal{M} \not\models \alpha^{an}(\bar{t})$ (by induction hypothesis) iff for every minimal model $\mathcal{M}$ of $\mathcal{T}(DB, IC)$, $\mathcal{M} \models \neg\alpha^{an}(\bar{t})$ iff $\mathcal{M} \models (\neg\alpha)^{an}(\bar{t})$ (by Lemma 3).
- $\varphi(\bar{x}) = \alpha(\bar{x_1}) \vee \beta(\bar{x_2}) = (\alpha \vee \beta)(\bar{x})$. $DB \models_c (\alpha \vee \beta)(\bar{t})$ iff for every repair DB' of DB it is true that $DB' \models_\Sigma \alpha(\bar{t_1})$ or $DB' \models_\Sigma \beta(\bar{t_2})$, where $\bar{t_i}$ is the restriction of substitution $\bar{t}$ to the variables $\bar{x_i}$, iff for every minimal model $\mathcal{M}$ of $\mathcal{T}(DB, IC)$, $\mathcal{M} \models \alpha^{an}(\bar{t_1})$ or $\mathcal{M} \models \beta^{an}(\bar{t_2})$ (by induction hypothesis) iff $\mathcal{T}(DB, IC) \models_\Delta (\alpha^{an} \vee \beta^{an})(\bar{t})$ iff $\mathcal{T}(DB, IC) \models_\Delta (\alpha \vee \beta)^{an}(\bar{t})$. □

Proofs for Section 5

Lemma 4. *If $\mathcal{M}$ is a coherent stable model of $\Pi^\star(DB, IC)$, i.e. a coherent minimal model of $(\Pi^\star(DB, IC))^\mathcal{M}$, then exactly one of the following cases holds:*

- *$p(\bar{a}, \mathbf{t_d})$, $p(\bar{a}, \mathbf{t}^\star)$ and $p(\bar{a}, \mathbf{t}^{\star\star})$ belong to $\mathcal{M}$, and no other $p(\bar{a}, v)$, for v an annotation value, belongs to $\mathcal{M}$.*
- *$p(\bar{a}, \mathbf{t_d})$, $p(\bar{a}, \mathbf{t}^\star)$, $p(\bar{a}, \mathbf{f_a})$, $p(\bar{a}, \mathbf{f}^\star)$ and $p(\bar{a}, \mathbf{f}^{\star\star})$ belong to $\mathcal{M}$, and no other $p(\bar{a}, v)$, for v an annotation value, belongs to $\mathcal{M}$.*
- *$p(\bar{a}, \mathbf{t_a})$, $p(\bar{a}, \mathbf{f}^\star)$, $p(\bar{a}, \mathbf{t}^\star)$ and $p(\bar{a}, \mathbf{t}^{\star\star})$ belong to $\mathcal{M}$, and no other $p(\bar{a}, v)$, for v an annotation value, belongs to $\mathcal{M}$.*
- *$p(\bar{a}, \mathbf{f}^\star)$ and $p(\bar{a}, \mathbf{f}^{\star\star})$ belongs to $\mathcal{M}$, and no other $p(\bar{a}, v)$, for v an annotation value, belongs to $\mathcal{M}$.*

Proof: For an atom $p(\bar{a})$ we have two possibilities:

1. $p(\bar{a}, \mathbf{t_d}) \in \mathcal{M}$. Then, $p(\bar{a}, \mathbf{t}^\star) \in \mathcal{M}$. Two cases are possible now: $p(\bar{a}, \mathbf{f_a}) \in \mathcal{M}$ or $p(\bar{a}, \mathbf{f_a}) \notin \mathcal{M}$. For the first one we also have $p(\bar{a}, \mathbf{f}^{\star\star})$, $p(\bar{a}, \mathbf{f}^\star) \in \mathcal{M}$ and $p(\bar{a}, \mathbf{t_a}) \notin \mathcal{M}$ (because $\mathcal{M}$ is coherent). For the second one, $p(\bar{a}, \mathbf{f}^\star) \notin \mathcal{M}$ (since $\mathcal{M}$ is minimal), $p(\bar{a}, \mathbf{t_a}) \notin \mathcal{M}$ (because $p(\bar{a}, \mathbf{f}^\star) \notin \mathcal{M}$ and $\mathcal{M}$ is minimal) and $p(\bar{a}, \mathbf{t}^{\star\star}) \in \mathcal{M}$. This covers the first two items in the lemma.
2. $p(\bar{a}, \mathbf{t_d}) \notin \mathcal{M}$. Then, $p(\bar{a}, \mathbf{f}^\star) \in \mathcal{M}$. Two cases are possible now: $p(\bar{a}, \mathbf{t_a}) \in \mathcal{M}$ or $p(\bar{a}, \mathbf{t_a}) \notin \mathcal{M}$. For the first one we also have $p(\bar{a}, \mathbf{t}^{\star\star})$, $p(\bar{a}, \mathbf{t}^\star) \in \mathcal{M}$ and $p(\bar{a}, \mathbf{f_a}) \notin \mathcal{M}$ (because $\mathcal{M}$ is coherent).
   For the second one, $p(\bar{a}, \mathbf{t}^\star) \notin \mathcal{M}$ (since $\mathcal{M}$ is minimal), $p(\bar{a}, \mathbf{f_a}) \notin \mathcal{M}$ (because $p(\bar{a}, \mathbf{t}^\star) \notin \mathcal{M}$ and $\mathcal{M}$ is minimal) and $p(\bar{a}, \mathbf{f}^{\star\star}) \in \mathcal{M}$. This covers the last two items in the lemma. □

From two database instances we can define a structure.

Definition 11. *For two database instances DB_1 and DB_2 over the same schema and domain, $\mathcal{M}^\star(DB_1, DB_2)$ is the Herbrand structure $\langle D, I_\mathcal{P}, I_\mathcal{B}\rangle$, where D is the domain of the database*[6] *and I_P, I_B are the interpretations for the database predicates (extended with annotation arguments) and the built-ins, respectively. I_P is defined as follows:*

- *If $p(\bar{a}) \in DB_1$ and $p(\bar{a}) \in DB_2$, then $p(\bar{a}, \mathbf{t_d})$, $p(\bar{a}, \mathbf{t}^\star)$ and $p(\bar{a}, \mathbf{t}^{\star\star}) \in I_P$.*
- *If $p(\bar{a}) \in DB_1$ and $p(\bar{a}) \notin DB_2$, then $p(\bar{a}, \mathbf{t_d})$, $p(\bar{a}, \mathbf{t}^\star)$, $p(\bar{a}, \mathbf{f_a})$, $p(\bar{a}, \mathbf{f}^\star)$ and $p(\bar{a}, \mathbf{f}^{\star\star}) \in I_P$.*
- *If $p(\bar{a}) \notin DB_1$ and $p(\bar{a}) \notin DB_2$, then $p(\bar{a}, \mathbf{f}^\star)$ and $p(\bar{a}, \mathbf{f}^{\star\star}) \in I_\mathcal{P}$.*
- *If $p(\bar{a}) \notin DB_1$ and $p(\bar{a}) \in DB_2$, then $p(\bar{a}, \mathbf{f}^\star)$, $p(\bar{a}, \mathbf{t_a})$, $p(\bar{a}, \mathbf{t}^\star)$ and $p(\bar{a}, \mathbf{t}^{\star\star}) \in I_\mathcal{P}$.*

The interpretation $I_\mathcal{B}$ is defined as expected: if q is a built-in, then $q(\bar{a}) \in I_\mathcal{B}$ iff $q(\bar{a})$ is true in classical logic, and $q(\bar{a}) \notin I_\mathcal{B}$ iff $q(\bar{a})$ is false. □

Notice that the database associated to $\mathcal{M}^\star(DB_1, DB_2)$ corresponds exactly to DB_2, i.e. $DB_{\mathcal{M}^\star(DB_1, DB_2)} = DB_2$.

Lemma 5. *If $DB' \models_\Sigma IC$, then there is a coherent model $\mathcal{M}$ of the program $(\Pi^\star(DB, IC))^\mathcal{M}$ such that $DB_\mathcal{M} = DB'$. Furthermore, the model $\mathcal{M}$ corresponds to $\mathcal{M}^\star(DB, DB')$.*

Proof: As $DB_{\mathcal{M}^\star(DB, DB')} = DB'$, we only need to show that $\mathcal{M}^\star(DB, DB')$ is a model of $(\Pi^\star(DB, IC))^{\mathcal{M}^\star(DB, DB')}$. Since $DB' \models_\Sigma \bigvee_{i=1}^n \neg p_i(\bar{a_i}) \vee \bigvee_{j=1}^m q_j(\bar{b_j}) \vee \varphi$, we have three possibilities to analyze with respect to the satisfaction of this clause. The first possibility is $DB' \models_\Sigma \neg p_i(\bar{a})$. Then, two cases arise

[6] Strictly speaking, the domain D now also contains the annotations values.

- $p_i(\bar{a}) \in DB$. Then, $p_i(\bar{a},\mathbf{f}^\star), p_i(\bar{a},\mathbf{t}_\mathbf{d}), p_i(\bar{a},\mathbf{f}_\mathbf{a}), p_i(\bar{a},\mathbf{t}^\star)$ and $p_i(\bar{a},\mathbf{f}^{\star\star})$ belong to $\mathcal{M}^\star(DB, DB')$, and the program $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$ contains the following clauses: $p_i(\bar{a},\mathbf{t}_\mathbf{d}) \leftarrow$, $p_i(\bar{a},\mathbf{t}^\star) \leftarrow p_i(\bar{a},\mathbf{t}_\mathbf{d})$, $p_i(\bar{a},\mathbf{t}^\star) \leftarrow p_i(\bar{a},\mathbf{t}_\mathbf{a})$, $p_i(\bar{a},\mathbf{f}^\star) \leftarrow p_i(\bar{a},\mathbf{f}_\mathbf{a})$, $p_i(\bar{a},\mathbf{t}^{\star\star}) \leftarrow p_i(\bar{a},\mathbf{t}_\mathbf{a})$ and $p_i(\bar{a},\mathbf{f}^{\star\star}) \leftarrow p_i(\bar{a},\mathbf{f}_\mathbf{a})$. Then, all these formulas are satisfied by $\mathcal{M}^\star(DB,DB')$. The program also contains the clause $\bigvee_{i=1}^n p_i(\bar{a},\mathbf{f}_\mathbf{a}) \vee \bigvee_{j=1}^m q_j(\bar{a},\mathbf{t}_\mathbf{a}) \leftarrow \bigwedge_{i=1}^n p_i(\bar{a},\mathbf{t}^\star) \wedge \bigwedge_{j=1}^m q_j(\bar{a},\mathbf{f}^\star) \wedge \bar{\varphi}$, which is satisfied since $p_i(\bar{a},\mathbf{f}_\mathbf{a})$ belongs to $\mathcal{M}^\star(DB,DB')$.
- $p_i(\bar{a}) \notin DB$. Then, $p_i(\bar{a},\mathbf{f}^\star)$ and $p_i(\bar{a},\mathbf{f}^{\star\star}) \in \mathcal{M}^\star(DB,DB')$, and $p_i(\bar{a},\mathbf{f}^\star)$, $p_i(\bar{a},\mathbf{t}^\star) \leftarrow p_i(\bar{a},\mathbf{t}_\mathbf{d})$, $p_i(\bar{a},\mathbf{t}^\star) \leftarrow p_i(\bar{a},\mathbf{t}_\mathbf{a})$, $p_i(\bar{a},\mathbf{f}^\star) \leftarrow p_i(\bar{a},\mathbf{f}_\mathbf{a})$, $p_i(\bar{a},\mathbf{f}^{\star\star}) \leftarrow$, $p_i(\bar{a},\mathbf{t}^{\star\star}) \leftarrow p_i(\bar{a},\mathbf{t}_\mathbf{a})$ and $p_i(\bar{a},\mathbf{f}^{\star\star}) \leftarrow p_i(\bar{a},\mathbf{f}_\mathbf{a})$ are in the program $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$. All these are satisfied by the model considered. Also the clause $\bigvee_{i=1}^n p_i(\bar{a},\mathbf{f}_\mathbf{a}) \vee \bigvee_{j=1}^m q_j(\bar{a},\mathbf{t}_\mathbf{a}) \leftarrow \bigwedge_{i=1}^n p_i(\bar{a},\mathbf{t}^\star) \wedge \bigwedge_{j=1}^m q_j(\bar{a},\mathbf{f}^\star) \wedge \bar{\varphi}$ is present, and is trivially satisfied since $p_i(\bar{a},\mathbf{t}^\star) \notin \mathcal{M}^\star(DB,DB')$.

The second possibility is $DB' \models_\Sigma q_j(\bar{a})$. The following cases arise:

- $q_j(\bar{a}) \in DB$. Then, $\mathcal{M}^\star(DB,DB')$ contains $q_j(\bar{a},\mathbf{t}_\mathbf{d})$, $q_j(\bar{a},\mathbf{t}^\star)$ and $q_j(\bar{a},\mathbf{t}^{\star\star})$, and program $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$ contains the formulas $q_j(\bar{a},\mathbf{t}_\mathbf{d}) \leftarrow$, $q_j(\bar{a},\mathbf{t}^\star) \leftarrow q_j(\bar{a},\mathbf{t}_\mathbf{d})$, $q_j(\bar{a},\mathbf{t}^\star) \leftarrow q_j(\bar{a},\mathbf{t}_\mathbf{a})$, $q_j(\bar{a},\mathbf{f}^\star) \leftarrow q_j(\bar{a},\mathbf{f}_\mathbf{a})$, $q_j(\bar{a},\mathbf{t}^{\star\star}) \leftarrow q_j(\bar{a},\mathbf{t}_\mathbf{d})$, $q_j(\bar{a},\mathbf{t}^{\star\star}) \leftarrow q_j(\bar{a},\mathbf{t}_\mathbf{a})$ and $q_j(\bar{a},\mathbf{f}^{\star\star}) \leftarrow q_j(\bar{a},\mathbf{f}_\mathbf{a})$. The structure $\mathcal{M}^\star(DB,DB')$ satisfies all these clauses. The clause $\bigvee_{i=1}^n p_i(\bar{a},\mathbf{f}_\mathbf{a}) \vee \bigvee_{j=1}^m q_j(\bar{a},\mathbf{t}_\mathbf{a}) \leftarrow \bigwedge_{i=1}^n p_i(\bar{a},\mathbf{t}^\star) \wedge \bigwedge_{j=1}^m q_j(\bar{a},\mathbf{f}^\star) \wedge \bar{\varphi}$ is also in the program, and is trivially satisfied since it holds that $q_j(\bar{a},\mathbf{f}^\star)$ does not belong to $\mathcal{M}^\star(DB,DB')$.
- $q_j(\bar{a}) \notin DB$. Then, $q_j(\bar{a},\mathbf{f}^\star)$, $q_j(\bar{a},\mathbf{t}_\mathbf{a})$, $q_j(\bar{a},\mathbf{t}^\star)$ and $q_j(\bar{a},\mathbf{t}^{\star\star})$ are in the structure $\mathcal{M}^\star(DB,DB')$, and the following formulas are in the program $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$: $q_j(\bar{a},\mathbf{f}^\star) \leftarrow$, $q_j(\bar{a},\mathbf{t}^\star) \leftarrow q_j(\bar{a},\mathbf{t}_\mathbf{d})$, $q_j(\bar{a},\mathbf{t}^\star) \leftarrow q_j(\bar{a},\mathbf{t}_\mathbf{a})$, $q_j(\bar{a},\mathbf{f}^\star) \leftarrow q_j(\bar{a},\mathbf{f}_\mathbf{a})$, $q_j(\bar{a},\mathbf{t}^{\star\star}) \leftarrow q_j(\bar{a},\mathbf{t}_\mathbf{d})$, $q_j(\bar{a},\mathbf{t}^{\star\star}) \leftarrow q_j(\bar{a},\mathbf{t}_\mathbf{a})$ and $q_j(\bar{a},\mathbf{f}^{\star\star}) \leftarrow q_j(\bar{a},\mathbf{f}_\mathbf{a})$. These are satisfied by $\mathcal{M}^\star(DB,DB')$. Also the clause $\bigvee_{i=1}^n p_i(\bar{a},\mathbf{f}_\mathbf{a}) \vee \bigvee_{j=1}^m q_j(\bar{a},\mathbf{t}_\mathbf{a}) \leftarrow \bigwedge_{i=1}^n p_i(\bar{a},\mathbf{t}^\star) \wedge \bigwedge_{j=1}^m q_j(\bar{a},\mathbf{f}^\star) \wedge \bar{\varphi}$ is in the program, and is satisfied since $q_j(\bar{a},\mathbf{t}_\mathbf{a})$ belongs to $\mathcal{M}^\star(DB,DB')$.

The third possibility is $DB' \models_\Sigma \varphi$. Then, φ is true. The clause $\bigvee_{i=1}^n p_i(\bar{a},\mathbf{f}_\mathbf{a}) \vee \bigvee_{j=1}^m q_j(\bar{a},\mathbf{t}_\mathbf{a}) \leftarrow \bigwedge_{i=1}^n p_i(\bar{a},\mathbf{t}^\star) \wedge \bigwedge_{j=1}^m q_j(\bar{a},\mathbf{f}^\star) \wedge \bar{\varphi}$ is in $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$, and is satisfied since $\mathcal{M}^\star(DB,DB') \not\models \bar{\varphi}$.

As the analysis was done for an arbitrary value $\bar{a}$, it holds that the Herbrand structure $\mathcal{M}^\star(DB,DB')$ is a model of $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$. Moreover, it is also coherent, since $\mathcal{M}^\star(DB,DB')$ was defined in such a way that does not contain both $p(\bar{a},\mathbf{t}_\mathbf{a})$ and $p(\bar{a},\mathbf{f}_\mathbf{a})$. □

The next lemma shows that if $\mathcal{M}$ is a coherent and minimal model of the program $(\Pi^\star(DB,IC))^{\mathcal{M}}$, and represents a finite database instance, then the instance satisfies the constraints.

Lemma 6. *If $\mathcal{M}$ is a coherent stable model of the program $\Pi^\star(DB,IC)$ and $DB_\mathcal{M}$ is finite, then $DB_\mathcal{M} \models_\Sigma IC$.*

Proof: We want to show $DB_\mathcal{M} \models_\Sigma \bigvee_{i=1}^n \neg p_i(\bar{x}_i) \vee \bigvee_{j=1}^m q_j(\bar{y}_j) \vee \varphi$, for every constraint in IC. Since $\mathcal{M}$ is a model of $(\Pi^\star(DB,IC))^{\mathcal{M}}$, we have that $\mathcal{M} \models \bigvee_{i=1}^n p_i(\bar{x}_i,\mathbf{f}_\mathbf{a}) \vee \bigvee_{j=1}^m q_j(\bar{y}_j,\mathbf{t}_\mathbf{a}) \leftarrow \bigwedge_{i=1}^n p_i(\bar{x}_i,\mathbf{t}^\star) \wedge \bigwedge_{j=1}^m q_j(\bar{y}_j,\mathbf{f}^\star) \wedge \bar{\varphi}$. Then, at least one of the following cases is satisfied:

- $\mathcal{M} \models p_i(\bar{a},\mathbf{f}_\mathbf{a})$. Then, $\mathcal{M} \models p_i(\bar{a},\mathbf{f}^{\star\star})$ and $p(\bar{a}) \notin DB_\mathcal{M}$ (by lemma 4). Hence, $DB_\mathcal{M} \models_\Sigma \neg p_i(\bar{a})$. Since the analysis was done for an arbitrary value $\bar{a}$, $DB_\mathcal{M} \models_\Sigma \bigvee_{i=1}^n \neg p_i(\bar{x}_i) \vee \bigvee_{j=1}^m q_j(\bar{y}_j) \vee \varphi$ holds.
- $\mathcal{M} \models q_j(\bar{a},\mathbf{t}_\mathbf{a})$. It is symmetrical to the previous one.
- It is not true that $\mathcal{M} \models \bar{\varphi}$. Then $\mathcal{M} \models \varphi$. Hence, φ is true, and $DB_\mathcal{M} \models_\Sigma \bigvee_{i=1}^n \neg p_i(\bar{x}_i) \vee \bigvee_{j=1}^m q_j(\bar{y}_j) \vee \varphi$ holds.
- $\mathcal{M} \not\models p_i(\bar{a},\mathbf{t}^\star)$. Given the model is coherent and minimal, just the last item in Lemma 4 holds. This means $\mathcal{M} \models p_i(\bar{a},\mathbf{f}^{\star\star})$, $p_i(\bar{a}) \notin DB_\mathcal{M}$ and $DB_\mathcal{M} \models_\Sigma \neg p_i(\bar{a})$. Since the analysis was done for an arbitrary value $\bar{a}$, $DB_\mathcal{M} \models_\Sigma \bigvee_{i=1}^n \neg p_i(\bar{x}_i) \vee \bigvee_{j=1}^m q_j(\bar{y}_j) \vee \varphi$ holds.
- $\mathcal{M} \not\models q_j(\bar{a},\mathbf{f}^\star)$. Given the model is coherent and minimal, just the first item in lemma 4 holds. Then, $\mathcal{M} \models q_j(\bar{a},\mathbf{t}^{\star\star})$, $q_j(\bar{a}) \in DB_\mathcal{M}$ and $DB_\mathcal{M} \models_\Sigma q_j(\bar{a})$. Since the analysis was done for an arbitrary value $\bar{a}$, $DB_\mathcal{M} \models_\Sigma \bigvee_{i=1}^n \neg p_i(\bar{x}_i) \vee \bigvee_{j=1}^m q_j(\bar{y}_j) \vee \varphi$ holds. □

Lemma 7. *Consider two database instances DB and DB' over the same schema and domain. If $\mathcal{M}$ is a coherent and minimal model of $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$, such that $\mathcal{M} \subsetneqq \mathcal{M}^\star(DB,DB')$, then there exists model $\mathcal{M}'$ such that $\mathcal{M}'$ is a coherent and minimal model of $(\Pi^\star(DB,IC))^{\mathcal{M}'}$ and $\Delta(DB,DB_{\mathcal{M}'}) \subsetneqq \Delta(DB,DB')$.*

Proof: Since $\mathcal{M}$ is a coherent and minimal model of $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$, we have that $p(\bar{a},\mathbf{t}_\mathbf{d}) \in \mathcal{M}$ iff $p(\bar{a}) \in DB$. By the way we defined $\mathcal{M}^\star(DB,DB')$ and given $\mathcal{M} \subsetneqq \mathcal{M}^\star(DB,DB')$, the only two ways that both models can differ is that, for some $p(\bar{a}) \in DB$, $\{p(\bar{a},\mathbf{f}_\mathbf{a}), p(\bar{a},\mathbf{f}^\star), p(\bar{a},\mathbf{f}^{\star\star})\} \subseteq \mathcal{M}^\star(DB,DB')$ and none of these atoms belong to $\mathcal{M}$, or for some $p(\bar{a}) \notin DB$, $\{p(\bar{a},\mathbf{t}_\mathbf{a}), p(\bar{a},\mathbf{t}^\star), p(\bar{a},\mathbf{t}^{\star\star})\} \subseteq \mathcal{M}^\star(DB,DB')$ and none of these atoms belong to $\mathcal{M}$. Now, some of the atoms in $\mathcal{M}$ may have not received an interpretation in terms of $\mathbf{t}^{\star\star}$ and $\mathbf{f}^{\star\star}$, *i.e.* $\mathcal{M}$ is not a minimal model of $(\Pi^\star(DB,IC))^{\mathcal{M}}$. Anyway, if we use the interpretation rules over $\mathcal{M}$, we will finish with a model $\mathcal{M}'$ that is a minimal model of $(\Pi^\star(DB,IC))^{\mathcal{M}'}$. From $\mathcal{M}$ the model $\mathcal{M}'$ is constructed as follows:

- If $p(\bar{a},\mathbf{t}_\mathbf{d}) \in \mathcal{M}$ and $p(\bar{a},\mathbf{f}_\mathbf{a}) \notin \mathcal{M}$, then $p(\bar{a},\mathbf{t}_\mathbf{d}), p(\bar{a},\mathbf{t}^\star)$ and $p(\bar{a},\mathbf{t}^{\star\star}) \in \mathcal{M}'$.
- If $p(\bar{a},\mathbf{t}_\mathbf{d}) \in \mathcal{M}$ and $p(\bar{a},\mathbf{f}_\mathbf{a}) \in \mathcal{M}$, then $p(\bar{a},\mathbf{t}_\mathbf{d}), p(\bar{a},\mathbf{t}^\star)$, $p(\bar{a},\mathbf{f}_\mathbf{a}), p(\bar{a},\mathbf{f}^\star)$ and $p(\bar{a},\mathbf{f}^{\star\star}) \in \mathcal{M}'$.
- If $p(\bar{a},\mathbf{t}_\mathbf{d}) \notin \mathcal{M}$ and $p(\bar{a},\mathbf{t}_\mathbf{a}) \notin \mathcal{M}$, then $p(\bar{a},\mathbf{f}^\star)$ and $p(\bar{a},\mathbf{f}^{\star\star}) \in \mathcal{M}'$.
- If $p(\bar{a},\mathbf{t}_\mathbf{d}) \notin \mathcal{M}$ and $p(\bar{a},\mathbf{t}_\mathbf{a}) \in \mathcal{M}$, then $p(\bar{a},\mathbf{f}^\star), p(\bar{a},\mathbf{t}_\mathbf{a}), p(\bar{a},\mathbf{t}^\star)$ and $p(\bar{a},\mathbf{t}^{\star\star}) \in \mathcal{M}'$.

It is clear that $\mathcal{M}'$ is a coherent and minimal model of $(\Pi^\star(DB,IC))^{\mathcal{M}'}$. It just rests to prove that $\Delta(DB,DB_{\mathcal{M}'}) \subsetneqq \Delta(DB,DB')$. First, we will prove $\Delta(DB,DB_{\mathcal{M}'}) \subseteq \Delta(DB,DB')$. Let us suppose $p(\bar{a}) \in \Delta(DB,DB_{\mathcal{M}'})$. Then, either $p(\bar{a}) \in DB$ and $p(\bar{a}) \notin DB_{\mathcal{M}'}$ or $p(\bar{a}) \notin DB$ and $p(\bar{a}) \in DB_{\mathcal{M}'}$. In the first case, $p(\bar{a},\mathbf{t}_\mathbf{d})$, $p(\bar{a},\mathbf{t}^\star)$, $p(\bar{a},\mathbf{f}_\mathbf{a})$ and $p(\bar{a},\mathbf{f}^\star)$ are in $\mathcal{M}'$. These atoms are also in $\mathcal{M}$ and, by our assumption, they are also in $\mathcal{M}^\star(DB,DB')$. Hence, $p(\bar{a}) \in \Delta(DB,DB')$. In the second case, $p(\bar{a},\mathbf{f}^\star)$, $p(\bar{a},\mathbf{t}_\mathbf{a})$ and $p(\bar{a},\mathbf{t}^\star)$ are in $\mathcal{M}'$. These atoms are also in $\mathcal{M}$ and, by our assumption, these are also in $\mathcal{M}^\star(DB,DB')$. Hence, $p(\bar{a}) \in \Delta(DB,DB')$.

We will now prove $\Delta(DB,DB_{\mathcal{M}'}) \subsetneqq \Delta(DB,DB')$. We know for some fact $p(\bar{a})$ there is an element related to it which is in $\mathcal{M}^\star(DB,DB')$ and which is not in $\mathcal{M}$. One possible case is $p(\bar{a},\mathbf{f}_\mathbf{a})$ and $p(\bar{a},\mathbf{f}^\star)$ are in $\mathcal{M}^\star(DB,DB')$ and not in $\mathcal{M}$. Then, $p(\bar{a}) \in \Delta(DB,DB')$, but $p(\bar{a}) \notin \Delta(DB,DB_{\mathcal{M}'})$. The other possible case is that $p(\bar{a},\mathbf{t}_\mathbf{a})$ and $p(\bar{a},\mathbf{t}^\star)$ are in $\mathcal{M}^\star(DB,DB')$ and not in $\mathcal{M}$. Then, $p(\bar{a}) \in \Delta(DB,DB')$, but $p(\bar{a}) \notin \Delta(DB,DB_{\mathcal{M}'})$. □

Proposition 5. *If DB' is a repair of DB with respect to IC, then there is a coherent stable model $\mathcal{M}$ of the program $\Pi^\star(DB,IC)$ such that $DB_\mathcal{M} = DB'$. Furthermore, the model $\mathcal{M}$ corresponds to $\mathcal{M}^\star(DB,DB')$.*

Proof: By Lemma 5 we have $\mathcal{M}^\star(DB,DB')$ is a coherent model of the program $\Pi^\star(DB,IC)^{\mathcal{M}^\star(DB,DB')}$. We just have to show it is minimal. Let us suppose first there exists a model $\mathcal{M}$ of $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$ such that it is the case that $\mathcal{M} \subsetneqq \mathcal{M}^\star(DB,DB')$ (it is also coherent since it is contained in $\mathcal{M}^\star(DB,DB')$). Since $\mathcal{M} \subsetneqq \mathcal{M}^\star(DB,DB')$, the model $\mathcal{M}$ contains the atom $p(\bar{a},\mathbf{t}_\mathbf{d})$ iff $p(\bar{a}) \in DB$. Then, we can assume without loss of generality that $\mathcal{M}$ is minimal (if it is not minimal, we can always generate from it a minimal model $\mathcal{M}'$, such that $\mathcal{M}' \subsetneqq \mathcal{M}$, by deleting its non-supported atoms).

By Lemma 7, there exists model $\mathcal{M}'$ such that $\Delta(DB,DB_{\mathcal{M}'}) \subsetneqq \Delta(DB,DB')$ and $\mathcal{M}'$ is a coherent and minimal model of $(\Pi^\star(DB,IC))^{\mathcal{M}'}$. By Lemma 6, $DB_{\mathcal{M}'} \models_\Sigma IC$. This contradicts our fact that DB' is a repair. □

Proposition 6. *If $\mathcal{M}$ is a coherent and minimal model of $(\Pi^\star(DB,IC))^{\mathcal{M}}$ and $DB_\mathcal{M}$ is finite, then $DB_\mathcal{M}$ is a repair of DB with respect to IC.*

Proof: From Lemma 6, we have $DB_\mathcal{M} \models_\Sigma IC$. We just have to show minimality. Let us suppose there is a database instance DB', such that $DB' \models_\Sigma IC$ and $\Delta(DB,DB') \subsetneqq \Delta(DB,DB_\mathcal{M})$. Then, by Lemma 5, $\mathcal{M}^\star(DB,DB')$ is a coherent model of $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$. We will first show it is the case that $\mathcal{M}^\star(DB,DB') \subseteqq \mathcal{M}$ and that $\mathcal{M}^\star(DB,DB')$ is a model of $(\Pi^\star(DB,IC))^{\mathcal{M}}$. Notice that since $\mathcal{M}$ is a minimal model of $(\Pi^\star(DB,IC))^{\mathcal{M}}$, this program contains the clause $p(\bar{a},\mathbf{f}^\star) \leftarrow$ for every $p(\bar{a}) \notin DB$. The rest of the program must look exactly like $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$. This is true because the only other clauses in $\Pi^\star(DB,IC)$ that contain negation in their bodies are the interpretation rules $p(\bar{a},\mathbf{f}^{\star\star}) \leftarrow \mathit{not}\ p(\bar{a},\mathbf{t}_\mathbf{d}), \mathit{not}\ p(\bar{a},\mathbf{t}_\mathbf{a})$ and $p(\bar{a},\mathbf{t}^{\star\star}) \leftarrow p(\bar{a},\mathbf{t}_\mathbf{d}), \mathit{not}\ p(\bar{a},\mathbf{f}_\mathbf{a})$. Since $\Delta(DB,DB') \subsetneqq \Delta(DB,DB_\mathcal{M})$, if $\mathcal{M}$ does not satisfy $p(\bar{a},\mathbf{f}_\mathbf{a})$ then $\mathcal{M}^\star(DB,DB')$ does not satisfy it either (this is, either both programs, $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$ and $(\Pi^\star(DB,IC))^{\mathcal{M}}$, contain the clause $p(\bar{a},\mathbf{t}^{\star\star}) \leftarrow p(\bar{a},\mathbf{t}_\mathbf{d})$ or both do not contain it) and if $\mathcal{M}$ does not satisfy $p(\bar{a},\mathbf{t}_\mathbf{a})$ then $\mathcal{M}^\star(DB,DB')$ does not satisfy it either (this is, either both programs, $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$ and $(\Pi^\star(DB,IC))^{\mathcal{M}}$, contain the clause $p(\bar{a},\mathbf{f}^{\star\star}) \leftarrow$ or both do not contain it). By Definition 11, for an arbitrary atom $p(\bar{a})$ in a model $\mathcal{M}^\star(DB,DB')$, we just have to analyze four cases:

1. Let us suppose just $p(\bar{a},\mathbf{t}^{\star\star})$, $p(\bar{a},\mathbf{t}^\star)$ and $p(\bar{a},\mathbf{t}_\mathbf{d})$ belong to $\mathcal{M}^\star(DB,DB')$. Then $p(\bar{a}) \in DB$ and $p(\bar{a}) \in DB'$. Since $p(\bar{a}) \notin \Delta(DB,DB')$, we have two possibilities. The first one saying $p(\bar{a}) \notin \Delta(DB,DB_\mathcal{M})$. Then, $p(\bar{a},\mathbf{t}^\star)$, $p(\bar{a},\mathbf{t}_\mathbf{d})$ and $p(\bar{a},\mathbf{t}^{\star\star})$ also belong to $\mathcal{M}$ and $\mathcal{M}^\star(DB,DB')$ is clearly a model of the clauses in $(\Pi^\star(DB,IC))^{\mathcal{M}}$ concerning $p(\bar{a})$. The second one saying $p(\bar{a}) \in \Delta(DB,DB_\mathcal{M})$. Again, $p(\bar{a},\mathbf{t}^\star)$, $p(\bar{a},\mathbf{t}_\mathbf{d})$ and $p(\bar{a},\mathbf{t}^{\star\star})$ belong to $\mathcal{M}$ and $\mathcal{M}^\star(DB,DB')$ is clearly a model of the clauses in $(\Pi^\star(DB,IC))^{\mathcal{M}}$ concerning $p(\bar{a})$.
2. Let us suppose now, just $p(\bar{a},\mathbf{f}^\star)$ and $p(\bar{a},\mathbf{f}^{\star\star})$ belong to $\mathcal{M}^\star(DB,DB')$. Again we have two possibilities. The first one says that $p(\bar{a}) \notin \Delta(DB,DB_\mathcal{M})$. Then, $p(\bar{a},\mathbf{f}^\star)$ and $p(\bar{a},\mathbf{f}^{\star\star})$ also belong to $\mathcal{M}$. The program $(\Pi^\star(DB,IC))^{\mathcal{M}}$ contains (among others) the clause $p(\bar{a},\mathbf{f}^\star) \leftarrow$, that is satisfied by the program $\mathcal{M}^\star(DB,DB')$. The rest of the clauses concerning $p(\bar{a})$ are satisfied because are also present in $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$. The second one says that $p(\bar{a}) \in \Delta(DB,DB_\mathcal{M})$. Again, $p(\bar{a},\mathbf{f}^\star)$ and $p(\bar{a},\mathbf{f}^{\star\star})$ belong to $\mathcal{M}$. The program $(\Pi^\star(DB,IC))^{\mathcal{M}}$ contains (among others) the clause $p(\bar{a},\mathbf{f}^\star) \leftarrow$, that is satisfied by $\mathcal{M}^\star(DB,DB')$. The rest of the clauses concerning $p(\bar{a})$ are satisfied because they are also present in $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$.
3. Let us suppose just $p(\bar{a},\mathbf{t}^\star)$, $p(\bar{a},\mathbf{t}_\mathbf{d})$, $p(\bar{a},\mathbf{f}_\mathbf{a})$, $p(\bar{a},\mathbf{f}^\star)$ and $p(\bar{a},\mathbf{f}^{\star\star})$ belong to the model $\mathcal{M}^\star(DB,DB')$. Then $p(\bar{a}) \in DB$ and $p(\bar{a}) \notin DB'$. Hence, $p(\bar{a}) \in \Delta(DB,DB')$, and due to our assumption $p(\bar{a}) \in \Delta(DB,DB_\mathcal{M})$. Therefore, $p(\bar{a},\mathbf{t}^\star)$, $p(\bar{a},\mathbf{t}_\mathbf{d})$, $p(\bar{a},\mathbf{f}_\mathbf{a})$, $p(\bar{a},\mathbf{f}^\star)$ and $p(\bar{a},\mathbf{f}^{\star\star})$ belong to $\mathcal{M}$. Moreover, $\mathcal{M}^\star(DB,DB')$ is clearly a model of the clauses in $(\Pi^\star(DB,IC))^{\mathcal{M}}$ concerning $p(\bar{a})$.
4. Finally, we will suppose just $p(\bar{a},\mathbf{f}^\star)$, $p(\bar{a},\mathbf{t}_\mathbf{a})$, $p(\bar{a},\mathbf{t}^\star)$ and $p(\bar{a},\mathbf{t}^{\star\star})$ belong to the model $\mathcal{M}^\star(DB,DB')$. Then, $p(\bar{a}) \notin DB$ and $p(\bar{a}) \in DB'$. Hence, $p(\bar{a}) \in \Delta(DB,DB')$, and due to our assumption $p(\bar{a}) \in \Delta(DB,DB_\mathcal{M})$. Therefore, $p(\bar{a},\mathbf{f}^\star)$, $p(\bar{a},\mathbf{t}^\star)$, $p(\bar{a},\mathbf{t}_\mathbf{a})$ and $p(\bar{a},\mathbf{t}^{\star\star})$ belong to $\mathcal{M}$. The program $(\Pi^\star(DB,IC))^{\mathcal{M}}$ contains (among others) the clause $p(\bar{a},\mathbf{f}^\star) \leftarrow$, that is satisfied by $\mathcal{M}^\star(DB,DB')$. The rest of the clauses concerning $p(\bar{a})$ are satisfied because are also present in $(\Pi^\star(DB,IC))^{\mathcal{M}^\star(DB,DB')}$.

We will now show $\mathcal{M}^\star(DB,DB') \subsetneqq \mathcal{M}$. We have assumed there is an element of $\Delta(DB,DB_\mathcal{M})$ that is not an element of $\Delta(DB,DB')$. Thus, for some element $p(\bar{a})$, either $p(\bar{a}) \in DB$, $p(\bar{a}) \in DB'$ and $p(\bar{a}) \notin DB_\mathcal{M}$, or $p(\bar{a}) \notin DB$, $p(\bar{a}) \notin DB'$ and $p(\bar{a}) \in DB_\mathcal{M}$. For the first one we have $\mathcal{M}^\star(DB,DB')$ satisfies $p(\bar{a},\mathbf{t}_\mathbf{d})$ and $p(\bar{a},\mathbf{t}^\star)$, and $\mathcal{M}$ satisfies $p(\bar{a},\mathbf{t}_\mathbf{d})$ and $p(\bar{a},\mathbf{t}^\star)$, but also satisfies $p(\bar{a},\mathbf{f}_\mathbf{a})$ and $p(\bar{a},\mathbf{f}^\star)$. In the second one, $\mathcal{M}^\star(DB,DB')$ satisfies $p(\bar{a},\mathbf{f}^\star)$ and $\mathcal{M}$ satisfies $p(\bar{a},\mathbf{f}^\star)$, but also $p(\bar{a},\mathbf{t}_\mathbf{a})$ and $p(\bar{a},\mathbf{t}^\star)$. Then, $\mathcal{M}$ is not a minimal model; a contradiction. □

Proof of of Theorem 1: From Propositions 5 and 6. □

Proofs for Section 7

The following is an extension of Lemma 4, considering the introduction of *null* values.

Lemma 8. *If $\mathcal{M}$ is a coherent stable model of $\Pi^\star(DB, IC)$, i.e. a coherent minimal model of $(\Pi^\star(DB, IC))^{\mathcal{M}}$, then exactly one of the following cases holds:*

- *$p(\bar{a}, \mathbf{t_d})$, $p(\bar{a}, \mathbf{t}^\star)$ and $p(\bar{a}, \mathbf{t}^{\star\star})$ belong to $\mathcal{M}$, and no other $p(\bar{a}, v)$, for v an annotation value, belongs to $\mathcal{M}$.*
- *$p(\bar{a}, \mathbf{t_d})$, $p(\bar{a}, \mathbf{t}^\star)$, $p(\bar{a}, \mathbf{f_a})$, $p(\bar{a}, \mathbf{f}^\star)$ and $p(\bar{a}, \mathbf{f}^{\star\star})$ belong to $\mathcal{M}$, and no other $p(\bar{a}, v)$, for v an annotation value, belongs to $\mathcal{M}$.*
- *$p(\bar{a}, \mathbf{t_a})$, $p(\bar{a}, \mathbf{f}^\star)$, $p(\bar{a}, \mathbf{t}^\star)$ and $p(\bar{a}, \mathbf{t}^{\star\star})$ belong to $\mathcal{M}$, and no other $p(\bar{a}, v)$, for v an annotation value, belongs to $\mathcal{M}$.*
- *$p(\bar{a}, \mathbf{f}^\star)$ and $p(\bar{a}, \mathbf{f}^{\star\star})$ belongs to $\mathcal{M}$, and no other $p(\bar{a}, v)$, for v an annotation value, belongs to $\mathcal{M}$.*
- *$p(\bar{a}, null, \mathbf{t_d})$ and $p(\bar{a}, null, \mathbf{t}^{\star\star})$ belongs to $\mathcal{M}$, and no other $p(\bar{a}, null, v)$ for v an annotation value, belongs to $\mathcal{M}$.*
- *$p(\bar{a}, null, \mathbf{t_a})$, $p(\bar{a}, null, \mathbf{t}^{\star\star})$ belongs to $\mathcal{M}$, and no other $p(\bar{a}, null, v)$, for v an annotation value, belongs to $\mathcal{M}$.*
- *$\nexists v\, p(\bar{a}, null, v)$ for v an annotation value.*

Proof: The first four cases where already proven in Lemma 4. The two new cases are deduced directly considering the new rules involving the referential ICs and the inclusion of *null* values. □

Definition 11 is extended to consider the atoms with *null* values as follows:

Definition 12. *For two database instances DB_1 and DB_2 over the same schema and domain, $\mathcal{M}^\star(DB_1, DB_2)$ is the Herbrand structure $\langle D, I_P, I_B\rangle$, where D is the domain of the database*[7] *and I_P, I_B are the interpretations for the database predicates (extended with annotation arguments) and the built-ins, respectively. I_P is defined as follows:*

- *If $p(\bar{a}) \in DB_1$ and $p(\bar{a}) \in DB_2$, then $p(\bar{a}, \mathbf{t_d})$, $p(\bar{a}, \mathbf{t}^\star)$ and $p(\bar{a}, \mathbf{t}^{\star\star}) \in I_P$.*
- *If $p(\bar{a}) \in DB_1$ and $p(\bar{a}) \notin DB_2$, then $p(\bar{a}, \mathbf{t_d})$, $p(\bar{a}, \mathbf{t}^\star)$, $p(\bar{a}, \mathbf{f_a})$, $p(\bar{a}, \mathbf{f}^\star)$ and $p(\bar{a}, \mathbf{f}^{\star\star}) \in I_P$.*
- *If $p(\bar{a}) \notin DB_1$ and $p(\bar{a}) \notin DB_2$, then $p(\bar{a}, \mathbf{f}^\star)$ and $p(\bar{a}, \mathbf{f}^{\star\star}) \in I_P$.*
- *If $p(\bar{a}) \notin DB_1$ and $p(\bar{a}) \in DB_2$, then $p(\bar{a}, \mathbf{f}^\star)$, $p(\bar{a}, \mathbf{t_a})$, $p(\bar{a}, \mathbf{t}^\star)$ and $p(\bar{a}, \mathbf{t}^{\star\star}) \in I_P$.*
- *If $p(\bar{a}, null) \in DB_1$ and $p(\bar{a}, null) \in DB_2$, then $p(\bar{a}, null, \mathbf{t_d})$ and $p(\bar{a}, null, \mathbf{t}^{\star\star}) \in I_P$.*
- *If $p(\bar{a}, null) \notin DB_1$ and $p(\bar{a}, null) \in DB_2$, then $p(\bar{a}, null, \mathbf{t_a})$ and $p(\bar{a}, null, \mathbf{t}^{\star\star}) \in I_P$.*

The interpretation I_B is defined as expected: if q is a built-in, then $q(\bar{a}) \in I_B$ iff $q(\bar{a})$ is true in classical logic, and $q(\bar{a}) \notin I_B$ iff $q(\bar{a})$ is false. □

Notice that, as before, the database associated to $\mathcal{M}^\star(DB_1, DB_2)$ corresponds exactly to DB_2, i.e. $DB_{\mathcal{M}^\star(DB_1, DB_2)} = DB_2$. The next lemma states that Lemma 6 still holds when considering universal and referential ICs.

Lemma 9. *If $\mathcal{M}$ is a coherent stable model of the program $\Pi^\star(DB, IC)$ and $DB_{\mathcal{M}}$ is finite, then $DB_{\mathcal{M}} \models_\Sigma IC$.*

Proof: As in Lemma 6 it was already proven that universal constraints are satisfied. As $\mathcal{M}$ satisfies: $\{aux(\bar{x}') \leftarrow q(\bar{x}', y, \mathbf{t_d}) \wedge\ not\ q(\bar{x}', y, \mathbf{f_a});\ aux(\bar{x}') \leftarrow q(\bar{x}', y, \mathbf{t_a});\ p(\bar{x}, \mathbf{f_a}) \vee q(\bar{x}', null, \mathbf{t_a}) \leftarrow p(\bar{x}, \mathbf{t}^\star) \wedge\ not\ aux(\bar{x}'),\ not\ q(\bar{x}', null, \mathbf{t_d})\}$ we have that it can be proved, as in Lemma 6 that the RICs of the form $p(\bar{x}) \rightarrow \exists y(q(\bar{x}', y))$ are satisfied by $\mathcal{M}$. □

The next lemma is a variation of Lemma 5 that considers universal and referential ICs and the fact that a database that is inconsistent wrt a RIC of the form $p(\bar{x}) \rightarrow \exists y(q(\bar{x}', y))$ can be repaired only deleting a tuple or inserting a tuple with the *null* value.

Lemma 10. *If DB' is a repair of DB, then there is a model $\mathcal{M}$ of $\Pi^\star(DB, IC)^{\mathcal{M}}$ such that $DB_{\mathcal{M}} = DB'$.*

Proof: This lemma is proved like Lemma 5, but instead of considering that $\mathcal{M} = \mathcal{M}^\star(DB, DB')$, it considers $\mathcal{M} = \mathcal{M}^\star(DB, DB') \cup \{aux_i(\bar{a}') \mid IC_i \in IC$ and IC_i is of the form $p(\bar{x}) \rightarrow \exists y\, q(\bar{x}', y)$ and $\exists y\, ((q(\bar{a}', y, \mathbf{t_d}) \in \mathcal{M}^\star(DB, DB')$ and $q(\bar{a}', y, \mathbf{f_a}) \notin \mathcal{M}^\star(DB, DB'))$ or $q(\bar{a}', y, \mathbf{t_a}) \in \mathcal{M}^\star(DB, DB'))\}$. □

The next proposition shows that Proposition 5 holds also for $\Pi^\star(DB, IC)$ extended for RICs.

Proposition 7. *If DB' is a repair of DB with respect to IC, then there is a coherent stable model $\mathcal{M}$ of $\Pi^\star(DB, IC)$ such that $DB_{\mathcal{M}} = DB'$.*

Proof: By Lemma 10 we have that $\mathcal{M} = \mathcal{M}^\star(DB, DB') \cup \{aux_i(\bar{a}') \mid IC_i \in IC$ and IC_i is of the form $p(\bar{x}) \rightarrow \exists y\, q(\bar{x}', y)$ and $\exists y\, ((q(\bar{a}', y, \mathbf{t_d}) \in \mathcal{M}^\star(DB, DB')$ and $q(\bar{a}', y, \mathbf{f_a}) \notin \mathcal{M}^\star(DB, DB'))\mathbf{or}q(\bar{a}', y, \mathbf{t_a}) \in \mathcal{M}^\star(DB, DB'))\}$ is a coherent model of the program $\Pi^\star(DB, IC)^{\mathcal{M}}$. Its minimality can be proved as done for $\mathcal{M}^\star(DB, DB')$ in Lemma 5. □

Proposition 8. *If $\mathcal{M}$ is a coherent and stable model of $\Pi^\star(DB, IC)$, and $DB_{\mathcal{M}}$ is finite, then $DB_{\mathcal{M}}$ is a repair of DB with respect to IC.*

Proof: From Lemma 9, we have $DB_{\mathcal{M}} \models_\Sigma IC$. We only need to prove that it is $\leq_{DB}$*-minimal.* This is proven in a similar way as it was done in Proposition 6, but considering $\leq_{DB}$ instead of minimality under set inclusion. □

Proof of of Theorem 2: From Propositions 7 and 8. □

[7] Strictly speaking, the domain D now also contains the annotations values.

Proofs for Section 8

Proof of of Theorem 3: ($\Leftarrow$) If the set of *ground*(IC) does not have a pair of bilateral literals in the same IC, we want to prove that the program $\Pi^\star(DB, IC)$ is HCF for any DB.

We will suppose that the program $\Pi^\star(DB, IC)$ is *not* HCF. Then the program *ground*$(\Pi(DB, IC))$ has a directed cycle that goes through two literals that belong to the head of the same rule from *ground*$(\Pi(DB, IC))$. The only rules with more than one literal in the head are the rules capturing the ICs, i.e. those of the form $\bigvee_{i=1}^{n} p_i(\bar{a_i}, \mathbf{f_a}) \vee \bigvee_{j=1}^{m} q_j(\bar{b_j}, \mathbf{t_a}) \leftarrow \bigwedge_{i=1}^{n} p_i(\bar{a_i}, \mathbf{t}^\star) \wedge \bigwedge_{j=1}^{m} q_j(\bar{b_j}, \mathbf{f}^\star) \wedge \bar{\varphi}$.

For the program no to be HCF there has to be a cycle involving:

- $P_1(\bar{a}_1, \mathbf{f_a})$ and $P_2(\bar{a}_2, \mathbf{f_a})$ or
- $Q_1(\bar{b}_1, \mathbf{t_a})$ and $Q_2(\bar{b}_2, \mathbf{t_a})$ or
- $P_1(\bar{a}_1, \mathbf{f_a})$ and $Q_1(\bar{b}_1, \mathbf{t_a})$

If we analyze the first case, we can consider that only $P_1(\bar{a}_1, \mathbf{f_a})$ might be bilateral. Figure 2 shows that no directed cycle involving $P_1(\bar{a}_1, \mathbf{f_a})$ and $P_2(\bar{a}_2, \mathbf{f_a})$ is possible. The dependency graph of the other two cases is analogous, and it is not possible to have cycles involving to literals of the head of a rule. So the program can not be HCF.

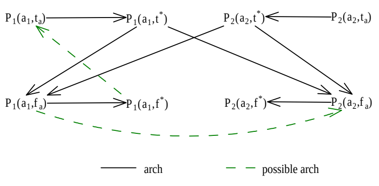


Fig. 2. Dependency Graph of P_1 and P_2

($\Rightarrow$) If the program $\Pi^\star(DB, IC)$ is HCF for any DBthen the set of instantiated ICs do not have a pair of bilateral literals in the same IC.

Let us suppose there is a pair of bilateral literals, $P_1(\bar{a}_1)$ and $Q_1(\bar{b}_1)$, in the same IC. As $P_1(\bar{a}_1)$ and $Q_1(\bar{b}_1)$ are in the same IC, there are three different cases to study. Note that P and Q can be the same predicate.

1. $P_1(\bar{a}_1)$ and $Q_1(\bar{b}_1)$ are in the head of the IC. In this case, $P_1(\bar{a}_1, \mathbf{f_a})$ and $Q_1(\bar{b}_1, \mathbf{f_a})$ are in the head of a rule of $\Pi^\star(DB, IC)$, and as it can be seen in Figure 3 there is a cycle that includes them, so the program is not HCF.

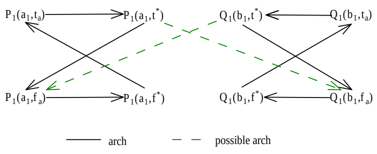


Fig. 3. Dependency Graph of P_1 and Q_1 with both of them in the head of an IC

2. $P_1(\bar{a}_1)$ and $Q_1(\bar{b}_1)$ are in the body of the IC. Analogous to first case.
3. $P_1(\bar{a}_1)$ is in the head and $Q_1(\bar{b}_1)$ is in the body of the IC. Analogous to the first case.

So, if there is a pair of bilateral literals in the same IC, the program can not be HCF, i.e. if the program is HCF, then it can not have a pair of bilateral literals in the same IC. □

Consistent Query Answering under Inclusion Dependencies

Loreto Bravo and **Leopoldo Bertossi**
Carleton University
School of Computer Science
Ottawa, Canada.
{lbravo,bertossi}@scs.carleton.ca

Abstract

For several reasons a database may not satisfy certain integrity constraints (ICs), for example, when it is the result of integrating several independent data sources. However, most likely most of the information in it is still consistent with the ICs; and could be retrieved when queries are answered. Consistent answers with respect to a set of ICs have been characterized as answers that can be obtained from every possible minimal repair of the database. In this paper we show and analyze how specify those repairs using disjunctive logic program with a stable model semantics in the presence of referential ICs. In this case, repairs are obtained by introduction of null values that do not propagate through other constraints, which makes the problem of consistent query answering decidable. We also present results about cases where the implementation of consistent query answering can be made more efficient due to the fact that the program can be simplified into a non-disjunctive program. Finally, we discuss several research issues around the implementation of system for retrieving consistent answers to queries from a DBMS.

1 Introduction

In databases, integrity constraints (ICs) capture the semantics of the application domain and help maintain the correspondence between this domain and its model provided by the database when updates on the database are performed. However, commercial database management systems (DBMSs) provide limited autonomic support to database maintenance, that is, to the process of keeping the database contents consistent with respect to certain ICs. Except for very restricted classes of integrity constraints that can be internally handled if the user has declared them together with the database schema; in general integrity constraints are maintained by means of user-defined triggers or mechanisms specified at the application level.

There are several reasons for a database to be or become inconsistent with respect to a given set of integrity constraints [9]. This could happen in several situations:

- The most common case is a DBMS that does not have a mechanism to maintain the satisfaction of certain ICs. The available DBMSs are able to maintain by themselves important classes of ICs, but not, e.g., the full class of first-order ICs.
- When data of different sources are being integrated, either virtually or under a materialized approach. In this case, even if the independent data sources are consistent with respect to certain ICs, the global integrated system might be inconsistent with respect to other global ICs [11, 12].
- If new constraints are to be imposed on a pre existing database, i.e., legacy data.
- Soft or user constraints that might be considered only when queries are answered, but without being enforced by the system.

In several cases it can be difficult, impossible or undesirable to repair the database in order to restore consistency [9]. The process may be too expensive; useful data may be lost; or it is not clear how to restore the consistency, for example, if extra information is needed. Furthermore, a user who wants to impose new constraints may have no permission to make changes on the data. In the case of data integration, the access to the sources might be restricted.

In those situations, possibly most of the data is still consistent and can be retrieved when queries are posed to the database. In [2] consistent data is characterized as the data that is invariant under all minimal restorations of consistency; i.e., as data that is present in all minimally repaired versions of the original instance (the *repairs*). In particular, an answer to a query is defined as consistent when it can be obtained as a standard answer to the query from *every possible* repair.

We envision the next DBMSs providing much more flexible and user friendly mechanisms for dealing with semantic constraints. In this direction, the system should allow the user to provide a set of ICs as another input to the query answering process, in such a way that those ICs are taken into account as answers to the query are computed. Those ICs could be entered as a another clause in a query expressed in an enhanced version of SQL, something like

```
SELECT      Name, Salary              (⋆)
FROM        Employee
WHERE       Position = 'manager'
CONSIST/W   FD: Name -> Salary;
```

where, for some reasons, the specified functional dependency (FD), which requests that attribute Name functionally determines attribute Salary, is not been maintained by the DBMS. The returned answers from the database should be only those that are consistent with FD. For example, if the underlying database is

Employee	Name	Salary	Position
	John	*55,000*	*manager*
	Peter	*50,000*	*manager*
	John	*60,000*	*manager*
	Ken	*40,000*	*secretary*

the only (consistent) answer returned by the system would be the tuple *(Peter, 50,000)*. This is because the only minimal repairs of the database are the instances

Employee1	Name	Salary	Position
	John	*55,000*	*manager*
	Peter	*50,000*	*manager*
	Ken	*40,000*	*secretary*

and

Employee2	Name	Salary	Position
	Peter	*50,000*	*manager*
	John	*60,000*	*manager*
	Ken	*40,000*	*secretary*

which are obtained by deleting each time only one of the conflicting tuples; and the only tuple that is an (usual) answer to the query (⋆) above (but without the consistency clause in the last line) in both repaired instances is *(Peter, 50,000)*.

With the same original database, if now the query is

```
SELECT      Name
From        Employee
WHERE       Position = 'manager'
CONSIST/W   FD: Name -> Salary;
```

the (consistent) answers are *(John)*, *(Peter)*, because these two names are returned as usual answers in both repairs.

We can see in this example that computing consistent query answers is different from data cleaning. We do not get rid of the tuples in the original database that participate in a violation of integrity constraints. In particular, in this case we do not lose the information about the existence of an employee named John. We can see that in consistent query answering (CQA), we could see (some of) the ICs as constraints on the query answers rather than on states of the database.

In [2, 14, 25, 3, 5], some mechanisms have been developed for CQA, that is, for retrieving consistent answer when queries are posed to an inconsistent database. All those mechanisms, in different degrees, work only with the original, inconsistent database, without restoring its consistency. Ideally, inconsistencies are solved at query time and the query is posed to the original database. For example, the (consistent) answers to the query (⋆) can be obtained posing as a standard SQL query the following rewriting of (⋆)

```
SELECT      Name, Salary
FROM        Employee
WHERE       Position = 'manager'
            AND NOT EXISTS (
   SELECT *
   FROM   Employee E
   WHERE  E.Name = Name AND
          E.Salary <> Salary);
```

which retrieves those employees, with their salaries, for which there is not other employee with the same name, but different salary. The usual answers to this query from the original database will be the consistent answers to query (⋆). No repair is needed to answer this query. Unfortunately, such first-order query rewriting based methodology provably works only for restricted classes of queries and constraints [6].

Since mechanisms that compute the consistent answers at query time without calculating the repairs [2, 14, 16] are restricted to some very limited classes of queries and constraints, more general methodologies, like the one we present in this paper, require more expressive languages to formulate the rewritings of the original queries [6]. Sometimes it is necessary to use Datalog extended with non stratified negation and disjunctive heads [1, 20]. In those cases, although the above mentioned repairs are intended to be an auxiliary concept to define the right semantics for consistent query answers, they become also an auxiliary intermediate computational step that, for complexity reasons, has to be reduced to a minimum.

In [5, 6] an algorithm is presented that deals with more general class of queries and constraints, e.g., all universal ICs and first-order queries. It is based on specifying the repairs of the database by using disjunctive logic programs with stable model semantics [20]. The complexity of this approach is higher than the complexity of the restricted ones, but the reason is that it matches the intrinsic complexity of the problem of computing consistent answers, which is provably a hard computational problem [15, 13].

In this paper we extend the methodology presented in [5, 6] in order to handle referential integrity constraints via introduction of null values that do not propagate through other ICs. The extended methodology is investigated and optimized, which allows to obtain lower complexity for some classes of ICs.

2 Repairs and Consistent Answers

We will consider a fixed relational schema $\Sigma = (\mathcal{U}, \mathcal{R} \cup \mathcal{B})$ where $\mathcal{U}$ is the possibly infinite database domain, $\mathcal{R}$ is a fixed set of database predicates, and $\mathcal{B}$ is a fixed set of built-in predicates.

A database instance can be seen as a finite collection D of ground atoms of the form $P(c_1, ..., c_n)$, where P is a predicate in $\mathcal{R}$ and $c_1, ..., c_n$ are constants in $\mathcal{U}$. Built-in predicates have a fixed and same extension in every database instance, not subject to changes.

In the following we will express integrity constraints and queries in the first-order language of relational calculus, and the latter sometimes as Datalog rules [1]. Database relations will be represented as sets of ground atoms, and not as tables as above.

A *universal integrity constraint* is a any first-order sentence that is logically equivalent to a sentence of the form

$$\overline{\forall}(\bigvee_{i=1}^{m} \neg P_i(x_i) \vee \bigvee_{j=1}^{n} Q_j(y_j) \vee \varphi), \qquad (1)$$

where $\overline{\forall}$ is a prefix of universal quantifiers, $P_i, Q_j \in \mathcal{R}$, and φ is a formula containing built-in atoms from $\mathcal{B}$ only. Notice that (1) is logically equivalent to

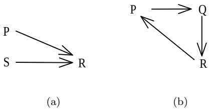
$$\forall(\bigwedge_{i=1}^{m} P_i(x_i) \ \rightarrow \ \bigvee_{j=1}^{n} Q_j(y_j) \vee \varphi). \qquad (2)$$

Example 1 For a database schema $\{Emp(id, dept),\ People(id)\}$ some universal ICs can be defined, for example, the functional dependency (FD) Emp: $id \rightarrow dept$ can be expressed by $\forall id\ dept_1\ dept_2\ (Emp(id, dept_1) \wedge Emp(id, dept_2) \rightarrow dept_1 = dept_2)$; and the full inclusion dependency (IND) $Emp[id] \subseteq People[id]$, by $\forall\ id\ dept\ (Emp(id, dept) \rightarrow People(id))$. We can see that the common universal ICs found in database praxis do not need the disjunction in the RHS of (2). □

A *referential integrity constraint* (RIC) is a sentence of the form

$$\forall x\ (P(x) \rightarrow \exists y\ Q(x', y)), \qquad (3)$$

where $x' \subseteq x$ and $P, Q \in \mathcal{R}$.



(a) (b)

Figure 1: Directed graphs for Example 3.

Example 2 For a database schema $\{Emp(id, dept),\ People(id, name)\}$, in order to represent the IND $Emp[id] \subseteq People[id]$ that states that employees are people, we use the RIC: $\forall id\ dept\ (Emp(id, dept) \rightarrow \exists name\ \ People(id, name))$. Here $x = (id, dept)$, $x' = (id)$, and $y = (name)$. □

These classes of ICs include those most common in the database praxis. We assume we have a fixed set IC of ICs that is logically consistent in the sense that is it is possible to find a database that satisfies them.

A set of RICs is said to be *acyclic* if there are no cycles in the directed graph whose vertices correspond to the relations in $\mathcal{R}$, and an edge from P to R corresponds to a RIC of the form (3).

Example 3 The RICs $\forall x(P(x) \rightarrow \exists y R(x, y))$ and $\forall x(S(x) \rightarrow \exists y R(x, y))$ are *acyclic* since there are no cycles in the directed graph as shown in Figure 1(a). On the other hand, the set of RICs: $\{\forall xz(P(x, z) \rightarrow \exists y Q(x, y)), \forall xy(Q(x, y) \rightarrow \exists z R(z, x)), \forall xy(R(x, y) \rightarrow \exists z P(x, z))\}$ is cyclic, as shown in Figure 1(b). □

A database instance D is inconsistent if it does not satisfy a given set IC of ICs. In the absence of null values it is clear when this happens, but if they are present or allowed in D, they should be treated as a special constant. Their presence in a tuple means that there are unknown values for the corresponding attributes; i.e., we have incomplete information. Since we do not have precise information about them, we will consider that no inconsistencies arise due to their presence. This leads to the following definition of consistency in the presence of the null value *null*:

Definition 1 [6] For a database instance D, whose domain $\mathcal{U}$ may contain the constant $null$, and a set of integrity constraints $IC = IC_U \cup IC_R$, where IC_U is a set of universal integrity constraints and IC_R is a set of referential integrity constraints, we say that D satisfies IC iff:

1. For each sentence in IC_U of the form $\overline{\forall}\varphi$, where $\overline{\forall}$ is a prefix of universal quantifiers and φ is a a quantifier-free formula, $\varphi[a] \in D$ for every ground tuple a of elements in $(\mathcal{U} - \{null\})$, and
2. For each sentence in IC_R of the form (3), if $P(a) \in D$, with a a ground tuple of elements in $(\mathcal{U} - \{null\})$, there exists a tuple b of constants in $\mathcal{U}$ for which $Q(a', b) \in D$. □

Intuitively, this means that a universal IC holds if it is satisfied by non-null values, and a RIC is satisfied considering only non-null values for universally quantified variables and any value for existentially quantified variables.

Example 4 Given a universal IC $\forall xy(P(x, y) \rightarrow R(x, y))$ and a RIC $\forall x(T(x) \rightarrow \exists y P(x, y))$, the database instance $D_1 = \{P(a, d),\ R(a, d),\ T(a),\ T(b),\ P(b, null)\}$ is consistent. The universal constraint is satisfied even in the presence of $P(b, null)$ since the incomplete information does not generate inconsistencies. □

If a database D is inconsistent with respect to a set of constraints IC, a *repair* of D is a new database with the same schema as D, that satisfies IC, and minimally differs from the original database under set inclusion of tuples. These repairs can be obtained from the original repair by adding or deleting tuples [2].

Example 5 Given a database with two tables and one tuple each: $\{P(a, b), R(c, e)\}$, and the universal IC $\forall xy(P(x, y) \rightarrow R(x, y))$; there are two ways of minimally repairing the database: add the tuple (a, b) to table R or delete (a, b) from table P, i.e., the repairs are $D_1 = \{P(a, b), R(c, e)R(a, b)\}$ and $D_2 = \{R(c, e)\}$. The database instance $D_3 = \{P(a, b), R(c, e), R(a, b), P(e, d)\}$ satisfies the ICs, but is not a repair because it unnecessarily adds the tuple $P(e, d)$. □

Example 5 shows how the repairs for universal ICs can be obtained. For RICs the process is different because of the presence of existential variables.

Example 6 Given a database $\{T(a)\}$ and the ICs $\forall x(T(x) \rightarrow \exists y P(x,y))$. One way of repairing the database is by deleting the tuple $T(a)$, corresponding to repair $D_1 = \{\}$. Another way would be to add a tuple $P(a,d)$ where d is any value in the database universe. In latter case, we would have as many repairs as elements in the domain. Instead of this second alternative, we will consider only one possible insertion based repair: $D_2 = \{T(a), P(a, null\}$. The null value in the second repair represents the fact that we know that there is a tuple in P with first argument a, but the second value is unknown. □

In order to formalize the concept of minimal repair, the distance between databases is defined as follows:

Definition 2 [2] 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 $\Delta(D,D') = (D \setminus D') \cup (D' \setminus D)$. □

Example 7 Consider the databases D_1, D_2 and D_3 in example 5. The distances between each of them and the original database are $\Delta(D,D_1) = \{R(a,b)\}$, $\Delta(D,D_2) = \{P(a,b)\}$ and $\Delta(D,D_3) = \{R(a,b), P(e,d)\}$. The elements in each Δ correspond to the elements added into or deleted from D to obtain D_i. □

In order to determine which databases are closer to the original one when repairing it, we define a partial order:

Definition 3 [6] Let D, D', D'' be database instances over the same schema and domain $\mathcal{U}$. It holds that $D' \leq_D D''$ iff:

1. For every atom $P(a) \in \Delta(D,D')$, with $a \in (\mathcal{U} - \{null\})$,[1] it holds that $P(a) \in \Delta(D,D'')$, and
2. For every atom $Q(a', null) \in \Delta(D,D')$, it holds that $Q(a', null) \in \Delta(D,\ D'')$ or $Q(a,b) \in \Delta(D,D'')$, for some $b \in (\mathcal{U} - \{null\})$. □

Example 8 (example 7 continued) From the distances we confirm that the instances that minimally differ from D are D_1 and D_2. Instance D_3 is not minimal, because $D_1 \leq_D D_3$, with $D_1 \neq D_3$. □

Using this partial order, we are now in position of formally define the *repairs* of an inconsistent database:

Definition 4 Given a database instance D and a set of universal and referential ICs, IC, a repair of D with respect to IC is a database instance D' over the same schema and domain (plus possibly *null* if it was not in the domain of D), such that D' satisfies IC and D' is $\leq_D$-*minimal* in the class of database instances that satisfy IC. We denote by $Rep(D)$ the set of repairs of D. □

In the absence of null-value based repairs, definitions 3 and 4 coincide with those given in [2], where RICs were not considered. The repairs of violations of universal ICs are obtained by either deleting or adding an atom without *null*. The repairs of violations of referential ICs are obtained by either deleting the atom that is generating the inconsistency or by adding an atom with a *null* value. In particular, if the instance is $\{P(a)\}$ and IC contains only $\forall x(P(x) \rightarrow \exists y Q(x,y))$, then $\{P(a), Q(a, null)\}$ will be a repair, but not $\{P(a), Q(a,b)\}$ for any $b \in \mathcal{U}, b \neq null$. In [3, 5, 13] repairs with values other than null have been considered.

Example 9 Consider the universal IC $\forall xy(P(x,y) \rightarrow R(x,y))$, together with the RIC $\forall x(T(x) \rightarrow \exists y P(x,y))$, and an inconsistent database $D = \{P(a,b), T(c)\}$ with domain $\mathcal{U} = \{a,b,c,u\}$. The repairs of D are:

i	D_i	$\Delta(D,D_i)$
1	$\{P(a,b), R(a,b), T(c),$ $P(c,null)\}$	$\{R(a,b), P(c,null)\}$
2	$\{P(a,b), R(a,b)\}$	$\{T(c), R(a,b)\}$
3	$\{T(c), P(c,null)\}$	$\{P(a,b), P(c,null)\}$
4	$\emptyset$	$\{P(a,b), T(c)\}$

We can see that in the first repair the atom $P(c, null)$ does not propagate through the universal constraint to $R(c, null)$. The instance $D_5 = \{P(a,b), R(a,b),\ T(c), P(c,a)\}$, where $P(c,a)$ has been introduced in order to satisfy the referential IC, does satisfy IC, but is not a repair because $\Delta(D,D_1) \leq_D \Delta(D,D_5) = \{R(a,b), P(c,a)\}$. □

If a database D is consistent with respect to a set of ICs, then it is its only repair.

[1] That $\bar{a} \in (\mathcal{U} - \{null\})$ means that each of the elements in tuple $\bar{a}$ belongs to $(\mathcal{U} - \{null\})$.

Definition 5 [2] Given a database instance D, a set of universal and referential ICs IC, and a first-order query $Q(x)$, we say that a ground tuple t is a *consistent answer* to Q with respect to IC iff for every $D' \in Rep(D)$, D' satisfies Q with variables x replaced by t (denoted $D' \models Q[t]$). □

Example 10 Given the IC $\forall x(T(x) \rightarrow \exists y P(x,y))$, the inconsistent database $D = \{P(a,d),\ R(a,d),\ T(a),\ T(b),\ R(b,e)\}$, and the queries $Q_1 : P(x,y)$ and $Q_2 : \forall xy\ (P(x,y) \rightarrow \exists z R(x,z))$. $Rep(D) = \{\ \{P(a,d),\ R(a,d),\ T(a),\ R(b,e)\},\ \{P(a,d),\ R(a,d),\ T(a),\ T(b),\ R(b,e), P(b, null)\}\}$. The consistent answer for Q_1 is $P(a,d)$ since that is the only element of P in both repairs. For Q_2 the consistent answer is *Yes* since the formula is satisfied in both repairs. □

3 Repair Logic Programs

The repairs of a relational database can be specified as stable models of disjunctive logic programs [24]. Once the specification has been given, in order to obtain consistent answers to a, say, first-order query Q, the latter is transformed into a query written as a logic program, which is a standard process [29, 1]. Next, this query program is "run" together with the program that specifies the repairs. This evaluation can be implemented on top of, e.g., *DLV*, a logic programming system that computes according to the stable models semantics [21, 28].

The repair programs introduced in [5, 6] use annotation constants with the intended semantics shown in the table below.

Annot.	Atom	Tuple $P(a)$ is...
$\mathbf{t_d}$	$P(a,\mathbf{t_d})$	a database fact
$\mathbf{f_d}$	$P(a,\mathbf{f_d})$	not a database fact
$\mathbf{t_a}$	$P(a,\mathbf{t_a})$	advised to be true
$\mathbf{f_a}$	$P(a,\mathbf{f_a})$	advised to be false
$\mathbf{t}^\star$	$P(a,\mathbf{t}^\star)$	true or becomes true
$\mathbf{f}^\star$	$P(a,\mathbf{f}^\star)$	false or becomes false
$\mathbf{t}^{\star\star}$	$P(a,\mathbf{t}^{\star\star})$	true in the repair

The intuitive idea behind these annotations is simple. We can think of each ground atom, say $P(c)$, in the database as annotated with the constant $\mathbf{t_d}$ in an extra argument. In consequence, we have the atom $P(c,\mathbf{t_d})$ as a fact of the program. On the other side, if an atom $P(c)$ does not belong to the database, we have the fact $P(c,\mathbf{f_d})$ in the program[2]. Now, when a violation of an IC happens, which can be expressed as the condition in the body of a program rule, then the disjunctive head of the same rule tells us how to restore consistency by deleting or inserting tuples. These recommendations are captured by means of the constants $\mathbf{t_a}, \mathbf{f_a}$, for making an atom true or false (or inserting or deleting it), resp. For example, if the IC is the full inclusion dependency $\forall x(P(x) \rightarrow Q(x))$, then we could have the program rule

$$P(x,\mathbf{f_a}) \vee Q(x,\mathbf{t_a}) \ \leftarrow\ P(x,\mathbf{t_d}), Q(x,\mathbf{f_d}),\ (4)$$

which in its body detects a violation of the IC (the tuple $P(x)$ is in the database, but not the tuple $Q(x)$). Its head advises then to either delete $P(x)$ or insert $Q(x)$ in order to restore consistency.

The problem is that there might be another IC, whose satisfaction is being restored by inserting, say $P(e)$, which is obtained by deriving the atom $P(e,\mathbf{t_a})$. If we repair in this way, but the tuple $Q(e)$ is not in the database, then there will be a new violation. The rule (4), with its body written in terms of the tuples in (outside) the original database, cannot be used to keep repairing. That is why we need to pass to intermediate notations $\mathbf{t}^\star, \mathbf{f}^\star$, that can be used to detect violations due to the original database values or to those obtained by the local repair steps. Then, instead of the program rule (4) we use the program rule

$$P(x,\mathbf{f_a}) \vee Q(x,\mathbf{t_a}) \ \leftarrow\ P(x,\mathbf{t}^\star), Q(x,\mathbf{f}^\star).\ (5)$$

The tuples with annotations $\mathbf{t}^\star$ are those obtained collecting those annotated with $\mathbf{t_d}$ or $\mathbf{t_a}$, which can be expressed by means of a new program rule. Similarly for annotation $\mathbf{f}^\star$. Finally, the atoms that can be found in a repair are those that became annotated with $\mathbf{t_a}$ or were originally annotated with $\mathbf{t_d}$, but did not become annotated with $\mathbf{f_a}$. Again, this can be expressed by program rules.

Definition 6 [6] The *repair program*, $\Pi(D, IC)$, of D with respect to IC contains the following clauses:

1. $dom(a)$ for each constant $a \in (\mathcal{U} - \{null\})$[3].
2. Fact $P(a,\mathbf{t_d})$ for every $P(a) \in D$.
3. For every predicate $P \in \mathcal{R}$, the clauses
$P(x,\mathbf{t}^\star) \leftarrow P(x,\mathbf{t_d}),\ dom(x)$.[4]
$P(x,\mathbf{t}^\star) \leftarrow P(x,\mathbf{t_a}),\ dom(x)$.
$P(x,\mathbf{f}^\star) \leftarrow P(x,\mathbf{f_a}),\ dom(x)$.
$P(x,\mathbf{f}^\star) \leftarrow dom(x),\ not\ P(x,\mathbf{t_d})$.[5]
4. For every global universal IC of form (1) the clause:
$\bigvee_{i=1}^n P_i(x_i,\mathbf{f_a}) \vee \bigvee_{j=1}^m Q_j(y_j,\mathbf{t_a}) \ \leftarrow\ \bigwedge_{i=1}^n P_i(x_i,\mathbf{t}^\star),$
$\bigwedge_{j=1}^m Q_j(y_j,\mathbf{f}^\star), dom(x),\ \varphi;$
where x is the tuple of all variables appearing in database atoms in the rule, and φ is a conjunction of built-ins equivalent to the negation of φ.
5. For every referential IC of form (3) the clauses:
$P(x,\mathbf{f_a}) \vee Q(x', null, \mathbf{t_a}) \leftarrow P(x,\mathbf{t}^\star), not\ aux(x'),$
$not\ Q(x', null, \mathbf{t_d}), dom(x).$
$aux(x') \leftarrow Q(x', y, \mathbf{t_d}),\ not\ Q(x', y, \mathbf{f_a}),\ dom(x', y).$
$aux(x') \leftarrow Q(x', y, \mathbf{t_a}),\ dom(x', y).$
6. For every predicate $P \in \mathcal{R}$, the interpretation clauses:
$P(x,\mathbf{t}^{\star\star})\ \leftarrow\ P(x,\mathbf{t_a})$.
$P(x,\mathbf{t}^{\star\star})\ \leftarrow\ P(x,\mathbf{t_d}),\ not\ P(x,\mathbf{f_a})$.
7. For every predicate $P \in \mathcal{R}$, the program denial constraint: $\leftarrow\ P(a,\mathbf{f_a}),\ P(a,\mathbf{t_a})$. □

[2] Actually, these atoms can be defined by rules of the form $P(\bar{x},\mathbf{f_d}) \leftarrow dom(\bar{x}),\ not\ P(\bar{x},\mathbf{t_d})$, where a domain predicate stores the admissible values for the tuples involved. This materialization of the closed world assumption [31] can be avoided, getting rid of annotation $\mathbf{f_d}$. Actually, the program in Definition 6 does not use it.

A logic program like this, that contains non stratified negation [1], has a stable model semantics [24]. The stable models of the program are the intended models of the program, and they sanction what is true with respect to the program. In general, a program like this will have several stable models.

Rules in 4. and 5. are the most important ones; they specify how the database is to be repaired when a violation of the IC is detected (in the body, i.e., the RHS, of the rule). The disjunction in the head (LHS) of a rule specifies the alternative ways to repair. An atom annotated with $\mathbf{t_a}$ indicates that there is an advice (what the **a** stands for) to make it true, i.e., to insert it into the database; whereas an atom annotated with $\mathbf{f_a}$ indicates an advice to make it false, i.e., to delete it from the database. The annotation constant $\mathbf{t}^\star$ is used in the bodies to give feedback to the repair rules in case there are interacting ICs. At the end, we are only interested in the atoms annotated with $\mathbf{t}^{\star\star}$ in the stable models of the repair program, since they correspond to the data elements in the repairs.

Definition 7 Let $\mathcal{M}$ be a stable model of program $\Pi(D, IC)$.
(a) The database associated to $\mathcal{M}$ is $D_{\mathcal{M}} = \{P(a) \mid P(a,\mathbf{t}^{\star\star}) \in \mathcal{M}\}$.
(b) $SM(D)$ is the set of databases associated to (the stable models of) $\Pi(D, IC)$. □

Example 11 (example 9 continued) The repair program $\Pi(D, IC)$ is the following:
1. $dom(a).\quad dom(b).\quad dom(c).\quad dom(u).$
2. $P(a,b,\mathbf{t_d}).\quad T(c,\mathbf{t_d}).$
3. $P(x,y,\mathbf{t}^\star) \leftarrow P(x,y,\mathbf{t_a}),\ dom(x),\ dom(y).$
$P(x,y,\mathbf{t}^\star) \leftarrow P(x,y,\mathbf{t_d}),\ dom(x),\ dom(y).$
(similarly for R and T)
4. $P(x,y,\mathbf{f_a}) \vee R(x,y,\mathbf{t_a}) \leftarrow P(x,y,\mathbf{t}^\star), R(x,y,\mathbf{f_a}),$
$dom(x),\ dom(y).$
$P(x,y,\mathbf{f_a}) \vee R(x,y,\mathbf{t_a})\ \leftarrow\ P(x,y,\mathbf{t}^\star),\ not$
$R(x,y,\mathbf{t_d}),\ dom(x),\ dom(y).$
5. $T(x,\mathbf{f_a}) \vee P(x, null, \mathbf{t_a}) \leftarrow T(x,\mathbf{t}^\star), not\ aux(x),$
$not\ P(x, null, \mathbf{t_d}),\ dom(x).$
$aux(x) \leftarrow P(x,y,\mathbf{t_d}),\ not\ P(x,y,\mathbf{f_a}),$
$dom(x,y).$
$aux(x) \leftarrow P(x,y,\mathbf{t_a}), dom(x,y).$
6. $P(x,y,\mathbf{t}^{\star\star}) \leftarrow P(x,y,\mathbf{t_a}).$
$P(x,y,\mathbf{t}^{\star\star}) \leftarrow P(x,y,\mathbf{t_d}),\ not\ P(x,y,\mathbf{f_a}).$
(similarly for R and T)
7. $\leftarrow P(x,y,\mathbf{t_a}), P(x,y,\mathbf{f_a}).$ (also for R, T)

Only rules 4. and 5. depend on the ICs. Rules 4. corresponds to the universal ICs. They are obtained by unfolding the annotation $\mathbf{f}^\star$ used in 4. in Definition 6 into its definition given in 3. in the same program. The rules in 5. correspond to the referential IC. These rules say how to repair the inconsistencies. Rules 2. contain the database atoms. Rules 7. are denial program constraints to discard models that contain an atom annotated with both $\mathbf{t_a}$ and $\mathbf{f_a}$. The program has four stable models:
$\mathcal{M}_1 = \{dom(a), dom(b), dom(c), dom(u), P(a,\ b,\ \mathbf{t_d}), P(a,b,\mathbf{t}^\star), T(c,\mathbf{t_d}), T(c,\mathbf{t}^\star), aux(a), T(c,\mathbf{f_a}), \underline{P(a,b,\mathbf{t}^{\star\star})}, R(a,\ b,\ \mathbf{t_a}), R(a,\ b,\ \mathbf{t}^\star), \underline{R(a,b,\mathbf{t}^{\star\star})}\}$
$\mathcal{M}_2 = \{dom(a),\ dom(b),\ dom(c), dom(u),\ P(a,\ b,\ \mathbf{t_d}),\ P(a,\ b,\ \mathbf{t}^\star),\ T(c,\ \mathbf{t_d}),\ T(c,\ \mathbf{t}^\star),\ aux(a), \underline{T(c,\mathbf{t}^{\star\star})}, P(c, null, \mathbf{t_a}), \underline{P(c, null, \mathbf{t}^{\star\star})}, \underline{P(a,b,\mathbf{t}^{\star\star})}, \underline{R(a,\ b,\ \mathbf{t_a})}, R(a,\ b,\ \mathbf{t}^\star), \underline{R(a,b,\mathbf{t}^{\star\star})}\}$
$\mathcal{M}_3 = \{dom(a),\ \underline{dom(b),\ dom(c)}, dom(u),\ P(a,\ b,\ \mathbf{t_d}),\ P(a,\ b,\ \mathbf{t}^\star),\ T(c,\ \mathbf{t_d}),\ T(c,\ \mathbf{t}^\star),\ aux(a), \underline{T(c,\mathbf{t}^{\star\star})}, P(c, null, \mathbf{t_a}), \underline{P(c, null, \mathbf{t}^{\star\star})}, P(a,b,\mathbf{f_a})\}$
$\mathcal{M}_4 = \{dom(a),\ dom(b),\ dom(c), dom(u),\ P(a,\ b,\ \mathbf{t_d}), P(a,\ b,\ \mathbf{t}^\star), T(c,\mathbf{t_d}), T(c,\mathbf{t}^\star), aux(a), T(c,\ \mathbf{f_a}), P(a,\ b,\ \mathbf{f_a})\}$

The databases associated to the program are obtained from the models by selecting the atoms annotated with $\mathbf{t}^{\star\star}$ (the underlined atoms): $D_1 = \{P(a,b), R(a,b)\}$, $D_2 = \{T(c),\ P(c, null),\ P(a,b),\ R(a,b)\}$ and $D_3 = \{T(c), P(c, null)\}$, $D_4 = \emptyset$. These repairs coincide with those obtained in example 9. □

When we have the general class of universal and referential ICs, it holds that for every repair of a database with respect to a set of ICs, there exists a model $\mathcal{M}$ of $\Pi(D, IC)$ such that its database associated is the repair, that is, all the repairs can be obtained from the repair program. In example 11 every stable model of the repair program corresponds to a repair. However, there are cases, c.f. example 12, where the database instance corresponding to a model is not a repair of the original database.

Example 12 The database instance $\{Emp(bill, ann), Emp(paul, john), Emp(john, john)\}$ stores the name of an employee with the one of his/her boss. The cyclic RIC: $\forall xy(Emp(x,y) \rightarrow \exists z Emp(y,z))$ states that each is boss is also an employee. The repairs are: $D_1 = \{Emp(paul, john), Emp(john, john)\}$ and $D_2 = \{Emp(bill, ann), Emp(ann, null), Emp(paul, john), Emp(john, john)\}$.

The repair program $\Pi(D, IC)$ is:
$dom(bill).\ \ dom(ann).\ \ dom(paul).\ \ dom(john).$
$Emp(bill, ann, \mathbf{t_d}).\quad Emp(paul, john, \mathbf{t_d}).$
$Emp(john, john, \mathbf{t_d}).$
$Emp(x,y,\mathbf{t}^\star) \leftarrow Emp(x,y,\mathbf{t_d}),\ dom(x),\ dom(y).$
$Emp(x,y,\mathbf{t}^\star) \leftarrow Emp(x,y,\mathbf{t_a}),\ dom(x),\ dom(y).$
$Emp(x,y,\mathbf{f_a}) \vee Emp(y, null, \mathbf{t_a}) \leftarrow Emp(x,y,\mathbf{t}^\star),$
$not\ Emp(y, null, \mathbf{t_d}),\ not\ aux(y),$
$dom(x),\ dom(y).$
$aux(y) \leftarrow Emp(y,z,\mathbf{t_d}),\ not\ Emp(y,z,\mathbf{f_a}),$
$dom(y),\ dom(z).$
$aux(y) \leftarrow Emp(y,z,\mathbf{t_a}),\ dom(y), dom(z).$
$Emp(x,y,\mathbf{t}^{\star\star}) \leftarrow Emp(x,y,\mathbf{t_d}), not\ Emp(x,y,\mathbf{f_a}).$
$Emp(x,y,\mathbf{t}^{\star\star}) \leftarrow Emp(x,y,\mathbf{t_a}).$
$\leftarrow Emp(x,y,\mathbf{t_a}), Emp(x,y,\mathbf{f_a}).$

The stable models of the program are:
$\mathcal{M}_1 = \{dom(bill),\ dom(ann),\ dom(paul),\ dom(john),\ Emp(bill,\ ann,\ \mathbf{t_d}),\ Emp(bill,\ ann,\ \mathbf{t}^\star),\ Emp(paul,\ john,\ \mathbf{t_d}),\ Emp(paul,\ john,\ \mathbf{t}^\star),\ Emp(john,\ john,\ \mathbf{t_d}),\ Emp(john,\ john,\ \mathbf{t}^\star),\ \underline{Emp(bill, ann, \mathbf{t}^{\star\star})},\ \underline{Emp(paul, john, \mathbf{t}^{\star\star})},\ \underline{Emp(john, john, \mathbf{t}^{\star\star})},\ Emp(ann,\ null,\ \mathbf{t_a}),\ aux(john),\ \underline{Emp(ann, null, \mathbf{t}^{\star\star})},\ aux(bill),\ aux(paul)\}$.
$\mathcal{M}_2 = \{dom(bill),\ dom(ann),\ dom(paul),\ dom(john),\ Emp(bill,\ ann,\ \mathbf{t_d}),\ Emp(bill,\ ann,\ \mathbf{t}^\star),\ Emp(paul,\ john,\ \mathbf{t_d}),\ Emp(paul,\ john,\ \mathbf{t}^\star),\ Emp(john,\ john,\ \mathbf{t_d}),\ Emp(john,\ john,\ \mathbf{t}^\star),\ Emp(bill,\ ann,\ \mathbf{f_a}),\ \underline{Emp(paul, john, \mathbf{t}^{\star\star})},\ \underline{Emp(john, john, \mathbf{t}^{\star\star})}, aux(john), aux(paul)\}$.
$\mathcal{M}_3 = \{dom(bill),\ dom(ann),\ dom(paul),\ dom(john),\ Emp(bill,\ ann,\ \mathbf{t_d}),\ Emp(bill,\ ann,\ \mathbf{t}^\star),\ Emp(paul,\ john,\ \mathbf{t_d}),\ Emp(paul,\ john,\ \mathbf{t}^\star),\ Emp(john,\ john,\ \mathbf{t_d}),\ Emp(john,\ john,\ \mathbf{t}^\star),\ \underline{Emp(bill, ann, \mathbf{t}^{\star\star})},\ Emp(paul,\ john,\ \mathbf{f_a}),\ Emp(john,\ john,\ \mathbf{f_a}),\ Emp(ann,\ null,\ \mathbf{t_a}),\ \underline{Emp(ann, null, \mathbf{t}^{\star\star})}, aux(bill)\}$.
$\mathcal{M}_4 = \{dom(bill),\ dom(ann),\ dom(paul),\ dom(john),\ Emp(bill,\ ann,\ \mathbf{t_d}),\ Emp(bill,\ ann,\ \mathbf{t}^\star),\ Emp(paul,\ john,\ \mathbf{t_d}),\ Emp(paul,\ john,\ \mathbf{t}^\star),\ Emp(john,\ john,\ \mathbf{t_d}),\ Emp(john,\ john,\ \mathbf{t}^\star),\ Emp(bill,\ ann,\ \mathbf{f_a}),\ Emp(paul,\ john,\ \mathbf{f_a}),\ Emp(john,\ john,\ \mathbf{f_a})\}$.

The databases associated to the first two models correspond to the repairs, but not the last two, which are consistent with the IC, but have unnecessarily deleted $Emp(john, john)$. This happens because this deletion, corresponding to the presence of the atom $Emp(john, john, \mathbf{f_a})$ in the models, stably satisfies the rules
$Emp(john, john, \mathbf{f_a}) \vee Emp(john, null, \mathbf{t_a}) \leftarrow$
$Emp(john, john, \mathbf{t}^\star)\ not\ aux(john),$
$not\ Emp(john, null, \mathbf{t_d}),\ dom(john).$
$aux(john) \leftarrow Emp(john, john, \mathbf{t_d}),$
$not\ Emp(john, john, \mathbf{f_a}),\ dom(john).$
$aux(john) \leftarrow Emp(john, john, \mathbf{t_a}),\ dom(john).$

This happens because there is a cycle that involves $Emp(john, john)$, without this tuple participating in the violation of an IC. These rules are satisfied if $Emp(john, john, \mathbf{f_a})$ belong to the model or not. □

In general, $SM(D) \supseteq Rep(D)$. If (and only if) the RICs are cyclic, the inclusion may be proper. However, all the elements of $SM(D)$ satisfy the ICs.

The repair program computes exactly the repairs for universal and *acyclic* referential ICs; i.e., we have a one-to-one correspondence between the repairs and the databases associated to the models of the repair program, i.e., $SM(D) = Rep(D)$.

[3] Since we want that atoms with *null* values do not generate inconsistencies we would need to add the literal $x \neq null$ to every rule with the predicate $dom(x)$. To avoid this $dom(null)$ is not included in the program.

[4] If $\bar{x} = (x_1, \ldots, x_n)$, we abbreviate $dom(x_1) \wedge \cdots \wedge dom(x_n)$ with $dom(\bar{x})$.

[5] Actually, as illustrated by Example 11, we can always get rid of annotation $\mathbf{f}^\star$.

Example 13 Consider a database instance $\{P(a,b,b), Q(b,c), Q(a,a)\}$, and the cyclic set of RICs: $\{\forall xyz(P(x,y,z) \rightarrow \exists v Q(y,v)), \forall xy(Q(x,y) \rightarrow \exists uw P(u,w,x))\}$. We would expect two ways to restore the consistency of the database, by deleting $Q(b,c)$ or adding the tuple $P(null, null, a)$, but the databases associated to the stable models of the repair program $\Pi(D, IC)$ are $D_{\mathcal{M}_1} = \{P(a,b,b),\ Q(a,a),\ Q(b,c),\ P(null, null, a)\}$, $D_{\mathcal{M}_2} = \{P(a,b,b),\ Q(b,c)\}$, $D_{\mathcal{M}_3} = \{Q(a,a),\ P(null, null, a)\}$ and $D_{\mathcal{M}_4} = \emptyset$. Only the first two are repairs. □

In summary, we have the following:

- D satisfies the ICs for every $D \in SM(D)$.
- $Rep(D) \subseteq SM(D)$.
- For universal and acyclic referential ICs, $Rep(D) = SM(D)$.

4 Consistent Query Answering

The repair program $\Pi(D, IC)$ computes exactly the repairs of the database for universal and acyclic RICs, and a superset when universal and cyclic RICs are considered. We want to use this specification in order to compute the consistent answers from a database with respect to a set of ICs.

We will first concentrate in the case of universal and acyclic RICs. In this case, in order to compute the consistent answers to a query Q, we need to collect the answers that we receive simultaneously from all the stable models of the program $\Pi(D, IC)$. This can be done by first replacing every atom $P(x)$ of the query by $P(x, \mathbf{t}^{\star\star})$. This will force to apply the query over the atoms that belong to every repair. This new query can be transformed into a query program $\Pi(Q)$ by a standard transformation [29, 1]. If this query program is run in combination with $\Pi(D, IC)$, the consistent answers to the query will be obtained.

Example 14 (example 10 continued) Queries Q_1 and Q_2 transformed into query programs are:

$\Pi(Q_1)$: $Ans(x,y) \leftarrow P(x,y,\mathbf{t}^{\star\star})$.
$\Pi(Q_2)$: $Ans \leftarrow\ not\ aux_2$.
$aux_2 \leftarrow P(x,y,\mathbf{t}^{\star\star}),\ not\ aux_1(x)$.
$aux_1(x) \leftarrow R(x,z,\mathbf{t}^{\star\star})$.

If we run $\Pi(D, IC) \cup \Pi(Q_1)$ we get (only the relevant part is shown): $\mathcal{M}_1 = \{\ldots, P(a,d,\mathbf{t}^{\star\star}),\ R(a,d,\mathbf{t}^{\star\star}),\ T(a,\mathbf{t}^{\star\star}),\ R(b,e,\mathbf{t}^{\star\star}),\ Ans(a,d)\}$, and $\mathcal{M}_2 = \{\ldots,\ P(a,d,\mathbf{t}^{\star\star}),\ R(a,d,\mathbf{t}^{\star\star}),\ T(a,\mathbf{t}^{\star\star}),\ T(b,\mathbf{t}^{\star\star}),\ R(b,e,\mathbf{t}^{\star\star}),\ P(b,null,\mathbf{t}^{\star\star}),\ Ans(a,d),\ Ans(b,null)\}$. The only Ans tuple in both repairs is (a,d), and therefore it is the only consistent answer to the query.

If we do the same for query Q_2, we get the following: $\mathcal{M}_1 = \{\ldots, P(a,d,\mathbf{t}^{\star\star}), R(a,d,\mathbf{t}^{\star\star}),\ T(a,\mathbf{t}^{\star\star}),\ R(b,e,\mathbf{t}^{\star\star}), aux_1(a), aux_1(b), Ans\}$, and $\mathcal{M}_2 = \{\ldots, P(a,d,\mathbf{t}^{\star\star}),\ R(a,d,\mathbf{t}^{\star\star}),\ T(a,\mathbf{t}^{\star\star}), T(b,\mathbf{t}^{\star\star}), R(b,e,\mathbf{t}^{\star\star}), P(b,null,\mathbf{t}^{\star\star}),\ aux_1(a),\ aux_1(b),\ Ans\}$. Since Ans is in both repairs the answer to the boolean query is *Yes*. □

Summarizing, in order to compute consistent answers under universal and acyclic RICs, the repair program has to be run with a query program, which will allow us to extract the answers that are true in all the stable models. We have successfully experimented with CQA based on specification of database repairs using the *DLV* system [21].

For cyclic constraints things are more complex. If repairs of RICs are obtained by adding arbitrary elements of the domain, the problem of consistent query answering becomes undecidable [13]. It is possible to prove that under our null values based repairs, the same problem is decidable. We still have the problem of obtaining undesirable models for the programs, those that do not correspond to repairs. We are currently extending the logic program based specification of repairs with local tests for minimality [30]. Using logic programs with priorities [10, 19] is an alternative, due to the fact that, even though all stable models are minimal, the minimality of those that do not correspond to repairs is related to the auxiliary predicates and not to the database predicates (c.f. example 12). In consequence, giving higher priorities to the latter seems to be the right approach. In [15] repairs of RICs by only tuple deletions are investigated. In this case, the problem becomes decidable, but as complex as the evaluation of disjunctive logic programs under the skeptical stable model semantics [20, 18].

5 Optimizations

Sometimes, the repair programs may be transformed into equivalent non-disjunctive programs, i.e., having the same stable models, and then, also specifying the same repairs. This is the case when the disjunctive programs are head-cycle-free [7] (see below). Non-disjunctive logic programs have lower computational complexity than general disjunctive programs [18, 27].

The *dependency graph* of a ground (or fully instantiated) disjunctive program Π is defined as a directed graph where each literal L in the program (i.e., atom or negation of atom) is a node; and there is an arch from L to L' iff there is a rule in which L appears positive in the body and L' appears in the head. Π is *head-cycle-free* (HCF) iff its dependency graph does not contain directed cycles that go through two literals that belong to the head of the same rule.

A disjunctive program Π is HCF if its ground version is HCF. If this is the case, Π can be transformed into a non-disjunctive normal program *sh*(Π) with the same stable models [7]. The non-disjunctive version is obtained by replacing every disjunctive rule of the form: $\bigvee_{i=1}^{n} P_i(x_i) \leftarrow \bigwedge_{j=1}^{m} Q_j(y_j)$ by the n rules $P_i(x_i) \leftarrow \bigwedge_{j=1}^{m} Q_j(y_j) \wedge \bigwedge_{k \neq i}\ not\ P_k(x_k),\ i = 1, ..., n$.

Transformations of this kind can be justified or discarded on the basis of a careful analysis of the intrinsic complexity of consistent query answering [15]. If the original program can be transformed into a non-disjunctive normal program, then also other efficient implementations could be used for query evaluation, e.g., *XSB* [32], that has been applied in interaction with an IBM DB2 DBMS in the context of consistent query answering via first-order query transformation, but only for non-existentially quantified conjunctive queries and limited classes of universal ICs [14].

In [6] it was proved that for the class of universal ICs, the repair programs are HCF, but there no results were reported on referential ICs. Now, we have been able to identify, on the basis of a general test to be applied to a set of ICs containing acyclic RICs, some useful classes of ICs for which the specification program becomes HCF. For example this is the case when *IC* only contains *denial constraints*; i.e., formulas of the form (1) without positive literals (e.g., FDs and range constraints fall in this class); plus *acyclic referential integrity constraints*. In consequence, for *acyclic foreign key constraints* the repair program becomes HCF.

Example 15 (example 11 continued) The program is HCF and therefore it can be transformed into a normal program by shifting one by one the literals in the disjunctive head to the body in negated form. In this case, the program $sh(\Pi(D, IC))$ is obtained from $\Pi(D, IC)$ by replacing rules in 4. and 5. by:

4. $P(x,y,\mathbf{f_a}) \leftarrow P(x,y,\mathbf{t}^{\star}),\ R(x,y,\mathbf{f_a}),\ dom(x),$
$not\ R(x,y,\mathbf{t_a}),\ dom(y)$.
$R(x,y,\mathbf{t_a}) \leftarrow P(x,y,\mathbf{t}^{\star}),\ R(x,y,\mathbf{f_a})\ dom(x),$
$not\ P(x,y,\mathbf{f_a}),\ dom(y)$.
$R(x,y,\mathbf{t_a}) \leftarrow P(x,y,\mathbf{t}^{\star}),\ not\ R(x,y,\mathbf{t_d}),$
$not\ P(x,y,\mathbf{f_a})\ dom(x),\ dom(y)$.
$P(x,y,\mathbf{f_a}) \leftarrow P(x,y,\mathbf{t}^{\star}),\ not\ R(x,y,\mathbf{t_d}),$
$not\ R(x,y,\mathbf{t_a})\ dom(x),\ dom(y)$.
5. $P(x,null,\mathbf{t_a}) \leftarrow T(x,\mathbf{t}^{\star}), not\ aux(x),\ dom(x),$
$not\ T(x,\mathbf{f_a}),\ not\ P(x,null,\mathbf{t_d})$.
$T(x,\mathbf{f_a}) \leftarrow T(x,\mathbf{t}^{\star}), not\ aux(x),\ dom(x),$
$not\ P(x,null,\mathbf{t_a}),\ not\ P(x,null,\mathbf{t_d})$.
$aux(x) \leftarrow P(x,y,\mathbf{t_d}),\ not\ P(x,y,\mathbf{f_a})$.
$aux(x) \leftarrow P(x,y,\mathbf{t_a})$.

The stable models of this program coincide with those of the original program $\Pi(D, IC)$. □

6 Implementation Issues

We are currently working on the implementation of a system for computing consistent query answers on the basis of the repair programs. The details will be presented somewhere else; however we can discuss here a few general issues.

It is clear that any implementation must optimize several processes that participate in consistent query answering. This is because, query answering from disjunctive logic programs has a rather high complexity [18, 20]. However, by using logic programs we are not exceeding the intrinsic complexity of the problem of consistent query answering. In other words, in the general case, the program evaluation and consistent query answering have the same complexity (actually, they are Π_2^P-complete in data complexity) [15, 13]. What is important is to be able to both identify those cases where the complexity of CQA is lower, and optimize

the program in order to match this lower complexity (as the class identified in section 5, for which the complexity of both consistent query answering and the evaluation of the non-disjunctive program can be brought down to the class *NP*). In a similar spirit, determining classes of ICs and queries for which the lower complexity *well-founded semantics* of logic programs [33] can be used is also interesting.

The interaction of a logic programming system and a DBMS is another and important source of complexity. Evaluation of stable models should be avoided whenever possible, trying to obtain as much direct information from the original database as possible. A first step in this direction would be to detect, when a query is to be consistently answered, if the inconsistencies in the data (if any) are relevant to the query at hand. A *consistency check* could determine if the query can be answered directly from the database or the repair program has to be used.

Building a component which decides if the database is consistent or not is simple: The queries

$C(\overline{x}) \leftarrow P_1(\overline{x}_1), \ldots, P_m(\overline{x}_m), not\ Q_1(\overline{y}_1),$
$\cdots,\ not\ Q_n(\overline{y}_n), \varphi,$

for each universal integrity constraint of the form (1), and

$C'(\overline{x})\ \leftarrow P_1(\overline{x}) \wedge\ not\ aux(\overline{x})$
$aux(\overline{x}) \leftarrow Q(\overline{x}', y),$

for each referential integrity constraint of the form (3), will detect the tuples participating in violations of the ICs. Predicates C and C' can be defined, stored and updated as SQL views in the the database, which would reduce the overhead of recomputing them.

If violations that are relevant to the query are detected (actually detecting the relevant ones is another interesting issue), a generator of the repair program, or the programs themselves, should be called. Notice that the repair programs depend on the ICs and not on the query, so, they can be reutilized. Next, a stable model generator, such as *DLV*, has to be used. Finally, the query can be evaluated by running it with the repair program in the logic programming environment.

Since the logic programming environment has to interact with the DBMS, which stores the facts of the program, it is important to avoid unnecessary data extraction. In this sense, it becomes relevant to determine those tables and portions of them which are involved in violation of ICs and relevant to the query. Some interesting ideas in this direction can be found in [22].

It is important to emphasize that we are not interested in the repairs *per se*. In principle, they are used as auxiliary means to characterize the consistent answers. In [2], for restricted classes of queries and ICs, it was possible to compute consistent answers by query the original database alone. However appealing to, complete or partial, computation of repairs, becomes necessary in other cases. In these circumstances, we must minimize their computations or the amount of data involved in the process. The emphasis should be on query answering and not on the computation of the repairs.

It is a problem that the state of art research and implementations of stable model semantics of logic programs are strong at computing (some) stable models, but not at query answering. Full instantiation of the program should be avoided as much as possible; and the system should allow to pose open queries. In this direction, recent research on *magic sets* techniques for disjunctive logic programs under stable model semantics is encouraging [17, 26]. They would allow to reduce the amount of data participating in query processing.

Another direction worth being explored consists in caching previous results of consistent query answering, trying to reuse them when new queries for consistent answers are received; this would avoid running more than desired a stable model generator/evaluator.

7 Conclusions

In this paper we have presented research results that go in the direction of providing mechanisms, to be implemented as as part of the core of a DBMS, that would allow a user to specify, together with a query, a set of integrity constraints -that are not necessarilly maintained by the DBMS- in such a way that the answers to the queries obtained from the system are consistent with the given semantic constraints.

Our approach uses some techniques from the area knowledge representation. At the current state of this line of research, the methodology provably works for any class of first-order ICs that contains universal constraints and acyclic referential constraints.

The current approach considers null-value based repairs under referential integrity constraints. Null values have a special treatment with respect to satisfaction of ICs, and as a consequence, they do not propagate in the repair process. In [3, 5, 13], repairs of RICs using normal domain values are considered. This, under cyclic sets of RICs, may lead to undecidability of consistent query answering. It would be interesting to study some sort of mixed approach, and also the possibility of limited propagation of null values. This is a direction that requires further investigation.

The general complexity of our approach does not exceed the intrinsic complexity of the problem of obtaining consistent query answers; however, as previously discussed, still much exciting research has to be done in terms of optimizing many aspects of the mechanisms, and implementing them in real DBMSs.

Some of the concepts and techniques developed for consistently querying single relational databases, like those presented here, have found applications in the context of virtual data integration. There, global integrity constraints are not maintained, and answers to global queries that are consistent with those constrains are expected to be returned by the mediator [11, 12].

Connections between consistent query answering, virtual data integration and query answering in peer-to-peer data exchange systems are established in [8]. Query answering from a peer has to consider the data exchange constraints and trust relationships with the other peers in the system.

Consistent query answering seem to have natural connections with the area of data exchange, where the main problem is to transfer data from a source database to a target schema that may be different from the schema of the source. In consequence, mappings have to be specified in order to establish the relationship between the data at the source and the data at the target [23], and the process of data transfer has to respect the formulas that express those mappings. However, there are some differences with CQA that deserve further investigation. For example, it is usually the case that for a given source instance, and in contrast with CQA, there are infinite instances at the target that are "solutions" to the problem. The typical syntactic form of the exchange constraints used to express the mappings causes that any superset of a solution is also a solution, whereas database repairs are always minimal.

On the other side, in data exchange some techniques have have been developed to show that some queries over the target schema are not rewritable as a queries that, over a materialized target instance, give a result that is semantically equivalent with the source [4]. It is possible that some of those techniques could be used to show that the consistent answers to some queries cannot be expressed as a first-order views over the underlying instance.

Acknowledgements: Research supported by NSERC Grant 250279-02, and IBM CAS (Toronto Lab.) and CITO awards. We are grateful to Mauricio Vines for his support with the implementation efforts of consistent query answering, and to Marcelo Arenas and Pablo Barcelo for some useful remarks. We also appreciate technical and stylistic comments by anonymous referees.

About the Authors: Loreto Bravo is a Bachelor in Sciences of Engineering and a Civil Transportation Engineer from the Catholic University of Chile (PUC). She is a graduate student and PhD candidate in Computer Science at Carleton University. She participates in a CITO/IBM-CAS (Toronto) Student Internship Program. Leopoldo Bertossi is a full professor of computer science at Carleton University. Before he was a professor at the Catholic University of Chile (PUC). He has held visiting academic positions at the universities of Toronto, Wisconsin-Milwaukee and Marseille (Luminy), and Technical of Berlin. He is a Faculty Fellow of the IBM Center for Advanced Studies, Toronto Lab.; and a member of the NSERC Computing and Information Sciences Grant Selection Committee (GSC 330), 2002 -2005.

References

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

[2] Arenas, M., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent

Databases. In *Proc. Symposium on Principles of Database Systems (PODS 99)*, ACM Press, 1999, pp. 68–79.

[3] Arenas, M., Bertossi, L. and Chomicki, J. Answer Sets for Consistent Query Answers. Theory and Practice of Logic Programming, 2003, 3(4-5), pp. 393-424.

[4] Arenas, M., Barcelo, P., Fagin, R. and Libkin, L. Locally Consistent Transformations and Query Answering in Data Exchange. In *Proc. Symposium on Principles of Database Systems (PODS 04)*, ACM Press, 2004, pp. 229-240.

[5] Barcelo, P. and Bertossi, L. Logic Programs for Querying Inconsistent Databases. In *Proc. Fifth International Symposium on Practical Aspects of Declarative Languages (PADL 03)*. Springer LNCS 2562, 2003, pp. 208–222.

[6] 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.

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

[8] Bertossi, L. and Bravo, L. Query Answering in Peer-to-Peer Data Exchange Systems. In *Proc. International Workshop on Peer-to-Peer Computing & DataBases (P2P&DB 04)*. To appear in Springer LNCS. Also in CORR repository under cs.DB/0401015.

[9] Bertossi, L. and Chomicki, J. Query Answering in Inconsistent Databases. In ‘Logics for Emerging Applications of Databases’, J. Chomicki, G. Saake and R. van der Meyden (eds.), Springer, 2003.

[10] Brewka, G. and Eiter, T. Preferred Answer Sets for Extended Logic Programs. *Artificial Intelligence*, 1999, 109(1-2):297-356.

[11] 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.

[12] 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*.

[13] Cali, A., Lembo, D. and Rosati, R. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *Proc. Symposium on Principles of Database Systems (PODS 03)*, ACM Press, 2003, pp. 260-271.

[14] Celle, A. and Bertossi, L. Querying Inconsistent Databases: Algorithms and Implementation. In ‘Computational Logic - CL 2000’, J. Lloyd et al. (eds.). Stream: 6th International Conference on Rules and Objects in Databases (DOOD 00). Springer LNAI 1861, 2000, pp. 942–956.

[15] Chomicki, J. and Marcinkowski, J. Minimal-Change Integrity Maintenance Using Tuple Deletions. arXiv.org paper cs.DB/0212004. To appear in *Information and Computation*.

[16] Chomicki, J., Marcinkowski, J. and Staworko, S. Hippo: A System for Computing Consistent Answers to a Class of SQL Queries. In *Advances in Database Technology - EDBT 2004*, Springer LNCS 2992, 2004, pp. 841-844.

[17] Cumbo, C., Faber, W., Greco, G. and Leone, N. Enhancing the Magic-Set Method for Disjunctive Datalog Programs. In *Proceedings of the 20th International Conference on Logic Programming (ICLP 04)*, 2004. To appear.

[18] Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 2001, 33(3): 374–425.

[19] Delgrande, J., Schaub, T. and Tompits, H. A Framework for Compiling Preferences in Logic Programs. *Theory and Practice of Logic Programming*, 2003, 3(2):129-187.

[20] Eiter, T., Gottlob, G. Complexity Aspects of Various Semantics for Disjunctive Databases. In *Proc. Symposium on Principles of Database Systems (PODS 93)*, ACM Press, 1993, pp. 158-167.

[21] 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.

[22] Eiter, T., Fink, M., Greco, G. and Lembo, D. Efficient Efficient Evaluation of Logic Programs for Querying Data Integration Systems. In *Proc. 19th International Conference on Logic Programming (ICLP 03)*, Springer LNCS 2916, 2003, pp. 163-177.

[23] Fagin, R., Kolaitis, P., Miller, R.J. and Popa, L. Data Exchange: Semantics and Query Answering. In *Proc. International Conference on Database Theory (ICDT 03)*, Springer LNCS 2572, 2003, pp. 207–224..

[24] Gelfond, M. and Lifschitz, V. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 1991, 9:365–385.

[25] Greco, G., Greco, S. and Zumpano, E. A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. In *Proc. International Conference on Logic Programming (ICLP 01)*, Springer LNCS 2237, 2001, pp. 348–364.

[26] Greco, G., Greco, S., Trubtsyna, I. and Zumpano, E. Optimization of Bound Disjunctive Queries with Constraints. arXiv.org paper cs.LO/0406013. To appear in *Theory and Practice of Logic Programming*.

[27] Leone, N., Rullo, P. and Scarcello, F. Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics, and Computation. *Information and Computation*, 1997, 135(2):69-112.

[28] 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*.

[29] Lloyd, J.W. *Foundations of Logic Programming*. Second ed., Springer-Verlag, 1987.

[30] Niemela, I. Implementing Circumscription Using a Tableau Method. In *Proc. European Conference on Artificial Intelligence (ECAI 96)*, 1996, pp. 80–84.

[31] Reiter, R. On Closed World Databases. In *Logic and Databases*, H. Gallaire and J. Minker (eds.), Plenum Press, 1978, pp. 55-76.

[32] Sagonas, K.F., Swift, T. and Warren, D.S. XSB as an Efficient Deductive Database Engine. In *Proc. International Conference on Management of Data (SIGMOD 94)*, ACM Press, 1994, pp. 442-453.

[33] Van Gelder, A., Ross, K.A., Schlipf, J.S. Unfounded Sets and Well-Founded Semantics for General Logic Programs. In *Proc. Symposium on Principles of Database Systems (PODS 88)*, ACM Press, 1988, pp. 221-230.

Scalar Aggregation in Inconsistent Databases*

Marcelo Arenas
Dept. of Computer Science
University of Toronto
marenas@cs.toronto.edu

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

Jan Chomicki†
Dept. CSE
University at Buffalo
chomicki@cse.buffalo.edu

Xin He
Dept. CSE
University at Buffalo
xinhe@cse.buffalo.edu

Vijay Raghavan
EECS Dept.
Vanderbilt University
raghavan@vuse.vanderbilt.edu

Jeremy Spinrad
EECS Dept.
Vanderbilt University
spin@vuse.vanderbilt.edu

Abstract

We consider here scalar aggregation queries in databases that may violate a given set of functional dependencies. We define consistent answers to such queries to be greatest lowest/least upper bounds on the value of the scalar function across all (minimal) repairs of the database. We show how to compute such answers. We provide a complete characterization of the computational complexity of this problem. We also show how tractability can be improved in several special cases (one involves a novel application of Boyce-Codd Normal Form) and present a practical hybrid query evaluation method.

1 Introduction

In this paper, we address the issue of obtaining *consistent* information from *inconsistent* databases – databases that violate given integrity constraints. Our basic assumption departs from everyday practice of database management systems. Typically, a database management system checks the satisfaction of integrity constraints and backs out those updates that violate them. Therefore, databases seemingly never become inconsistent. However, we list below several practical scenarios in which inconsistent databases do occur.

Integration of autonomous data sources. The sources may separately satisfy the constraints but, when the sources are integrated together, the constraints may stop to hold. For instance, consider different, conflicting addresses for the same person in the taxpayer and the voter registration databases. Each of those databases separately satisfies the functional dependency that associates a single address with each person, yet together they violate this dependency. Morever, since the sources are autonomous they can not be simply fixed to satisfy the dependency by removing all but one of the conflicting tuples.

Unenforced integrity constraints. Even though integrity constraints capture an important part of the semantics of a given application, they may still fail to be enforced for a variety of reasons. A data source may be a legacy system that does not support the notion of integrity checking altogether. Integrity checking may be too costly (this is often the reason for dropping some integrity constraints from the database schema). Finally, the DBMS itself may support only a limited class of constraints. For example, SQL2 DBMS typically support only *key* functional dependencies, not arbitrary ones. Therefore, if the relations in a data warehouse are *denormalized* for efficiency reasons, some functional dependencies may become unenforceable.

Temporary inconsistencies. It may often be the case that the database consistency is only temporarily violated and further updates or transactions are expected to restore it. This phenomenon is becoming more and more common, as databases are increasingly involved in a variety of long-running activities or *workflows*.

Conflict resolution. Removing tuples from a database to restore consistency leads to information loss, which may be undesirable. For example, one may want to keep multiple addresses for a person if it is not clear which is the correct one. In general, the process of conflict resolution may be complex, costly, and non-deterministic. In real-time decision-making applications, there may not be enough time to resolve all conflicts relevant to a query.

To formalize the notion of consistent information obtained from a (possibly inconsistent) database in response to a user query, we proposed in [3] the notion of a *consistent query answer*. A consistent answer is, intuitively, true regardless of the way the database is fixed to remove constraint violations. Thus answer consistency serves as an indication of its reliability. The different ways of fixing an inconsistent database are formalized using the notion of *repair*: another database that is consistent and minimally differs from the original database.

For instance, in the case of multiple addresses of a single person, one can still consistently determine the addresses of those people who have only a single address in the integrated database. Or, more interestingly, if all tuples for the same person have the same birthdate, then the birthdate can be returned as a consistent answer, although there may be multiple conflicting addresses. Also, the different addresses may have a common part, e.g., the state name, that can be consistently returned and will suffice for some queries, e.g., those concerned with taxation. These examples show that simply discarding conflicting data will lead to information loss.

In [3], in addition to a formal definition of a consistent query answer, a computational mechanism for obtaining such answers was presented. However, the queries considered were just *first-order queries*. Here we address in the same context the issue of *aggregation queries*. Aggregation queries are important in OLAP and data warehousing – precisely the context in which inconsistent databases may occur (see above). We limit, however, ourselves to single relations that possibly violate a given set of functional dependencies (FDs).

In defining consistent answers to aggregation queries we distinguish between queries with *scalar* and *aggregation* functions. The former return a single value for the entire relation. The latter perform grouping on an attribute (or a set of attributes) and return a single value for each group. Both kinds of queries use the same standard set of SQL-2 aggregate operators: `MIN`, `MAX`, `COUNT`, `SUM`, and `AVG`. In this paper, we address only *aggregation queries with scalar functions*.

Example 1 *Consider the following example. Suppose the results of an election in which two candidates, Brown and Green are running, are kept in two relations:* BrownVotes *and* GreenVotes.

BrownVotes

County	*Date*	*Tally*
A	*11/07*	*541*
A	*11/11*	*560*
B	*11/07*	*302*

GreenVotes

County	*Date*	*Tally*
A	*11/07*	*653*
A	*11/11*	*730*
B	*11/07*	*101*

Vote tallies in every county should be unique. Consequently, the functional dependency County → Tally should hold in both relations. On the other hand, we may want to keep multiple tallies corresponding to different counts (and recounts). Clearly, both relations will have two repairs each, depending on whether the first or the second count for county A is picked. Altogether, the original database has thus four repairs.

The total tally for Brown is 843 in one repair and 862 in the other. For Green, the corresponding figures are 754 and 831. It is clear that there is no single consistent answer to the aggregation query:

```
SELECT SUM(Tally)
FROM BrownVotes
```

and the same holds for the similar query involving the relation GreenVotes. *Therefore, the notion of consistent query answer from [3] needs to be adapted in the context of aggregation queries. For such queries, we propose to return* ranges *of values: [843,862] for Brown and [754,831] for Green. Note that in this case we can safely say that Brown won the election, since the minimum vote for Brown is greater than the maximum vote for Green.* □

The plan of the paper is as follows. In Section 2, we provide a general definition of consistent answer to an aggregation query with a scalar function. We also define a graph-theoretical representation of database repairs, which is specifically geared towards FDs. In Section 3, we study data complexity of the problem of computing consistent answers to aggregation queries in inconsistent databases. In Section 4, we show how to reduce in practice the computational cost of computing such answers by decomposing the computation into two parts: one that involves standard relational query evaluation and one that computes the consistent answers in a smaller instance. In Section 5, we show that the complexity of computing consistent answers can be reduced by exploiting special properties of the given set of FDs or the given instances. In Section 6 we discuss related and further work.

2 Basic Notions

In this paper we assume that we have a fixed database schema containing only one relation schema R with the set of attributes U. We will denote elements of U by $A, B, \ldots$, subsets of U by $X, Y, \ldots$, and the union of X and Y by XY. We also have two fixed, disjoint infinite database domains: D (uninterpreted constants) and N (rational numbers). We assume that elements of the domains with different names are different. The database instances can be seen as finite first-order structures that share the domains D and N. Every attribute in U is typed, thus all the instances of R can contain only elements either of D or of N in a single attribute. Since each instance is finite, it has a finite active domain which is a subset of $D \cup N$. As usual, we allow built-in predicates $(=, \neq, <, >, \leq, \geq)$ over N that have infinite, fixed extensions. There is also a set of integrity constraints F over R that captures the semantics of the database. E.g., it may express the property that an employee has only a single salary. The instances of the database do not have to satisfy F. A database that satisfies a given set of integrity constraints F, denoted by $r \models F$, is called *consistent*, otherwise *inconsistent*. In this paper we consider only integrity contraints that are *functional dependencies* (FDs).

2.1 Repairs

The following definitions are adapted from [3].

Definition 1 *For the instances* r, r', r'', $r' \leq_r r''$ *if* $r - r' \subseteq r - r''$. □

Definition 2 *Given a set of integrity constraints F and database instances r and r', we say that r' is a* repair *of r w.r.t. F if $r' \models F$ and r' is $\leq_r$-minimal in the class of database instances that satisfy F.* □

We denote by $Repairs_F(r)$ the set of repairs of r w.r.t. F. Examples 1 (earlier) and 2 (below) illustrate the notion of repair.

Because we consider only functional dependencies here and for such constraints all the repairs of an instance are obtained by deleting tuples from it, the notion of repair from [3] can be simplified here. A repair is simply a maximal consistent subset of an instance. Clearly, there are only finitely many repairs, since the relations are finite. Also, in this case the union of all repairs of any instance r is equal to r. These properties are not necessarily shared by other classes of integrity constraints.

Definition 3 *The* core *of r is defined as*

$$Core_F(r) = \bigcap_{r' \in Repairs_F(r)} r'.$$

□

The *core* is a new database instance. If r consists of a single relation, then the core is the intersection of all the repairs of r. The core of r itself is not necessarily a repair of r.

Example 2 *In Example 1, the relation* BrownVotes *has two repairs*

r_1

County	*Date*	*Tally*
A	*11/07*	*541*
B	*11/07*	*302*

r_2

County	*Date*	*Tally*
A	*11/11*	*560*
B	*11/07*	*302*

The core of the relation BrownVotes *consists of the single tuple*

County	*Date*	*Tally*
B	*11/07*	*302*

and is not a repair. It satisfies the functional dependency County → Tally but is not a maximal consistent subset of the original instance. □

[*] An earlier version of this paper appeared in [5].

[†] Corresponding author. Address: Department of Computer Science and Engineering, 201 Bell Hall, University at Buffalo, Buffalo, NY 14260-2000, USA. Phone: (716)645-3180, ext. 103. Fax: (716) 645-3464.

2.2 Consistent Query Answers

2.2.1 First Order Queries

Query answers for first order queries are defined in the standard way.

Definition 4 *A ground tuple $\bar{t}$ is an* answer *to a query $Q(\bar{x})$ in a database instance r if $r \models Q(\bar{t})$, i.e., the query $Q(\bar{x})$ is true of $\bar{t}$ in the instance r.* □

Consistent query answers were first defined in [3]. We present here a slightly modified but equivalent definition.

Definition 5 *A ground tuple $\bar{t}$ is a* consistent answer *to a query $Q(\bar{x})$ with respect to a set of integrity constraints F in a database instance r if for every $r' \in Repairs_F(r)$, $r' \models Q(\bar{t})$. We denote the set of consistent answers to Q w.r.t. F in r by $Cqa^Q_F(r)$.* □

Example 3 *The query*

```
SELECT * FROM BrownVotes
```

has the following consistent answer in the instance of Example 1:

Brown	*B*	*11/07*	*302*

In the same instance the query

```
SELECT County FROM BrownVotes WHERE Tally > 400
```

has A as the only consistent answer. Notice that this answer cannot be obtained by evaluating the query in the original instance from which the conflicting tuples have been removed. □

2.2.2 Aggregation Queries

The aggregation queries we consider are queries of the form

SELECT f FROM R

where f is one of: `MIN(A)`, `MAX(A)`, `COUNT(A)`, `SUM(A)`, `AVG(A)`, or `COUNT(*)`, where `A` is an attribute of the schema R. These queries return single numerical values by applying the corresponding *scalar function*, i.e., for `MIN(A)` the minimum `A`-value in the given instance, etc. In general, f will also denote an aggregation query (or a scalar function itself). Thus, $f(r)$ will denote the result of applying f to the given instance r of R.

In contrast with first-order queries, there is no single intuitive notion of consistent query answer for aggregation queries. It is likely (see Example 5 below) that aggregation queries return different answers in different repairs, and thus there will be no single consistent answer in the sense of Definition 5. In order to obtain more informative answers even in such a case, we explore therefore several alternative definitions of consistent query answers.

Definition 6 *Given a set of integrity constraints F, an aggregation query f and a database instance r, the* set of possible answers *$Poss^f_F(r)$ is defined as*

$$Poss^f_F(r) = \{f(r') \mid r' \in Repairs_F(r)\}.$$

The greatest-lower-bound (glb) answer *$glb^f_F(r)$ to f w.r.t. F in r is defined as*

$$glb^f_F(r) = glb\ Poss^f_F(r).$$

The least-upper-bound (lub) answer *$lub^f_F(r)$ to f w.r.t. F in r is defined as*

$$lub^f_F(r) = lub\ Poss^f_F(r).$$

□

Example 4 *In the instance of Example 1 and the query*

```
SELECT SUM(Tally) FROM BrownVotes
```

the set of possible answers is $\{843, 862\}$, the glb-answer is 843 and the lub-answer is 862. □

Based on Definition 6, one can envision several possible notions of consistent query answer for aggregation queries:

1. the set of possible answers $Poss^f_F(r)$,
2. the range of possible answers $[glb^f_F(r), lub^f_F(r)]$,
3. some aggregate, for example average, of all possible answers, or
4. some representation of the distribution of all possible answers.

We conjecture that each of those notions makes sense in the context of some application. In this paper, we study the second notion, that of the range of all possible answers $[glb^f_F(r), lub^f_F(r)]$, for the reasons outlined below.

Example 5 *Consider the functional dependency $A \rightarrow B$ and the following family of relation instances r_n, $n > 0$:*

A	B
1	0
1	1
2	0
2	2
…	
i	0
i	2^{i-1}
…	
n	0
n	2^{n-1}

We use this example to illustrate two points. First, the instance r_n has 2^n different repairs. Therefore, the approach to computing consistent query answers to any aggregation query (or any other query for that matter) by evaluating the query in every repair separately and then collecting the results is infeasible. Second, note that the aggregation query `SUM(B)` admits a different result in every repair. Actually, every integer in the answer range $[0, 2^n - 1]$ is the result of the query `SUM(B)` in some repair. In spite of that, glb- and lub-answers have polynomial size (since the bounds can be represented in binary). This is not be the case if we represent all the possible values as a set, a distribution, or some form of disjunctive information e.g., an OR-object [22] or a C-table [21]. (An OR-object is a special domain value specified as a set of atomic values and interpreted as *one* of those values. A C-table is a table with null values that have to satisfy conditions associated with individual rows or the entire table. For a discussion of the relationship between tables with OR-objects and sets of all repairs, see Section 6). □

It is easy to see that glb- and lub-answers in our framework are always polynomially-sized and thus exponentially more succinct than set-, distribution-, or disjunction-based representations. However, representing a set of values as a range may lead to information loss. For instance, while we guarantee that the value of the scalar function in every repair falls within the returned range, clearly not every value in this range will necessarily correspond to the value of the function obtained in some repair.

Further aggregating the values of an aggregation query over all repairs, e.g., taking the average, leads to further information loss. In fact, presented with such an answer the user can no longer say anything about the values the query has in the individual repairs.

We should note that regardless of whether a range- or a set-based representation is used, the obtained result is semantically not a standard relation, so it cannot directly serve as input to other SQL queries. In the first case, the obtained range $[a, b]$ can be represented as a pair but in fact should be interpreted as a *condition* $a \leq v \leq b$ on the repair-dependent value v of the scalar function. In the second case, the result is a set and thus requires going beyond First Normal Form. Moreover, the set needs to be interpreted as a condition too, in this case disjunctive. (The condition is $x = v_1 \vee \cdots \vee x = v_k$ where $\{v_1, \ldots, v_k\}$ is the set of possible values of the scalar function.)

We will also consider other auxiliary notions of query answer in inconsistent databases. *Core* answers are used for hybrid evaluation in Section 4 and *union* answers are defined for symmetry with core answers.

Definition 7 *A number v is a* core answer *to f w.r.t. F in r if*

$$v = f(Core_F(r)) = f(\bigcap_{r' \in Repairs_F(r)} r').$$

A number v is a union answer *to f w.r.t. F in r if*

$$v = f(\bigcup_{r' \in Repairs_F(r)} r').$$

□

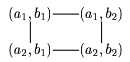
However, union answers are trivial for FDs, as the union of all the repairs of r is r itself, so the union answer reduces to $f(r)$.

2.3 Graph Representation

Given a set of FDs F and an instance r, all the repairs of r w.r.t. F can be succinctly represented as a graph.

Definition 8 *The* conflict graph *$G_{F,r}$ is an undirected graph whose set of vertices is the set of tuples in r and whose set of edges consists of all the edges (t_1, t_2) such that $t_1 \in r$, $t_2 \in r$, and there is a dependency $X \rightarrow Y \in F$ for which $t_1[X] = t_2[X]$ and $t_1[Y] \neq t_2[Y]$.* □

Example 6 *Consider a schema $R(AB)$, the set F of two functional dependencies $A \rightarrow B$ and $B \rightarrow A$, and an instance $r = \{(a_1, b_1), (a_1, b_2), (a_2, b_2), (a_2, b_1)\}$ over this schema. The conflict graph $G_{F,r}$ looks as follows:*



□

Definition 9 *An* independent set *S in an (undirected) graph $G = (V, E)$ is a subset of the set of vertices V of this graph, such that there is no edge in the set of edges E connecting two vertices in S. A* maximal *independent set is an independent set which is not a proper subset of any other independent set. A* maximum *independent set is an independent set of maximum cardinality.* □

Proposition 1 *Each repair in $Repairs_F(r)$ corresponds to a maximal independent set in $G_{F,r}$ and vice versa.* □

Conflict graphs are geared specifically towards FDs. The repairs of other classes of constraints do not necessarily have similar representations.

We also note that, for a given set of FDs F over R, one can write an SQL2 query that for any instance r of R computes the edges of the conflict graph $G_{F,r}$.

2.4 Computational Complexity

2.4.1 Data Complexity

The data complexity notion [8, 31] makes it possible to study the complexity of query processing as a function of the number of tuples in the database instance. We define separately the data complexity of checking repairs, the data complexity of computing consistent query answers to first-order queries, and that of computing consistent query answers to aggregation queries.

Definition 10 *Given a class of databases $\mathcal{D}$ and a class of integrity constraints, the* data complexity *of checking repairs is defined to be the complexity of determining the membership of the sets*

$$D_F = \{(r, r') \mid r \in \mathcal{D} \wedge r' \in Repairs_F(r)\}$$

*for a fixed finite set F of integrity constraints. This problem is C-*data-hard *for a complexity class C if there is a finite set of integrity constraints F_0 such that D_{F_0} is C-hard.* □

Lemma 1 *For a given set F of FDs, the data complexity of checking whether an instance r' is a repair of r is in PTIME.*

Proof: Checking whether r' satisfies F is in PTIME. The repair r' has also to be $\leq_r$-minimal among those instances that satisfy F. For FDs, it means that r' has to be a maximal subset of r that satisfies F. Checking this property can be done as follows: try all the tuples $\bar{t}$ in $r - r'$, one by one. If $r' \cup \{\bar{t}\}$ satisfies F, then r' is not maximal. Otherwise, if for no such tuple $\bar{t}$, $r' \cup \{\bar{t}\}$ satisfies F, no superset of r' can satisfy F (violations of FDs cannot be removed by adding tuples) and r' is maximal. □

Definition 11 *Given a class of databases $\mathcal{D}$, a class of first-order queries $\mathcal{L}$ and a class of integrity constraints, the* data complexity *of computing consistent query answers is defined to be the complexity of determining the membership of the sets*

$$D_{F,\phi} = \{(r,\bar{t}) \mid r \in \mathcal{D} \wedge \bar{t} \in Cqa_F^{\phi}(r)\}$$

*for a fixed $\phi \in \mathcal{L}$ and a fixed finite set F of integrity constraints. This problem is C-*data-hard *for a complexity class C if there is a query $\phi_0 \in \mathcal{L}$ and a finite set of integrity constraints F_0 such that D_{F_0,ϕ_0} is C-hard.* □

¿From Lemma 1, we can immediately obtain:

Corollary 1 *For any set of FDs F and first-order query Q, the data complexity of checking whether a tuple $\bar{t}$ is a consistent answer to Q is in co-NP.* □

In section 3, we will see that the above problem is in fact co-NP-hard (Corollary 2).

Definition 12 *Given a class of databases $\mathcal{D}$, a class of aggregation queries $\mathcal{F}$ and a class of integrity constraints, the* data complexity *of computing the glb-answer (resp. lub-answer) is defined to be the complexity of determining the membership of the sets*

$$D_{F,f} = \{(r,k) \mid r \in \mathcal{D} \wedge glb_F^f(r) \leq k\}$$

and

$$D_{F,f} = \{(r,k) \mid r \in \mathcal{D} \wedge lub_F^f(r) \geq k\},$$

*respectively, for a fixed aggregation query $f \in \mathcal{F}$ and a fixed finite set F of integrity constraints. This problem is C-*data-hard *for a complexity class C if D_{F_0,f_0} is C-hard for some aggregation query $f_0 \in \mathcal{F}$ and a finite set of integrity constraints F_0.* □

In our case, each class of aggregation queries $\mathcal{F}$ contains only queries that use scalar functions of the same kind, e.g., `MIN(A)` for some attribute `A` of R.

Proposition 2 *For every class of aggregation queries F that contains only queries with scalar functions of the same kind, computing the glb- and the lub-answer is in NP.*

Proof: Consider computing the glb-answer (the other case is symmetric). We have that $glb_F^f(r) \leq k$ if and only if there is a repair $r' \in Repairs_F(r)$ such that $f(r') \leq k$. The latter condition can be clearly checked in NP, in the view of Lemma 1. □

Our PTIME results will yield algorithms that compute the glb-answer $glb_F^f(r)$ (or $lub_F^f(r)$), which is clearly sufficient to determine the truth of the condition $glb_F^f(r) \leq k$ (resp. $lub_F^f(r) \geq k$).

3 Complexity of Scalar Aggregation

We have seen (Example 5) that there may be exponentially many repairs even in the case of one functional dependency. Therefore, it is computationally infeasible to evaluate a scalar aggregation query in every repair. In [3] and this paper, we have identified two ways of computing consistent answers by querying the given, possibly inconsistent database instance, without having to compute all the repairs. Query transformation modifies the original query, Q, into a new query, $T(Q)$, that returns only consistent answers. We have applied this approach in [3] to restricted first order queries and universal integrity constraints. Except in some simple cases, this approach does not seem applicable to aggregation queries. For example, even when `MAX(A)` and `MIN(A)` queries can be written as first order queries, their resulting syntax does not allow the application of the methodology developed in [3] to them. Moreover, as argued earlier in the paper, aggregation queries seem to require a different notion of consistent query answer than first-order queries. Therefore, we use instead the fact that for FDs, the set of all repairs of an instance can be compactly represented as the conflict graph. We develop techniques and algorithms geared specifically towards this representation.

We start by considering core answers – an easy case. Then we consider several aggregate operators – `MIN`, `MAX`, `SUM` and `COUNT(*)` – together. They share common properties: for each of them computing glb- and lub-answers is tractable only in the case of a single functional dependency and the proof of tractability uses the same technique of building an appropriate single repair. Subsequently, we consider the `AVG` operator which requires a much more involved tractability proof. Finally, we study `COUNT(A)`, for which even the single-dependency case is not tractable.

In the following r denotes an instance of the schema R. The input of the problem of computing consistent query answers will consist of r and a numerical parameter k (as required by Definition 12).

3.1 Core Answers

For some aggregate operators, e.g., `COUNT` and `SUM` of nonnegative values, a core answer is a lower-bound-answer, but not necessarily the glb-answer. As we will see in Section 4, computing core answers to aggregation queries can be useful for computing consistent answers.

Theorem 1 *The data complexity of computing core answers for any scalar function is in PTIME.*

Proof: The core consists of all the isolated vertices in the conflict graph. □

We note that, for a given set of FDs F over R, one can write an SQL2 query that computes for any instance r of R the set of isolated vertices in the conflict graph $G_{F,r}$.

In general, computing glb-answers and lub-answers is considerably more involved than computing core answers.

3.2 Aggregation using `MIN`, `MAX`, `SUM`, and `COUNT(*)`

3.2.1 One functional dependency

Consider `MAX(A)` (`MIN(A)` is symmetric). In this case computing the lub-answer in r w.r.t. an arbitrary set of FDs F consists of evaluating `MAX(A)` in r, thus it is clearly in PTIME.

However, it is not obvious how to compute the glb-answer, namely the minimum of the set of maximums obtained by posing the query `MAX(A)` in every repair. Computing `MAX(A)` in $Core_F(r)$ gives us only a lower-bound-answer which does not have to be the glb-answer.

Theorem 2 *The data complexity of computing $glb_F^f(r)$ in r for a set of FDs F consisting of a single FD $X \rightarrow Y$ and $f \in \{$`MAX(A)`, `SUM(A)`, `COUNT(*)`$\}$ is in PTIME.*

Proof: The approach for all of the above scalar functions is essentially identical and consists of constructing a repair that minimizes the value of the scalar function. Call an (X,Y)-*cluster* a maximal set of tuples of r that have the same attribute values in X and Y. Clearly, in a single repair we can have only one (X,Y)-cluster for every given value of X. For every value of the attribute X we pick that (X,Y)-cluster that minimizes the scalar function and apply the scalar function to this cluster. Finally, we aggregate the obtained values across all values of X (and combine the (X,Y)-clusters if we want to obtain a repair minimizing f). This approach gives the minimum of the scalar function over all repairs. For `MAX(A)` it can be defined in SQL2 as the following sequence of views:

```
CREATE VIEW S(X,Y,C) AS
  SELECT X,Y,MAX(A) FROM R
  GROUP BY X,Y;

CREATE VIEW T(X,C) AS
  SELECT X, MIN(C) FROM S
  GROUP BY X;

SELECT MAX(C) FROM T;
```

For `SUM(A)`, we only have to replace `MAX` in the above by `SUM`. For `COUNT(*)`, we replace `MAX(A)` by `COUNT(*)` and `MAX(C)` by `SUM(C)`. Evaluating all those SQL2 queries can be done in PTIME. □

It is clear that there is a symmetric result to Theorem 2 for lub-answers to `MIN(A)`. Note that

$$lub_F^{\mathtt{MIN(A)}}(r) = -glb_F^{\mathtt{MAX(A)}}(r^-)$$

where r^- contains identical tuples to r except that their A-values are inverted (every A-value v is changed to $-v$).

We show now that Theorem 2 exhausts the tractable cases for the scalar functions in question.

3.2.2 Two functional dependencies and `MAX(A)`

Theorem 3 *There is a set of 2 FDs F_0 for which deciding whether*

$$glb_{F_0}^{\mathtt{MAX(A)}}(r) \leq k$$

in r is NP-data-hard.

Proof: Reduction from 3SAT. Consider a propositional formula $\varphi = C_1 \wedge \cdots \wedge C_n$ in CNF. Let $p_1, \ldots p_m$ be the propositional variables in φ. Construct a relation r with the attributes A, B, C, D, and containing exactly the following tuples:

1. $(p_i, 1, C_j, 1)$ if making p_i true makes C_j true,
2. $(p_i, 0, C_j, 1)$ if making p_i false makes C_j true,
3. $(w, 2, C_j, 2)$, $1 \leq j \leq n$, where w is a new symbol.

The set of FDs F_0 consists of $A \rightarrow B$ (each propositional variable cannot have more than one truth value) and $C \rightarrow D$. Also, $k = 1$. We show that $glb_{F_0}^{\mathtt{MAX(D)}}(r) = 1$ iff φ is satisfiable.

Assume $glb_{F_0}^{\mathtt{MAX(D)}}(r) = 1$. Then there is a repair r_0 of r in which the attribute D assumes only the value 1. If for some j the repair r_0 does not contain any tuple of the form $(\_, \_, C_j, 1)$, then r_0 has to contain the tuple $(w, 2, C_j, 2)$ and `MAX(D)` returns 2 in this repair, a contradiction. ¿From r_0 we can build a satisfying assignment for φ by reading off the values of the attributes A and B for each conjunct C_j. Notice that r_0 has to satisfy the FD $A \rightarrow B$ and thus each propositional variable receives in this way only a single value.

Assume now that φ is satisfiable. Then, given a satisfying assignment, we build a database instance r_1 in the following way: For every propositional variable p_i made true by the assignment and every conjunct C_j in which this variable occurs positively, we include the tuple $(p_i, 1, C_j, 1)$ in r_1. The variables made false by the assignment are treated symmetrically. Clearly, r_1 satisfies $A \rightarrow B$. Since the assignment satisfies φ, for every conjunct C_j there is a tuple in r_1 which has C_j as the value of the attribute C. Therefore, r_1 cannot contain any tuples of the third kind, and has to satisfy $C \rightarrow D$ as well. It is also maximal, and thus a repair. Since in every repair of r, `MAX(D)` returns a value greater or equal to 1, and `MAX(D)` returns 1 in r_1, then $glb_{F_0}^{\mathtt{MAX(D)}}(r) = 1$. □

The above reduction yields also a lower bound for checking consistent query answers for first-order queries.

Corollary 2 *There is a set of 2 FDs F_0 and a first-order query Q for which the problem of checking whether $\bar{t}$ is a consistent answer to Q is co-NP-data-hard.*

Proof: We use the same reduction and the same set of FDs F_0 as in Theorem 3. We note that the formula φ is unsatisfiable iff 2 is a consistent answer to the query

$$\exists x, y, z.\ R(x, y, z, w).$$

□

Corollary 2 should be contrasted with the results of [3], which imply that in the presence of FDs the data complexity of computing consistent query anwers for first-order queries consisting only of quantifier-free conjunctions of positive and negative literals is in PTIME. Thus Corollary 2 identifies the existential quantifier as a source of intractability.

3.2.3 Two functional dependencies and `COUNT(*)`

We consider now `COUNT(*)`.

Lemma 2 *There is a set of 2 FDs F_1 for which the problem of determining the existence of a repair of r of size $\geq k$ is NP-data-hard.*

Proof: Reduction from 3-COLORABILITY. Given a graph $G = (N, E)$, with $N = \{1, 2, \ldots, n\}$, such that $(i, i) \notin E$ for each $i \in [1, n]$, and given colors w (white), b (blue) and r (red), we define the relation p with attributes A, B, C, D and the following tuples:

1. for every $1 \leq i \leq n$, $(i, w, i, w) \in p$, $(i, b, i, b) \in p$ and $(i, r, i, r) \in p$.

2. for every $(i,j) \in E$, $(i,w,j,b) \in p$, $(i,w,j,r) \in p$, $(i,b,j,w) \in p$, $(i,b,j,r) \in p$, $(i,r,j,w) \in p$ and $(i,r,j,b) \in p$.

We consider the set of functional dependencies $F_1 = \{A \to B, C \to D\}$. We will show that G is 3-colorable iff there is a repair p' of p with exactly $n + 2 \cdot |E|$ tuples (the maximum possible number of tuples in a repair). That property follows from Lemmas 3, 4 and 5. □

Lemma 3 *Assuming p is defined as in the proof of Lemma 2, every repair p' of p has at most $n + 2 \cdot |E|$ tuples.*

Proof: by induction on n. If n is equal to 1, then p is equal to

p

1	w	1	w
1	b	1	b
1	r	1	r

and, therefore, it has three repairs:

p_1

1	w	1	w

p_2

1	b	1	b

p_3

1	r	1	r

Thus, $|p_1| = |p_2| = |p_3| = 1 \leq n + 2 \cdot |E|$.

Suppose that the theorem is satisfied in every graph with n nodes. Let (N, E) be a graph containing $n+1$ nodes, p be a table constructed from (N, E) as we showed above and p' be a repair of p. Define $N^* = N - \{n+1\}$, $E^* = E \cap N^* \times N^*$, $p^* = p \cap N^* \times \{w,b,r\} \times N^* \times \{w,b,r\}$ and $(p^*)' = p' \cap N^* \times \{w,b,r\} \times N^* \times \{w,b,r\}$.

$(p^*)'$ satisfies the set of functional dependencies and it only contains tuples from table p^*. Then, there exists a repair $(p^*)''$ of p^* such that $(p^*)' \subseteq (p^*)''$. Thus, by induction hypothesis we conclude that $|(p^*)''| \leq n + 2 \cdot |E^*|$, and, therefore, $|(p^*)'| \leq n + 2 \cdot |E^*|$.

In order to know how many tuples p' could have, we need to know how many tuples $p' - (p^*)'$ could contain, which can be established by considering the following:

(I) This set could contains at most one of the following tuples: $(n+1, w, n+1, w)$, $(n+1, b, n+1, b)$, $(n+1, r, n+1, r)$.

(II) For each $(i, n+1) \in E$, this set could contains at most two tuples of the form $(i, color_1, n+1, color_2)$, $(n+1, color_3, i, color_4)$.

By (I) and (II) we conclude that

$$|p' - (p^*)'| \leq 1 + 2 \cdot |E - E^*|$$

and, therefore,

$$|p'| \leq n + 2 \cdot |E^*| + 1 + 2 \cdot |E - E^*| \leq n + 1 + 2 \cdot |E|.$$

□

Lemma 4 *Assuming p is defined as in the proof of Lemma 2, if it is possible to color the graph (N, E) where $N = \{1, 2, \ldots, n\}$, with colors w, b and r, then there exists a repair of p with $n + 2 \cdot |E|$ tuples.*

Proof: Suppose that C_i is the color assigned to the node i in the graph. Define p' as follows:

1. For every $1 \leq i \leq n$, $(i, C_i, i, C_i) \in p'$.
2. For every $(i,j) \in E$, $(i, C_i, j, C_j) \in p'$ and $(j, C_j, i, C_i) \in p'$.

Clearly, p' satisfies the integrity constraints $A \to B$ and $C \to D$. But $|p'| = n + 2 \cdot |E|$ and, therefore, by the previous lemma we conclude that p' is a repair of p. □

Lemma 5 *Assuming p is defined as in the proof of Lemma 2, if there is a repair p' of p with $n + 2 \cdot |E|$ tuples, then is possible to color the graph (N, E) by using colors w, b and r.*

Proof: Let $q = \{(i, x, i, x) \mid 1 \leq i \leq n$ and x is equal to w, b or $r\}$. For every $(i,j) \in E$, there are 12 tuples in p mentioning i and j:

$$\begin{array}{lll} (i,w,j,b) & (i,r,j,w) & (j,b,i,w) \\ (i,w,j,r) & (i,r,j,b) & (j,b,i,r) \\ (i,b,j,w) & (j,w,i,b) & (j,r,i,w) \\ (i,b,j,r) & (j,w,i,r) & (j,r,i,b) \end{array}$$

A repair of p must have at most two tuples from this set and, therefore, $|p' - q| \leq 2 \cdot |E|$. Thus, $|p' \cap q|$ must be equal to n, since $|p'| = n + 2 \cdot |E|$. Hence, for every node i there exists a color C_i such that $(i, C_i, i, C_i) \in p'$. We will prove that if we choose color C_i for painting node i, then we have a coloring for the graph.

Let $(i,j) \in E$. There are at most two tuples in p' that mention i and j together. If we have zero or one of these kind of tuples, $|p' - q| < 2 \cdot |E|$ and, therefore, $|p'| < n + 2 \cdot |E|$, a contradiction. Thus, we have exactly two tuples in p' mentioning i and j together. But these tuples together with (i, C_i, i, C_i) and (j, C_j, j, C_j) cannot violate the set of functional dependencies, because p' is a repair. Then $(i, C_i, j, C_j) \in p$ and $(j, C_j, i, C_i) \in p$. By the definition of p, we conclude that $C_i \neq C_j$. □

Lemma 6 *There is a set of 2 FDs F_2 for which the problem of determining the existence of a repair of r of size $\leq k$ is NP-data-hard.*

Proof: Modification of the lower bound proof of Theorem 3. We build the instance by using the same tuples of the first and second kinds, as well as "sufficiently many tuples" of the third kind, each with a different new symbol w. It is enough to have $3n+1$ tuples of the third kind for each clause (where n is the number of clauses), thus the instance will have the total of $3n + n(3n+1)$ tuples. Every repair that contains a tuple of the third kind, has to contain at least $3n+1$ such tuples (by maximality). The formula φ is satisfiable iff there is a repair of size $\leq 3n$. □

Lemmas 2 and 6 imply the following theorems, resp.

Theorem 4 *There is a set of two FDs F_1 for which determining whether*

$$lub_{F_1}^{\mathtt{COUNT(*)}}(r) \geq k$$

in r is NP-data-hard. □

Theorem 5 *There is a set of two FDs F_2 for which determining whether*

$$glb_{F_2}^{\mathtt{COUNT(*)}}(r) \leq k$$

in r is NP-data-hard. □

Analogous results to Theorems 4 and 5 can be obtained for `SUM(A)` and the proofs are easy modifications of the above proofs (`COUNT(*)` can be mimicked by `SUM` over an additional attribute that has the value 1 in each tuple).

3.3 Aggregation using `AVG`

3.3.1 One functional dependency

We reduce the problem of computing glb- and lub-answers to `AVG(A)` queries w.r.t. a single FD $X \to Y$ to the following problem of MAXIMUM AVERAGE WEIGHT (MAW):

> There are m bins, each containing weighted, colored objects. No two bins have objects of the same color, although any particular bin may contain more than one object of the same color. Choose exactly one color for each bin in such a way that the sum of the weights of all objects of the chosen colors divided by the total number of such objects (i.e., the average weight AVG of objects of the chosen color) is maximized.

In the reduction of the problem of computing lub-answers to MAW, bins correspond to different X-values, and objects of the same color have the same Y-values. Each object corresponds to a tuple in which the attribute A represents the weight of the object. Different objects of the same color can have different weight, since A does not have to be a member of Y or be functionally dependent on Y. For glb-answers, we use an *inverted* database, as in the remarks after the proof of Theorem 2.

To solve MAW, consider the well-known "*2-OPT*" strategy of starting with an arbitrary selection $\langle c_1, c_2, ..., c_m \rangle$ of one color each from each of the m bins. The *2-OPT* strategy is simply to replace a color from one bin with a different color from the same bin if so doing increases the value of the average weight of objects of the colors in the selection.

More precisely, let $c = \langle c_1, c_2, ..., c_m \rangle$ be a selection of colors such that c_i is the color chosen from the i^{th} bin. Let $AVG(c)$ be the average weight of objects with colors from c. Let OPT be the maximum value over all choices of c of $AVG(c)$. Then *2-OPT* is the end result of the following strategy:

```
Let $c$ be any arbitrary selection of $m$ colors, one from each bin.
while there is a color $c'_i$ in bin $i$ : $AVG(\langle c_1, c_2, ..., c_{i-1}, c'_i, c_{i+1}, ..., c_m \rangle) > AVG(c)$ do
    $c := \langle c_1, c_2, ..., c_{i-1}, c'_i, c_{i+1}, ..., c_m \rangle$
endwhile
$\mathit{2\text{-}OPT} := AVG(c)$
```

We establish the proof of the main theorem through two intermediate lemmas.

Lemma 7 2-OPT = OPT

Proof: Let n_i denote the number of objects of color c_i and let w_i denote the total weight of objects of color c_i. For any color c'_i in bin i, it can be verified (after a little bit of arithmetic) that $AVG(\langle c_1, c_2, ..., c_{i-1}, c'_i, c_{i+1}, ..., c_m \rangle) > AVG(c)$ if and only if one of the following holds (where n'_i is the number of objects of color c'_i):

(1) $n_i = n'_i$ and $w'_i > w_i$

(2) $n_i < n'_i$ and $(w'_i - w_i)/(n'_i - n_i) > AVG(c)$

(3) $n_i > n'_i$ and $(w'_i - w_i)/(n'_i - n_i) < AVG(c)$.

Intuitively, the conditions above may be interpreted in the following way. Let the density of a set of objects be the sum of the weight of the objects divided by the number of objects. A swap is beneficial if and only if the net changes made correspond to adding objects with density greater than the current average density of the solution, or deleting objects with density smaller than the current average density of the solution.

Now if *2-OPT* $<$ OPT, then there is some c for which none of the above 3 conditions holds for any choice of c'_i in any bin i and yet OPT is larger than $AVG(c)$. Specifically, let the coloring for OPT be $d = \langle d_1, d_2, ..., d_m \rangle$ where the total weight of objects of color d_i is u_i and the total number of such objects is m_i. We will show that for some i, the choice $c'_i = d_i$ will satisfy one of the above 3 conditions, yielding a contradiction.

First observe that, for any i such that $n_i = m_i$, having $w_i < u_i$ would immediately give a contradiction. Also, if $u_i < w_i$, then OPT can be improved by replacing d_i with c_i, again a contradiction. Therefore, if $n_i = m_i$ it must be that $u_i = w_i$ and we may as well assume that $d_i = c_i$ for all such i.

Next consider the colors which are different in d and c. We will use the elementary fact that if

$$\frac{p+A+B}{q+C+D} > \frac{p}{q} \text{ then either } \frac{p+A}{q+C} > \frac{p}{q} \text{ or } \frac{p+B}{q+D} > \frac{p}{q}.$$

In particular, let $q = \sum_i n_i$ and $p = \sum_i w_i$. Let $E = \sum_{i:d_i \neq c_i} u_i - w_i$ and $F = \sum_{i:d_i \neq c_i} m_i - n_i$. Observe that $\frac{p+E}{q+F} = OPT > \mathit{2\text{-}OPT} = \frac{p}{q}$. If the sum in E runs over only one index i then $c'_i = d_i$ satisfies (2) or (3) above, a contradiction. Otherwise, E may be partitioned into two sums A and B and F into corresponding sums C and D such that the above fact guarantees that either $\frac{p+A}{q+C} > \mathit{2\text{-}OPT}$ or $\frac{p+B}{q+D} > \mathit{2\text{-}OPT}$. If the former is true, we replace E and F with A and C; otherwise we replace E and F with B and D. Repeated application of the above fact in this manner will eventually result in finding some i such that $c'_i = d_i$ satisfies (2) or (3), a contradiction. □

We have just shown that the simple *2-OPT* strategy will converge to the value OPT. However, it does not necessarily follow that the number of iterations of the *2-OPT* strategy is polynomial. For this, we need another idea.

Let c be any selection of colors with one color from each bin. We say that color c_i is *stable* if there exists no c'_i in the i^{th} bin for which the condition (1), (2), or (3) holds. Note that if color c_i can be replaced by any color in bin i to produce an increase in the value of AVG, then there exists a stable color with which it can be replaced, this simply being the color which results in the largest value of AVG obtained by maintaining the colors in all bins other than i fixed while trying different colors from the i^{th} bin. Clearly such a stable color can be found by simply cycling through the choices for the i^{th} bin. This leads to the following "*Stable-2-OPT*" strategy.

Let c be any arbitrary selection of m colors, one from each bin.

```
while there is a color $c'_i$ in bin $i$ : $AVG(\langle c_1, c_2, ..., c_{i-1}, c'_i, c_{i+1}, ..., c_m\rangle) > AVG(c)$ do
        Find a stable color $d_i$ for bin $i$.
        $c := \langle c_1, c_2, ..., c_{i-1}, d_i, c_{i+1}, ..., c_m\rangle$
endwhile
$Stable\text{-}2\text{-}OPT := AVG(c)$
```

¿From Lemma 7, it follows that *Stable-2-OPT* = *OPT* also. In addition, we claim the following.

Lemma 8 *Any color c_i is chosen in the* Stable-2-OPT *strategy at most once as a stable color for the i^{th} bin.*

Proof: Consider the situation when a color c_i is chosen as a stable color for the first time. At this point in time, none of the conditions (1), (2), or (3) holds with respect to other colors c'_i in bin i. This means that none of the colors c'_i with $n_i < n'_i$ can ever take the i^{th} position as a stable color since *AVG* increases monotonically in the run of the strategy and color c_i will always be preferred over such a color c'_i. Similarly, none of the colors c'_i with $n'_i = n_i$ can ever take the i^{th} position as a stable color. Finally, if a color c'_i replaces c_i as a stable color, it must be because *AVG* has increased to such an extent that condition (3) now holds; subsequently the monotonicity of *AVG* ensures that color c'_i will always be preferred to color c_i as a stable color for the i^{th} bin and hence color c_i will never ever be chosen again. □

Each iteration of the while loop chooses a new stable color and by Lemma 8, a color is chosen at most once. It follows that the number of iterations of the while loop is at most the number of colors available. Therefore *Stable-2-OPT* finishes in polynomial time.

Now the main theorem follows from Lemmas 7 and 8.

Theorem 6 *If the set of FDs F consists of a single dependency $X \rightarrow Y$, with $X \cap Y = \emptyset$, then the data complexity of computing both $glb_F^{\mathtt{AVG(A)}}(r)$ and $lub_F^{\mathtt{AVG(A)}}(r)$ in an instance r is in PTIME.* □

3.3.2 Two functional dependencies

Theorem 7 *There is a set of two FDs F_3 for which determining whether*

$$glb_{F_3}^{\mathtt{AVG(A)}}(r) \leq k$$

in r is NP-data-hard.

Proof: We can use the same reduction from 3SAT as in theorem 3. Given a set of clauses, there is a satisfying assignment if and only if there is a repair of the corresponding database r for which `AVG(D)` $= 1$ (since otherwise the glb-answer is greater than 1). This is the case if and only if $glb_{F_3}^{\mathtt{AVG(D)}}(r) \leq 1$. □

Theorem 8 *There is a set of two FDs F_4 for which determining whether*

$$lub_{F_4}^{\mathtt{AVG(A)}}(r) \geq k$$

in r is NP-data-hard.

Proof: We reduce 3SAT to our problem. Change the tuples of the instance in the proof of theorem 3 as follows:

3'. $(w, 2, C_j, d)$, $1 \leq j \leq n$, where w is a new symbol and $d < 1$.

Given a set of clauses, there is a satisfying assignment if and only if there is a repair of the corresponding database r for which `AVG(D)` $= 1$. This is the case if and only if $lub_{F_4}^{\mathtt{AVG(D)}}(r) \geq 1$. □

3.4 Aggregation using `COUNT(A)`

We assume here that distinct values of A are counted (`COUNT (DISTINCT A)`).

Theorem 9 *There is a single FD $d_0 = B \rightarrow A$ for which determining whether*

$$glb_{d_0}^{\mathtt{COUNT}(A)}(r) \leq k$$

in r is NP-data-hard.

Proof: To see that the lower bound holds, we will encode an instance of the HITTING SET problem in r (whose schema is $R(A, B)$). The HITTING SET problem [14] is formulated as follows: Given a collection $C = \{S_1, \ldots, S_n\}$ of sets, is there a set H (called a *hitting set*) with k or fewer elements that intersects all the members of C? For every set S_i in C and every element $x \in S_i$ we put the tuple (x, i) in r. There is in C a hitting set of size less than or equal to k if and only if there is a repair of r with at most k different values of the first attribute A. □

Theorem 10 *There is a single FD $d_1 = A \rightarrow B$ for which determining whether*

$$lub_{d_1}^{\mathtt{COUNT}(C)}(r) \geq k$$

in r is NP-data-hard.

Proof: We reduce SAT to this problem. Let $\varphi = C_1 \wedge \ldots \wedge C_n$. Consider the functional dependency $A \rightarrow B$ and the database instance r over the schema ABC with the following tuples:

1. $(p_i, 1, C_j)$ if making p_i true makes C_j true.
2. $(p_i, 0, C_j)$ if making p_i false makes C_j true.

Then, φ is satisfiable iff $lub_{d_1}^{\mathtt{COUNT(C)}}(r) \geq n$. □

3.5 Summary of Complexity Results

The following is a tabular summary of the results presented in this section. The membership in NP is from Proposition 2.

	glb-answer		lub-answer	
	$\lvert F\rvert = 1$	$\lvert F\rvert \geq 2$	$\lvert F\rvert = 1$	$\lvert F\rvert \geq 2$
`MIN(A)`	PTIME	PTIME	PTIME	NP-complete
`MAX(A)`	PTIME	NP-complete	PTIME	PTIME
`COUNT(*)`	PTIME	NP-complete	PTIME	NP-complete
`COUNT(A)`	NP-complete	NP-complete	NP-complete	NP-complete
`SUM(A)`	PTIME	NP-complete	PTIME	NP-complete
`AVG(A)`	PTIME	NP-complete	PTIME	NP-complete

4 Hybrid Computation

As we have seen, determining glb-answers and lub-answers is often computationally hard. However, it seems that hard instances of those problems are unlikely to occur in practice. We expect that in a typical instance a large majority of tuples are not involved in any conflicts. If this is the case, it is advantageous to break up the computation of the lub-answer (or the glb-answer) to f in r into three parts:

1. the computation of f in the core of r,
2. the computation of the lub-answer to f in the complement of the core of r (which should be small), and
3. the combination of the results of the first two steps using an operator g (which depends on f).

The first step can be done using a DBMS because the core of r can be computed using a first-order query (Theorem 1).

Definition 13 *The scalar function f admits a* g-decomposition *of its lub-answers (resp. glb-answers) w.r.t. a set of FDs F if for every instance r of R, the lub-answer (resp. glb-answer) v to f satisfies the condition*

$$v = g(f(Core_F(r)), v')$$

where $v' = lub_F^f(r - Core_F(r))$ (resp. $v' = glb_F^f(r - Core_F(r))$). □

Theorem 11 *The following pairs describe g-decompositions admitted by scalar functions f:*

1. *$f =$ `MIN(A)`, $g = \min$;*
2. *$f =$ `MAX(A)`, $g = \max$;*
3. *$f =$ `COUNT(*)`, $g = +$;*
4. *$f =$ `SUM(A)`, $g = +$.*

Proof: First, notice that every repair r' of r w.r.t. a set of FDs F is a union of $Core_F(r)$ and a repair of $r - Core_F(r)$. Now to see that the first decomposition holds for $f =$ `MIN(A)` consider:

$$lub\ Poss_F^f(r) = \min(f(Core_F(r)), lub\ Poss_F^f(r - Core_F(r)))$$

and similarly for glb-answers and other decompositions. □

5 Special Cases

We consider here several cases when the conflict graph has a special form that could be used to reduce the complexity of computing answers to aggregation queries. We only consider lub-answers to `COUNT(*)` queries. It is an open question whether our approach will generalize to other classes of scalar aggregation queries.

5.1 BCNF

We show here that if the set of FDs F has two dependencies and the schema R is in Boyce-Codd Normal Form (BCNF), computing lub-answers to `COUNT(*)` queries can be done in PTIME. This should be contrasted with Theorem 4 which showed that two dependencies without the BCNF assumption are sufficient for NP-hardness.

Given a set of FDs F in a schema R, we say that the schema R is in *BCNF* if all the dependencies in F are of the form $X \rightarrow Y$ where X contains a key of R (w.r.t. F). This definition can be found in every database textbook. BCNF is often satisfied in practice, since schemas in BCNF are considered good, by the virtue of being free of redundancies and insertion/deletion/update anomalies. For example, the relation instance in the proof of Theorem 3 is not in BCNF, since neither A nor C is a key of this relation.

We pursue here two different approaches to BCNF schemas. The first [5] is based on the observation that for 2 FDs in BCNF the conflict graph is claw-free. For such graphs computing a maximum independent set (an independent set of maximum cardinality) can be done in PTIME. The second approach is direct and yields a subquadratic time complexity bound.

Definition 14 *A FD $X \rightarrow Y$ is a* partition dependency *over R if $X \cup Y = U$ (where U is the set of all the attributes of R) and $X \cap Y = \emptyset$.* □

Lemma 9 *For any instance r of R and any partition dependency $d = X \rightarrow Y$ over R, the conflict graph $G_{d,r}$ is a union of disjoint cliques.*

Proof: Assume (t_1, t_2) and (t_2, t_3) are two edges in $G_{d,r}$ such that $t_1 \neq t_3$. Then $t_1[X] = t_2[X]$, $t_1[Y] \neq t_2[Y]$, $t_2[X] = t_3[X]$, and $t_2[Y] \neq t_3[Y]$. Therefore $t_1[X] = t_3[X]$. Also, $t_1[Y] \neq t_3[Y]$ because otherwise t_1 and t_3 would be the same tuple. So (t_1, t_3) is an edge in $G_{d,r}$. □

Lemma 10 *If R is in BCNF and F is equivalent to a set of FDs with k dependencies, then F is equivalent to a set of FDs with at most k partition dependencies.*

Proof: We build a set of partition dependencies equivalent to F by replacing every non-trivial dependency $d = X \rightarrow Y$, $d \in F$, by the partition dependency $X \rightarrow U - X$. □

Therefore, in the case $|F| = 2$ we can assume that $F = \{d_1, d_2\}$ where d_1 and d_2 are different partition dependencies. (The case of $|F| = 1$ has already been shown to be in PTIME, even without the BCNF assumption.) Note that Lemma 10 does not have to hold for arbitrary FDs.

We consider now the first class of graphs for which maximum independent set can be computed in PTIME: *claw-free* graphs. Since repairs correspond to maximal independent sets in the conflict graph, the size of a maximum independent set provides the lub-answer to a `COUNT(*)` aggregation query.

Definition 15 *A graph is* claw-free *if it does not contain an induced subgraph (V_0, E_0) where $V_0 = \{t_1, t_2, t_3, t_4\}$ and $E_0 = \{(t_2, t_1), (t_3, t_1), (t_4, t_1)\}$.* □

Lemma 11 *If R is in BCNF over $F = \{d_1, d_2\}$, then for every instance r of R, the conflict graph $G_{\{d_1,d_2\},r}$ is claw-free.*

Proof: Assume that the conflict graph contains a claw (V_0, E_0) where $V_0 = \{t_1, t_2, t_3, t_4\}$ and $E_0 = \{(t_2, t_1), (t_3, t_1), (t_4, t_1)\}$. Then two of the edges in E_0, say (t_2, t_1) and (t_3, t_1) come from one of $G_{d_1,r}$ or $G_{d_2,r}$. By Lemma 9, the edge (t_3, t_2) also belongs to that graph, and consequently to $G_{\{d_1,d_2\},r}$. Thus the subgraph induced by V_0 is not a claw. □

We note that it can also be shown that the conflict graph is perfect in this case [5]. (A graph is *perfect* if its chromatic number is equal to the size of its maximum clique.)

Theorem 12 *If the relational schema R is in BCNF and the given set of FDs F is equivalent to one with at most two dependencies, computing $lub_F^{\texttt{COUNT}(*)}(r)$ in any instance r of R can be done in PTIME.*

Proof: The theorem follows from Lemma 11 and the fact that a maximum independent set in a claw-free graph can be found in polynomial time [28, 26]. □

We show now the second approach that directly yields an $O(n^{1.5})$ complexity bound.

Theorem 13 *If the relational schema R is in BCNF and the given set of FDs F is equivalent to one with at most two dependencies, computing $lub_F^{\texttt{COUNT}(*)}(r)$ in any instance r of R can be done in $O(n^{1.5})$ time where n is the number of tuples in r.*

Proof: Suppose that $G_{d_1,r} = (V, E_1)$ and $G_{d_2,r} = (V, E_2)$. Then $G_{\{d_1,d_2\},r} = (V, E_1 \cup E_2)$. By Lemma 9, both $G_{d_1,r} = (V, E_1)$ and $G_{d_2,r} = (V, E_2)$ are unions of disjoint cliques. Let $U_1, U_2, \ldots, U_{k_1}$ be the cliques in $G_{d_1,r}$. Let $W_1, W_2, \ldots, W_{k_2}$ be the cliques in $G_{d_2,r}$.

In order to find a maximum independent set in $G_{\{d_1,d_2\},r} = (V, E_1 \cup E_2)$, we construct a bipartite graph $H = (U \cup W, E_H)$ as follows: $U = \{u_1, u_2, \ldots, u_{k_1}\}$ and $W = \{w_1, w_2, \ldots, w_{k_2}\}$. For each vertex $v \in V$, v is in exactly one clique U_i and in exactly one clique W_j. We add an edge (u_i, w_j) into E_H. H contains only these edges.

A *matching* of H is a subset $M \subseteq E_H$ such that no two edges in M share a common end vertex. The crucial observation is that the independent sets in $G_{\{d_1,d_2\},r} = (V, E_1 \cup E_2)$ one-to-one correspond to the matchings in H. To see this, first note that the vertices of $G_{\{d_1,d_2\},r}$ one-to-one correspond to the edges of H. Consider two vertices x, y in $G_{\{d_1,d_2\},r}$. Suppose that $e = (x, y)$ is an edge in $G_{\{d_1,d_2\},r}$. Without loss of generality, we may assume $e \in E_1$. Then both x and y are in the same clique U_i. Hence, the two edges in H corresponding to x and y share a common vertex u_i in H. Conversely, suppose that x and y are not adjacent in $G_{\{d_1,d_2\},r}$. Then x and y are in different cliques in $G_{\{d_1\},r}$, say they are in U_i and $U_{i'}$, where $i \neq i'$, respectively. Similarly, x and y are in different cliques in $G_{\{d_2\},r}$, say they are in W_j and $W_{j'}$, where $j \neq j'$, respectively. Thus the edge (u_i, v_j) in H corresponding to x and the edge $(u_{i'}, v_{j'})$ in H corresponding to y share no common end vertex in H. Therefore, a subset S of vertices in $G_{\{d_1,d_2\},r}$ is an independent set if and only its corresponding edge set is a matching in H. Hence, finding a maximum independent set in $G_{\{d_1,d_2\},r}$ is equivalent to finding a maximum matching in the bipartite graph H. This can be done in $O((|U| + |W|)^{1/2}|E_H|)$ time by using the algorithm in [20]. Since $|U| \leq n, |W| \leq n$ and $|E_H| = n$, the total time needed is $O(n^{1.5})$. □

We show now that more than two FDs, even in BCNF, push the problem of computing lub-answers beyond tractability.

Theorem 14 *If the relational schema R is in BCNF and the given set of FDs F is equivalent to one with three dependencies, the data complexity of computing $lub_F^{\texttt{COUNT}(*)}(r)$ in an instance r of R is NP-hard.*

Proof: Let d_1, d_2, d_3 be three partition dependencies. As before, the graph $G_{d_i,r} = (V, E_i)$ $(1 \leq i \leq 3)$ is a union of disjoint cliques. We also have $G_{\{d_1,d_2,d_3\},r} = (V, E_1 \cup E_2 \cup E_3)$. Our problem is equivalent to finding a maximum independent set in $G_{\{d_1,d_2,d_3\},r}$.

To show the problem is NP-hard, we reduce the 3-Dimensional Matching (3DM) problem [14] to it. The 3DM problem is defined as follows:

An instance of 3DM is a tuple (X, Y, Z, M), where X, Y, Z are three disjoint sets of the same cardinality, and $M \subseteq X \times Y \times Z$. A *matching* of the instance is a subset $M' \subseteq M$ such that no two elements in M' agree in any coordinate. The goal is to determine the existence of a (maximum) matching of size $|X|$.

Given an instance (X, Y, Z, M) of 3DM, we construct a graph $D = (V_D, E_D)$ as follows: $V_D = M$. Suppose $X = \{x_1, x_2, \ldots, x_t\}$. Partition M into $M_1, \ldots, M_t$ such that $M_i = \{(x_i, y, z) \in M\}$ (for $1 \leq i \leq t$). For each i $(1 \leq i \leq t)$, we add a clique into E_D whose vertices are exactly the triples in M_i. Denote the set of the edges added this way by E_X. Note that the graph (V_D, E_X) is a union of disjoint cliques. Similarly, we perform the same action for Y and Z, and let E_Y and E_Z be the sets of the edges added, respectively. We set $E_D = E_X \cup E_Y \cup E_Z$.

Note that the maximum matchings of the instance (X, Y, Z, M) one-to-one corresponds to the maximum independent sets of the graph D. Also note that $D = G_{\{d_1,d_2,d_3\},r}$ for the instance $r = M$ and partition dependencies $d_1 = A \rightarrow BC$, $d_2 = B \rightarrow AC$, and $d_3 = C \rightarrow AB$, where ABC is the schema of r. Thus, there is a maximum matching of size $|X|$ iff $glb_{\{d_1,d_2,d_3\}}^{\texttt{COUNT}(*)}(r) = |X|$. This completes the reduction. □

5.2 Other tractable cases

There are other, simpler cases where the conflict graph has a structure that makes it possible to determine the cardinality of a maximum independent set in PTIME.

Theorem 15 *If an instance r is the disjoint union of two instances that separately satisfy F, the data complexity of computing $lub_F^{\texttt{COUNT}(*)}(r)$ is in PTIME.*

Proof: In this case, the only conflicts are between the parts of r that come from different instances. Thus the conflict graph is a bipartite graph. For bipartite graphs determining the cardinality m of a maximum independent set can be done in PTIME. This follows from the fact that $m = n - k$ where n is the number of vertices in the graph and k is the cardinality of the minimum vertex cover. The latter is equal to the cardinality of the maximum matching in the graph (König-Egervary Theorem [23]). □

Note that the assumption in Theorem 15 is satisfied when the instance r is obtained by merging together two consistent databases in the context of database integration.

Theorem 16 *If every tuple in an instance r is in conflict with at most two tuples in the same instance, the data complexity of computing $lub_F^{\texttt{COUNT}(*)}(r)$ is in PTIME.*

Proof: In this case, each vertex in the conflict graph has degree at most 2, thus the conflict graph is a union of disjoint components each of which is an isolated vertex, a non-cyclic path, or a single cycle. Finding the cardinality of a maximum independent set in such a graph can clearly be done in PTIME. □

6 Related and Further Work

We have provided a complete classification of the tractable/intractable cases of the problem of computing glb- and lub-answers to aggregation queries with scalar functions in the presence of functional dependencies. We have also shown how tractability can be obtained in several special cases and presented a practical hybrid computation method.

We only briefly survey the related work here. A more comprehensive discussion can be found in [3]. The need to accommodate violations of functional dependencies is one of the main motivations for considering disjunctive databases [22, 30] and has led to various proposals in the context of data integration [2, 6, 13, 25]. A purely proof-theoretic notion of consistent query answer comes from Bry [7]. This notion, described only in the propositional case, corresponds to our notion of core answer. None of the above approaches considers aggregation queries.

There seems to be an intriguing connection between relation repairs w.r.t. FDs and databases with disjunctive information [30]. For example, the set of repairs of the relation *BrownVotes* from Example 1 can be represented as a disjunctive database D consisting of the formulas

$$BrownVotes(A, 11/07, 541) \vee BrownVotes(A, 11/11, 560)$$

and

$$BrownVotes(B, 11/07, 302).$$

Each repair corresponds to a minimal model of D and vice versa. We conjecture that the set of all repairs of an instance w.r.t. a set of FDs can be represented as a disjunctive table (with rows that are disjunctions of atoms with the same relation symbol). This is not as obvious as it seems, as the repairs require an *exclusive* representation of disjunctions, which is forced through the minimal model semantics of disjunctive formulas. The relationship in the other direction does not hold. E.g., the set of minimal models of the formula

$$(p(a_1, b_1) \vee p(a_2, b_2)) \wedge p(a_3, b_3)$$

cannot be represented as a set of repairs of any set of FDs. However, we are not aware of any work on aggregation in general disjunctive databases (but see below).

The relationship between sets of repairs and databases with OR-objects [22, 9] is more complicated.

Example 7 *The set of repairs of the relation* BrownVotes *in Example 1 cannot be represented as a table with OR-objects. However, the set of repairs of the projection of* BrownVotes *on the first and third attributes:*

County	*Tally*
A	*541*
A	*560*
B	*302*

can be represented as

County	*Tally*
A	*OR(541,560)*
B	*302*

□

In the example above, the the schema of the relation `BrownVotes` was not in BCNF. But even under BCNF, there is still a mismatch.

Example 8 *Consider the following set of FDs $F = \{A \rightarrow B, A \rightarrow C\}$, which is in BCNF. The set of all repairs of the instance $\{(a_1, b_1, c_1), (a_1, b_2, c_2)\}$ cannot be represented as a table with OR-objects.* □

The relationship in the other direction, from tables with OR-objects to sets of repairs, also does not hold.

Example 9 *Consider the following table with OR-objects:*

OR(a,b)	*c*
a	*OR(c,d)*

It does not represent the set of all repairs of any instance under any set of FDs. □

A correspondence between sets of repairs and tables with OR-objects occurs only in the very restricted case when a relation is binary, say $R(A, B)$, and there is one FD $B \rightarrow A$.

Several people [12, 9] studied aggregation in databases with OR-objects. As in our case, the query results in this case are indefinite. The dissertation [12] suggests, like we do, to return ranges of values of the aggregate functions. On the other hand, the paper [9] proposes to return sets of all possible values of such functions. The second approach runs into the problem that the set of possible values may have exponential size, c.f., Example 5. The paper [9] discusses not only scalar aggregation but also aggregation functions (`GROUP BY` in SQL). Possibly, some of the techniques of that paper can be adapted if we extend the present results in that direction. Due to the above-mentioned lack of correspondence between sets of repairs and tables with OR-objects the results from our paper cannot be directly transferred to the context of [9], except in a very restricted case, and vice versa.

Incidentally, the paper [9] incorrectly claims that the greatest lower bound on the value of the aggregate function `COUNT(A)` can be computed in PTIME in tables with OR-objects. This is contradicted by our Theorem 9, which shows in an equivalent setting that checking whether the glb bound is less than or equal to k is an NP-complete problem. The paper [9] provides a greedy PTIME algorithm (Algorithm 3.1) for computing the glb of `COUNT(A)` but the algorithm is incorrect. To see this consider the set of OR-objects $S = \{OR(a, b), OR(a, c), OR(a, d), b, c, d\}$. The algorithm will compute 4 as the lower bound on the number of different values that cover all the OR-objects in S. However, this bound is actually $3 = |\{b, c, d\}|$.

There are several proposals for language constructs specifying nondeterministic queries that are related to our approach (*witness* [1], *choice* [15, 16, 18]). Essentially, the idea is to construct a maximal subset of a given relation that satisfies a given set of functional dependencies. Since there is usually more than one such subset, the approach yields nondeterministic queries in a natural way. Clearly, maximal consistent subsets (choice models [15]) correspond to repairs. Datalog with choice [15] is, in a sense, more general than our approach, since it combines enforcing functional dependencies with inference using Datalog rules. Answering queries in all choice models ($\forall G$-queries [18]) corresponds to our notion of computation of consistent query answers for first-order queries (Definition 5). However, in [18] the former problem is shown to be co-NP-complete and no tractable cases are identified. One of the sources of complexity in this case is the presence of Datalog rules, absent from our approach. Moreover, the procedure proposed in [18] runs in exponential time if there are exponentially many repairs, as in Example 5. Also, only conjunctions of literals are considered as queries in [18]. Arbitrary first-order or aggregation queries are not studied.

As mentioned earlier, the paper [3] contains a general method for transforming first-order queries in such a way that the transformed query computes the consistent answers to the orginal query. In that paper, soundness, completeness and termination of the transformation are studied, and some classes of constraints and queries for which consistent query answers can be computed in PTIME are identified. Representing repairs as stable models of logic programs with disjunction and classical negation has been proposed in [4, 17]. Those papers consider computing consistent answers to first-order queries (but not to aggregation queries). No tractable cases beyond those of [3] are identified in [4, 17], which is not surprising in view of Corollary 2.

Many further questions suggest themselves. First, is it possible to identify more tractable cases and to reduce the degree of the polynomial in those already identified? Second, is it possible to use approximation in the intractable cases? The INDEPENDENT SET problem is notoriously hard to approximate [19], but perhaps the special structure of the conflict graph may be helpful. Finally, it would be very interesting to see if our approach can be generalized to broader classes of queries and integrity constraints. In most implementations of SQL2, only functional dependencies in BCNF are supported (using `PRIMARY KEY` and `UNIQUE` constraints). Therefore, the approaches described in Section 5 may be applicable there. It is not obvious, however, how to generalize our approach to broader classes of queries. Is it possible to combine the approach of this paper with that of [3]?

There is some recent work done on rewriting aggregation queries in terms of *aggregation views* [29, 10, 11]. It would be interesting to explore how to take advantage of those results when computing consistent answers to aggregation queries. Another possible avenue is to consider *aggregation constraints* [24, 27].

Finally, alternative definitions of repairs and consistent query answers that include, for example, preferences are left for future work. Also, one can apply further aggregation to the results of aggregation queries in different repairs, e.g., the average of all `MAX(A)` answers.

Acknowledgments: Work supported in part by FONDECYT Grant 1000593, NSF grant IIS-0119186, and NSF grant INT-9901877/CONICYT Grant 1998-02-083. The comments of the anonymous referees are gratefully acknowledged.

References

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

[2] S. Agarwal, A.M. Keller, G. Wiederhold, and K. Saraswat. Flexible Relation: An Approach for Integrating Data from Multiple, Possibly Inconsistent Databases. In *IEEE International Conference on Data Engineering*, 1995.

[3] M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *ACM Symposium on Principles of Database Systems*, pages 68–79, 1999.

[4] M. Arenas, L. Bertossi, and J. Chomicki. Specifying and Querying Database Repairs Using Logic Programs with Exceptions. In *International Conference on Flexible Query Answering Systems*, pages 27–41. Springer-Verlag, 2000.

[5] M. Arenas, L. Bertossi, and J. Chomicki. Scalar Aggregation in FD-Inconsistent Databases. In *International Conference on Database Theory*, pages 39–53. Springer-Verlag, LNCS 1973, 2001.

[6] C. Baral, S. Kraus, J. Minker, and V.S. Subrahmanian. Combining Knowledge Bases Consisting of First-Order Theories. *Computational Intelligence*, 8:45–71, 1992.

[7] F. Bry. Query Answering in Information Systems with Integrity Constraints. In *IFIP WG 11.5 Working Conference on Integrity and Control in Information Systems.* Chapman &Hall, 1997.

[8] A. K. Chandra and D. Harel. Computable Queries for Relational Databases. *Journal of Computer and System Sciences*, 21:156–178, 1980.

[9] A. L. P. Chen, J-S. Chiu, and F. S. C. Tseng. Evaluating Aggregate Operations Over Imprecise Data. *IEEE Transactions on Knowledge and Data Engineering*, 8(2):273–284, 1996.

[10] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting Aggregate Queries Using Views. In *Proc. ACM PODS'99*, pages 155–166, 1999.

[11] S. Cohen, W. Nutt, and A. Serebrenik. Algorithms for Rewriting Aggregate Queries Using Views. In *Proc. Symposium on Advances in Databases and Information Systems (ADBIS-DASFAA' 2000), Prague, Sept. (to appear)*, 2000.

[12] L. G. De Michiel. *Performing Database Operations over Mismatched Domains.* PhD thesis, Stanford University, 1989.

[13] Phan Minh Dung. Integrating Data from Possibly Inconsistent Databases. In *International Conference on Cooperative Information Systems*, Brussels, Belgium, 1996.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP Completeness.* W. H. Freeman and Co., 1979.

[15] F. Giannotti, S. Greco, D. Sacca, and C. Zaniolo. Programming with Non-determinism in Deductive Databases. *Annals of Mathematics and Artificial Intelligence*, 19(3-4), 1997.

[16] F. Giannotti and D. Pedreschi. Datalog with Non-deterministic Choice Computes NDB-PTIME. *Journal of Logic Programming*, 35:75–101, 1998.

[17] G. Greco, S. Greco, and E. Zumpano. A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. In *International Conference on Logic Programming*, pages 348–364. Springer-Verlag, LNCS 2237, 2001.

[18] S. Greco, D. Sacca, and C. Zaniolo. Datalog Queries with Stratified Negation and Choice: from P to D^P. In *International Conference on Database Theory*, pages 82–96. Springer-Verlag, 1995.

[19] D.S. Hochbaum. Approximating Covering and Packing Problems: Set Cover, Vertex Cover, Independent Set, and Related Problems. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Co., 1997.

[20] J. E. Hopcroft and R. M. Karp. An $O(n^{5/2})$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[21] T. Imieliński and W. Lipski. Incomplete Information in Relational Databases. *Journal of the ACM*, 31(4):761–791, 1984.

[22] T. Imieliński, S. Naqvi, and K. Vadaparty. Incomplete Objects - A Data Model for Design and Planning Applications. In *ACM SIGMOD International Conference on Management of Data*, pages 288–297, Denver, Colorado, May 1991.

[23] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[24] A. Y. Levy and I. Singh Mumick. Reasoning with Aggregation Constraints. In *Proc. EDBT*, pages 514–534, 1996.

[25] J. Lin and A. O. Mendelzon. Merging Databases under Constraints. *International Journal of Cooperative Information Systems*, 7(1):55–76, 1996.

[26] G. J. Minty. On Maximal Independent Sets of Vertices in Claw-Free Graphs. *Journal of Combinatorial Theory B*, 28:284–304, 1980.

[27] K. A. Ross, D. Srivastava, P. J. Stuckey, and S. Sudarshan. Foundations of Aggregation Constraints. *Theoretical Computer Science*, 193(1-2):149–179, 1998.

[28] M. Sbihi. Algorithme de Recherche d'un Stable de Cardinalité Maximum dans un Graphe sans Étoile. *Discrete Mathematics*, 29:53–76, 1980.

[29] D. Srivastava, S. Dar, H. V. Jagadish, and A. Y. Levy. Answering Queries with Aggregation Using Views. In *Proc. VLDB'96*, pages 318–329, 1996.

[30] R. van der Meyden. Logical Approaches to Incomplete Information: A Survey. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 10. Kluwer Academic Publishers, Boston, 1998.

[31] M. Y. Vardi. The Complexity of Relational Query Languages. In *ACM Symposium on Theory of Computing*, pages 137–146, 1982.

Consistent Query Answers in Virtual Data Integration Systems

Leopoldo Bertossi and **Loreto Bravo**
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 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.

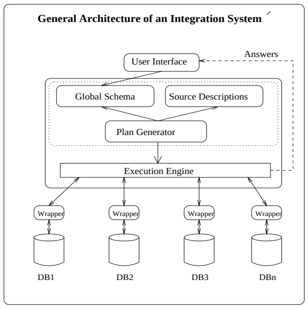


Fig. 1. Architecture of an Integration System

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

Page 62

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 $Q: \quad Ans(L) \quad Contract(NorahJones, 2002, L)$.

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 $\texttt{SELECT}_{X=NorahJones, Y=2002}\ \texttt{Contract(X,Y,L)}$.

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

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

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

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

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

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

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*,

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

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

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

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.

$$\begin{aligned} Ans'(Album, Artist, Song) \quad & \underline{DB_1(Album, Artist, 2003)}, \\ & \underline{\underline{DB_1(Album, Artist, Year), DB_3(Album, Song)}}. \\ Ans'(Album, Artist, Song) \quad & \underline{DB_2(Album, Artist, 2003, Label)}, \\ & \underline{\underline{DB_1(Album, Artist, Year), DB_3(Album, Song)}}. \end{aligned}$$

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

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

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: $CD(Album, Artist, Year)$, $Songs(Album, Song)$, $Contract(Artist, Year, Label)$. □

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:

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

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:

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

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.

[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.

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 $\forall x \varphi(x)$, where $\varphi(x)$ is a quantifier-free formula; and also *referential integrity constraints* of the form $\forall x(P(x) \rightarrow \exists y(Q(x', y))$, where $x' \quad x$.

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:

$$\begin{array}{llll} \mathcal{G}: & V_1(X_1) \quad \varphi_1(X_1'); & v_1 & (1) \\ & & & \\ & V_n(X_n) \quad \varphi_n(X_n'); & v_n & \end{array}$$

Here, $X_j \quad X_j'$, 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 $\mathcal{G}$.

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

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

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

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) $v_1 \quad \varphi_1(D) = \{(a, b), (c, d), (a, c), (d, e)\}$; and (b) $v_2 \quad \varphi_2(D) = \{(a, b), (c, d), (a, c), (d, e)\}$. 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 $\mathcal{D} = \{a, b, c, \ldots\}$ be the underlying domain. Consider the integration system $\mathcal{G}_2$ defined by

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

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 $v_1 \quad \varphi_1(D) = \{(a, b)\}$ and $v_2 \quad \varphi_2(D) = \{(a, c), (a, z)\}$. 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]:

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

Example 8. (example 6 continued) Consider the following global query Q posed to system $\mathcal{G}_1$: $\mathit{Ans}(X, Y) \quad R(X, Y)$. 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.

[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 $D = \{R(a, b), R(c, d), ...\}$.

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

$$\begin{array}{lll} S_1: & V_1(X, Z) \quad R_1(X, Y), R_2(Y, Z), & (2) \\ S_2: & V_2(X, Y) \quad R_3(X, Y). & (3) \end{array}$$

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 "$\subseteqq$" R_3, or, equivalently, V_2 "$\Rightarrow$" R_3. More precisely, we invert the rule in the description of V_2, obtaining

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

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

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

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 $R_1(X, Y) \quad V_1(X, Z)$ and $R_2(Y, Z) \quad V_1(X, Z)$, 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: $V_1(X, Z) \quad R_1(X, Y), R_2(Y, Z)$ is equivalent to $V_1(X, Z) \quad \exists Y\ (R_1(X, Y) \wedge R_2(Y, Z))$ (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 t is a *consistent answer* to Q in D wrt IC whenever 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 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.

Page 65

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.

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

Table 1. Semantic of Annotation Constants

Example 12. (example 10 continued) The repair program (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 $\mathbf{t}^\star$ $(\mathbf{f}^\star)$:

2. $P(X, \mathbf{t}^\star) \quad P(X, \mathbf{t_d})$
   $P(X, \mathbf{t}^\star) \quad P(X, \mathbf{t_a})$
   $P(X, \mathbf{f}^\star) \quad \mathit{not}\ P(X, \mathbf{t_d})$
   $P(X, \mathbf{f}^\star) \quad P(X, \mathbf{f_a})$ ... the same for R ...

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

   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. $P(X, \mathbf{t_a}), P(X, \mathbf{f_a})$
   $R(X, \mathbf{t_a}), R(X, \mathbf{f_a})$

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. $P(X, \mathbf{t}^{\star\star}) \quad P(X, \mathbf{t_a})$
   $P(X, \mathbf{t}^{\star\star}) \quad P(X, \mathbf{t_d}),\ \mathit{not}\ P(X, \mathbf{f_a})$
   $P(X, \mathbf{f}^{\star\star}) \quad P(X, \mathbf{f_a})$
   $P(X, \mathbf{f}^{\star\star}) \quad \mathit{not}\ P(X, \mathbf{t_d}),\ \mathit{not}\ P(X, \mathbf{t_a})$. ... etc.

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

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

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

If we want the consistent answers to the query $(P(x) \wedge R(x))$, for example, we run the repair program (r, IC) together with query program $Ans(X)$ $P(X, \mathbf{t}^{\star\star}), Q(X, \mathbf{t}^{\star\star})$, obtaining the answer $Ans = \emptyset$, as expected. With the query $Ans(X) \quad P(X, \mathbf{t}^{\star\star}), Q(X, \mathbf{f}^{\star\star})$, 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 $Ans(X) \quad P(X, \mathbf{t}^{\star\star})$ and $Ans(X) \quad R(X, \mathbf{t}^{\star\star})$, 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: $\quad R(X, Y)$, that $\mathit{Certain}_{\mathcal{G}_1}(Q) = \{(a, b), (c, d), (a, c),\ (d, e)\}$. Now assume that we have global functional dependency $FD\colon X \to Y$. 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 $V_1\colon\ X \to Y, \quad V_2\colon\ X \to Y$ 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 $\mathit{Mininst}(\mathcal{G})$ 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 $D \in \mathit{Mininst}(\mathcal{G})$ 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 $Q\colon Ans(X, Y) \quad R(X, Y)$, 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 $Q'\colon Ans(X, Y) \quad \neg R(X, Y)$. 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 $\mathit{Certain}_{\mathcal{G}}(Q)$ $\subseteqq \mathit{Minimal}_{\mathcal{G}}(Q)$; 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 IC a set of global ICs.
(a) A *repair* of $\mathcal{G}$ wrt to IC is a global instance that satisfies IC, and minimally differs from a minimal instance (wrt to inclusion of sets of tuples). We denote by $\mathit{Repairs}^{IC}(\mathcal{G})$ the set of repairs of $\mathcal{G}$ wrt IC.
(b) A ground tuple t is a *consistent answer* to a global query Q wrt IC if for every $D \in \mathit{Repairs}^{IC}(\mathcal{G})$, it holds $D \models Q[t]$, i.e. t is an answer to Q in D. We denote by $\mathit{Consis}^{IC}_{\mathcal{G}}(Q)$ set of consistent answers to Q. □

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

Now, for the query $Q\,:\ Ans(X, Y) \quad R(X, Y)$, we have $\mathit{Consis}^{FD}_{\mathcal{G}_1}(Q) =$ $\{(c, d), (d, e)\}$, as expected. For the existential query $Q''(X)\colon Ans(X) \quad R(X, Y)$, we have $\mathit{Consis}^{FD}_{\mathcal{G}_1}(Q'') = \{a, c, d\}$. 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 $\mathit{Consis}^{IC}_{\mathcal{G}}(Q) \subseteqq \mathit{Minimal}_{\mathcal{G}}(Q)$ [10]. If $\mathcal{G}$ is consistent wrt IC, then $\mathit{Consis}^{IC}_{\mathcal{G}}(Q) = \mathit{Minimal}_{\mathcal{G}}(Q)$. 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 $\forall x \forall y (x = a \wedge y = e \to R(x, y))$, saying that tuple (a, e) belongs to R. In this case, there is only

Page 66

one repair for $\mathcal{G}_1$, namely $D^3 = \{(a,e),(c,d),(d,e)\}$. Now, $\mathit{Consis}^{IC}_{\mathcal{G}_1}(Q) = \{(a,e),(c,d),(d,e)\}\ /\ \mathit{Minimal}_{\mathcal{G}_1}(Q)$. □

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

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

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

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

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

In our example, the stable models of $SV(\ \ (\mathcal{G}_2))$ 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 $\quad(\mathcal{G}_2)$ and the minimal instances of $\mathcal{G}_2$. □

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

1. Facts: $\quad dom(a)$ for every constant $a \in \mathcal{D}$, and $V_i(a)$ whenever $a \in v_i$ for a source extension v_i in $\mathcal{G}$.
2. For every view (source) predicate V_i with definition $V_i(X) \quad P_1(X_1),\ldots, P_n(X_n)$, the rule
$$P_j(X_j) \quad V_i(X), \bigwedge_{X_l \in (X_j \setminus X)} F_i^{X_l}(X, X_l).$$
3. For every predicate $F_i^{X_l}(X,X_l)$ introduced in 2., the rule
$$F_i^{X_l}(X,X_l) \quad V_i(X), dom(X_l), choice((X),(X_l)).$$

It can be proved [16] that

$$Mininst(\mathcal{G}) \quad \text{class of stable models of } SV(\ \ (\mathcal{G})) \quad Legal(\mathcal{G}). \tag{4}$$

Queries expressed as logic programs can be answered by running them together with $\quad(\mathcal{G})$ 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 $\quad(\mathcal{G})$ 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

$$\begin{aligned} V_1(X) &\quad P(X,Y); & v_1 &= \{a\} \\ V_2(X,Y) &\quad P(X,Y); & v_2 &= \{(a,c)\}. \end{aligned}$$

Here we have $Mininst(\mathcal{G}_3) = \{\{P(a,c)\}\}$, however, the legal global instances corresponding to stable models of $\quad(\mathcal{G}_3)$ are of the form $\quad\{\{P(a,c), P(a,z)\} \mid z \in \mathcal{D}\}$, 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

$$Ans \quad P(X,Y), Y \neq c \tag{5}$$
$$Ans(Y) \quad P(a,Y), Y \neq c. \tag{6}$$

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 $P(X, f(X)) \quad V_1(X)$ and $P(X,Y) \quad V_2(X,Y)$. 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 $Ans = true$, whereas query (6) gets the answer $Ans = \{f(a)\}$, which, after elimination of the f-value, becomes $Ans = \emptyset$.

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 $V_3(X,Y) \quad R(X,Y);\ v_3 = \{(a,c)\}$. 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

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

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 $\quad(\mathcal{G})$ to specify only the minimal instances, then the program has to be refined. The new version $\quad(\mathcal{G})$ detects in which cases it is necessary to use the

[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.

function predicates. This is achieved by means of a stronger condition, $add_{V_i}(X)$, in the choice rules, i.e. $F_i^{X_l}(X, X_l) \quad add_{V_i}(X), dom(X_l), choice((X),(X_l))$, where $add_{V_i}(X)$ 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.

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

Table 2. Semantic of Annotation Constants for Minimal Models

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

$$dom(a), dom(c), ..., V_1(a), V_2(a,c). \tag{9}$$
$$P(X,Y,\mathbf{v_1}) \quad add_{V_1}(X), F_1^Y(X,Y). \tag{10}$$
$$add_{V_1}(X) \quad V_1(X),\ not\ aux_{V_1}(X). \tag{11}$$
$$aux_{V_1}(X) \quad var_{V_1,Z}(X,Z). \tag{12}$$
$$var_{V_1,Z}(X,Z) \quad P(X,Z,\mathbf{nv_1}). \tag{13}$$
$$F_1^Y(X,Y) \quad add_{V_1}(X), dom(Y), choice((X),(Y)). \tag{14}$$
$$P(X,Y,\mathbf{o}) \quad V_2(X,Y). \tag{15}$$
$$P(X,Y,\mathbf{nv_1}) \quad P(X,Y,\mathbf{o}). \tag{16}$$
$$P(X,Y,\mathbf{t_d}) \quad P(X,Y,\mathbf{v_1}). \tag{17}$$
$$P(X,Y,\mathbf{t_d}) \quad P(X,Y,\mathbf{o}). \tag{18}$$

Rules (10) to (13) ensure that if there is an atom in source V_1, e.g. $V_1(a)$, and if an atom of the form $P(a,Y)$ was not added by view V_2, then it is added by rule (10) with a Y value given by the functional predicate $F_1^Y(a,Y)$. 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 $\{dom(a), dom(c), \ldots, V_1(a), V_2(a,c),$ $\underline{P(a,c,\mathbf{t_d})}$, $P(a,c,\mathbf{o})$, $P(a,c,\ \mathbf{nv_1})$, $aux_{V_1}(a)\}$, 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 $(\mathcal{G})$ 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 $(\mathcal{G})$ 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 $Q(X)$ into the first-order query $T^\omega(Q(X))$ using IC.[5]
2. Transform $T^\omega(Q(X))$ into a recursion-free Datalog$^\neg$ query program $(T^\omega($ $Q))$ (this is straightforward [64]).
3. Find a query plan, $Plan($ $(T^\omega(Q)))$ to answer the query $(T^\omega(Q))$ 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 $(T^\omega(Q))$ may contain negation, that is introduced at the first step. We give some examples.

Example 22. Consider the integration system

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

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.

[5] We are assuming here that $T^\omega(Q(X))$ 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.

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: $Ans(X,Y) \quad P(X,Y),\ not\ R(X,Y)$. □

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 $Q\colon R(X,Y)?$ 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

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

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

$$Ans(X,Y) \quad R(X,Y),\ not\ S(X,Y) \tag{22}$$
$$S(X,Y) \quad R(X,Z), dom(Y), Y \neq Z \tag{23}$$
$$dom(a), dom(b), dom(c), dom(d), dom(e), ... \tag{24}$$

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

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

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

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, $(T^\omega(Q))$ 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 $Ans(X,Y) \quad P(X,Y)$, *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 $Ans(X,Y) \quad P(X,Y,\mathbf{t_d}),\ not\ R(X,Y,\mathbf{t_d})$. □

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.

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, $(\mathcal{G}, IC)$, 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 $Mininst(\mathcal{G}_3) =$

[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.

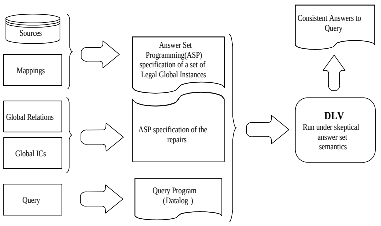

Sources
Mappings
Global Relations
Global ICs
Query
Answer Set Programming(ASP) specification of a set of Legal Global Instances
ASP specification of the repairs
Query Program (Datalog)
Consistent Answers to Query
DLV
Run under skeptical answer set semantics

Fig. 2. Computing Consistent Answers

$\{\{P(a,c)\}\}$, an the only instance does not satisfy *Sym*, the system is inconsistent.

The repair program, $(\mathcal{G}_3, Sym)$, consists of two layers. The first one is exactly program $(\mathcal{G}_3)$ 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:

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

The stable models of this program are:

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

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

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 $Q(x)$ 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 $Q(\quad P(u) \quad \neg R(v) \quad)$, we pass to $Q' := Q(\quad P(u, \mathbf{t}^{\star\star}) \quad R(v, \mathbf{f}^{\star\star}) \quad)$. Next, a query program (Q') with an $Ans(X)$ predicate is produced from Q (this is standard [64]). Finally, the program $:= \quad (Q') \cup \quad (\mathcal{G}, IC)$ is run under the stable model semantics; and the ground atoms $Ans(t) \in \bigcap\{S \mid S$ is a stable model of $\}$ 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 $Q': \quad P(X,Y,\mathbf{t}^{\star\star})$, which in its turn is transformed into the query program $(Q'):\ Ans(X,Y) \quad P(X,Y,\mathbf{t}^{\star\star})$. Next, we form $= \quad (\mathcal{G}_3, Sym) \cup \quad (Q')$, with $(\mathcal{G}_3, Sym)$ as in Example 25.

Now, the models of program are those of $(\mathcal{G}_3, Sym)$ 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:

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

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

Repair subprogram:

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

Query subprogram:

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

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 $V(X) \leftarrow \varphi_V(X)$ 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 $\mathcal{G}_4$:

$$V_1(X,Z) \leftarrow P(X,Y), R(Y,Z); \qquad v_1 = \{(a,b)\} \quad \mathit{open} \qquad (25)$$
$$V_2(X,Y) \leftarrow P(X,Y); \qquad v_2 = \{(a,c)\} \quad \mathit{clopen} \qquad (26)$$

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 $\leftarrow P_1(X_1), \dots, P_n(X_n),\ not\ V(X)$, for each closed (or clopen) source v with view definition $V(X) \leftarrow P_1(X_1), \dots, P_n(X_n)$. 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:

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

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

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 $\{dom(a), \dots, V_1(a,b), V_2(a,c), \underline{P(a,c)}, F_1^Y(a,b,c), \underline{R(c,b)}\}$, 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

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

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.

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 '$111\ldots11$'. 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.

Consistent Answers from Integrated Data Sources

Leopoldo Bertossi[1], **Jan Chomicki**[2]
Alvaro Cortés[3], and **Claudio Gutiérrez**[4]

[1] Carleton University, School of Computer Science, Ottawa, Canada. bertossi@scs.carleton.ca
[2] State University of New York at Buffalo, Dept. of Computer Science and Engineering. chomicki@cse.buffalo.edu
[3] Pontificia Universidad Catolica de Chile, Departamento de Ciencia de Computacion, Santiago, Chile. acortes@ing.puc.cl
[4] Universidad de Chile, Center for Web Research, Departamento de Ciencias de la Computación, Santiago, Chile. cgutierr@dcc.uchile.cl

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

1 Introduction

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

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

In order to perform this query transformation, the processor needs a mapping between the mediated schema and the information sources. Two general paradigms have been proposed to provide this mapping. One of them, called the *Local-as-View* (LAV) approach [15], considers each base information source as a view defined in terms of relations in the global schema. The *Global-as-View* (GAV) approach, considers each global predicate as a view defined in terms of the source relations [20, 22].[1]

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

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

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

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

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

$$conference(Paper, Conference)$$
$$year(Paper, Year)$$
$$location(Conference, Year, Location)$$

plus the functional dependencies (FDs):

$$conference: \ Paper \rightarrow Conference$$
$$year: \ Paper \rightarrow Year$$
$$location: \ Conference, Year \rightarrow Location$$

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

$$s_1(P, C, Y) \leftarrow conference(P, C), year(P, Y) \quad (1)$$
$$s_2(P, L) \leftarrow conference(P, C), year(P, Y), location(C, Y, L) \quad (2)$$

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

$$ans(L) \leftarrow s_1(P, pods, 1989), s_2(P, L).$$

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

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

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

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

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

$$V_1(X, Y) \leftarrow R(X, Y) \qquad V_2(X, Y) \leftarrow R(X, Y).$$

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

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

Example 3. (example 2 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 tuple $(c, d), (d, e)$ should be returned as answers that are consistent wrt the global FD. □

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

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

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

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

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

2 Preliminaries

2.1 Global schemas and view definitions

A global schema, $\mathcal{R}$, is modeled by a finite set of relations $\{R_1, R_2, ..., R_n\}$ and a possibly infinite domain $\mathcal{D}$. With these predicate symbols and the elements of $\mathcal{D}$ treated as constants, a first order language $\mathcal{L}(\mathcal{R})$ can be defined. This language can be extended with new defined predicates.

[1] It is also possible to specify this mapping using description logics [5].

[2] Even in the local-as-view approach where, from a theoretical perspective, sources are seen as views over the global relations: sources are still materialized, but global relations are virtual.

A view, denoted by a new global predicate V, can be defined by means of an $\mathcal{L}(\mathcal{R})$-formula of the form φ_V: $V(\bar{t}) \leftarrow \mathit{body}(\varphi_V)$, where $\bar{t}$ is a tuple containing variables and/or constants, and $\mathit{body}(\varphi_V)$ is a conjunction of global atoms. The formula φ_V is implicitly universally quantified. That is, a view is defined by a conjunctive query [1].

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

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

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

2.2 Global instances

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

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

$$\begin{aligned}\mathit{Linst}(\mathfrak{G}) = \{D \text{ instance over } \mathcal{R} \mid\ & v_i \subseteq \varphi_i(D) \text{ for all open sources } S_i \in \mathfrak{G},\\ & \text{and } v_j \supseteq \varphi_j(D) \text{ for all closed sources } S_j \in \mathfrak{G},\\ & \text{and } v_k = \varphi_k(D) \text{ for all clopen sources } S_k \in \mathfrak{G}\}.\end{aligned}$$

Here, v_i is the extension in the source S_i of the view defined by φ, $\varphi_i(D)$ is the set of tuples obtained by applying the view definition φ_i to instance D. □

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

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

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

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

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

Definition 2. *(a)* [2] *The ground tuple $\bar{a}$ is a certain answer to a query Q posed to a global system $\mathfrak{G}$ if for every instance $D \in \mathit{Linst}(\mathfrak{G})$, $\bar{a} \in Q(D)$, where $Q(D)$ is the answer set for Q in D.*
(b) We denote by $\mathit{Certain}_{\mathfrak{G}}(Q)$ the set of certain answers to query Q in $\mathfrak{G}$.

2.3 Short review of the inverse rules method

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

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

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

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

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

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

3 Global Systems and Consistency

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

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

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

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

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

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

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

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

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

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

This definition is particularly relevant in the case of open sources[3]. In this case, there is only one minimal instance if the intersection of the elements in $\mathit{Linst}(\mathfrak{G})$ is again an element of $\mathit{Linst}(\mathfrak{G})$.

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

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

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

$$S_1 = \langle V_1(X,Y) \leftarrow P(X,Z) \wedge Q(Z,Y), \{V_1(a,b)\}\rangle$$
$$S_2 = \langle V_2(X,Y) \leftarrow P(X,Y), \{V_2(a,c)\}\rangle.$$

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

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

Definition 5. *(a) The ground tuple $\bar{a}$ is a minimal answer to a query Q posed to a global system $\mathfrak{G}$ if for every minimal instance $D \in \mathit{mininst}(\mathfrak{G})$, $\bar{a} \in Q(D)$, where $Q(D)$ is the answer set for Q in D.*
(b) We denote by $\mathit{Minimal}_{\mathfrak{G}}(Q)$ the set of minimal answers to query Q in $\mathfrak{G}$.

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

[3] For closed sources, the only minimal instance is empty.

3.1 Repairs of global systems

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

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

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

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

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

Definition 7. *Let $\mathfrak{G}$ be a global system and IC a set of global ICs. A* repair *of $\mathfrak{G}$ wrt IC is a global database instance D', i.e. an instance over global schema $\mathcal{R}$, such that:*
(a) $D' \models IC$ and
(b) D' is $\leq_D$-minimal for some $D \in mininst(\mathfrak{G})$. □

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

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

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

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

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

[4] Notice from definition 6 that built-in predicates do not contribute to the Δs, because they have fixed extensions, identical in every database instance

Example 10. Consider the global system $\mathfrak{G}_5 = \{S_1, S_2\}$, with

$$S_1 = \langle V_1(X, Y) \leftarrow R(X, Y), \{V_1(a, b_1)\}\rangle,$$
$$S_2 = \langle V_2(X, Y) \leftarrow R(X, Y), \{V_2(a, b_2)\}\rangle,$$

and the FD $R(X, Y): X \rightarrow Y$. The only element in $mininst(\mathfrak{G}_5)$ is $D_0 = \{R(a, b_1), R(a, b_2)\}$, that does not satisfy IC. The global system is inconsistent. The only repairs are the global instances that minimally differ from D_0 and satisfy the FD, namely $D_0^1 = \{R(a, b_1)\}$ and $D_0^2 = \{R(a, b_2)\}$. Notice that they do not belong to $Linst(\mathfrak{G}_5)$. □

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

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

and the inclusion dependency IC: $\forall XYZ(R_1(X, Y, Z), Z > 0 \rightarrow R_3(X, Y))$. The elements in $mininst(\mathfrak{G}_6)$ are of the form

$$D_{c_1c_2c_3} = \{\ R_1(a, b, 1), R_1(e, f, -5), R_1(l, m, 3), R_2(a, c_1), R_2(e, c_2), R_2(l, c_3), R_3(a, b)\},$$

where c_1, c_2, c_3 are any elements in the underlying domain. They do not satisfy IC because $R_3(l, m) \notin D_{c_1c_2c_3}$. The global system is inconsistent. The next two global instances minimally differ from $D_{c_1c_2c_3}$ and satisfy IC, in consequence they are repairs of $D_{c_1c_2c_3}$:

$$D^1_{c_1c_2c_3} = \{R_1(a, b, 1), R_1(e, f, -5), R_2(a, c_1), R_2(e, c_2), R_2(l, c_3), R_3(a, b)\}$$
$$D^2_{c_1c_2c_3} = \{R_1(a, b, 1), R_1(e, f, -5), R_1(l, m, 3), R_2(a, c_1), R_2(e, c_2), R_2(l, c_3), R_3(a, b), R_3(l, m)\}$$ □

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

3.2 Consistent answers to global queries

There are algorithms in the literature for obtaining certain answers to a global query Q from a global system $\mathfrak{G}$. This system may violate desired, global integrity constraints IC. Our goal is to characterize those answers to Q that are consistent with IC, even when the global system is inconsistent as a whole. As in [3], we formalize this notion appealing to the repairs of the global system. Nevertheless, in this new scenario, we do not start form a single, possibly inconsistent relational databases instance, but from a possibly inconsistent global system, with a collection of implicit instances.

Definition 8. *(a) Given a global system $\mathfrak{G}$, 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, denoted by $\mathfrak{G} \models_c Q[\bar{t}]$, 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 query Q in $\mathfrak{G}$. □

Example 12. (example 10 continued) For the query $Q_1(X): \exists Y\ R(X, Y)$, a is a consistent answer, i.e. $\mathfrak{G}_5 \models_c \exists Y\ R(X, Y)[a]$. Instead, the query $Q_2(X, Y): R(X, Y)$, does not have any consistent answer. Nevertheless, a query plan for Q_2, e.g. found according to [10], without caring about inconsistency, would be:

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

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

Example 13. (example 11 continued) For the query $Q_1(X, Y, Z): R_1(X, Y, Z)$, $[a, b, 1]$ and $[e, f, -5]$ are consistent answers, i.e. $\mathfrak{G}_6 \models_c R_1(X, Y, Z)[a, b, 1]$ and $\mathfrak{G}_6 \models_c R_1(X, Y, Z)[e, f, -5]$. However the certain answers to query $Q_1(X, Y, Z)$ are $[a, b, 1]$, $[e, f, -5]$ and $[l, m, 3]$. □

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

4 Computing Consistent Answers

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

4.1 The rewriting operator for CQA

We consider *universal* first order integrity constraints expressed in the so-called "standard format" [3]: $\forall(\bigvee_{i=1}^{m} l_i(\bar{x}_i) \vee \varphi)$, where l_i is a database literal and φ is a formula containing built-in predicates only.

The new query $T^\omega(\varphi(\bar{x}))$ is computed via the iterative T operator, which transforms an arbitrary query by appending the corresponding *residue* to each database literal appearing in the query until a fixed point is reached. The residue of a databas literal forces the local satisfaction of the ICs for the tuples satisfying the literal. Residues can be obtained by resolution between the table and the ICs.

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

$$T^1(\varphi(x)) = Q(x) \wedge (R(x) \vee \neg P(x)) \wedge P(x).$$
$$T^2(\varphi(x)) = T(T^1(\varphi(x))) = Q(x) \wedge (T(R(x)) \vee T(\neg P(x))) \wedge T(P(x)).$$
$$= Q(x) \wedge (R(x) \vee (\neg P(x) \wedge \neg Q(x))) \wedge P(x) \wedge (R(x) \vee \neg Q(x)).$$
$$T^3(\varphi(x)) = Q(x) \wedge (R(x) \vee (\neg P(x) \wedge T(\neg Q(x)))) \wedge P(x) \wedge (T(R(x)) \vee T(\neg Q(x))).$$

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

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

4.2 Computing consistent answer to global queries

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

Algorithm:

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

1. Rewrite $Q(\bar{X})$ into the first order query $T^\omega(Q(\bar{X}))$ applying the algorithm presented in 4.1 using IC as the input set of ICs.

2. Using a standard methodology [1, 16], transform $T^{\omega}(Q(\bar{X}))$ into a Datalog$^{\neg}$ program, i.e. a Datalog program with negation, $\Pi(T^{\omega}(Q))$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$T^{\omega}(Q(X,Y)):\ R_1(X,Y) \wedge R_2(X,Y).$$

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

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

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

$$Ans(X,Y) \leftarrow R_1(X,Y), R_2(X,Y)$$

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

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

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

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

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

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

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

Here, the only element in $mininst(\mathfrak{G}_8)$ is $D_0 = \{R(a,b), R(a,c), R(c,d)\}$. The only minimal instance violates FD through the tuples $[a,b]$, $[a,c]$. In consequence, the global system $\mathfrak{G}_8$ is inconsistent. It has two repairs, $D_0^1 = \{R(a,b), R(c,d)\}$ and $D_0^2 = \{R(a,c), R(c,d)\}$. The only consistent answer to the query is the tuple $[c,d]$.

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

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

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

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

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

We now apply the Algorithm for CQA. First of all, we need the FD expressed in a first order language, it becomes $\forall XYZ(R_1(X,Y) \wedge R_1(X,Z) \rightarrow Y = Z)$.

At step 1. the query has to be rewritten. We obtain

$$T^{\omega}(Q(\bar{X})):\ R(X,Y) \wedge \neg\exists Z(R(X,Z) \wedge Z \neq Y). \tag{3}$$

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

$$Ans(X,Y) \leftarrow R(X,Y),\ not\ S(X,Y)$$
$$S(X,Y) \leftarrow R(X,Z), Y \neq Z.$$

We need a query plan to answer this query program containing negation. □

5 Plans for Queries with Negation

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

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

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

Example 17. Consider the global system $\mathfrak{G}_9 = \{S_1, S_2\}$ with

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

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

The global query Q is:

$$Ans(X,Z) \leftarrow R_1(X,Y), R_2(Y,Z),\ not\ R_4(X,Y)$$
$$R_4(X,Y) \leftarrow R_3(X,Y),\ not\ R_5(X,Y) \tag{4}$$
$$R_7(X) \leftarrow R_1(X,Y), R_6(X,Y). \tag{5}$$

The inverses rules $\mathcal{V}^{-1}$ are obtained from the source descriptions:

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

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

$$Ans(X,Z) \leftarrow R_1(X,Y), R_2(Y,Z),\ not\ R_4(X,Y)$$
$$R_4(X,Y) \leftarrow R_3(X,Y).$$

The literal $notR_5(X,Y)$ was eliminated from rule (4) because it does not appear in any source description. For the same reason (the literal $R_6(X,Y)$ does not appear in any source description), rule (5) was eliminated. Then, the query plan $P : (Q^{-}, V^{-1})$ is:

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

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

5.1 Containment

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

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

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

$$Ans(X,Z) \leftarrow P_1(X,Y), P_2(Y,Z),\ not\ P_4(X,Y)$$
$$P_4(X,Y) \leftarrow P_3(X,Y)$$
$$P_1(X, f_1(X,Z)) \leftarrow V_1(X,Z) \tag{6}$$
$$P_2(f_1(X,Z), Z) \leftarrow V_1(X,Z) \tag{7}$$
$$P_3(X,Y) \leftarrow V_2(X,Y). \tag{8}$$

[5] It should be easy to extend this methodology to stratified Datalog$^{\neg}$ queries, but we do not need this extension here.

The expansion is the following Datalog$^\neg$ query P^{exp}:

$$\begin{aligned} Ans(X,Z) &\leftarrow P_1(X,Y), P_2(Y,Z),\ \mathit{not}\ P_4(X,Y)\\ P_4(X,Y) &\leftarrow P_3(X,Y)\\ P_1(X,f_1(X,Z)) &\leftarrow R_1(X,U), R_2(U,Z)\\ P_2(f_1(X,Z),Z) &\leftarrow R_1(X,U), R_2(U,Z)\\ P_3(X,Y) &\leftarrow R_3(X,Y). \qquad \Box \end{aligned}$$

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

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

Definition 9. *(maximally-contained plan* [17]*) A query plan P is maximally contained in a query Q using views $\mathcal{V}$ if $P^{exp} \subseteq Q$ and for every query plan P' such that $(P')^{exp} \subseteq Q$, it holds $P' \subseteq P$.*[6] □

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

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

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

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

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

[6] Here, $P^{exp} \subseteq Q$ means that the extension of the query predicate in P^{exp} in included in the extension of Q for every database instance over the global schema.

6 The CQA Algorithm Revisited

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

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

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

$$Ans'(X,Y) \leftarrow R(X,Y),\ \mathit{not}\ S(X,Y)$$

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

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

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

Theorem 4. *Given an open global system $\mathfrak{G}$, IC a set of general ICs, and a conjuntive global query Q, Plan$(\Pi(T^\omega(Q))$ retrieves exactly Consis$_{\mathfrak{G}}(Q)$.* □

7 Conclusions

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

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

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

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

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

References

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. 9th Annual ACM Symp. on the Theory of Computing*, ACM Press, 1998, pp. 254-263.
3. Arenas, M.; Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. In *Proc. ACM Symposium on Principles of Database Systems (ACM PODS'99)*, ACM Press, 1999, pp. 68–79.
4. 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.L. arsen, J. Kacprzyk, S. Zadrozny, H. Christiansen (eds.), Springer, 2000, pp. 27–41.
5. Beeri, C.; Levy, A. and Rousset, M. Rewriting Queries using Views in Description Logics. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'97)*, 1997, pp.
6. Bertossi, L. and Celle, A. Querying Inconsistent Databases: Algorithms and Implementation. In *Computational Logic - CL 2000*, J. Lloyd et al. (eds.)., Springer LNAI 1861, 2000, pp. 942–956.
7. Barcelo, P. and Bertossi, L. Repairing Databases with Annotated Predicate Logic. In *Proc. Ninth International Workshop on Non-Monotonic Reasoning (NMR'2002). Special session on Changing and Integrating Information: From Theory to Practice.* S. Benferhat and E. Giunchiglia (eds.), Morgan Kaufmann Publishers, 2002, pp. 160 – 170.
8. Cali, A.; Calvanese, D.; De Giacomo, G. and Lenzerini, M. Data integration under Integrity Constraints. In *Proc. of the 14th Conf. on Advanced Information Systems Engineering (CAiSE 2002)*, 2002. To appear.
9. Duschka, O.; Genesereth, M. and Levy, A. Recursive Query Plans for Data Integration. *Journal of Logic Programming*, 2000, 43(1):49-73.
10. Grahne, G. and Mendelzon, A. Tableau Techniques for Quering Information Sources through Global Schemas. In *Proc. of the Int. Conf. on Database Theory (ICDT'99)*, Springer LNCS1540, 1999, pp. 332-347.
11. Grant, J. and Minker, M. A Logic-based Approach to Data Integration. *Theory and Practice of Logic Programming journal.* To appear.
12. Greco, G.; Greco, S. and Zumpano, E. A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. In *Proc. 17th International Conference on Logic Programming (ICLP'01)*, Ph. Codognet (ed.), LNCS 2237, Springer, 2001, pp. 348–364.
13. Gryz, J. Query Rewriting Using Views in the Presence of Functional and Inclusion Dependencies. *Information Systems*, 1999, 24(7):597–612.
14. Halevy, A. Answering Queries using Views: A Survey. *VLDB Journal.* To appear.
15. Levy, A.; Rajaraman, A. and Ordille, J. Quering Heterogeneous Information Sources using Source-Descriptions. In *Proceedings of the 22nd International Conference on Very Large Databases (VLDB'96)*, Morgan Kaufmann Publishing Co., 1996, pp. 251–262.
16. Lloyd, J. *Foundations of Logic Programming.* Springer, 1987.
17. Millstein, T.; Levy, A. and Friedman, M. Query Containment for data Integration Systems. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'00)*, 2000, pp.

18. Pottinger, R. and Levy, A. A Scalable Algorithm for Answering Queries Using Views. In *Proceedings of the 26th VLDB Conference*, 2000, pp.
19. Ramakrishnan, R.; Sagiv, Y.; Ullman, J.D. and Vardi, M.Y. Proof-tree Transformation Theorems and their Applications. In *Proceedings ACM SIGACT-SIGMOD-SIGART Symposium on Princ. of DBS*,1989, pp. 172–181.
20. Ullman, J. Information Integration using Logical Views. In *Proc. of the Int. Conf. on Database Theory (ICDT'97)*, Springer LNCS1186, 1997, pp. 19–40.
21. Wang, J.; Maher, M. and Topor, R. Rewriting General Conjunctive Queries Using Views. In *Proc. Australasian Database Conference 2002*. To appear
22. Ullman, J.D. Information Integration Using Logical Views. *Theoretical Computer Science*, 2000, 239(2):189-210.

Appendix A: Proof trees

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

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

1. *The label of the root is*
   (a) (d, A) if $C = A$, with a positive A
   (b) $(-d, A)$ if $C = \neg A$, with a positive A.
2. *Each leaf is labeled by either:*
   (a) $(+1, A)$, where A is a fact in I, or
   (b) $(-1, B)$, where B is not a fact in I and B does not appear in the head of any ground instantiation of a rule in Π.
3. *For each internal vertex with label $(+1, A)$, there exists a ground instantiation $A \leftarrow A_1, \ldots, A_n$ of a program rule in Π and for each A_j there is a child which is the root of a $(+1)$-proof-tree of A_j.*
4. *For each internal vertex with label $(-1, B)$, it holds that for each ground instantiation $B \leftarrow B_1, \ldots, B_n$ of a program rule, there is some j and a child which is the root of a (-1)-proof-tree of B_j.*

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

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

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

Appendix B: Proofs

Proof of Proposition 1 (a) Let $\bar{t} \in Consis_{\mathfrak{G}}(Q)$. Then for every repair D of $\mathfrak{G}$, $D \models Q(\bar{t})$. By the hypothesis on IC, $D \models Q(\bar{t})$ for every $D \in mininst(\mathfrak{G})$, then $\bar{t} \in Minimal_{\mathfrak{G}}(Q)$.
(b) In this case, the repairs coincide with the elements of $mininst(\mathfrak{G})$, and then $Minimal_{\mathfrak{G}}(Q) = Consis_{\mathfrak{G}}(Q)$. □

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

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

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

If there is no occurrence of a view predicate V [7], then this is a proof tree from Q and D, and we are done: $Ans(\bar{t}) \in Q(D)$. If some V occurs in T, due to the form of the inverse rules, it must occur as an internal node with children $p_1, \ldots p_n$ that are ground facts in D, in particular, without functions. Due to the form of the inverse rules, its parent must be one of the atoms p_i. Notice that there is at most one occurrence of predicate V in a branch. Then, for each such positive occurrence of a view predicate V we can prune the subtree with root V. In case of a negative occurrence of $V(\bar{a})$ ($\neg V(\bar{a})$, or equivalently, $(-, V(\bar{a}))$, again by the form of the rules, the parent must be of the form $(-, P(\bar{a})$, with P a global predicate. In this case we can prune the subtree with root $(-, V)$, because if we developed the node $(-, P)$ with ist child $(-, V)$, that is because $P(\bar{a})$ was not in D. Pruning all V nodes in this way, leaves us with a proof tree for $Ans(\bar{t})$, but now from Q and D.

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

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

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

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

Proof of Proposition 2: Let $\bar{t} \in Consis_{\mathfrak{G}}(Q)$. Then, for every $D \in mininst(\mathfrak{G})$ and every repair D' of D: $\bar{t} \in Q(D')$, with $Q(D')$ the answer set to Q in D'. Then, by the results in [3], for every $D \in mininst(\mathfrak{G})$, $\bar{t} \in T^{\omega}(Q)(D)$, that is, $\bar{t} \in Minimal_{\mathfrak{G}}(T^{\omega}(Q))$. Due to the correctness and completeness of the T^{ω} operator for this kind of queries [3], the other inclusion can be established similarly. □

Proof of Theorem 4: From theorem 3 and proposition 2. □

Appendix B: Other Sources

Example 20. Consider now the global system $\mathfrak{G}_2 = \{S_1\}$, with

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

and IC, the functional dependency $R(X,Y) : X \rightarrow Y$. The only instance in $Linst(\mathfrak{G}_2)$ is $D = \{R(a,b), R(c,d)\}$. Since D satisfies IC, $\mathfrak{G}_2$ would be consistent wrt IC. □

Example 21. (adapted from [10]) Consider the global system $\mathfrak{G}_7 = \{S_1, S_2, S_3\}$ with

$$\begin{aligned} S_1 &= \langle V_1(U,W) \leftarrow S(U,W),\ open, \{V_1(b,c), V_1(e,f)\}\rangle \\ S_2 &= \langle V_2(U) \leftarrow R(U,X),\ open, \{V_2(a)\}\rangle \\ S_3 &= \langle V_3(Z) \leftarrow R(a,Z),\ closed, \{V_3(b), V_3(b')\}\rangle, \end{aligned}$$

and IC, the inclusion dependency $S(X,U) \rightarrow R(Z,X)$. The elements in $mininst($ $\mathfrak{G}_7)$ are of the form $D_d = \{S(b,c), S(e,f), R(a,d)\}$, where d is any element in the underlying domain. All of them violate IC[8], so the global system is inconsistent.

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

[7] Assuming we kept the view predicates in the expanded plan as indicated in remark 2.

[8] We are applying the *Unique Names Assumption*, that tells us that $b \neq e$, so cannot make d to be equal to both b and e.

D_e gives rise to the repairs $D_e^v = \{S(b,c), S(e,f), R(a,e), R(v,b)\}$, and $D'_e = \{S(e,f), R(a,e)\}$. Otherwise, D_d, for $d \neq b, e$, gives rise to the repairs $D_d^{u,v} = \{S(b,c), S(e,f), R(a,d), R(u,e), R(v,b)$ and $D'_d = \{R(a,d)\}$. □

Deductive Databases for Computing Certain and Consistent Answers from Mediated Data Integration Systems

Loreto Bravo and **Leopoldo Bertossi**
Carleton University
School of Computer Science
Ottawa, Canada.
{lbravo,bertossi}@scs.carleton.ca

Abstract. We address the problem of retrieving certain and consistent answers to queries posed to a mediated data integration system under the local-as-view paradigm with open sources and conjunctive and disjunctive view definitions. For obtaining certain answers a query program is run under the cautious stable model semantics on top of a normal deductive database with *choice* operator that specifies the class of minimal legal instances of the integration system. This methodology works for all monotone Datalog queries. To compute answers to queries that are consistent wrt given global integrity constraints, the specification of minimal legal instances is combined with another disjunctive deductive database that specifies the repairs of those legal instances. This allows to retrieve the answers to any Datalog$^\neg$ query that are consistent wrt global universal and referential integrity constraints.

1 Introduction

Usually independent and autonomous data sources are virtually integrated by means of a mediator, which is a program that 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 to be given back to the user.

The "Local-As-View" (LAV) approach to virtual data integration requires that each data source is described as a set of views over the global schema. On the other side, the "Global-As-View" (GAV) approach, defines every global relation as a view of the set of relations in the sources (see [33] for a survey on these and mixed approaches). Query answering is harder under LAV [2]. On the other side, LAV offers more flexibility to accept or release sources into/from an existing system.

In these virtual integration setting, inconsistencies wrt to global integrity constraints (ICs), i.e. that refer to the relations at the virtual level, are likely to occur. This is due to the autonomy of the participating sources, the lack of a central maintenance mechanism; and also to the flexibility to add or delete sources, without having to consider the other sources in the system.

Example 1. Consider the LAV based global integration system $\mathcal{G}_1$ 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 $\{(a, b), (a, c)\}$. □

Inconsistencies are not exclusive to integration systems. For several reasons also single databases may become inconsistent wrt certain ICs. Restoring consistency may be undesirable, difficult or impossible [10]. In such a situation, possibly most of the data is still consistent and can be retrieved when queries are posed to the database. In [3] consistent data in a stand-alone relational database is characterized as the data that is invariant under all minimal restorations of consistency, i.e. as data that is present in all repaired versions of the original instance (the *repairs*). In particular, an answer to a query is defined as consistent when it can be obtained as a standard answer to the query from every possible repair.

In [3, 17, 30, 4, 5], some mechanisms have been developed for consistent query answering (CQA), i.e. for retrieving consistent answer when queries are posed to such an inconsistent database. All those mechanisms, in different degrees, work only with the original, inconsistent database, without restoring its consistency. That is, inconsistencies are solved at query time. The above mentioned repairs provide an auxiliary concept that allows defining the right semantics for consistent query answers. Furthermore, in some of the query evaluation methodologies, repairs are also an auxiliary computational intermediate step that, for complexity reasons, has to be kept to a minimum.

In virtual data integration systems, there is also an intuitive notion of consistent answer to a query.

Example 2. (example 1 continued) If we pose to the global system the query $Q\colon Ans(X, Y) \leftarrow R(X, Y)$, 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 $R\colon X \rightarrow Y$. □

Several algorithms for deriving query plans to obtain query answers from virtual data integration systems have been proposed in the last few years (see [36] for a survey). However they are not designed for obtaining the consistent answers to queries. Even more, some of those algorithms assume that certain ICs hold at the global level [31, 21, 29]; what may not be a realistic assumption due to the independence of the different data sources and the lack of a central, global maintenance mechanism. Only a few exceptions, including this paper, consider the problem of CQA in virtual integration systems [32, 9, 13, 16].

In a virtual data integration system, the mediator should solve potential inconsistencies when the query plan is generated; again without attempting to bring the whole system into a global consistent material state. Such an enhanced query plan generator should produce query plans that are guaranteed to retrieve all and only the consistent answers to global queries.

In this spirit and under the LAV approach, in [9] 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. This method uses the query rewriting approach to CQA presented in [3]; and in consequence inherits its limitations in terms of the queries and ICs that it can handle, actually queries that are conjunctions of tables and universal ICs. Once the query is transformed, query plans are generated for the new query. However, [9] provides the right semantics for CQA in mediated integrated systems (see Section 2).

In this paper, under the LAV approach and assuming that sources are open (or incomplete) [2], we solve the problem of retrieving consistent answers to global queries. We consider arbitrary universal ICs and referential ICs; that is, the ICs that are most used in database praxis [1]. View definitions are conjunctive queries, and disjunctions thereof. Global queries are expressed in Datalog and its extensions with negation.

The methodology can be summarized as follows. In a first stage, we specify, using a deductive database with *choice operator* [25] and stable model semantics [24], the class of all minimal legal global instances of a virtual integration system. This approach is inspired by the inverse-rules algorithm [21] and uses auxiliary Skolem predicates whose functionality is enforced with the choice operator.

In order to obtain answers to global queries from the integration system, a query program has to be combined with the deductive database that specifies the minimal instances as its stable models, and then be run under the skeptical stable model semantics. It turns out that *minimal answers*, i.e. answers that are true in all minimal instances, can be retrieved for *Datalog*$^\neg$ queries. The *certain answers*, i.e. those true in all legal global instances, can be obtained for all monotone queries, a result that generalizes those found so far in the literature.

In a second stage, we address the computation of consistent answers. We first observe that an integration system is consistent if all of its minimal legal instances satisfy the integrity constraints [9]. Consistent answers from an inconsistent integration systems are those that can be obtained from all the repairs of all the minimal legal instances wrt the global ICs [3, 9]. In consequence, in order to retrieve consistent answers, the specification of the minimal instances has to be combined with a specification of their repairs wrt to given ICs. The latter is a disjunctive deductive database that specifies the repairs as its stable models; and uses annotation constants as in the case of repairs of single relational databases [3] as presented in [6, 5]. We have experimented with this query answering mechanism (and the computation of minimal instances and their repairs) with the *DLV* system [22, 35], which implements the stable model and answer set semantics of disjunctive extended deductive databases.

The paper is structured as follows. In Section 2 we review some basic notions we need in the rest of this paper. 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. In Section 4, the repairs of the minimal global instances are specified as the stable models of disjunctive logic programs with annotation constants, like those used to specify repairs of single relational

databases for CQA [6]. In Section 5, consistent answers to queries are obtained by running a query program in combination with the previous two specification programs. In Section 6 several issues and possible extensions around the specification presented in the previous sections are discussed in detail. Finally, in Section 7, we draw some final conclusions, and we point to related and future work. Appendix A.1 contains the proofs of the main results in this paper.

This paper is an extended version of [13] that now includes the most general specification of minimal instances, the proofs, an extension to disjunctive view definitions, and an analysis of: complexity, the underlying assumptions about the domain, a comparison between the use of the choice operator and the use of Skolem functions.

2 Preliminaries

2.1 Global schemas and view definitions

A *global schema* $\mathcal{R}$ consists of a finite set of relations $\{R_1, R_2, ..., R_m\}$ over a fixed, possibly infinite 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 defined and built-in predicates, like (in)equality. In particular, we will extend the global schema with a *local schema* $\mathcal{S}$, i.e. a finite set of new view predicates $V_1, V_2, ..., V_n$, that will be used to describe the relations in the local sources.

A *view*, denoted by a new predicate V, is defined by means of conjunctive query [1], i.e. an $\mathcal{L}(\mathcal{R} \cup \mathcal{S})$-formula $\varphi_{_V}$ of the form $V(\bar{t}) \leftarrow body(\varphi_{_V})$, where $\bar{t}$ is a tuple containing variables and/or constants, and $body(\varphi_{_V})$ is a conjunction of $\mathcal{R}$-atoms. In general, $V \in \mathcal{S}$.

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 . An instance D satisfies , denoted $D \models$, if is true in D.

Given a database instance D over schema $\mathcal{R}$, and a view definition $\varphi_{_V}$, $\varphi_{_V}(D)$ denotes the extension of V obtained by applying the definition $\varphi_{_V}$ to D. 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 $\varphi_{_V}(D)$; i.e. $v \subseteq \varphi_{_V}(D)$, and we say the view extension v is *open* wrt D [2]. Most mechanisms for deriving query plans assume that sources are open, e.g. [21].

A *source* S is a pair $\langle \varphi, v \rangle$, where φ is the view definition, and v is an extension for the view defined by φ. An *open global system* $\mathcal{G}$ 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*, which consists of all the constants appearing in the view extensions v_i of the sources, and in their definitions. When considering global integrity constraints the *active domain* also includes the constants in them. A global system $\mathcal{G}$ defines a set of legal global instances [33].

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

Example 3. (example 2 continued) Let us denote by φ_1, φ_2 the view definitions of V_1, V_2, resp. in $\mathcal{G}_1$. $D = \{R(a,\, b),\, R(c,\, d),\, R(a,\, c),\, R(d,\, e)\}$ is a legal global instance, because $v_1 = \{V_1(a,\, b),\, V_1(c,\, d)\} \quad \varphi_1(D) = \{V_1(a,\, b),\, V_1(c,\, d),\, V_1(a,\, c),\, V_1(d,\, e)\}$ and $v_2 = \{V_2(a,\, c),\, V_2(d,\, e)\} \quad \varphi_2(D) = \{V_2(a,\, b),\, V_2(c,\, d),\, V_2(a,\, c),\, V_2(d,\, e)\}$. Supersets of D are also legal instances; but proper subsets are not. □

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. [2] Given an open global system $\mathcal{G}$ and a global query $Q(X) \in \mathcal{L}(\mathcal{R})$, a ground tuple t is a *certain answer* to Q in $\mathcal{G}$ if for every global instance $D \in\ Linst(\mathcal{G})$, it holds $D \models Q[t]$.[1] We denote with $Certain_{\mathcal{G}}(Q)$ the set of certain answers to Q in $\mathcal{G}$. □

The inverse-rules algorithm [21] 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: $V(X) \quad P_1(X_1), \ldots, P_n(X_n)$, with $X \quad \bigcup_i X_i$. Since the queries posed to the system are expressed in terms of the global relations, that now appear in the bodies of the view definitions (contrary to the GAV approach), those definitions cannot be directly applied. The rules need to be "inverted".

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

Example 4. Consider the integration system $\mathcal{G}_2$ with global schema $\mathcal{R} = \{P, R\}$. The set $\mathcal{V}$ of local view definitions consists of $V_1(X,Z) \quad P(X,Y),\ R(Y,Z)$, and $V_2(X,Y) \quad P(X,Y)$. The set $\mathcal{V}^{-1}$ consists of the rules $P(X,\ f(X,Z)) \quad V_1(X,Z);\ R(f(X,Z),Z) \quad V_1(X,Z);$ and $P(X,Y) \quad V_2(X,Y)$.

For a view definition, we need as many Skolem functions as existential variables in it. For example, if instead of $V_1(X,Z) \quad P(X,Y),\ R(Y,Z)$ we had, say $V_1(X,Z) \quad P(X,Y),\ R(Y,Z,W)$, we would need two Skolem functions for that view, and the inverse rules arising from that view would be $P(X,\ f(X,Z)) \quad V_1(X,Z)$ and $R(f(X,Z),Z,g(X,Z)) \quad V_1(X,Z)$. □

[1] $D \models Q[t]$ means that query $Q(X)$ becomes true in instance D, when tuple of variables X is assigned the values in the tuple t of database elements.

The inverse rules are then used to answer Datalog queries expressed in terms of the global relations, that now, through the inverse rules, have definitions in terms of the sources. The query plan obtained with the inverse rule algorithm is maximally contained in the query [21], and the answers it produces coincide with the certain answers [2].

2.2 Global systems and consistency

We assume that we have a set of global integrity constraints $IC \quad \mathcal{L}(\mathcal{R})$ that is consistent as a set of logical sentences, and *generic*, in the sense that it does not entail any ground database literal by itself, i.e independently of concrete instance [10]. ICs used in database praxis are always generic. The ICs can be universal, i.e. a sentence of the form $\overline{\forall}\varphi$, where $\overline{\forall}$ is a prefix of universal quantifiers and φ a quantifier-free formula; or referential, i.e. of the form

$$\forall X(P(X) \rightarrow \exists Y Q(X', Y)), \quad X' \quad X.^2 \tag{1}$$

Definition 3. [9] (a) Given a global system $\mathcal{G}$, an instance D is *minimal* if $D \in Linst(\mathcal{G})$ and is minimal wrt set inclusion, i.e. there is no other instance in $Linst(\mathcal{G})$ that is a proper subset of D (as a set of atoms). We denote by $Mininst(\mathcal{G})$ the set of minimal legal global instances of $\mathcal{G}$ wrt set inclusion.
(b) A global system $\mathcal{G}$ is *consistent* wrt IC, if for all $D \in Mininst(\mathcal{G}),\ D \models IC$. □

Example 5. (example 4 continued) Assume that $\mathcal{G}_2$ has the source contents $v_1 = \{V_1(a,b)\}$, $v_2 = \{V_2(a,c)\}$, and that $\mathcal{U} = \{a, b, c, u, ...\}$. Then, the elements of $Mininst(\mathcal{G}_2)$ are of the form $D_z = \ \{P(a,z), R(z,b),\ P(a,c)\}$ for some $z \in \mathcal{U}$. The global FD $\ P(X,Y) \colon X \rightarrow Y$ is violated exactly in those minimal legal instances D_z for which $z \neq c$. Thus, $\mathcal{G}_2$ is inconsistent. □

Definition 4. [9] The ground tuple a is a *minimal answer* to a query Q posed to $\mathcal{G}$ if for every $D \in Mininst(\mathcal{G})$, $a \in Q(D)$, where $Q(D)$ is the answer set for Q in D. The set of minimal answers is denoted by $Minimal_{\mathcal{G}}(Q)$. □

Clearly $Certain_{\mathcal{G}}(Q) \quad Minimal_{\mathcal{G}}(Q)$. For monotone queries [1], the two notions coincide [9]. Nevertheless, in Example 5 the query $Ans(X,Y) \quad \neg P(X,Y)$ 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.

Definition 5. [3] (a) Given a database instance D, we denote by $\quad (D)$ the set of ground atomic formulas $\{P(a) \mid P \in \mathcal{R}$ and $D \models P(a)\}$.
(b) Let D, D' be database instances over the same schema and domain. The *distance*, $\quad (D, D')$, between D and D' is the symmetric difference $\quad (D, D') = (\quad (D) \setminus \quad (D')) \cup (\quad (D') \setminus \quad (D))$. □

[2] To keep the presentation simple, Y is a single variable, however it could be a tuple of variables, actually interleaved with those in X'.

We may assume that the original data sources and the global legal instances do not contain null values, however when dealing with referential integrity constraints (RICs), we will consider the possibility of having them, in order to restore the consistency of the database. If no RICs are present, we will assume that null values are not available either. However, if necessary, the null value *null* will be treated as a new, special constant. Its presence in a tuple means that there is an unknown value for the correspondent attribute, i.e. we have incomplete information. Since we do not have precise information about it, we will consider that no inconsistencies arise due to its presence. This leads to the following definition of consistency in the presence of null values:

Definition 6. [6] For a database instance D, whose domain $\mathcal{U}$ may contain the constant *null* and a set of integrity constraints $IC = IC_U \cup IC_R$, where IC_U is a set of universal integrity constraints and IC_R is a set of referential integrity constraints, we say that D satisfies IC, written $D \models IC$, iff:

1. For each $\overline{\forall}\varphi \in IC_U,\ D \models \varphi[a]$ for every ground tuple a of elements in $(\mathcal{U} \quad \{null\})$, and
2. For each sentence in IC_R of the form (1), if $D \models P[a]$, with a a ground tuple of elements in $(\mathcal{U} \quad \{null\})$, then $D \models \exists Y Q(a, Y)$. □

Example 6. Consider the universal IC $\forall xy(P(x,y) \rightarrow R(x,y))$ and the referential IC $\forall x(T(x) \rightarrow \exists y P(x,y))$. The database instance $D = \{P(a,d),\ R(a,d),\ T(a),\ T(b),\ P(b, null)\}$ is consistent. The universal constraint is satisfied even in the presence of $P(b, null)$ since the incomplete information cannot generate inconsistencies. □

Definition 7. [6] Let D, D', D'' be database instances over the same schema and domain $\mathcal{U}$. It holds $D' \quad_D D''$ iff:

1. For every atom $P(a) \in \quad (D, D')$, with $a \in (\mathcal{U} \quad \{null\})$,[3] it holds $P(a) \in \quad (D, D'')$, and
2. For every atom $Q(a, null) \in \quad (D, D')$, it holds $Q(a, null) \in \quad (D,\ D'')$ or $Q(a,b) \in \quad (D, D'')$ with $b \in (\mathcal{U} \quad \{null\})$. □

Definition 7 defines which databases are closer to the original one in the presence of *null* values. This partial order is used in the next definition for repairs in the presence of universal and referential ICs.

Definition 8. (based on [3]) Let $\mathcal{G}$ be a global system and IC a set of global ICs. A *repair* of $\mathcal{G}$ wrt IC is a global database instance D', such that $\ D' \models IC$ and D' is $\quad_D$-minimal for some $D \in Mininst(\mathcal{G})$. □

According to this definition the repairs of violations of referential ICs are obtained by either deleting the atom that is generating the inconsistency or by adding an atom with a *null* value. In particular, if the instance D is $\{P(a)\}$ and IC contains only $\forall x(P(x) \rightarrow \exists y Q(x,y))$, then $\{P(a), Q(a, null)\}$ will be a repair, but not $\{P(a), Q(a,b)\}$, with $b \in \mathcal{U}$ and $b \neq null$. In the absence of *null* values, i.e. without null values in the original instance nor in the repair process, Definitions 7 and 8 coincide with the ones given in [3]. In [4, 5, 15] repairs with non *null* values have been considered.

[3] That $a \in (\mathcal{U} - \{null\})$ means that each of the elements in tuple a belongs to $(\mathcal{U} - \{null\})$.

Example 7. Consider the universal integrity constraint $\forall xy(P(x,y) \rightarrow R(x,y))$ together with the referential integrity constraint $\forall x(T(x) \rightarrow \exists y P(x,y))$ and an inconsistent minimal instance of an integration system $D = \{P(a,b), T(c)\}$. The repairs for the latter are:

i	D_i	(D, D_i)
1	$\{P(a,b), R(a,b), T(c), P(c, null)\}$	$\{R(a,b), P(c, null)\}$
2	$\{P(a,b), R(a,b)\}$	$\{T(c), R(a,b)\}$
3	$\{T(c), P(c, null)\}$	$\{P(a,b), P(c, null)\}$
4	$\emptyset$	$\{P(a,b), T(c)\}$

In the first repair it can be seen that the atom $P(c, null)$ does not propagate through the universal constraint to $R(c, null)$. We also have that the instance $D_5 = \{P(a,b), R(a,b),\ T(c), P(c,a)\}$, where we have introduced $P(c,a)$ in order to satisfy the referential IC, does satisfy IC, but is not a repair because $\quad (D, D_1) \quad_D \quad (D, D_7) = \{R(a,b), P(c,a)\}$. □

We can see that a repair of a global system is a global database instance that satisfies IC and minimally differs, in the sense of Definition 7, from a minimal legal global database instance. If $\mathcal{G}$ is already consistent, then the repairs are the elements of $Mininst(\mathcal{G})$. In Definition 8 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 $Linst(\mathcal{G})$. If we do not allow this this flexibility, a global system might not be repairable. Repairs are used as an auxiliary concept to define the notion of consistent answer.

Example 8. (example 1 continued) The only element in $Mininst(\mathcal{G}_1)$ is $D_0 = \{R(a,b), R(c,d), R(a,c), R(d,e)\}$, that does not satisfy IC. Then, $\mathcal{G}_1$ 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 $Linst(\mathcal{G}_1)$. □

Definition 9. [9] (a) Given a global system $\mathcal{G}$, a set of global integrity constraints IC, and a global first-order query $Q(X)$, we say that a (ground) tuple t is a *consistent answer* to Q wrt IC iff for every repair D of $\mathcal{G}$, $\ D \models Q[t]$. (b) We denote by $Consis_{\mathcal{G}}(Q)$ the set of consistent answers to Q in $\mathcal{G}$. □

Example 9. (example 8 continued) For the query $Q_1(X) \colon \exists Y\ R(X,Y)$, the consistent answers are a, c, d. $\ Q_2(X,Y)$: $R(X,Y)$ has $(c,d), (d,e)$ as consistent answers. □

If $\mathcal{G}$ is consistent wrt IC, then $Consis_{\mathcal{G}}(Q) = Minimal_{\mathcal{G}}(Q)$. Furthermore, if the ICs are generic, then for any $\mathcal{G}$ it holds $Consis_{\mathcal{G}}(Q) \quad Minimal_{\mathcal{G}}(Q)$ [9]. Notice also that the notion of consistent answer can be applied to queries expressed in Datalog or its extensions with built-ins and negation.

3 Specification of Minimal Instances

The specification of the class $Mininst(\mathcal{G})$ for system $\mathcal{G}$ is given using normal deductive databases, 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 [26]. We consider global system all of whose sources are open.

3.1 The Simple Program

In this section we will present a first approach to the specification of legal instances. In Section 3.2 we present the definitive program, that refines the one given in this section. We proceed in this way, because the program we give now, although it may not be suitable for all situations (as discussed later in this section), is simpler to understand than its refined version, and already contains the key ideas.

Definition 10. Given an open global system $\mathcal{G}$, the logic program $(\mathcal{G})$, contains the following clauses:
1. Fact $dom(a)$ for every constant $a \in \mathcal{U}$; and the fact $V_i(a)$ whenever $a \in v_i$ for some source extension v_i in $\mathcal{G}$.
2. For every view (source) predicate V_i in the system with description $V_i(X) \quad P_1(X_1), \ldots, P_n(X_n)$, the rules
$$P_j(X_j) \quad V_i(X), \ \textstyle\bigwedge_{Z_l \in (X_j \setminus X)} F_i^l(X, Z_l), \quad j = 1, \ldots n.$$
3. For every predicate $F_i^l(X, Z_l)$ introduced in 2., the rule
$$F_i^l(X, Z_l) \quad V_i(X), dom(Z_l), choice((X), (Z_l)).$$ □

In this specification, the predicate $F_i^l(X, Z_l)$ replaces the Skolem function based atom $f_i^l(X) = Z_l$ introduced in Section 2.1, and, via the choice predicate, it assigns values in the domain to the variables in the head of the rule in 3. that are not in 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 $choice((X), (Z_l))$ ensures that for every (tuple of) value(s) for X, only one (tuple of) value(s) for Z_l is non deterministically chosen between the constants of the active domain.

Example 10. (examples 4 and 5 continued) Program $(\mathcal{G}_2)$ contains the following rules:

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

2. $P(X, Z) \quad V_1(X, Y), F_1(X, Y, Z).$
   $R(Z, Y) \quad V_1(X, Y), F_1(X, Y, Z).$
   $P(X, Y) \quad V_2(X, Y).$
3. $F_1(X, Y, Z) \quad V_1(X, Y), dom(Z),\ choice((X, Y), (Z)).$

In this section we will restrict ourselves to a finite domain $\mathcal{U}$, what is necessary to run the program in real implementations. In this example we have $\mathcal{U} = \{a, b, c, u\}$ (the extension of predicate *dom*). In section 6.2 we study how to handle infinite domains by adding to the active domain a finite number of extra constants, like constant u here.[4] □

For every program with the choice operator, there is its *stable version* $SV(\ \ )$, whose stable models correspond to the so-called *choice models* of [26]. The program $SV(\ \ )$ is obtained as follows:
(a) Each choice rule $r\colon\ H \quad B, choice((X), (Y))$ in is replaced by the rule $H \quad B, chosen_r(X, Y)$.
(b) For each rule as in (a), the following rules are added
$$chosen_r(X, Y) \quad B, not\ diffChoice_r(X, Y).$$
$$diffChoice_r(X, Y) \quad chosen_r(X, Y'), Y \neq Y'.$$

The rules defined in (b) ensure that, for every tuple X where B is satisfied, the predicate $chosen_r(X, Y)$ satisfies the functional dependency $X \rightarrow Y$.

Example 11. (example 10 continued) Program $SV(\ \ (\mathcal{G}_2))$ contains the following rules:
1. $dom(a).\ \ dom(b).\ \ dom(c).\ \ dom(u).\ \ V_1(a, b).\ \ V_2(a, c).$
2. $P(X, Z) \quad V_1(X, Y), F_1(X, Y, Z).$
   $R(Z, Y) \quad V_1(X, Y), F_1(X, Y, Z).$
   $P(X, Y) \quad V_2(X, Y).$
3. $F_1(X, Y, Z) \quad V_1(X, Y), dom(Z), chosen_1(X, Y, Z).$
4. $chosen_1(X, Y, Z) \quad V_1(X, Y), dom(Z),\ not\ diffChoice_1(X, Y, Z).$
   $diffChoice_1(X, Y, Z) \quad chosen_1(X, Y, Z'),\ dom(Z), Z' \neq Z.$

Its stable models are:
$\mathcal{M}_1 = \{dom(a),\ dom(b),\ dom(c),\ dom(u),\ V_1(a, b),\ V_2(a, c),\ \underline{P(a, c)},\ diffChoice_1(a, b, a),\ chosen_1(a, b, b),\ diffChoice_1(a, b, c),\ diffChoice_1(a, b, u),\ F_1(a, b, b),\ \underline{R(b, b)},\ \underline{P(a, b)}\}$
$\mathcal{M}_2 = \{dom(a),\ dom(b),\ dom(c),\ dom(u),\ V_1(a, b),\ V_2(a, c),\ \underline{P(a, c)},\ chosen_1(a, b, a),\ diffChoice_1(a, b, b),\ diffChoice_1(a, b, c),\ diffChoice_1(a, b, u),\ F_1(a, b, a),\ \underline{R(a, b)},\ \underline{P(a, a)}\}$

[4] In principle, *null* could be in the domain, and then we should include $dom(null)$ among the atoms, and, since we do not want legal instances to contain the null value, the literal $Z \neq null$ in the body of the rule in 3. Instead, to keep things simpler, we will not include $dom(null)$ in $(\mathcal{G})$, even if *null* belongs to the undelying domain $\mathcal{U}$.

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

The underlined atoms of the models correspond to the elements in which we are interested, namely the global relations of the integration system. □

Definition 11. The global instance associated to a choice model $\mathcal{M}$ of $(\mathcal{G})$ is $D_{\mathcal{M}} = \{P(a) \mid P \in \mathcal{R} \text{ and } P(a) \in \mathcal{M}\}$. □

Example 12. (example 11 continued) $D_{\mathcal{M}_1}$, $D_{\mathcal{M}_2}$, $D_{\mathcal{M}_3}$, $D_{\mathcal{M}_4}$ are the elements of $Mininst(\mathcal{G}_3)$, 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. It holds that
$$Mininst(\mathcal{G}) \quad \{D_{\mathcal{M}} \mid \mathcal{M} \text{ is a choice model of } (\mathcal{G})\} \quad Linst(\mathcal{G}).$$ □

From the inclusions in the theorem it is clear that for monotone queries Q, answers obtained using $(\mathcal{G})$ under the skeptical or cautious stable model semantics -that sanctions as true what is true of all the stable models of the program- coincide with $Certain_{\mathcal{G}}(Q)$ and $Minimal_{\mathcal{G}}(Q)$. This may not be the case for queries with negation, as pointed out in the remark after Definition 4.

In Example 12 the stable models are in a one to one correspondence with the minimal legal instances, but this may not be always the case.

Example 13. Consider an integration system $\mathcal{G}_3$ with global schema $\mathcal{R} = \{P\}$. The set $\mathcal{V}$ of local view definitions consists of $V_1(X) \qquad P(X,\ Y)$, and $V_2(X, Y) \quad P(X, Y)$ with source contents $v_1 = \{V_1(a)\}$, $v_2 = \{V_2(a, c)\}$, resp. We have that $Mininst(\mathcal{G}_3) = \{\{P(a, c)\}\}$. However, the global instances corresponding to models of $(\mathcal{G}_3)$ are of the form $\{\{P(a, c), P(a, z)\} \mid z \in \mathcal{U}\}$. As V_2 is open, it forces $P(a, c)$ to be in all legal instances, and with this, the same condition on V_1 is automatically satisfied, and no other values for Y are needed. But the choice operator still has freedom to chose other values (the $z \in \mathcal{U}$). This is why we get more legal instances than the minimal ones. □

Now we investigate sufficient conditions under which the simple program of Definition 10 captures the minimal instances. This is important because the general program to be presented in Section 3.2 is much more complex than the simple version presented so far.

We define a *section of a view* V_i as a set S_i^l consisting either of all the predicates in the body of its definition that share a same existential variable Z_l or all the atoms without existential variables, in which case $l = 0$ and the view section is denoted with S_i^0. For example, the view defined by $V(X, Y) \quad P(X, Z_1), R(Z_1, Y), T(X, Y)$ has two sections: $S_1^1 = \{P(X, Z_1),\ R(Z_1, Y)\}$ and $S_1^0 = \{T(X, Y)\}$. *Sec* denotes the set of all view sections for system $\mathcal{G}$.

Given a view section S_i^l, we denote by $Const(S_i^l)$, $UVar(S_i^l)$ and $EVar(S_i^l)$ the sets of constants, universal variables and existential variables, respectively, that occur in predicates in S_i^l.

Let μ, ε be two new constants. For a view section S_i^l, an *admissible mapping* is any mapping $h\colon Const(S_i^l) \cup UVar(S_i^l) \cup EVar(S_i^l) \rightarrow Const(S_i^l) \cup \{\mu, \varepsilon\}$, such that: (a) $h(c) = c$ for every $c \in Const(S_i^l)$; (b) $h(X) = D$ with $D \in Const(S_i^l) \cup \{\mu\}$ for every $X \in UVar(S_i^l)$; (c) $h(Z) = F$ with $F \in Const(S_i^l) \cup \{\mu, \varepsilon\}$ for every $Z \in EVar(S_i^l)$.

A particular admissible mapping L is given by (a) $L(c) = c$ for every $c \in Const(S_i^l)$; (b) $L(X) = \mu$ for every $X \in UVar(S_i^l)$; (c) $L(Z) = \varepsilon$ for every $Z \in EVar(S_i^l)$. For an admissible mapping h, $h(S_i^l)$ denotes the set of atoms obtained from S_i^l by applying h to the arguments in S_i^l.

Theorem 2. Given an integration system G, if for every view section S_i^l with existential variables, there is no admissible mapping h for S_i^l, such that $h(S_i^l) \quad \bigcup_{S \in (Sec \smallsetminus \{S_i^l\})} L(S)$, then the instances associated to the stable models of the simple version of $(\mathcal{G})$ are exactly the minimal legal instances of $\mathcal{G}$. □

Basically, the theorem says that if there is an admissible mapping, such that $h(S_i^l) \quad \bigcup_{S \in (Sec \smallsetminus \{S_i^l\})} L(S)$, then it is possible to have some view contents for which the openness will be satisfied by the other sections in *Sec*, and then it will not be necessary to compute values for the existential variables in section S_i^l. Since the simple version will always compute values for them, it may specify more legal instances than the minimal ones.

Example 14. (example 13 continued) The first view is defined by $V_1(X) \qquad P(X,\ Y)$, and has only one section $S_1^Y = \{P(X, Y)\}$. For the admissible mapping h defined by $h(X) = h(Y) = \mu$, we have that $h(S_1^Y) = \{P(\mu, \mu)\} \quad L(S_2^0)$. The conditions of the theorem are not satisfied, and there is no guarantee that the simple version will calculate exactly the minimal instances of $\mathcal{G}_3$. Actually, we already know that this is not the case. □

Example 15. (examples 4 and 5 continued) There are two view sections: $S_1^Z = \{P(X, Z), Q(Z, Y)\}$ and $S_2^0 = \{P(X, Y)\}$, where X and Y are universal variables and Z is an existential variable. It is easy to see that there is no mapping h for which $h(S_1^Z) \quad L(S_2^0)$ nor $h(S_2^0) \quad L(S_1^Z)$. In consequence, for any source contents, the simple version of $(\mathcal{G}_2)$ will calculate exactly the minimal instances of $\mathcal{G}_2$. □

3.2 The Refined Program

In the general case, if we want to compute only the elements of $Mininst(\mathcal{G})$, we need to refine the program $(\mathcal{G})$ given in the previous section. For this we will introduce auxiliary annotation constants that will be used as extra arguments in the database predicates. They and their intended semantics are given in the following table

annotation	atom	the tuple $P(a)$ is ...
$\mathbf{t_d}$	$P(a,\mathbf{t_d})$	an atom of the minimal legal instances
$\mathbf{t_o}$	$P(a,\mathbf{t_o})$	is an obligatory atom in all the minimal legal instances
$\mathbf{v_i}$	$P(a,\mathbf{v_i})$	an optional atom introduced to satisfy the openness of view v_i
$\mathbf{nv_i}$	$P(a,\mathbf{nv_i})$	an optional atom introduced to satisfy the openness of view that is not v_i

Definition 12. Given an open global system $\mathcal{G}$, the refined program $\quad(\mathcal{G})$, contains the following clauses:

1. Fact $\ \mathit{dom}(a)$ for every constant $a \in \mathcal{U}$
2. Fact $V_i(a)$ whenever $a \in v_i$ for some source extension v_i in $\mathcal{G}$.
3. For every view (source) predicate V_i in the system with description $\ V_i(X) \quad P_1(X_1), \ldots, P_n(X_n)$:

   (a) For every P_k with no existential variables, the rules

   $$P_k(X_k,\mathbf{t_o}) \quad V_i(X).$$

   (b) For every set S_{ij} of predicates of the description's body that are related by common existential variables $\{Z_1, \ldots, Z_m\}$, the rules,

   $$P_k(X_k,\mathbf{v_{ij}}) \quad \mathit{add}_{v_{ij}}(X'),\ \textstyle\bigwedge_{Z_l \in (X_k \setminus X')} F_i^l(X', Z_l), \text{ for } P_k \in S_{ij}.$$

   $$\mathit{add}_{v_{ij}}(X') \quad V_i(X),\ \mathit{not}\ \mathit{aux}_{v_{ij}}(X'), \text{ where } X' = X \cap \{\textstyle\bigcup_{P_k \in S_{ij}} X_k\}.$$

   $$\mathit{aux}_{v_{ij}}(X') \quad \textstyle\bigwedge_{l=1}^{m} \mathit{var}_{v_{ij}Z_l}(X_{Z_l}).$$

   $$\mathit{var}_{v_{ij}Z_l}(X_{Z_l}) \quad \textstyle\bigwedge_{P_k \in S_{ij} \& Z_l \in X_k} P_k(X_k,\mathbf{nv_{ij}}),$$
   $$\text{where } X_{Z_l} = \{\textstyle\bigcup_{P_k \in S_{ij} \& Z_l \in X_k} X_k\}, \text{ for } l = 1, \ldots m.$$

4. For every predicate $F_i^l(X', Z_l)$ introduced in 3.(b), the rules,

   $$F_i^l(X', Z_l) \quad \mathit{add}_{v_{ij}Z_l}(X'), \mathit{dom}(Z_l), \mathit{choice}((X'),(Z_l)).$$

   $$\mathit{add}_{v_{ij}Z_l}(X') \quad \mathit{add}_{v_{ij}}(X'),\ \mathit{not}\ \mathit{aux}_{v_{ij}Z_l}(X'), \text{ for } l = 1, \ldots m.$$

   $$\mathit{aux}_{v_{ij}Z_l}(X') \quad \mathit{var}_{v_{ij}Z_l}(X_{Z_l}), \textstyle\bigwedge_{Z_k \neq Z_l \& Z_k \in X_{Z_l}} F_i^k(X', Z_k),$$
   $$\text{for } l = 1, \ldots\ m.$$

5. For every global relation $P(X)$ the rules

   $$P(X,\mathbf{nv_{ij}}) \quad P(X,\mathbf{v_{hk}}), \quad \text{for } \{(ij,hk) | P(X) \in S_{ij} \cap S_{hk}, ij \neq hk\}.$$

   $$P(X,\mathbf{nv_{ij}}) \quad P(X,\mathbf{t_o}), \quad \text{for } \{(ij) | P(X) \in S_{ij}\}.$$

   $$P(X,\mathbf{t_d}) \quad P(X,\mathbf{v_{ij}}), \quad \text{for } \{(ij) | P(X) \in S_{ij}\}.$$

   $$P(X,\mathbf{t_d}) \quad P(X,\mathbf{t_o}).$$

□

Example 16. (example 13 continued) The refined program $\quad(\mathcal{G}_3)$ is:

$$\mathit{dom}(a). \quad \mathit{dom}(c). \tag{2}$$
$$v_1(a). \quad v_2(a,c). \tag{3}$$
$$P(X,Z,\mathbf{v_1}) \quad \mathit{add}_{v_1}(X), F_z(X,Z). \tag{4}$$
$$\mathit{add}_{v_1}(X) \quad v_1(X),\ \mathit{not}\ \mathit{aux}_{v_1}(X). \tag{5}$$
$$\mathit{aux}_{v_1}(X) \quad \mathit{var}_{v_1z}(X,Z). \tag{6}$$
$$\mathit{var}_{v_1z}(X,Z) \quad P(X,Z,nv_1). \tag{7}$$
$$F_z(X,Z) \quad \mathit{add}_{v_1}(X), \mathit{dom}(Z), \mathit{chosen}_{v_1z}(X,Z). \tag{8}$$
$$\mathit{chosen}_{v_1z}(X,Z) \quad \mathit{add}_{v_1}(X), \mathit{dom}(Z),\ \mathit{not}\ \mathit{diff\ choice}_{v_1z}(X,Z). \tag{9}$$
$$\mathit{diff\ choice}_{v_1z}(X,Z) \quad \mathit{chosen}_{v_1z}(X,Z'), \mathit{dom}(Z), Z' \neq Z. \tag{10}$$
$$P(X,Y,\mathbf{t_o}) \quad v_2(X,Y). \tag{11}$$
$$P(X,Y,\mathbf{nv_1}) \quad P(X,Y,\mathbf{t_o}). \tag{12}$$
$$P(X,Y,\mathbf{t_d}) \quad P(X,Y,\mathbf{v_1}). \tag{13}$$
$$P(X,Y,\mathbf{t_d}) \quad P(X,Y,\mathbf{t_o}). \tag{14}$$

Rules (4) to (7) ensure that if there is an atom in source V_1, e.g. $V_1(a)$, and if an atom of the form $P(a,Z)$ was not added by view V_2, then it is added by rule (4) with a Z value given by the function predicate $F_z(a,Z)$. This function predicate is calculated by rules (8) to (10). Rule (11) enforces the satisfaction of the openness of V_2 by adding obligatory atoms to predicate P and rule (12) 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 and that are in the minimal legal instances. The stable model of this program is $\{\mathit{dom}(a), \mathit{dom}(c),\ v_1(a),\ v_2(a,c),\ \underline{P(a,c,\mathbf{t_d})},$ $P(a,c,\mathbf{t_o}),\ P(a,c,\ \mathbf{nv_1}),\ \mathit{aux}_{v_1}(a)\}$, which corresponds to the only minimal legal instance $\{P(a,c)\}$. □

Theorem 3. If $\mathcal{M}$ is a stable model of $SV(\quad(\mathcal{G}))$, then $D_{\mathcal{M}} := \{P(a) \mid P \in \mathcal{R}$ and $P(a,\mathbf{t_d}) \in \mathcal{M}\} \in \mathit{Mininst}(\mathcal{G})$. Furthermore, the minimal legal instances obtained in this way are all the minimal legal instances of $\mathcal{G}$. □

The program $\quad(\mathcal{G})$ (or its stable version) can be used to compute $\mathit{Minimal}_{\mathcal{G}}(Q)$, where Q is a query expressed as a, say Datalog$^{\neg}$ program $\quad(Q)$. This can be done by running the combined program under the skeptical stable model semantics. The following corollary for monotone queries, e.g. a Datalog queries, can be immediately obtained from Theorem 3 and the fact that for those queries $\mathit{Certain}_{\mathcal{G}}(Q) = \mathit{Minimal}_{\mathcal{G}}(Q)$.

Corollary 1. The certain answers to monotone queries posed to an open integration system $\mathcal{G}$ can be computed by running, under the skeptical stable model semantics, the query program in combination with the program $\quad(\mathcal{G})$ that specifies the minimal legal instances of $\mathcal{G}$. □

We know that under the hypothesis of Theorem 2, the simple and refined programs compute the same legal database instances, namely the minimal ones. Beyond this, it is worth mentioning that, under the same hypothesis, there is a simple mechanical, syntactic transformation of the refined program into a simple program (in the sense of Section 3.1) that has the same stable models, and then, in particular, produces the same database instances (see Appendix A.2).

4 Specification of Repairs of a Global System

In [6], repairs of single relational databases are specified as stable models of disjunctive logic programs. We briefly explain those programs, because they will be used to specify repairs of instances of integration systems.

First, the database predicates are expanded with an extra argument to be filled with one of a set of new annotation constants. An atom inside (outside) the original database is annotated with $\mathbf{t_d}$ ($\mathbf{f_d}$).[5] 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 $\mathbf{t_a}$, this is seen as an advice to insert it into the database.

Example 17. (example 7 continued) Consider the ICs $\forall x(P(x,y) \rightarrow R(x,y))$ and $\forall x(T(x) \rightarrow \exists y P(x,y))$, together with the inconsistent database instance $D = \{P(a,b), T(c)\}$ and a domain $\mathcal{U} = \{a,b,c,u\}$. 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 2. and 5. 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 3. 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 7. below, that are used to interpret the models as database repairs. The facts of rule 1. correspond to all the elements of the domain except for the *null* constant, which is left outside of *dom*. The following is the program:

1. $\mathit{dom}(a). \quad \mathit{dom}(b). \quad \mathit{dom}(c). \quad \mathit{dom}(u).$
2. $P(x,y,\mathbf{f}^\star) \quad P(x,y,\mathbf{f_a}),\ \mathit{dom}(x),\ \mathit{dom}(y).$
   $P(x,y,\mathbf{t}^\star) \quad P(x,y,\mathbf{t_a}),\ \mathit{dom}(x),\ \mathit{dom}(y).$
   $P(x,y,\mathbf{t}^\star) \quad P(x,y,\mathbf{t_d}),\ \mathit{dom}(x),\ \mathit{dom}(y).$ (similarly for R and T)
3. $P(x,y,\mathbf{f_a}) \vee R(x,y,\mathbf{t_a}) \quad P(x,y,\mathbf{t}^\star), R(x,y,\mathbf{f}^\star),\ \mathit{dom}(x),\ \mathit{dom}(y).$
   $T(x,\mathbf{f_a}) \vee P(x,\mathit{null},\mathbf{t_a}) \quad T(x,\mathbf{t}^\star), \mathit{not}\ \mathit{aux}(x),\ \mathit{not}\ P(x,\mathit{null},\mathbf{t_d}),\ \mathit{dom}(x).$
   $\mathit{aux}(x) \quad P(x,y,\mathbf{t_d}),\ \mathit{not}\ P(x,y,\mathbf{f_a}).$
   $\mathit{aux}(x) \quad P(x',y,\mathbf{t_a}).$
4. $P(a,\mathbf{t_d}) \quad .$
5. $P(x,y,\mathbf{f}^\star) \quad \mathit{dom}(x),\ \mathit{dom}(y), \mathit{not}\ P(x,y,\mathbf{t_d}).$ (similarly for R and T)
6. $\quad P(x,\mathbf{t_a}), P(x,\mathbf{f_a}). \qquad R(x,\mathbf{t_a}), R(x,\mathbf{f_a}).$
7. $P(x,y,\mathbf{t}^{\star\star}) \quad P(x,y,\mathbf{t_a}),\ \mathit{dom}(x),\ \mathit{dom}(y).$
   $P(x,y,\mathbf{f}^{\star\star}) \quad P(x,y,\mathbf{f_a}),\ \mathit{dom}(x),\ \mathit{dom}(y).$
   $P(x,y,\mathbf{t}^{\star\star}) \quad P(x,y,\mathbf{t_d}),\ \mathit{not}\ P(x,y,\mathbf{f_a}),\ \mathit{dom}(x),\ \mathit{dom}(y).$
   $P(x,y,\mathbf{f}^{\star\star}) \quad \mathit{dom}(x),\ \mathit{dom}(y),\ \mathit{not}\ P(x,y,\mathbf{t_d}),\ \mathit{not}\ P(x,y,\mathbf{t_a}).$
   (similarly for R and T)

Only rules 3. depend on the ICs. The first rule in 3. corresponds to the universal ICs and the rest to the referential IC. These rules say how to repair the inconsistencies. Rules 4. contain the database atoms. Rules 5. capture the *closed world assumption* (CWA) [40]. Rules 6. are denial program constraints to discard models that contain an atom annotated with both $\mathbf{t_a}$ and $\mathbf{f_a}$. The program has four stable models. The repairs are obtained from them by selecting the atoms annotated with $\mathbf{t}^{\star\star}$: $D_1 = \{P(a,b), R(a,b)\}$, $D_2 = \{P(a,b), R(a,b), T(c), P(c,\mathit{null})\}$ and $D_3 = \{T(c), P(c,\mathit{null})\}$, $D_4 = \emptyset$. As expected, they coincide with the ones obtained in Example 7. □

It can be proved [6] in the context of single relational databases that the stable models of these disjunctive programs are in a one to one correspondence with the repairs of the original database, for any combination of universal and acyclic referential integrity constraints. If there are cycles between the referential ICs, then the specification programs may produce a class of stable models that properly extends the class of repairs [12]. Those models that do not correspond to repairs still satisfy the ICs, but may not be minimal repairs. In this case the stable models that do not correspond to (minimal) repairs can be pruned by comparison with the other stable models [12]. These properties will be inherited by our application of this kind of programs to the specification of the repairs of the minimal instances of an integration system.

The next definition combines into one program the refined version that specifies the minimal legal instances and the specification of the repairs of those minimal instances.

Definition 13. The *repair program*, $\quad(\mathcal{G}, IC)$, of $\mathcal{G}$ wrt IC contains the following clauses:

1. The same rules as in Definition 12.
2. For every predicate $P \in \mathcal{R}$, the clauses
   $P(X,\mathbf{t}^\star) \quad P(X,\mathbf{t_d}),\ \mathit{dom}(X).$[6]
   $P(X,\mathbf{t}^\star) \quad P(X,\mathbf{t_a}),\ \mathit{dom}(X).$
   $P(X,\mathbf{f}^\star) \quad P(X,\mathbf{f_a}),\ \mathit{dom}(X).$
   $P(X,\mathbf{f}^\star) \quad \mathit{dom}(X),\ \mathit{not}\ P(X,\mathbf{t_d}).$

[5] The annotation $\mathbf{t_d}$ is the same we had in the previous section, actually the program there will provide the contents of the minimal instances in terms of $\mathbf{t_d}$; next, in the repair process, the new annotations introduced here will be generated.

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

3. For every first-order global universal IC of the form $\overline{\forall}(Q_1(Y_1) \vee \quad \vee Q_n(Y_n) \quad P_1(X_1) \wedge \quad \wedge P_m(X_m) \wedge \varphi)$, where $P_i, Q_j \in \mathcal{R}$, and φ is a conjunction of built-in atoms, the clause:
   $\bigvee_{i=1}^{n} P_i(X_i, \mathbf{f_a}) \bigvee_{j=1}^{m} Q_j(Y_j, \mathbf{t_a}) \qquad \bigwedge_{i=1}^{n} P_i(X_i, \mathbf{t}^\star),\ \bigwedge_{j=1}^{m} Q_j(Y_j, \mathbf{f}^\star),$
   $dom(X),\ \varphi;$
   where X is the tuple of all variables appearing in database atoms in the rule.
4. For every referential IC of the form $\forall X(P(X) \rightarrow \exists Y Q(X', Y))$, with X' X, the clauses
   $P(X, \mathbf{f_a}) \vee Q(X', null, \mathbf{t_a}) \quad P(X, \mathbf{t}^\star), not\ aux(X'),\ not\ Q(X', null, \mathbf{t_d}),$
   $dom(X).$
   $aux(X') \quad Q(X', Y, \mathbf{t_d}),\ not\ Q(X', Y, \mathbf{f_a}),\ dom(X', Y).$
   $aux(X') \quad Q(X', Y, \mathbf{t_a}),\ dom(X', Y).$
5. For every predicate $P \in \mathcal{R}$, the interpretation clauses:
   $P(a, \mathbf{f}^{\star\star}) \qquad P(a, \mathbf{f_a}).$
   $P(a, \mathbf{f}^{\star\star}) \qquad not\ P(a, \mathbf{t_d}),\ not\ P(a, \mathbf{t_a}).$
   $P(a, \mathbf{t}^{\star\star}) \qquad P(a, \mathbf{t_a}).$
   $P(a, \mathbf{t}^{\star\star}) \qquad P(a, \mathbf{t_d}),\ not\ P(a, \mathbf{f_a}).$ □

Rules 4 repair referential ICs by deletion of tuples or insertion of null values that are not propagated through other ICs [6]. For this purpose, $dom(null)$ is not considered as a fact and therefore the *null* values will not propagate. Optimizations of the repair part of the program, like avoiding the materialization of the CWA, are analyzed in [6].

The choice models of program $(\mathcal{G}, IC)$ that do not contain a pair of literals of the form $\{P(a, \mathbf{t_a}), P(a, \mathbf{f_a})\}$ are called *coherent models*. Only coherent models can be obtained for the program if the denial constraints of the form $P(x, \mathbf{t}^{\star\star}), P(x, \mathbf{f}^{\star\star})$ are included in the program.

Definition 14. The global instance associated to a choice model $\mathcal{M}$ of $(\mathcal{G}, IC)$ is $D_{\mathcal{M}} = \{P(a) \mid P \in \mathcal{R}$ and $P(a, \mathbf{t}^{\star\star}) \in \mathcal{M}\}$. □

The repair program can be split [37] into the specification of the minimal instances and the specification of their repairs. Therefore, the minimal legal instances can be calculated first, and then the repairs of them. Each minimal model calculated by the first part of $(\mathcal{G}, IC)$ can be seen as a simple, relational database, which is repaired afterwards by the second part of $(\mathcal{G}, IC)$. This gives us the following theorem straightforwardly.

Theorem 4. Let IC be an arbitrary class of universal and acyclic referential integrity constraints. If $\mathcal{M}$ is a coherent choice model of $(\mathcal{G}, IC)$, then $D_{\mathcal{M}}$ is a repair of $\mathcal{G}$ wrt IC. Furthermore, the repairs obtained in this way are all the repairs of $\mathcal{G}$ wrt IC. □

In the case in which a cyclic set of referential ICs is considered, the global instances associated to the choice models of the program will be a superset of the repairs of $\mathcal{G}$ wrt IC, and in order to obtain the repairs, the choice models will have to be compared to choose those minimally differ from the minimal legal instance [12].

5 Consistent Answers

Now we can obtain the answers to queries posed to a system $\mathcal{G}$ that are consistent wrt to IC. First we will consider universal and acyclic referential ICs. We do the following:

1. We start with a query Q that is expressed, e.g. as a stratified Datalog program, (Q), whose extensional predicates are elements of the global schema $\mathcal{R}$. Each positive occurrence of those predicates, say $P(t)$, is replaced by $P(t, \mathbf{t}^{\star\star})$; and each negative occurrence, say *not* $P(t)$, by $P(t, \mathbf{f}^{\star\star})$. This query program has a query predicate *Ans* that collects the answers to Q. In particular, first order queries can be expressed as stratified Datalog programs [1].
2. Program (Q) is appended to the program $SV(\quad (\mathcal{G}, IC))$, 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 $(Q) \cup SV(\quad (\mathcal{G}, IC))$.

Example 18. (example 11 continued) We have the integration system $\mathcal{G}_2$ with the local view definitions $V_1(X, Z) \quad P(X, Y), R(Y, Z)$, and $V_2(X, Y) \quad P(X, Y)$, and source contents $v_1 = \{V_1(a, b)\}$ and $v_2 = \{V_2(a, c)\}$ respectively. Consider the global symmetry integrity constraint $sim:\ \forall x \forall y(R(x, y) \rightarrow R(y, x))$ on $\mathcal{G}_2$. We want the consistent answers to the query $Q: P(x, y)$. First, the query is written as the query program clause $Ans(X, Y) \quad P(X, Y, \mathbf{t}^{\star\star})$. This query program, (Q), is run with the revised version of $SV(\quad (\mathcal{G}_3, sim))$ that has the following rules:

% Subprogram for minimal instances
$dom(a). \quad dom(b). \quad dom(c). \quad dom(u).$
$v_1(a, b). \quad v_2(a, c).$
$P(X, Y, \mathbf{nv_1}) \quad P(X, Y, \mathbf{t_o}).$
$P(X, Y, \mathbf{nv_2}) \quad P(X, Y, \mathbf{v_1}).$
$P(X, Y, \mathbf{t_d}) \quad P(X, Y, \mathbf{v_1}).$
$P(X, Y, \mathbf{t_d}) \quad P(X, Y, \mathbf{t_o}).$
$R(X, Y, \mathbf{t_d}) \quad R(X, Y, \mathbf{v_1}).$
% Specification of V_1
$P(X, Y, \mathbf{v_1}) \quad add_{v_1}(X, Z), F_1^Y(X, Z, Y).$
$R(Y, Z, \mathbf{v_1}) \quad add_{v_1}(X, Z), F_1^Y(X, Z, Y).$
$add_{v_1}(X, Z) \quad v_1(X, Z),\ not\ aux_{v_1}(X, Z).$
$aux_{v_1}(X, Z) \quad var_{v_1 Y}(X, Y, Z).$
$var_{v_1 Y}(X, Y, Z) \quad P(X, Y, \mathbf{nv_1}), R(Y, Z, \mathbf{nv_1}).$
$F_1^Y(X, Z, Y) \quad add_{v_1 Y}(X, Z), dom(Y), chosen_{v_1 Y}(X, Z, Y).$
$chosen_{v_1 Y}(X, Z, Y) \quad add_{v_1 Y}(X, Z), dom(Y),\ not\ diffchoice_{v_1}(X, Z, Y).$
$diffchoice_{v_1}(X, Z, Y) \quad chosen_{v_1 Y}(X, Z, Y'), dom(Y), Y' \neq Y.$
$add_{v_1 Y}(X, Z) \quad add_{v_1}(X, Z),\ not\ aux_{v_1 Y}(X, Z).$
$aux_{v_1 Y}(X, Z) \quad var_{v_1 Y}(X, Y, Z).$
% Specification of V_2
$P(X, Y, \mathbf{t_o}) \quad v_2(X, Y).$

% Repair subprogram
$P(X, Y, \mathbf{t}^\star) \quad P(X, Y, \mathbf{t_a}), dom(X), dom(Y).$
$P(X, Y, \mathbf{t}^\star) \quad P(X, Y, \mathbf{t_d}), dom(X), dom(Y).$
$P(X, Y, \mathbf{f}^\star) \quad dom(X), dom(Y),\ not\ P(X, Y, \mathbf{t_d}).$
$P(X, Y, \mathbf{f}^\star) \quad P(X, Y, \mathbf{f_a}), dom(X), dom(Y).$
$R(X, Y, \mathbf{t}^\star) \quad R(X, Y, \mathbf{t_a}), dom(X), dom(Y).$
$R(X, Y, \mathbf{t}^\star) \quad R(X, Y, \mathbf{t_d}), dom(X), dom(Y).$
$R(X, Y, \mathbf{f}^\star) \quad dom(X), dom(Y),\ not\ R(X, Y, \mathbf{t_d}).$
$R(X, Y, \mathbf{f}^\star) \quad R(X, Y, \mathbf{f_a}), dom(X), dom(Y).$
$R(X, Y, \mathbf{f_a}) \vee R(Y, X, \mathbf{t_a}) \quad R(X, Y, \mathbf{t}^\star), R(Y, X, \mathbf{f}^\star), dom(X), dom(Y).$
$P(X, Y, \mathbf{t}^{\star\star}) \quad P(X, Y, \mathbf{t_a}), dom(X), dom(Y).$
$P(X, Y, \mathbf{t}^{\star\star}) \quad P(X, Y, \mathbf{t_d}), dom(X), dom(Y),\ not\ P(X, Y, \mathbf{f_a}).$
$P(X, Y, \mathbf{f}^{\star\star}) \quad P(X, Y, \mathbf{f_a}), dom(X), dom(Y).$
$P(X, Y, \mathbf{f}^{\star\star}) \quad dom(X), dom(Y),\ not\ P(X, Y, \mathbf{t_d}),\ not\ P(X, Y, \mathbf{t_a}).$
$R(X, Y, \mathbf{t}^{\star\star}) \quad R(X, Y, \mathbf{t_a}), dom(X), dom(Y).$
$R(X, Y, \mathbf{t}^{\star\star}) \quad R(X, Y, \mathbf{t_d}), dom(X), dom(Y),\ not\ R(X, Y, \mathbf{f_a}).$
$R(X, Y, \mathbf{f}^{\star\star}) \quad R(X, Y, \mathbf{f_a}), dom(X), dom(Y).$
$R(X, Y, \mathbf{f}^{\star\star}) \quad dom(X), dom(Y),\ not\ R(X, Y, \mathbf{t_d}),\ not\ R(X, Y, \mathbf{t_a}).$
$\qquad R(X, Y, \mathbf{t_a}), R(X, Y, \mathbf{f_a}).$
$\qquad P(X, Y, \mathbf{t_a}), P(X, Y, \mathbf{f_a}).$

This program has five stable models with the following associated repairs: (a) $D_{\mathcal{M}_1^r} = \{\ P(a,\ b),\ R(b,\ b),\ P(a,\ c)\ \}$, corresponding to the already consistent minimal instance $D_{\mathcal{M}_1}$ in Example 12; (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 $(Q) \cup SV(\quad (\mathcal{G}_3,\ sim))$ 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 $\mathcal{G}$ 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.

6 Further Analysis, Extensions and Discussion

6.1 Complexity

The complexity analysis of consistent query answering in integration of open sources under the LAV approach can be split according to the main two layers of the combined program, namely, the specification of minimal instances and the specification of the repairs of those minimal instances.

Query evaluation from the program $(\mathcal{G})$ with choice under the skeptical stable model semantics is in *coNP* (the case singularized as *certainty semantics* in [41]). Actually, if the choice operator program is represented in its "classical" stable version (see Section 3.1), we are left with a normal (non-disjunctive), but non-stratified program whose query answering complexity under the skeptical stable model semantics is *coNP*-complete [19, 35] in data complexity [1], in our case, in terms of the combined sizes of the sources. This complexity of computing minimal answers is inherited by the computation of certain answers when the two notions coincide, e.g. for monotone queries like Datalog queries. This complexity result is consistent and matches the theoretical complexity lower bound on computing certain answers to Datalog queries under the LAV approach [2]. With disjunctive views, as considered in Section 6.4, the complexity of the program goes up to being ${}^{P}_{2}$-complete.

The complexity of query evaluation wrt the disjunctive normal program $(\mathcal{G}, IC)$ that specifies the repair of minimal instances is ${}^{P}_{2}$-complete in data complexity [19], which matches the complexity of consistent query answering [10, 18, 15].

There are some cases studied in [6], e.g. only universal ICs, where the repair part of the program for CQA is *head-cycle free* (HCF) and therefore the complexity is reduced to *coNP* [7, 34]. This *coNP*-completeness result can be extended to some cases where both universal and RICs are considered. It is possible to show [12] that the program $(\mathcal{G}, IC)$ is HCF for a combination of: (a) *Denial constraints*, i.e. formulas of the form $\bigvee_{i=1}^{n} P_i(t_i) \rightarrow \varphi$, where $P_i(t_i)$ is an atom and φ is a formula containing built-in predicates only; (b) *Acyclic referential integrity constraints*, i.e. without cycles in the dependency graph.

This case includes the usual integrity constraints found in database practice, like (non cyclic) foreign key constraints. In [18, 15] some cases where functional dependencies and referential integrities coexist are presented, for which the problem of CQA becomes ${}^{P}_{2}$-complete. Actually, in the case when repairs wrt cyclic RICs is done by introducing arbitrary, non null elements of the underlying domain, the problem of consistent query answering becomes undecidable [15]. However, if repairs wrt cyclic RICs are obtained by introducing null values that do not propagate via ICs, the problem of consistent query answering becomes decidable [12].

6.2 Infinite vs. Finite Domain

In Section 2.1 we considered the possibility of having an infinite underlying domain $\mathcal{U}$. At the purely specification level there is not problem in admitting, in the first item of Definition 10, an infinite number of facts. Our soundness and completeness theorems hold. However, in the logic programs we have presented in the examples we had a finite domain, c.f. Example 10 (the finite domain is specified by the *dom* predicate), but also an extra constant u that does not appear in the active domain of the integration system, that consists of all the constants in the sources plus those that appear in the view definitions. The reason is that we need a finite domain to run the programs, but at the same time we need to capture the potential infiniteness of the domain and the openness of the sources. Furthermore, we should not be forced to use only the active domain, because doing so might assign the wrong semantics to the integration system.

Example 19. Consider an integration system $\mathcal{G}_4$ with one source defined by the view $V(X) \leftarrow R(X,Y)$ and the query $Q(Y) \leftarrow R(X,Y)$. If the view extension has only one tuple, say $\{(a)\}$, we have that the active domain is $\{a\}$ and that $R(a,a)$ is in all the legal instances of $\mathcal{G}_4$ if only this domain is used; and we would have $Certain_{\mathcal{G}_4}(Q) = \{a\}$. Now, if the view extension becomes $\{(a),(b)\}$, the active domain is $\{a,b\}$, and there is a global instance containing just the tuple $R(a,b)$, and another containing just $\{R(a,a)\}$. In consequence, there will be no certain answers. This simple example shows that a positive query may have an undesirable non-monotonic behavior □

In Example 10, introducing one extra constant (u) is good enough to correctly answer conjunctive queries (see below). In the general case, the number of extra constants may vary depending on the situation.

It is necessary to make all these considerations, because, the set of minimal legal instances may depend on underlying domain, as we saw in Example 5, where $Mininst(\mathcal{G}_2) = \{\{P(a,c),\, P(a,z),\, R(z,a)\} \mid z \in \mathcal{U} = \{a,b,c,...\}\}$.

Since we want only the certain answers, those that can be obtained from all the stable models, it is easy to see that the values taken by the "free variables", like z above, will not appear in a certain answer. However, the absence of the extra, new constants may sanction as certain some answers that are not if the domain is restricted to the active domain (see Example 19). In consequence, we need a larger domain, with enough variables to represent the relations and differences between the free variables. Depending on the query, there is a finite domain that generates the same certain and minimal answers as the infinite domain. It can be shown that if the query is conjunctive, then adding only one new constant to the active domain is good enough (see Example 10).

If the query is disjunctive, then the smallest "equivalent" finite domain is the active domain plus n new constants, where n is the maximum number of instantiations of existential variables in a minimal legal instance. This number of instantiations cannot be obtained from the view definitions alone, because it also depends on the number of elements in the sources associated to the Skolem predicates. An upper bound on the number of constants to be added to the active domain to correctly answer disjunctive queries is the sum over all sources of the product of the number of existential variables in a view definition with the number of atoms in the corresponding source.

Example 20. Given an integration system $\mathcal{G}_5$:
$V_1(X,Y) \leftarrow P(X,Z_0), R(Z_0,Y). \qquad \{V_1(a,b)\}$
$V_2(X,Y) \leftarrow P(X,Z_1), R(Z_2,Y). \qquad \{V_2(a,b), V_2(c,d)\}$
The set of minimal legal instances is $\{\{P(a,z_1),\, R(z_1,b),\, P(c,z_2),\, R(z_3,d)\} \mid z_1,$ $z_2,\, z_3 \in \mathcal{U}\}$. By looking at this representation, we see that in order to obtain correct certain answers to disjunctive queries, it is good enough to add to the active domain $\{a,b,c,d\}$ three extra constants, obtaining, say $\mathcal{U} = \{a,b,c,d,e,f,g\}$, a

finite domain that is able to simulate an infinite domain wrt disjunctive queries. Instead of inspecting the minimal instances to determine the number of new constants, we can use an upper bound, in this case, five, which can be computed as: 1 existential variable times 1 atom plus 2 existential variables times 2 atoms. So, we could use a domain $\mathcal{U}$ with five extra constants. □

6.3 Choice Models vs. Skolem Functions

In this paper we have used the *choice* operator to replace the Skolem functions used in the inverse rules algorithm. In this way we were able to specify the minimal global instances, which was one of our original goals, is interesting in itself, and allows us to specify the repairs of the integration system wrt the ICs. However, if we are interested in query answering only, it becomes relevant to analyze if it is possible to retrieve the minimal, certain and consistent answers by keeping the Skolem functions in the program, evaluating it, and then filtering out the final answers that contain those functions (as done in [21]).

We first analyze the case of the simple program (see Section 3.1), in which we want to consider using the Skolem functions instead of the functional predicate together with the choice operator. For example, we would have $P(X, f(X)) \leftarrow V(X)$ instead of the couple of rules $P(X,Y) \leftarrow V(X)$, $F(X,Y)$ and $F(X,Y) \leftarrow V(X)$, $dom(Y)$, $choice((X),(Y))$.

In this case, the program will have the same rules $\mathcal{V}^{-1}$ as in the inverse rules algorithm. The resulting definite program is positive and, therefore, its stable model corresponds to the minimal model. That model will have atoms with instantiated Skolem functions, and can be seen as a compact representation of the collection of stable models of the choice program, in the sense that the latter can be recovered by considering the different ways in which the Skolem functions can be defined in the underlying domain.

If a query is posed to the program with Skolem functions, the answer set may contain or not answers with Skolem functions. Those answers with Skolem functions correspond to answers that would be different in different stable models of the choice program, because in a sufficiently rich domain (see Section 6.2) the functions may be defined in different ways. This is why if we delete those answers with functions, we get the same answers as from the choice program $\Pi(\mathcal{G})$ under the cautious stable model semantics. In consequence, for computing the certain answers to a monotone query, we can indistinctly use the program with Skolem functions (pruning the answers with Skolem functions at the end) or the choice program.

Let us now consider the refined program (see Section 3.2). In this case, if Skolem functions are used instead of the choice operator, the resulting program is a normal program that may have several stable models.

Example 21. Consider an integration system $\mathcal{G}$ with
$V_1(X) \leftarrow P(X_1,Y_1,Z_1), S(Y_1) \qquad V_1(a)$
$V_2(X,Y) \leftarrow P(X_2,Y_2,Z_2) \qquad V_2(a,e)$

The following is the program with Skolem functions:

$\%V_1$
$P(X, f_1(X), f_2(X), \mathbf{v_1}) \leftarrow add_{v_1}(X), add_{v_1 Y}(X), add_{v_1 Z}(X).$
$S(f_1(X), \mathbf{v_1}) \leftarrow addv_1(X).$
$add_{v_1}(X) \leftarrow v_1(X),\ not\ aux_{v_1}(X).$
$aux_{v_1}(X) \leftarrow var_{v_1 Y}(X,Y,Z), var_{v_1 Z}(X,Y,Z).$
$var_{v_1 Y}(X,Y,Z) \leftarrow P(X,Y,Z,\mathbf{nv_1}), S(Y,\mathbf{nv_1}).$
$var_{v_1 Z}(X,Y,Z) \leftarrow P(X,Y,Z,\mathbf{nv_1}).$
$add_{v_1 Y}(X) \leftarrow add_{v_1}(X),\ not\ aux_{v_1 Y}(X).$
$aux_{v_1 Y}(X) \leftarrow var_{v_1 Y}(X,Y,Z), Z = f_2(X).$
$add_{v_1 Z}(X) \leftarrow add_{v_1}(X),\ not\ aux_{v_1 Z}(X).$
$aux_{v_1 Z}(X) \leftarrow var_{v_1 Z}(X,Y,Z), Z = f_1(X).$
$\%V_2$
$P(X, Y, f_3(X,Y), \mathbf{v_2}) \leftarrow add_{v_2}(X,Y), add_{v_2 Z}(X,Y).$
$add_{v_2}(X,Y) \leftarrow v_2(X,Y),\ not\ aux_{v_2}(X,Y).$
$aux_{v_2}(X,Y) \leftarrow var_{v_2 Z}(X,Y,Z).$
$var_{v_2 Z}(X,Y,Z) \leftarrow P(X,Y,Z,\mathbf{nv_2}).$
$add_{v_2 Z}(X,Y) \leftarrow add_{v_2}(X,Y),\ not\ aux_{v_2 Z}(X,Y).$
$aux_{v_2 Z}(X,Y) \leftarrow var_{v_2 Z}(X,Y,Z).$
$P(X,Y,Z,\mathbf{nv_1}) \leftarrow P(X,Y,Z,\mathbf{v_2}).$
$P(X,Y,Z,\mathbf{nv_2}) \leftarrow P(X,Y,Z,\mathbf{v_1}).$
$P(X,Y,Z,\mathbf{t_d}) \leftarrow P(X,Y,Z,\mathbf{v_1}).$
$P(X,Y,Z,\mathbf{t_d}) \leftarrow P(X,Y,Z,\mathbf{v_2}).$
$S(Y,\mathbf{t_d}) \leftarrow S(Y,\mathbf{v_1}).$
□

The stable models of the refined program with Skolem functions are calculated under the *unique names assumption* [40]. As a consequence of this, the program may not be able to distinguish those cases where the openness condition for a source can be satisfied because the condition already holds for another source (see the discussion at the end of Section 3.1). For example, if two atoms, say $P(a, f1(a), f2(a))$ and $P(a, e, f3(a,e))$, are added to the stable models in order to satisfy the openness conditions for two different views, the program will treat those two atoms as different, what may not be the case when the Skolem functions are interpreted. As a consequence, stable models that are larger than needed might be produced. If each of these stable models is seen as a compact representation of a set of intended global instances, which can be recovered through all possible instantiations of the Skolem functions in the model, we may end up generating global instances that are not minimal. In other words, the class of stable models of the refined program with Skolem functions represents a

class that possibly properly extends the one of minimal instances, by including global instances that are legal but not minimal.

Example 22. (example 21 continued) The minimal instances of this integration system can be represented by $\{\{P(a,e,\, f_3(a,e)),\, P(a, f_1(a), f_2(a)),\, S(f_1(a))\} \mid$ $f_3(a,e) \in \mathcal{U},\, f_2(a) \in \mathcal{U},\, f_1(a) \in \mathcal{U} \smallsetminus \{e\}\} \bigcup \{\{P(a,e,f_3(a,e)),\, S(e)\} \mid f_3(a,e) \in$ $\mathcal{U}\}$. By interpreting the Skolem functions in the underlying domain, we obtain all and only the minimal instances. Notice that in this case, it is necessary to give all the possible values in the domain to the existential variables (or function symbols), the only exception being when the existential variable Y_1 is made equal to e. In that case it is good enough to give values to Z_1 *or* Z_2 in order to satisfy the openness conditions for V_1 and V_2.

In the context of the refined program with function symbols, due to the unique names assumption, $f_1(a)$ will always be considered different from e, and therefore the program will not realize that there is a minimal model that does not contain the tuple $P(X, f_1(X), f_2(X), v_1)$. In consequence, the program will generate the stable model $\{P(a,e,f_3(a,e)),\ P(a, f_1(a), f_2(a)),\ S(f_1(a))\}$, that represents a proper superclass of the minimal legal instances. For example, it represents the instance $\{P(a,e,u), P(a,e,v), S(e)\}$ that is not minimal. □

The possibly strict superset of the minimal instances that is represented by the models of the program with functions can be used to correctly compute the minimal and certain answers to monotone queries (in this case it is better to use the simple program though), but not for queries with negation.

We now consider the repair program. In those cases where the stable models of the simple or revised programs with Skolem functions do not represent the minimal legal instances, it is clear that it is not possible to compute their repairs. When the stable models do represent the minimal legal instances, it is not possible for the repair program to detect all the inconsistencies in them because of the underlying unique names assumption.

Example 23. (examples 4 and 5 continued) The minimal legal instances are represented via Skolem functions by $\mathcal{M} = \{P(a, f(a,b)), R(f(a,b),b), P(a,c)\}$, which can be obtained as a model of by the simple program with Skolem functions. This model is inconsistent wrt $IC\colon \forall x \forall y (R(X,Y) \rightarrow R(Y,X))$.

The repair program $\Pi(\mathcal{G}, IC)$ has the rule

$$R(X,Y,\mathbf{f_a}) \vee R(Y,X,\mathbf{t_a}) \leftarrow R(X,Y,\mathbf{t^\star}), R(Y,X,\mathbf{f^\star}).$$

that will produce the set of repairs $D_{\mathcal{M}_1} = \{P(a, f(a,b)), P(a,c)\}$ and $D_{\mathcal{M}_2} =$ $\{P(a, f(a,b)), R(f(a,b),b), R(b, f(a,b)), P(a,c)\}$, which represent a superset of the real repairs of the minimal legal instances. Because of the unique names assumption, the program will not detect that for $f(a,b) = b$ the instance is consistent wrt IC. □

Additional remarks on this issue can be found in [8].

6.4 Disjunctive Sources

In Section 3 we considered sources defined as conjunctive views only. If sources are now described as disjunctive views, i.e. with more than one conjunctive rule [20], then the program $(\mathcal{G})$ has to be extended in order to capture the minimal instances. In this case, a *source* S_i is a pair $\langle\ _i, v_i\rangle$, where $_i$ is a set of conjunctive rules defining the same view, say $\varphi_{i1},\ldots,\varphi_{im}$, and v_i is the given extension of the source.

Definition 15. Given an open global system $\mathcal{G} = \{\langle\ _1, v_1\rangle, \ldots, \langle\ _n, v_n\rangle\}$, the set of legal global instances is $Linst(\mathcal{G}) = \{D$ instance over $\mathcal{R} \mid v_i \quad \bigcup_k \varphi_{ik}(D)$, for $i = 1,\ldots,n\}$. □

Example 24. Consider the global integration system $\mathcal{G}_7$ with global relations $\{R(X,Y), S(X), T(X,Y)\}$ and two source relations v_1 and v_2 with the following view definitions and extensions:

Source	Extension	View Definitions
v_1	$\{V_1(a,b), V_1(c,d)\}$	$\mathcal{V}_{11} : V_1(X,Y) \quad R(X,Y), S(Y)$ $\mathcal{V}_{12} : V_1(X,d) \quad T(X,d)$
v_2	$\{V_2(b), V_2(a)\}$	$\mathcal{V}_{21} : V_2(X) \quad S(X)$

Examples of legal instances are $\{S(b), S(a), R(a,b), T(c,d)\}$, $\{S(b), S(a), R(a,b), R(c,d), S(d)\}$ and $\{S(b), S(a), R(a,b), T(c,d), T(a,b)\}$. □

If we have disjunctive view definitions, in order to satisfy the openness of a source, it is necessary that one or more views generate each of its tuples. To capture this, in [20] the concepts of *truly disjunctive* view and *witness* are introduced, together with an *exclusion condition*. Informally, a set of views is *truly disjunctive* if there is a tuple t that can be generated by any of the views. This tuple is called a *witness*. The *exclusion condition* is a constraint on the *witness* that determines for which tuples the *truly disjunctive* views are the most general.

Example 25. (example 24 continued) The atoms of v_1 that have the constant d as the second attribute can be generated either by $\mathcal{V}_{11}$ or $\mathcal{V}_{12}$. On the other hand, if the second attribute is not d, the atom can only be generated by $\mathcal{V}_1$. This is expressed in terms of truly disjunctive views, most general witness and exclusion condition by the following table:

truly disjunctive views	most general witness	exclusion condition
$\mathcal{V}_1$	(X_1, X_2)	second attribute $\neq d$
$\mathcal{V}_1, \mathcal{V}_2$	(X_1, d)	true

□

In order to extend the simple version of $(\mathcal{G})$, incorporating disjunctive view definitions, we need to take into account the different sets of truly disjunctive views with their witnesses and exclusion conditions. For example, for the second truly disjunctive set in Example 25, the following rule needs to be imposed

$$(R(X,d) \wedge S(d)) \vee T(X,d) \quad V(X,d), \tag{15}$$

which is equivalent to the pair of disjunctive Datalog rules

$$R(X,d) \vee T(X,d) \quad V(X,d) \tag{16}$$

$$S(d) \vee T(X,d) \quad V(X,d). \tag{17}$$

For each set of truly disjunctive views, rules like (16) and (17) will have to be satisfied by the legal instances. These remarks motivate the following program as an specification of the minimal legal instances.

Definition 16. Given an open global system $\mathcal{G}$, the program, ${}^{\vee}(\mathcal{G})$, contains the following clauses:
1. Fact $dom(a)$ for every constant $a \in \mathcal{U}$; and the fact $V_i(a)$ whenever $a \in v_i$ for some source extension v_i in $\mathcal{G}$.
2. For every set of truly disjunctive views for a source V_i of the form
$\mathcal{V}_{i1} : \quad V_i(X_1) \quad P_{11}(X_{11}), \ldots, P_{1n}(X_{1n_1})$

$\mathcal{V}_{ik} : \quad V_i(X_k) \quad P_{k1}(X_{k1}), \ldots, P_{kn}(X_{kn_k})$,
where the variables in each view are different (fresh), for its more general witness W and its most general exclusion condition φ, the rules
$P_{1\delta_1}(X'_{1\delta_1}) \vee \quad \vee P_{k\delta_k}(X'_{k\delta_k}) \quad V_i(W) \wedge \varphi \wedge \bigwedge_{Z_l \in (X' \setminus W)} F_i^l(W, Z_l)$,
where $X' = \bigcup_{j=1}^{k} X'_{j\delta_j}$ and $\delta_l \in \{1,\ldots,n_k\}$ for $l = 1,\ldots,k$.
The vectors $X'_{1\delta_1},\ldots,X'_{k\delta_k}$ are those obtained by the substitution of X_i by W in all the view definitions. These rules represent all the possible combinations of k predicates where each of them is chosen from a different view definition.
3. For every predicate $F_i^l(X, Z_l)$ introduced in 2., the rule
$F_i^l(X, Z_l) \quad V_i(X), dom(Z_l), choice((X),(Z_l))$. □

Example 26. (example 25 continued) The program ${}^{\vee}(\mathcal{G}_7)$ is:

$$dom(a).\ dom(b). \quad dom(c).\ dom(d). \tag{18}$$

$$R(X,Y) \quad V_1(X,Y), Y \neq d. \tag{19}$$

$$S(Y) \quad V_1(X,Y), Y \neq d. \tag{20}$$

$$T(X,d) \vee R(X,Y) \quad V_1(X,Y). \tag{21}$$

$$T(X,d) \vee S(Y) \quad V_1(X,Y). \tag{22}$$

$$S(X) \quad V_2(X). \tag{23}$$

Rules (19)-(20) and (21)-(22) represent, respectively, the first and second truly disjunctive set for source v_1. Rule (23) is for the non-disjunctive source v_2. □

If all the sources are defined by conjunctive views, then is easy to see that ${}^{\vee}(\mathcal{G})$ becomes the simple program $(\mathcal{G})$ introduced in Section 3.1. As before, it holds that

$$Mininst(\mathcal{G}) \quad \{D_{\mathcal{M}} \mid \mathcal{M} \text{ is a stable model of } {}^{\vee}(\mathcal{G})\} \quad Linst(\mathcal{G}).$$

For monotone queries Q, the answers obtained using ${}^{\vee}(\mathcal{G})$ coincide with $Certain_{\mathcal{G}}(Q)$ and $Minimal_{\mathcal{G}}(Q)$. This might not be the case of queries with negation. It is possible to give a refined version, corresponding to the non disjunctive program in Section 3.2, for which $Mininst(\mathcal{G}) = \{D_{\mathcal{M}} \mid \mathcal{M}$ is a stable model of ${}^{\vee}(\mathcal{G})\}$ also holds.

7 Conclusions

We have presented a general approach to specifying, by means of disjunctive deductive databases with 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 (without considering any global ICs at this level yet). 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. The 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 specification of the minimal legal instances of the integration system allows obtaining the *minimal answers* to arbitrary queries; and the *certain answers* to monotone queries, what extends previous results in the literature related to query plan generation under the LAV approach.

The methodology for specifying minimal legal instances, computing certain answers and CQA works for conjunctive view definitions and disjunctions of them. Wrt the ICs and queries this approach can handle, the solution is sound and complete for combinations of universal ICs and acyclic referential ICs, and queries expressed as Datalog$^{\neg}$ programs. In consequence, the current approach to consistent query answering (CQA) subsumes and extends the methodologies presented in [9] for integration systems, and the one in [6] for stand alone relational databases. Also the complexity of query evaluation using the logic programs presented here matches the theoretical lower bounds for computing certain and consistent answers.

For reasons of space, we just mention a few optimizations of the specification programs and their execution (more on optimization of repair programs can be found in [6]). The materialization of the CWA present in $(\mathcal{G}, IC)$ can be avoided by program transformation. We have identified classes of common ICs for which $SV(\ (\mathcal{G}, IC))$ becomes head-cycle-free, and in consequence, can be transformed into a non disjunctive program [7, 34]. Transformations are shown in [6].

The program for CQA can be split [37] into: (1) the program that specifies minimal legal instances; (2) the program that specifies their repairs; and (3) the query program. If the simple version can be used in (1), that layer is a stratified program. Otherwise, if the refined version is used, that layer is not stratified, but its models can be computed bottom-up as fixpoints of an iterative operator [27]. The second layer, i.e. the repair part, is *locally stratified* [39]. 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.

We have already indicated that in the case the set of ICs contain referential ICs with cycles between them the stable models of the specification programs we gave may correspond to a superclass of the repairs of the global system [12]. Non minimal repairs may appear as models of the program. It should be possible to modify the given program by adding a new layer of rules that does the job of pruning all the stable models of the original program that do not correspond to (minimal) repairs. In this direction the answer set programming based specification of some "local test" for minimality as given in [38] (and used in [11] in the context of database repairs) could be attempted.

For CQA from integration systems we have successfully experimented with *DLV* [22, 35]. 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. First optimizations of this kind have been reported in [23].

Wrt related papers, query answering in mediated integration systems *under the assumption* that certain global ICs hold has been treated in [31, 21, 29, 14]. However, in CQA, we do not assume that global ICs hold. Logic programming specifications of repairs of single relational databases have been presented in [4, 30, 5].

In [9], 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 [3] 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$^{\neg}$ queries. This approach is limited by the restrictions of the query transformation methodology. In particular, it can be applied only to queries that are conjunctions of literals and universal ICs.

Integration systems under the GAV approach that do not satisfy global key dependencies are considered in [32]. 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 [16] and still under the GAV approach, this work is extended by introducing rewriting techniques to retrieve the consistent query answers without constructing the "repairs". More related work is discussed in the survey [8].

With respect to current and future work, apart from considering all kinds of implementation and optimization issues around the programs and their interaction with a database, we have extended [8] our treatment of CQA in integration systems to the mixed case where open, closed and sources that are both open and closed (clopen) coexist [28]; and to particular, but common and natural combinations of them. We are working on identifying conditions on the view definitions that make it possible to compute, from the program $(\mathcal{G})$, the certain answers to possibly non-monotonic queries.

In this paper we have considered null values based repairs under RICs. The null values have a special treatment wrt to satisfaction of ICs, and as a consequence, they do not propagate in the repair process. In [4, 5, 15], repairs of RICs using normal domain values are considered. This, under cyclic sets of RICs, may lead to undecidability of consistent query answering. It would be interesting to study some sort of mixed approach, and also the possibility of limited propagation of null values.

Research related to the design of virtual data integration systems and its impact on global query answering has been mostly neglected. Most of the research in the area starts from a given set of view definitions, but the conditions on them hardly go beyond classifying them as conjunctive, disjunctive, Datalog, etc. However, other conditions, imposed when the systems is being designed, could have an impact on, e.g. query plan derivation. Much research is needed in this direction.

Acknowledgments: Research funded by DIPUC, CONICYT, Carleton University Start-Up Grant 9364-01, NSERC Grant 250279-02, and a grant from the CITO/IBM-CAS Student Internship Program. L. Bertossi is Faculty Fellow of the IBM Center for Advanced Studies, Toronto Lab. We also appreciate a CoLogNet Scholarship for Loreto Bravo to attend the Workshop on Logic-Based Method for Information Integration (Vienna, August 2003). We are grateful to Alberto Mendelzon, Pablo Barceló, Jan Chomicki, Enrico Franconi, Andrei Lopatenko for useful conversations, and to the anonymous referees for [13], a first version of this paper, for useful remarks.

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. ACM Symposium on Principles of Database Systems (PODS 99)*, 1999, pp. 68–79.
4. Arenas, M., Bertossi, L. and Chomicki, J. Answer Sets for Consistent Query Answers. Theory and Practice of Logic Programming, 2003, 3(4-5), pp. 393-424.
5. Barcelo, P. and Bertossi, L. Logic Programs for Querying Inconsistent Databases. In *Proc. Fifth International Symposium on Practical Aspects of Declarative Languages (PADL 03)*. Springer Lecture Notes in Computer Science 2562, 2003, pp. 208–222.

6. 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.
7. Ben-Eliyahu, R. and Dechter, R. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics in Artificial Intelligence*, 1994, 12:53-87.
8. Bertossi, L. and Bravo, L. Consistent Query Answers in Virtual Data Integration Sistems. To appear as a book chapter in 'Inconsistency Tolerance in Knowledgebases, Databases and Software Specifications'. Springer.
9. Bertossi, L., Chomicki, J., Cortes, A. and Gutierrez, C. Consistent Answers from Integrated Data Sources. In *Proc. Flexible Query Answering Systems (FQAS 02)*, Springer LNAI 2522, 2002, pp. 71–85.
10. Bertossi, L. and Chomicki, J. Query Answering in Inconsistent Databases. In 'Logics for Emerging Applications of Databases', J. Chomicki, G. Saake and R. van der Meyden (eds.), Springer, 2003.
11. Bertossi, L. and Schwind, C. Database Repairs and Analytic Tableaux. *Annals of Mathematics and Artificial Intelligence*, 2004, 40(1-2): 5-35.
12. Bertossi, L. and Bravo, L. In preparation, 2004.
13. Bravo, L. and Bertossi, L. Logic Programs for Consistently Querying Data Sources In *Proc. of the Eighteenth International Joint Conference on Artificial Intellience (IJCAI 03)*, Morgan Kaufmann, 2003, pp. 10–15.
14. 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.
15. Cali, A., Lembo, D. and Rosati, R. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *Proc. ACM Symposium on Principles of Database Systems (PODS 03)*, 2003, pp. 260-271.
16. Cali, A., Lembo, D., and Rosati, R. Query Rewriting and Answering under Constraints in Data Integration Systems. In *Proc. of the Eighteenth International Joint Conference on Artificial Intellience (IJCAI 03)*. Morgan Kaufmann, 2003, pp. 16-21.
17. Celle, A. and Bertossi, L. Querying Inconsistent Databases: Algorithms and Implementation. In *'Computational Logic - CL 2000'. Stream: 6th International Conference on Rules and Objects in Databases (DOOD 00)*, Springer Lecture Notes in Artificial Intelligence 1861, 2000, pp. 942–956.
18. Chomicki, J. and Marcinkowski, J. Minimal-Change Integrity Maintenance Using Tuple Deletions. arXiv.org paper cs.DB/0212004. To appear in *Information and Computation*.
19. Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. Complexity And Expressive Power Of Logic Programming. *ACM Computer Surveys*. 2001, 33(3), 374-425.
20. Duschka, O. *Query Planning and Optimization in Information Integration*. PhD Thesis, Stanford University, December 1997.
21. Duschka, O., Genesereth, M. and Levy, A. Recursive Query Plans for Data Integration. *Journal of Logic Programming*, 2000, 43(1):49-73.
22. 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.
23. Eiter, T., Fink, M., Greco, G. and Lembo, D. Efficient Evaluation of Logic Programs for Querying Data Integration Systems. In *Proc. 19th International Conference on Logic Programming (ICLP 03)*, Springer LNCS 2916, 2003, pp. 163-177.
24. Gelfond, M. and Lifschitz, V. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 1991, 9:365–385.

25. Giannotti, F., Greco, S., Sacca, D. and Zaniolo, C. Programming with Nondeterminism in Deductive Databases. *Annals of Mathematics and Artificial Intelligence*, 1997, 19(1-2):97–125.
26. Giannotti, F., Pedreschi, D., Sacca, D. and Zaniolo, C. Non-Determinism in Deductive Databases. In *Proc. International Conference on Rules and Objects in Databases (DOOD 91)*, Springer LNCS 566, 1991, pp. 129–146.
27. Giannotti, F., Pedreschi, D. and Zaniolo, C. Semantics and Expressive Power of Nondeterministic Constructs in Deductive Databases. *J. Computer and System Sciences*, 2001, 62(1):15–42.
28. Grahne, G. and Mendelzon, A. Tableau Techniques for Querying Information Sources through Global Schemas. In *Proceedings International Conference on Database Theory (ICDT 99)*, Springer LNCS 1540, 1999, pp. 332–347.
29. Grant, J. and Minker, M. A Logic-based Approach to Data Integration. *Theory and Practice of Logic Programming*, 2002, 2(3):323-368.
30. Greco, G., Greco, S. and Zumpano, E. A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. In *Proc. International Conference on Logic Programming (ICLP 01)*, Springer LNCS 2237, 2001, pp. 348–364.
31. Gryz, J. Query Rewriting Using Views in the Presence of Functional and Inclusion Dependencies. *Information Systems*, 1999, 24(7):597–612.
32. Lembo, D.; Lenzerini, M. and Rosati, R. Source Inconsistency and Incompleteness in Data Integration. In *Proc. Workshop on Knowledge Representation meets Databases (KRDB 02)*, 2002.
33. Lenzerini, M. Data Integration: A Theoretical Perspective. In *Proc. ACM Symposium on Principles of Database Systems (PODS 02)*, 2002, pp. 233-246.
34. Leone, N., Rullo, P. and Scarcello, F. Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics, and Computation. *Information and Computation*, 1997, 135(2):69-112.
35. 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*.
36. Levy, A. Logic-Based Techniques in Data Integration. Chapter in 'Logic Based Artificial Intelligence', J. Minker (ed.), Kluwer, 2000, pp. 575-595.
37. Lifschitz, V. and Turner, H. Splitting a Logic Program. In *Proc. International Conference on Logic Programming (ICLP 94)*. The MIT Press, 1994, pp. 23–37.
38. Niemela, I. Implementing Circumscription Using a Tableau Method. In *Proc. European Conference on Artificial Intelligence (ECAI 96)*, 1996, pp. 80–84.
39. Przymusinski, T. Stable Semantics for Disjunctive Programs. *New Generation Computing*, 1991, 9(3/4):401–424.
40. Reiter, R. Towards a Logical Reconstruction of Relational Database Theory. In 'On Conceptual Modeling', M.L. Brodie, J. Mylopoulos, and J.W. Schmidt (eds.). Springer-Verlag, 1984, pp. 191-233.
41. Wang, H. and Zaniolo, C. Nonmonotonic Reasoning in $\mathcal{LDL}^{++}$. In 'Logic-Based Artificial Intelligence', J. Minker (ed.), Kluwer 2000, pp. 523-544.

A Appendix

A.1 Proof of Results

Proof of Theorem 1: Consider (G) as in Definition 10. First we prove:

$$\{D_{\mathcal{M}} \mid \mathcal{M} \text{ is a choice model of } \quad (\mathcal{G})\} \quad Linst(\mathcal{G}). \tag{24}$$

Assume that there is a stable model $\mathcal{M}$ of $(\mathcal{G})$ such that its associated database $D_{\mathcal{M}}$ is not a legal instance. Then there is a view V_i for which $v_i\ /\ \varphi_i(D_{\mathcal{M}})$, that is, for some a:

- $a \in v_i$, and then by rules 1. of (G), $V_i(a)$ is true in any model of the program, in particular, in $\mathcal{M}$.
- $a \notin \varphi_i(D_{\mathcal{M}})$, i.e. in $\mathcal{M}$, it holds $\neg\exists z(P_1(a_1, z_1)\wedge\ldots\wedge P_n(a_n, z_n))$, for $a_i \quad a$, and $z_i \quad z$. This is equivalent to

$$\forall z\ \ (\neg P_1(a_1, z_1) \vee \ldots \vee \neg P_n(a_n, z_n)) \tag{25}$$

A consequence of (25) and rules 2. of $(\mathcal{G})$ is the following:

$$\forall z\ \ (\neg V_i(a) \vee \bigvee_l \neg F_i^l(a, z_l)). \tag{26}$$

Since $V_i(a) \in \mathcal{M}$ and rules 3. of (G) are satisfied by $\mathcal{M}$ we have that for some b's in the domain the atoms $F_i^l(a, b) \in \mathcal{M}$. But we had that equation (26) holds. We have reached a contradiction because (26) is false in $\mathcal{M}$; and (24) is proven.

Now we want to prove: $Mininst(\mathcal{G}) \quad \{D_{\mathcal{M}} \mid \mathcal{M}$ is a choice model of $(\mathcal{G})\}$.

The program $(\mathcal{G})$ can be split [37] into the bottom program $_B$, that contains the facts and rules in 1. and 3. of (G), and the top program, $_T$, that contains the rules in 2.. If $\mathcal{M}_B$ is a stable model of $_B$ and $\mathcal{M}_T^B$ is a stable model of $_T^{\mathcal{M}_B}$ (the top program partially evaluated by the atoms in $\mathcal{M}_B$), then $\mathcal{M}_B \cup \mathcal{M}_T^B$ is a stable model of $(\mathcal{G})$, and all the models of latter can be obtained in this way. The bottom program contains the choice operator and therefore its stable models will correspond to all the possible combinations of values for the Skolem predicates subject to the condition of functionality [41]. Since $_T^{\mathcal{M}_B}$ is a non-disjunctive-positive program (without the choice operator), there will be a unique stable model for each $\mathcal{M}_B$ that will correspond to its minimal model.

We will now prove that every minimal legal instance is of the form $D_{\mathcal{M}}$, where $\mathcal{M}$ is of the form $\mathcal{M}_B \cup \mathcal{M}_T^B$ with $\mathcal{M}_B$ a stable model of $_B$ and $\mathcal{M}_T^B$ a minimal model of $_T^{\mathcal{M}_B}$.

Let D be a minimal legal instance of $\mathcal{G}$. Let us define a structure $\mathcal{M}$ for the program $(\mathcal{G})$ containing the following ground atoms:

1. The atoms in D;
2. $V_i(a)$ whenever $a \in v_i$, where v_i is a source extension in $\mathcal{G}$;
3. $dom(a)$ for every constant $a \in \mathcal{U}$;
4. For each view $V_i(x)$, consider the rules $F_i^l(x, z_l) \quad body(\varphi_{V_i})$, for each variable z_l from the body that does not belong to x. Evaluate the bodies according to the atoms in 1. When the body is true, add to $\mathcal{M}$ the corresponding atom in the head.
5. If for a view V_i, $a \in v_i$ and $F_i^l(a, b) \in \mathcal{M}$, add $choice(a, b)$ to $\mathcal{M}$.

Note that $D_{\mathcal{M}} = D$. Now we have to prove that the structure $\mathcal{M}$ is a stable model of $(\mathcal{G})$. This can be shown by proving, first, that $\mathcal{M}_B := (\mathcal{M} \smallsetminus D)$ is a stable model of $_B$, and, next, that $\mathcal{M}_T^{\mathcal{M}_B} = D$ is a minimal model of $_T^{\mathcal{M}_B}$.

${}_B$ contains rules 1. and 3. of $(\mathcal{G})$. By construction $\mathcal{M}_B$ will satisfies rules 1. For $\mathcal{M}_B$ to satisfy rules 3. it is sufficient to prove that for each $V_i(a) \in \mathcal{M}_B$ there is exactly one $F_i^l(a,b) \in \mathcal{M}_B$ with $b \in \mathcal{U}$ for each z_l and that if $V_i(a) \notin \mathcal{M}$ then there is no $F_i^l(a,z)$ in $\mathcal{M}_B$. This is enough because it is proven that the choice operator will enforce that $F_i^l(x,z)$ satisfies a functional dependency between x and z.

Let us suppose by contradiction that for $V_i(a) \in \mathcal{M}_B$ there are two atoms $F_i^l(a,b_1) \in \mathcal{M}_B$ and $F_i^l(a,b_2) \in \mathcal{M}_B$. This would imply by construction of $\mathcal{M}$ that the following rules are satisfied by evaluating the bodies with the elements of D: $F_i^l(a,b_l) \quad body(\varphi_{V_i})$ and $F_i^l(a,b_2) \quad body(\varphi_{V_i})$. This would imply that D has two set of atoms satisfying the mapping $V_i(a) \quad body(\varphi_{V_i})$ and therefore D is not minimal. Since D is minimal we have reached a contradiction.

Now we have to prove that if $V_i(a) \notin \mathcal{M}$ then there is no $F_i^l(a,z)$ in $\mathcal{M}_B$. Let us suppose by contradiction that there for a given value $b \in \mathcal{U}$, $F_i^l(a,b) \in \mathcal{M}_B$. This would imply by construction of $\mathcal{M}$ that it holds, by evaluating the bodies with the elements of D, $F_i^l(a,b) \quad body(\varphi_{V_i})$. This implies that D satisfies $body(\varphi_{V_i})$ without $V_i(a)$ belonging to the source. Then D is not minimal. Since D is minimal we have reached a contradiction. This proves that $\mathcal{M}_B := (\mathcal{M} \setminus D)$ is a stable model of ${}_B$. Now we have to prove that D is a minimal model of ${}_T^{\mathcal{M}_B}$.

The program ${}_T^{\mathcal{M}_B}$ contains only facts of the form $\quad V_i(a,b) \qquad$ where $V_i(a) \in \mathcal{M}_B$ and b is constructed from all the function predicates $F_i^l(a,b_1) \in \mathcal{M}_B$. By construction this facts are exactly the elements of D. Then, D is a minimal model of ${}_T^{\mathcal{M}_B}$. This proves that $\mathcal{M}$ is a stable model of $(\mathcal{G})$ and since $D_{\mathcal{M}} = D$ we have that every minimal legal instance has a stable model of $(\mathcal{G})$ associated. □

Proof of Theorem 2: Let us suppose by contradiction that we have an integration system G that has no admissible mapping h for S_i^l (with $i \neq 0$), such that $h(S_i^l) \quad \bigcup_{S \in (Sec \setminus \{S_i^l\})} L(S)$, and that there is a stable model $\mathcal{M}$ of the simple version of $(\mathcal{G})$ such that the database associated $D_{\mathcal{M}}$ is not a minimal legal instance.

Since $D_{\mathcal{M}}$ is not minimal, there is a minimal legal instance E such that $E \subsetneqq D_{\mathcal{M}}$. From Theorem 1 we have that there is a model $\mathcal{M}'$ of $(\mathcal{G})$ such that $D_{\mathcal{M}'} = E$. Then there should be a non empty set $\mathcal{C}$, such that $\mathcal{C} \in \mathcal{M}$ and $\mathcal{C} \notin \mathcal{M}'$.

From the proof of Theorem 1 we have that the program $(\mathcal{G})$ can be divided into two parts ${}_B$ and ${}_T^{\mathcal{M}_B}$, where the second is a result of an evaluation of the model $\mathcal{M}_B$ of ${}_B$ over the rules of $(\mathcal{G})$ that do not belong to ${}_B$. The interesting thing is that the program ${}_T^{\mathcal{M}_B}$ turns out to be a set of facts of global relations. This shows that the different models will be determined only by the functional predicates atoms of the form $F_i^l(a,b)$ chosen in each model. Each of this atom will generate exactly one global atom for each relation that has the existential variable z_l in the view V_i. Then, we have that the only way that one model might generate a legal instance of $\mathcal{G}$ with less elements than other model is

if two functional predicate atoms generate the same global atom. Then, $\mathcal{C}$ has to be formed by instantiations of sections with existential variables. For simplicity and without lost of generality let us suppose that $\mathcal{C}$ has exactly one instantiation of one section. For $\mathcal{C}$ to belong to $\mathcal{M}$ and not to $\mathcal{M}'$, $\mathcal{M}$ should have different values of the existential variables that generate the instantiations of $\mathcal{C}$ than the ones assigned in $\mathcal{M}$ and the rest values should be the same (since $D_{\mathcal{M}'} \subsetneqq D_{\mathcal{M}}$). Furthermore, the values given in $\mathcal{M}'$ should generate the same set of predicates that another section or sections generates in $\mathcal{M}$ and in $\mathcal{M}'$. Then, if $\mathcal{C}$ is the instantiation of a section S_i^l, we have that the following has to hold for every value a_k in position k of the atom $P(a) \in \mathcal{C}$, being this atom an instantiation of $P(x_1, \ldots, x_k, \ldots, x_n) \in S_i^l$:

1. If $x_k \in Const(S_i^l)$ then there is a different section S_j^m such that $P(\ldots, x_k, \ldots) \in S_j^m$ and $x_k \in Const(S_j^m)$ and $x_k = a_k$.
2. If $x_k \in UVar(S_i^l)$ then there are two options:
   (a) There is other section S_j^m such that $P(\ldots, x_k, \ldots) \in S_j^m$, $x_k \in Const(S_j^m)$ and $x_k = a_k$.
   (b) There is other section S_j^m such that $P(\ldots, x_k, \ldots) \in S_j^m$, $x_k \in UVar(S_j^m)$ and $(\ldots, a_k, \ldots) \in v_j$.
3. If $x_k \in EVar(S_i^l)$ then there are three options:
   (a) There is other section S_j^m such that $P(\ldots, x_k, \ldots) \in S_j^m$, $x_k \in Const(S_j^m)$ and $x_k = a_k$.
   (b) There is other section S_j^m such that $P(\ldots, x_k, \ldots) \in S_j^m$, $x_k \in UVar(S_j^m)$ and $(\ldots, a_k, \ldots) \in v_j$.
   (c) There is other section S_j^m such that $P(\ldots, x_k, \ldots) \in S_j^m$, $x_k \in EVar(S_j^m)$ and $F_j^k(b, a_k) \in \mathcal{M}'$ for $(b) \in v_j$.

Consider a mapping h defined by the different cases just described, for example if we are in case (2b) we have that $h(x_k) = \mu$ and in case (3a) we have that $h(x_k) = a_k$. By construction this mapping is such that $h(S_i^l) \quad \bigcup_{S \in (Sec \setminus \{S_i^l\})} L(S)$. We have reached a contradiction since we assumed the mapping h did not exists. Therefore we have proved Theorem 2. □

The following intermediate results refer to the refined program $(\mathcal{G})$ introduced in Section 3.2.

Lemma 1. If $\mathcal{M}$ is a stable model of $SV(\quad(\mathcal{G}))$, then $D_{\mathcal{M}}$ is a legal instance of $\mathcal{G}$.

Proof. In the proof we use the same notation as in the Definition 12 of $(\mathcal{G})$. Assume that $D_{\mathcal{M}}$ is not legal. Then there must be a view V_i, with definition $\varphi_i:\ V_i(x) \quad \bigwedge_{u=1}^{n} P_u(x_u, z_u)$, for which $v_i \ / \ \varphi_i(D_{\mathcal{M}})$. More specifically, there is a such that $a \in (v_i \setminus \varphi_i(D_{\mathcal{M}}))$. If $a \in v_i$ then $V_i(a) \in \mathcal{M}$.

For every global relation P_u without existential variables in the view definition φ_i, we can conclude from rules (3a) of $(\mathcal{G})$ that $P_u(a_u, \mathbf{t_o}) \in \mathcal{M}$ with $a_u \quad a$. Then, by rules (5), $P_l(a_u, \mathbf{t_d}) \in \mathcal{M}$ and therefore $P_u(a_u) \in D_{\mathcal{M}}$.

Now we will analyze the case of global relation with existential variables treated by rules defined in (3b). For a certain S_{ij}, in order to satisfy the second rule of (3b), we have to analyze two cases:

1. $V_i(a) \in \mathcal{M}$ and $aux_{v_{ij}}(a') \notin \mathcal{M}$. Then, $add_{v_{ij}}(a') \in \mathcal{M}$. From the third rule of (3b) we have that there exists a non-empty set $\mathcal{L}$ such that $var_{v_{ij}Z_l}(a_{Z_l}) \notin \mathcal{M}$ for $l \in \mathcal{L}$. Now let us take a look at rules in (4). ¿From the 3^{rd} rule, we have that for every $l \in \mathcal{L}$, $aux_{v_{ij}Z_l}(a') \notin \mathcal{M}$. Then, from the 2^{nd} rule and since $add_{v_{ij}}(a') \in \mathcal{M}$ we have that for every $l \in \mathcal{L}$, $add_{v_{ij}Z_l}(a') \in \mathcal{M}$. Now, from the first rule, the choice operator will assign one value of the domain to Z_l, e.g. b_l for each $l \in \mathcal{L}$. Then we will have $F_i^l(a', b_l) \in \mathcal{M}$ for every $l \in \mathcal{L}$. Now let us have a look at the rules in (3b). For $P_k \in S_{ij}$, there are two cases to analyze wrt the first rule:
   (a) $\{Z_l | Z_l \in (X_k \setminus X')\} \quad \{Z_l | l \in \mathcal{L}\}$. Then $P_k(a_k, \mathbf{v_{ij}}) \in \mathcal{M}$ where a_k is a projection of a and the b_l of the functional predicates. Hence $P_k(a_k, \mathbf{t_d}) \in \mathcal{M}$ and therefore $P_k(a_k) \in D_{\mathcal{M}}$.
   (b) $\{Z_l | Z_l \in (X_k \setminus X')\} \ / \ \{Z_l | l \in \mathcal{L}\}$. For every $Z_{l'} \in \{\{Z_l | Z_l \in (X_k \setminus X')\} \setminus \{Z_l | l \in \mathcal{L}\}\}$ we have that since $l' \notin \mathcal{L}$, $var_{v_{ij}Z_{l'}}(a_{Z_{l'}}) \in \mathcal{M}$. Since the only way for an atom to belong to a model is to have a rule with it in the head and the body satisfied, we have that the body of the fourth rule of (3b) has to be true. This implies that $P_k(a_k, \mathbf{nv_{ij}}) \in \mathcal{M}$. We also have that since $F_i^{l'}(a', b_l) \notin \mathcal{M}$ for any value of b_l, then $add_{v_{ij}Z_{l'}}(a') \notin \mathcal{M}$ and therefore $aux_{v_{ij}Z_{l'}}(a') \in \mathcal{M}$. Then because of the third rule of (3b) we have that the values associated to the existential variables that are not $Z_{l'}$ in $P_k(a_k, \mathbf{nv_{ij}})$ coincide with the values given by the functional predicates of the view. Since $P_k(a_k, \mathbf{nv_{ij}}) \in \mathcal{M}$ we have from rules in (5) that $P_k(a_k, \mathbf{nv_{hk}})$ (with $hk \neq ij$) or $P_k(a_k, \mathbf{t_o})$ belong to $\mathcal{M}$ and therefore that $P_k(a_k, \mathbf{t_d}) \in \mathcal{M}$. Then $P_k(a_k) \in D_{\mathcal{M}}$ sharing the same existential variable that the ones generated by the previews case considered.

   Then we have that $a_{S_{ij}} \in \varphi_{iS_{ij}}(D_{\mathcal{M}})$[7]
2. $V_i(a) \in \mathcal{M}$ and $aux_{v_{ij}}(a') \in \mathcal{M}$. Then, $add_{v_{ij}}(a') \notin \mathcal{M}$. From the 3^{rd} rule of (3b) $var_{v_{ij}Z_l}(a_{Z_l}) \in \mathcal{M}$ for all Z_l. Then, from the fourth rule of (3b) $P_k(a_k, \mathbf{nv_{ij}}) \in \mathcal{M}$ for all $P_k \in S_{ij}$ such that $Z_l \in X_k$. From rules in (5),with $hk \neq ij$, $P_k(a_k, \mathbf{nv_{hk}})$ or $P_k(a_k, \mathbf{t_o})$ belong to $\mathcal{M}$ and therefore that $P_k(a_k, \mathbf{t_d}) \in \mathcal{M}$. Then $P_k(a_k) \in D_{\mathcal{M}}$. Then we have that $a_{S_{ij}} \in \varphi_{iS_{ij}}(D_{\mathcal{M}})$

Now, since the different S_{ij} do not share existential variables we have that $\varphi_i(D_{\mathcal{M}}) = \bowtie_{S_{ij} \in V_i} \ \varphi_{iS_{ij}}(D_{\mathcal{M}})$. Then since $a_{S_{ij}} \in \varphi_{iS_{ij}}(D_{\mathcal{M}})$, $a \in \varphi_i(D_{\mathcal{M}})$. We have reached a contradiction and the Lemma is proven. □

Lemma 2. If D is a minimal instance of $\mathcal{G}$, then there is a stable model $\mathcal{M}$ of $SV(\quad(\mathcal{G}))$, such that $D_{\mathcal{M}} = D$.

Proof. We need to define a Herbrand structure that will be our candidate to be the stable model $\mathcal{M}$ that generates instance D. For doing this, we use the same notation as in the Definition 12 of $(\mathcal{G})$. We put the following facts into $\mathcal{M}$:

[7] $a_{S_{ij}}$ corresponds to the atom a restricted to the variables of the view φ_i that belong to S_{ij}, and $\varphi_{iS_{ij}}$ is the view definition φ_i restricted to the predicates in S_{ij} and its variables

1. $P_k(a, \mathbf{t_d})$ for every global atom $P_k(a) \in D$. No other atom annotated with $\mathbf{t_d}$ belongs to $\mathcal{M}$.
2. $dom(a)$ iff $a \in \mathcal{U}$.
3. $V_i(a)$ iff $a \in v_i$ for $v_i \in \mathcal{G}$.
4. $P_k(a_k, \mathbf{t_o})$ iff there is a view $V_i(X) \quad P_1(X_1), \ldots, P_k(X_k), \ldots P_n(X_n)$, in which P_k has no existential variables and such that $a \in v_i$.
5. For every atom $P_k(a_k) \in D$, where $P_k(a_k, \mathbf{t_o}) \notin \mathcal{M}$, we need to check which views had the potential of generating it. After some considerations we will specify at the end of this item what new atoms go into $\mathcal{M}$ and which do not. We have that for each view section S_i^l with an existential variable z_l[8], such that $P_k \in S_i^l$, define the following views:

   $$P_k(X'_k, S_i^l) \qquad \bigwedge_{P_j(X_j) \in S_i^l} P_j(X_j) \wedge V_i(X),$$

   where S_i^l is considered as an annotation constant in the second argument of head of the view. This view will contain the information of which

   Let P be the result of instantiating these views over the atoms in D and the source extensions. P contains the possible section that might have generated the presence of each global atom in D. We will define $S^{P_k} = \{S_i^l \mid P_k(a_k, S_i^l) \in P\}$, i.e. S^{P_k} contains al the sections from which $P_k(a_k$ could have been generated. Note that there is only one S_{ij}[9] in $\mathcal{G}$ such that $S_{ij} \supseteq S_i^m$. Then, for each section $S_i^l \in S^{P_k}$ that does not have an admissible mapping [10] such that $h(S_i^l) \quad \bigcup_{S \in (Sec \setminus \{S_i^l\})} L(S)$ do the following: $P_k(a_k, \mathbf{v_{ij}}) \in \mathcal{M}$, $add_{v_{ij}}(a') \in \mathcal{M}$, $V_i(a) \in \mathcal{M}$, $aux_{v_{ij}}(a') \notin \mathcal{M}$, $var_{v_{ij}z_l}(a_{z_l}) \notin \mathcal{M}$, $aux_{v_{ij}Z_l}(a') \notin \mathcal{M}$, $add_{v_{ij}z_l}(a') \in \mathcal{M}$. For all the rest of the sections of S^{P_k}, e.g. S_i^m, we have that the $var_{v_{in}z_m}(a_{z_m}) \in \mathcal{M}$. If for all the sections in a view $var_{v_{in}z_m}(a_{z_m}) \in \mathcal{M}$ then $aux_{v_{ij}}(a') \in \mathcal{M}$ and $add_{v_{ij}}(a') \notin \mathcal{M}$.
6. For every $P_k(a_k, \mathbf{v_{ij}}) \in \mathcal{M}$, we add the fact $P_k(a_k, \mathbf{nv_{km}})$ to $\mathcal{M}$ for every $S_{km} \neq S_{ij}$.
7. For every $add_{v_{ij}z_l}(a'), P_k(a_k, \mathbf{v_{ij}}) \in \mathcal{M}$, add $F_i^l(X, z_l)$ into $\mathcal{M}$, where z_l is the value of that existential variable in $P_k(a_k, \mathbf{v_{ij}})$.

By construction $\mathcal{M}$ minimally satisfies rules (1), (2), (3a), (5) and the first rule of (3b) in the program $(\mathcal{G})^{\mathcal{M}}$. If $aux_{v_{ij}}(a') \in \mathcal{M}$, $(\mathcal{G})^{\mathcal{M}}$ does not include the second type of rules of (3b). If $aux_{v_{ij}}(a') \notin \mathcal{M}$, $(\mathcal{G})^{\mathcal{M}}$ has the rule $add_{v_{ij}}(X') \quad V_i(X)$ corresponding to second type of rules of (3b). This rule is satisfied by $\mathcal{M}$ because of the facts added to $\mathcal{M}$ in item 5. For the section S_i^l such have no admissible mapping such that $h(S_i^l) \quad \bigcup_{S \in (Sec \setminus \{S_i^l\})} L(S)$, we have that no other views can can generate the facts for this section and therefore that the body of the fourth rules in 3b will not be satisfied. Since in that case $var_{v_{ij}z_l}(a_{z_l}) \notin \mathcal{M}$, the whole rule is satisfied. For the sections that are not in this case, i.e there is an admissible mapping, then the body of the fourth rules in 3b

[8] The S_i^l are the view sections introduced in Section 3.1.
[9] Here the S_{ij} are those appearing in Definition 12.
[10] As defined in section 3.1

will be satisfied and since $var_{v_{ij} z_l}(a_{z_l}) \in \mathcal{M}$, the whole rule will be satisfied. If all the sections are in the situation last described, $aux_{v_{ij}}(a') \in \mathcal{M}$ and therefore the third rules in 3b will be satisfied. Following the same analysis and the fact that the choice operator will choose any value of the domain, it is easy to see that rules in (4) are also minimally satisfied. $\mathcal{M}$ is a minimal model of $(\mathcal{G})^{\mathcal{M}}$and therefore there is a stable model of $(\mathcal{G})$, $\mathcal{M}$, such that $D_{\mathcal{M}}$ corresponds to the minimal legal instance D. □

Lemma 3. If $\mathcal{M}$ is a stable model of $SV(\ (\mathcal{G}))$, then $D_{\mathcal{M}}$ is a minimal instance of $\mathcal{G}$.

Proof. The legality of $D_{\mathcal{M}}$ was established in Lemma 1. Assume, by contradiction that $D_{\mathcal{M}}$ is not a minimal instance of $\mathcal{G}$. Then there must be a minimal instance D such that $D \subsetneq D_{\mathcal{M}}$. By Lemma 2 we have that there is a model $\mathcal{M}'$ such that $D_{\mathcal{M}'} = D$. Then, $D_{\mathcal{M}'} \subsetneq D_{\mathcal{M}}$. In particular, we have that there is an atom of a global relation, say $P_k(a, \mathbf{t_d})$, such that $P_k(a, \mathbf{t_d}) \in \mathcal{M}$ and $P_k(a, \mathbf{t_d}) \notin \mathcal{M}'$. If $P_k(a, \mathbf{t_d}) \in \mathcal{M}$ we have two options:

1. $P_k(a, \mathbf{t_o}) \in \mathcal{M}$. Then there is a view v_i in which P_k has no existential variables. In that case $P_k(a, \mathbf{t_o})$ belongs to all the models and in particular to $\mathcal{M}'$. We have reached a contradiction since $P_k(a, \mathbf{t_d}) \notin \mathcal{M}'$
2. $P_k(a, \mathbf{v_{ij}}) \in \mathcal{M}$. This implies that $add_{v_{ij}}(a') \in \mathcal{M}$ and for all $a_l \in (a \setminus a')$, $F_i^l(a', a_l) \in \mathcal{M}$. Hence there is an atom $V_i(A) \in \mathcal{M}$ such that the first rule of (3b) is satisfied. We can also conclude that $var_{v_{ij} Z_l}(a_{Z_l}) \notin \mathcal{M}$. Then there is no other view that satisfies this section S_i^l. This implies that if $\mathcal{M}'$ does not contain $P_k(a, \mathbf{t_d})$ then, in order to satisfy the openness of view v_i it must add a new predicate annotated with $\mathbf{t_d}$. But $D'_{\mathcal{M}} \subsetneq D_{\mathcal{M}}$. We have reached a contradiction

As we reached a contradiction in both cases, we have proven that $D_{\mathcal{M}}$ is a minimal legal instance of $\mathcal{G}$. □

Proof of Theorem 3: Directly from Lemma 2 and 3 □

A.2 Obtaining the Simple Program from the Refined Program

Assume the hypothesis of Theorem 2 hold. We denote the view sections with S_i^l as in Section 3.1. The sections S_i^l are all associated to the definition of view V_i. We show now a syntactic transformation of the refined version of the program $(\mathcal{G})$. We justify each step of the transformation, so that at the end it will be clear that they have the same models.

Since there is no admissible mapping, each S_i^l can only be generated by view V_i. In consequence, for every model $\mathcal{M}$ of the refined version of $Pi(\mathcal{G})$, we have that for all a, $var_{v_{ij} Z_l}(a) \notin \mathcal{M}$. This implies that for every model $\mathcal{M}$ and a, $aux_{v_{ij}}(a) \notin \mathcal{M}$ and $aux_{v_{ij} Z_l}(a) \notin \mathcal{M}$. Since those atoms will never appear in a model of the refined version of $Pi(\mathcal{G})$, we can delete the rules with those predicates in their heads. We can also delete them from the bodies of the rules where they appear negated. We obtain the following program:

1. Fact $dom(a)$ for every constant $a \in \mathcal{U}$.
2. Fact $V_i(a)$ whenever $a \in v_i$ for some source extension v_i in $\mathcal{G}$.
3. For every view (source) predicate V_i in the system with description $V_i(X)$ $P_1(X_1), \ldots, P_n(X_n)$:
   (a) For every P_k with no existential variables, the rules
   $P_k(X_k, t_o) \quad V_i(X)$.
   (b) For every set S_{ij} of predicates of the description's body that are related by common existential variables $\{Z_1, \ldots, Z_m\}$, the rules,
   $P_k(X_k, v_{ij}) \quad add_{v_{ij}}(X'), \ \bigwedge_{Z_l \in (X_k \setminus X')} F_i^l(X', Z_l)$, for $P_k \in S_{ij}$.
   $add_{v_{ij}}(X') \quad V_i(X)$, where $X' = X \cap \{\bigcup_{P_k \in S_{ij}} X_k\}$.
4. For every predicate $F_i^l(X', Z_l)$ introduced in 3.b., the rules,
   $F_i^l(X', Z_l) \quad add_{v_{ij} Z_l}(X'), dom(Z_l), choice((X'), (Z_l))$.
   $add_{v_{ij} Z_l}(X') \quad add_{v_{ij}}(X')$, for $l = 1, \ldots m$.
5. For every global relation $P(X)$ the rules
   $P(X, nv_{ij}) \quad P(X, v_{hk})$, for $\{(ij, hk) | P(X) \in S_{ij}$ and $S_{hk}\}$.
   $P(X, nv_{ij}) \quad P(X, t_o)$, for $\{(ij) | P(X) \in S_{ij}\}$.
   $P(X, t_d) \quad P(X, v_{ij})$, for $\{(ij) | P(X) \in S_{ij}\}$.
   $P(X, t_d) \quad P(X, t_o)$.

This is a positive program with choice. Because of the second rule in 3.(b) and the second rule in 4., we can replace every occurrence of $add_{v_{ij}}(X')$ and $add_{v_{ij} Z_l}(X')$ by $V_i(X)$. Also from the third and fourth rules in 5., we can replace every occurrence of $P(X, t_o)$ and $P(X, v_{ij})$ by $P(X, t_d)$. It is also easy to see that the first two rules in 5. will generate atoms that are useless in the calculation of the the global predicates; then these rules can be deleted. We obtain the following program:

1. Fact $dom(a)$ for every constant $a \in \mathcal{U}$.
2. Fact $V_i(a)$ whenever $a \in v_i$ for some source extension v_i in $\mathcal{G}$.
3. For every view (source) predicate V_i in the system with description $V_i(X)$ $P_1(X_1), \ldots, P_n(X_n)$:
   (a) For every P_k with no existential variables, the rules
   $P_k(X_k, t_d) \quad V_i(X)$.
   (b) For every set S_{ij} of predicates of the description's body that are related by common existential variables $\{Z_1, \ldots, Z_m\}$, the rules,
   $P_k(X_k, t_d) \quad V_i(X), \ \bigwedge_{Z_l \in (X_k \setminus X')} F_i^l(X', Z_l)$, for $P_k \in S_{ij}$.
4. For every predicate $F_i^l(X', Z_l)$ introduced in 3.b., the rules,
   $F_i^l(X', Z_l) \quad V_i(X), dom(Z_l), choice((X'), (Z_l))$.

By merging rules 3.(a) and 3.(b), the revised version of $(\mathcal{G})$ is eventually syntactically transformed to the simple version of the program.

Query Answering in Peer-to-Peer Data Exchange Systems[*]

Leopoldo Bertossi and **Loreto Bravo**
Carleton University, School of Computer Science, Ottawa, Canada.
{bertossi,lbravo}@scs.carleton.ca

Abstract. The problem of answering queries posed to a peer who is a member of a peer-to-peer data exchange system is studied. The answers have to be consistent wrt to both the local semantic constraints and the data exchange constraints with other peers; and must also respect certain trust relationships between peers. A semantics for *peer consistent answers* under exchange constraints and trust relationships is introduced and some techniques for obtaining those answers are presented.

1 Introduction

In this paper the problem of answering queries posed to a peer who is a member of a peer-to-peer data exchange system is investigated. When a peer P receives a query and is going to answer it, it may need to consider both its own data and the data stored at other peers' sites if those other peers are related to P by data exchange constraints (DECs). Keeping the exchange constraints satisfied, may imply for peer P to get data from other peers to complement its own data, but also not to use part of its own data. In which direction P goes depends not only on the exchange constraints, but also on the *trust relationships* that P has with other peers. For example, if P trust another peer Q's data more than its own, P will accommodate its data to Q's data in order to keep the exchange constraints satisfied. Another element to take into account in this process is a possible set of local semantic constraints that each individual peer may have.

Given a network of peers, each with its own data, and a particular peer P in it, a *solution for* P is -loosely speaking- a global database instance that respects the exchange constraints and trust relationships P has with its immediate neighbors and stays as close as possible to the available data in the system. Since the answers from P have to be consistent wrt to both the local semantic constraints and the data exchange constraints with other peers, the *peer consistent answers* (PCAs) from P are defined as those answers that can be retrieved from P's portion of data in *every* possible solution for P. This definition may suggest that P may change other peers' data, specially of those he considers less reliable, but this is not the case. The notion of solution is used as an auxiliary notion to characterize the correct answers from P's point of view. Ideally, P should be able to obtain its peer consistent answers just by querying the already available local instances. This resembles the approach to *consistent query answering* (CQA) in databases [1, 6], where answers to queries that are consistent with given ICs are computed without changing the original database.

[*] This is slightly extended version of the paper that appeared in the Proceedings of the International Workshop on Peer-to-Peer Computing & Databases (P2P&DB 2004), collocated with EDBT 04, and published by Springer Verlag in its LNCS series.

We give a precise semantics for peer consistent answers to first-order queries. First for the *direct case*, where transitive relationships between peers via ECs are not automatically considered; and at the end, the *transitive case*. We also illustrate, by means of extended and representative examples, mechanisms for obtaining PCAs. One of them is first order (FO) query rewriting, where the original query is transformed into a new query, whose standard answers are the PCAs to the original one. This methodology has intrinsic limitations. The second, more general, approach is based on a specification of the solutions for a peer as the stable models of a logic program, which captures the different ways the system stabilizes after satisfying the DECs and the trust relationships.

We first recall the definition of database repair that is used to characterize the consistent answers to queries in single relational databases wrt certain integrity constraints (ICs) [1]. Given a relational database instance r with schema $\mathcal{R}$ (which includes a domain D), $\Sigma(r)$ is the set of ground atomic formulas $\{P(\bar{a}) \mid P \in \mathcal{R} \text{ and } r \models P(\bar{a})\}$.

Definition 1. [1] (a) Let r_1, r_2 be database instances over $\mathcal{R}$. The *distance*, $\Delta(r_1, r_2)$, between r_1 and r_2 is the symmetric difference $\Delta(r_1, r_2) = (\Sigma(r_1) \smallsetminus \Sigma(r_2)) \cup (\Sigma(r_2) \smallsetminus \Sigma(r_1))$. (b) For database instances r, r_1, r_2, we define $r_1 \leq_r r_2$ if $\Delta(r, r_1) \subseteq \Delta(r, r_2)$. (c) Let IC be a set of ICs on $\mathcal{R}$. A *repair* of an instance r wrt IC is a $\leq_r$-minimal instance r', such that $r' \models IC$. □

A repair of an instance r is a consistent instance that minimally differs from r.

2 A Framework for P2P Data Exchange

In this section we will describe the framework we will use to formalize and address the problem of query answering in P2P systems.

Definition 2. A *P2P data exchange system* $\mathfrak{P}$ consists of:
(a) A finite set $\mathcal{P}$ of peers, denoted by A, B, C, ..., P, Q, ...
(b) For each peer P, a database schema $\mathcal{R}(\mathtt{P})$, that includes a domain $D(\mathtt{P})$, and relations $R(\mathtt{P}), \ldots$. However, it may be convenient to assume that all peers share a common, fixed, possibly infinite domain, D. Each $\mathcal{R}(\mathtt{P})$ determines a FO language $\mathcal{L}(\mathtt{P})$. We assume that the schemas $\mathcal{R}(\mathtt{P})$ are disjoint, being the domains the only possible exception. $\mathcal{R}$ denotes the union of the $\mathcal{R}(\mathtt{P})$s.
(c) For each peer P, a database instance $r(\mathtt{P})$ corresponding to schema $\mathcal{R}(\mathtt{P})$.
(d) For each peer P, a set of $\mathcal{L}(\mathtt{P})$-sentences $IC(\mathtt{P})$ of ICs on $\mathcal{R}(\mathtt{P})$.
(e) For each peer P, a collection $\Sigma(\mathtt{P})$ of *data exchange constraints* $\Sigma(\mathtt{P}, \mathtt{Q})$ consisting of sentences written in the FO language for the signature $\mathcal{R}(\mathtt{P}) \cup \mathcal{R}(\mathtt{Q})$, and the Q's are (some of the) other peers in $\mathcal{P}$.
(f) A relation $\mathit{trust} \subseteq \mathcal{P} \times \{\mathit{less}, \mathit{same}\} \times \mathcal{P}$, with the intended semantics that when $(\mathtt{A}, \mathit{less}, \mathtt{B}) \in \mathit{trust}$, peer A trusts itself less than B; while $(\mathtt{A}, \mathit{same}, \mathtt{B}) \in \mathit{trust}$ indicates that A trusts itself the same as B. In this relation, the second argument functionally depends on the other two. By default a peer trusts its own data more than that of other peers. □

Each peer P is responsible for the update and maintenance of its material instance wrt $IC(\mathtt{P})$, independently from other peers. In particular, we assume $r(\mathtt{P}) \models IC(\mathtt{P})$. However, when local data is virtually changed to accommodate to other peers' data, the local ICs could be virtually violated. It is possible to keep the local ICs satisfied also at query time by using methodologies for consistent query answering, i.e. for consistently answering queries in databases that fail to satisfy certain ICs [6] (see Section 3.2). A peers may submit queries to other peers in accordance with the restrictions imposed its DECs and using the other peer's relations appearing in them.

Definition 3. (a) We denote with $\overline{\mathcal{R}}(\mathtt{P})$ the schema consisting of $\mathcal{R}(\mathtt{P})$ extended with the other peers' schemas that contain predicates appearing in $\Sigma(\mathtt{P})$. (b) For a peer P and an instance r on $\mathcal{R}(\mathtt{P})$, we denote by $\bar{r}$, the database instance on $\overline{\mathcal{R}}(\mathtt{P})$, consisting of the union of r with all the peers' instances whose schemas appear in $\overline{\mathcal{R}}(\mathtt{P})$. (c) If r is an instance over a certain schema $\mathcal{S}$ and $\mathcal{S}'$ is a subschema of $\mathcal{S}$, then $r|\mathcal{S}'$ denotes the restriction of r to $\mathcal{S}'$. In particular, if $\mathcal{R}(\mathtt{P}) \subseteq \mathcal{S}$, then $r|\mathtt{P}$ denotes the restriction of r to $\mathcal{R}(\mathtt{P})$. (d) We denote by $\mathcal{R}(\mathtt{P})^{less}$ the union of all schemas $\mathcal{R}(\mathtt{Q})$, with $(\mathtt{P}, \mathit{less}, \mathtt{Q}) \in \mathit{trust}$. Analogously is $\mathcal{R}(\mathtt{P})^{same}$ defined. □

From the perspective of a peer P, its own database may be inconsistent wrt the data owned by another peer Q and the DECs in $\Sigma(\mathtt{P}, \mathtt{Q})$. Only when P trust Q the same as or more than itself, it has to consider Q's data. When P queries its database, these inconsistencies may have to be taken into account. Ideally, the answers to the query obtained from P should be consistent with $\Sigma(\mathtt{P}, \mathtt{Q})$ (and its own ICs $\Sigma(\mathtt{P})$). In principle, P, who is not allowed to change other peers' data, could try to repair its database in order to satisfy $\Sigma(\mathtt{P}) \cup IC(\mathtt{P})$. This is not a realistic approach. Rather P should solve its conflicts or shortcomings at query time, when it queries its own database and those of other peers. Any answer obtained in this way should be sanctioned as correct wrt to a precise semantics.

The semantics of peer consistent query answers for a peer P is given in terms of all possible minimal, virtual, simultaneous repairs of the local databases that lead to a satisfaction of the DECs while respecting P's trust relationships to other peers. This repair process may lead to alternative global databases called the *solutions* for P. Next, the peer consistent answers from P are those that are invariant wrt to all its solutions. A peer's solution captures the idea that only some peers' databases are relevant to P, those whose relations appear in its trusted exchange constraints, and are trusted by P at least as much as it trusts its own data. In this sense, this is a "local notion", because it does not take into consideration transitive dependencies (but see Section 4.3).

Definition 4. (direct case) Given a peer P in a P2P data exchange system and an instance $\bar{r}$ on $\mathcal{R}$, an instance $\bar{r}'$ on $\mathcal{R}$ is a *solution for* P if $\bar{r}'$ is a repair of $\bar{r}$ wrt to $\Sigma(\mathtt{P}) \cup IC(\mathtt{P})$ that does not change the more trusted relations, more precisely: (a) $\bar{r}' \models \bigcup\{\Sigma(\mathtt{P}, \mathtt{Q}) \mid (\mathtt{P}, \mathit{less}, \mathtt{Q}) \text{ or } (\mathtt{P}, \mathit{same}, \mathtt{Q}) \in \mathit{trust}\} \cup IC(\mathtt{P})$; (b) $\bar{r}'|P = \bar{r}|P$ for every predicate $P \in \mathcal{R}(\mathtt{Q})$, where Q is a peer with $(\mathtt{P}, \mathit{less}, \mathtt{Q}) \in \mathit{trust}$; (c) $\bar{r}'$ minimally differs from $\bar{r}$ in the sense that $(\bar{r}' \smallsetminus \bar{r}) \cup (\bar{r} \smallsetminus \bar{r}')$ is minimal under set inclusion among those instances that satisfy (a) and (b). □

Intuitively, a solution for P repairs the global instance wrt the DECs with peers that P trusts more than or the same as itself, but leaving unchanged the tables that belong to more trusted peers. As a consequence of the definition, tables belonging to peers that are not related to P or are less trustable are not changed. That is, P tries to change its own tables according to what the dependencies to more or equally trusted peers prescribe.

The *solutions* for a peer are used as a conceptual, auxiliary tool to characterize the peer consistent answers; and we are not interested in them *per se*. Solutions are virtual and may be only partially computed if necessary, if this helps us to compute the correct answers obtained in/from a peer. The "changes" that are implicit in the definition of solution via the set differences are expected to be minimal wrt to sets of tuples which are inserted/deleted into/from the tables.

In these definitions we find clear similarities with the characterization of consistent query answers in single relational databases [6]. However, in P2P query answering, repairs may involve data associated to different peers, and also a notion of priority that is related to the trust relation.

Example 1. Consider a P2P data exchange system with peers P1, P2, P3, schemas $\mathcal{R}_i = \{R^i, \ldots\}$, instances $r(\mathtt{Pi})$, $i = 1, 2, 3$, resp., with: (a) $r(\mathtt{P1}) = \{R^1(a, b), R^1(s,\ t)\}$, $r(\mathtt{P2}) = \{R^2(c, d), R^2(a, e)\}$, $r(\mathtt{P3}) = \{R^3(a, f), R^3(s, u)\}$. (b) $\mathit{trust} = \{\ (\mathtt{P1}, \mathit{less},\ \mathtt{P2}),\ (\mathtt{P1}, \mathit{same}, \mathtt{P3})\ \}$. (c) $\Sigma(\mathtt{P1}, \mathtt{P2}) = \{\ \forall xy(R^2(x, y) \rightarrow R^1(x, y))\ \}$; $\Sigma(\mathtt{P1}, \mathtt{P3}) = \{\ \forall xyz(R^1(x, y) \wedge R^3(x, z) \ \rightarrow\ y = z)\ \}$.

Here, the global instance is $r = \{R^1(a, b), R^1(s, t), R^2(c, d), R^2(a, e), R^3(a, f), R^3(s, u)\}$. The solutions for P1 are obtained by first repairing r wrt the relationship between P1 and P2. Thus, r_1 in condition (c2) in Definition 4 is $r_1 = \{R^1(a, b), R^1(s, t),\ R^1(c, d), R^1(a, e),\ R^2(c, d), R^2(a, e),\ R^3(a, f), R^3(s, u)\}$. So, we have only one repair at this stage, which now has to be repaired wrt the DEC between P1 and P3 (but keeping the relationship between P1 and P2 satisfied). There are two sets of tuples violating $\Sigma(\mathtt{P1}, \mathtt{P3})$ in r_1: $\{R^1(s, t), R^3(s, u)\}$ and $\{R^1(a, b), R^1(a, e), R^3(a, f)\}$. The first violation can be repaired by deleting any, but only one, of the two tuples. The second one, by deleting tuple $R^3(a, f)$ only (otherwise we would violate the relationship between P1 and P2). In consequence, we obtain two repairs, $r' = \{R^1(a, b), R^1(s, t), R^1(c, d), R^1(a, e), R^2(c, d), R^2(a, e)\}$; and $r'' = \{\ R^1(a, b), R^1(c, d), R^1(a, e), R^2(c, d),\ R^2(a,\ e), R^3(s, u)\}$. □

The minimization involved in a solution is similar to a minimization with some fixed predicates as found in non-monotonic reasoning [26]. Actually, the notion to be given now of peer consistent answer, that captures the intended answers from a peer's perspective, is -as the notion of CQA- non-monotonic [6].[1]

Definition 5. Given a FO query $Q(\bar{x}) \in \mathcal{L}(\mathtt{P})$ posed to P, a ground tuple $\bar{t}$ is a *peer consistent* answer to Q for P iff $r'|\mathtt{P} \models Q(\bar{t})$ for every solution r' for P. □

Example 2. (example 1 continued) The query $Q\colon R^1(x, y)$ posed to P1 has as peer consistent answers the tuples: $(a, b), (c, d), (a, e)$, because those are the tuples found in relation R^1 in the restriction to P1's schema in every solution for P. □

[1] A circumscriptive approach to database repairs was given in [7]. It should not be difficult to extend that characterization to capture the peer solutions.

Notice that this definition is relative to a fixed peer, and not only because the query is posed to one peer and in its query language, but also because this notion is based on the "direct or local" notion of solution for a single peer, which considers its "direct neighbors" only. This is a first step towards the general case of transitive dependencies, that will be explored in Section 4.3. However, this restricted case is the basis for the transitive case, because P does not see beyond its neighbors; and when P requests data to a neighbor, say Q, the latter may have to find local solutions of its own by considering its direct neighbors. The transitive case has to combine these local solutions.

Peer consistent answers to queries can be obtained by using techniques similar to those for CQA, e.g. query rewriting [1,6]. However, there are important differences, because now we have some fixed predicates in the repair process.

Example 3. (example 1 continued) If P1 is posed the query $Q: R^1(x,y)$, asking for the tuples in relation R^1, we first rewrite the query using the DEC in $\Sigma(\mathtt{P1},\mathtt{P2})$, obtaining $Q': R^1(x,y) \vee R^2(x,y)$, with the effect of bringing P2's data into P1. Next, considering the DEC in $\Sigma(\mathtt{P1},\mathtt{P3})$ the query is rewritten as

$$Q'': [R^1(x,y) \wedge \forall z_1((R^3(x,z_1) \wedge \neg\exists z_2 R^2(x,z_2)) \rightarrow z_1 = y)] \vee R^2(x,y). \quad (1)$$

To answer this query, P1 first issues a query to P2 to retrieve the tuples in R^2 that will be in R^1 for all the solutions due to $\Sigma(\mathtt{P1},\mathtt{P2})$. Next, a query is issued to P3 to leave aside the tuples of R^1 that have the same first but not the same second argument in R^3, as long as there does not exist a tuple in R^2 that "protects" the tuple in R^1. The tuple $R^1(a,b)$ is protected by $R^2(a,e)$ because, as $R^1(a,e)$ belongs to all the solutions, the only way to repair a violation of mapping $\Sigma(\mathtt{P1},\mathtt{P3})$ is by deleting the tuple of R^3, and therefore the tuple R^1 will be in the answer. The answers to (1) are $(a,b),(c,d),(a,e)$, precisely the peer consistent answers from P1 to query Q according to Definition 5. □

Notice that a query Q may have peer consistent answers for a peer which are not answers to Q when the peer is considered in isolation, which makes sense, because the peer may import data from other peers.[2]

This query rewriting approach differs from the one used for CQA. In the latter case, literals in a query are resolved (by *resolution*) against ICs in order to generate residues that are iteratively appended as extra conditions to the query. In the case of P2P data systems, the query may have to be modified in order to include new data that is located at a different peer's site. This cannot be achieved by imposing extra conditions alone, but instead, by relaxing the query in some sense. Since query answering in P2P systems includes sufficiently complex cases of CQA, a FO query rewriting approach to P2P query answering is bound to have limitations in terms of completeness [6]. Instead, we will propose (see Section 3) a more general methodology based on answer set programming.

[2] Another difference with CQA, where all consistent answers are answers to the original query; at least for conjunctive queries and *generic* ICs [6], i.e. those that do not require the presence/absence of any particular ground tuple in/from the database.

3 Referential Data Exchange Constraints

In most applications we may expect the DECs to be, as $\Sigma(\mathtt{P1},\mathtt{P2})$ in Example 1), inclusion dependencies or referential constraints, i.e. formulas of the form

$$\forall \bar{x} \exists \bar{y}(R^{\mathsf{Q}}(\bar{x}) \wedge \cdots \rightarrow R^{\mathsf{P}}(\bar{z},\bar{y}) \wedge \cdots), \quad (2)$$

where $R^{\mathsf{Q}}, R^{\mathsf{P}}$ are relations for peers Q and P, resp., the dots indicate some possible additional conditions, most likely expressed in terms of built-ins, $\bar{z} \subseteq \bar{x}$.

An exchange constraint of the form (2) will most likely belong to $\Sigma(\mathtt{P},\mathtt{Q})$, i.e. to peer P, who wants to import data from the more trusted peer Q. It could also belong to Q, if this peer wants to validate its own data against P's data.

An answer set programming approach to the specification of solutions for a peer can be developed. Those specifications will be similar to those of repairs of single relational databases under referential integrity constraints [2]. However, as we have seen, there are important differences with CQA. In Section 3.1 we give an example of an even more more involved referential constraint that shows the main issues around this kind of specifications.

3.1 An extended example

Consider a P2P system with peers P and Q, with schemas $\{R_1(\cdot,\ \cdot), R_2(\cdot,\cdot)\}$, $\{S_1(\cdot,\cdot),\ S_2(\cdot,\cdot)\}$, resp.; and assume that P is querying its database subject to its DEC that mixes tables of the two peers on each side of the implication:

$$\forall x \forall y \forall z \exists w(R_1(x,y) \wedge S_1(z,y) \rightarrow R_2(x,w) \wedge S_2(z,w)), \quad (3)$$

We consider the case where $(\mathtt{P}, \mathit{less}, \mathtt{Q}) \in \mathit{trust}$, i.e. P considers Q's data more reliable than its own. If (3) is satisfied by the combination of the data in P and Q, then the current global instance constitutes P's solution. Otherwise, alternative solutions for P have to be found, keeping Q's data fixed in the process. This is the case, when there are ground tuples $R_1(d,m) \in r(\mathtt{P}), S_1(a,m) \in r(\mathtt{Q})$, such that for no t it holds both $R_2(d,t) \in r(\mathtt{P})$ and $S_2(a,t) \in r(\mathtt{Q})$.

Obtaining peer consistent answers for peer P amounts to virtually restoring the satisfaction of (3), by virtually modifying P's data. In order to specify P's modified relations, we introduce virtual versions R_1', R_2' of R_1, R_2, containing the data in peer P's solutions. In consequence, at the solution level, we have the relations R_1', R_2', S_1, S_2. Since P is querying its database, its original queries will be expressed in terms of relations R_1', R_2' only (plus, possibly, built-ins).

The contents of the virtual relations R_1', R_2' are obtained from the material sources R_1, R_2, S_1, S_2.[3] Since S_1, S_2 are fixed, the satisfaction of (3) requires R_1' to be a subset of R_1, and R_2', a superset of R_2. The specification of these relations is done in extended disjunctive logic programs with answer set (stable model) semantics [17]. The first rules for the specification program Π are:

$$R_1'(x,y) \leftarrow R_1(x,y),\ \mathit{not}\ \neg R_1'(x,y) \quad (4)$$

$$R_2'(x,y) \leftarrow R_2(x,y),\ \mathit{not}\ \neg R_2'(x,y), \quad (5)$$

[3] We can observe that the virtual relations can be seen as virtual global relations in a virtual data integration system [25, 22].

which specify that, by default, the tuples in the source relations are copied into the new virtual versions, but with the exception of those that may have to be removed in order to satisfy (3) (with R_1, R_2 replaced by R_1', R_2'). Some of the exceptions for R_1' are specified by

$$\neg R_1'(x,y) \leftarrow R_1(x,y), S_1(z,y),\ \mathit{not}\ \mathit{aux}_1(x,z),\ \mathit{not}\ \mathit{aux}_2(z) \quad (6)$$

$$\mathit{aux}_1(x,z) \leftarrow R_2(x,w), S_2(z,w) \quad (7)$$

$$\mathit{aux}_2(z) \leftarrow S_2(z,w). \quad (8)$$

That is, $R_1(x,y)$ is deleted if it participates in a violation of (3) (what is captured by the first three literals in the body of (6) plus rule (7)), and there is no way to restore consistency by inserting a tuple into R_2, because there is no possible matching tuple in S_2 for the possibly new tuple in R_2 (what is captured by the last literal in the body of (6) plus rule (8)). In case there is such a tuple in S_2, we can either delete a tuple from R_1 or insert a tuple into R_2:

$$\begin{aligned}\neg R_1'(x,y) \vee R_2'(x,w) \leftarrow R_1(x,y), S_1(z,y),\ \mathit{not}\ \mathit{aux}_1(x,z), S_2(z,w),&\\ \mathit{choice}((x,z),w).& \quad (9)\end{aligned}$$

That is, in case of a violation of (3), when there is tuple of the form (a,t) in S_2 for the combination of values (d,a), then the *choice operator* [18] non deterministically chooses a unique value for t, so that the tuple (d,t) is inserted into R_2 as an alternative to deleting (d,m) from R_1. The *choice* predicate can be replaced by a standard predicate plus extra rules that choose a unique value for t [18]. No exceptions are specified for R_2', which makes sense since R_2' is a superset of R_2. Then, the negative literal in the body of (5) can be eliminated. However, new tuples can be inserted into R_2', what is captured by rule (9). Finally, the program must contain as facts the tuples in the original relations R_1, R_2, S_1, S_2.

If P equally trusts itself and Q, both P and Qs' relations are flexible when searching for a solution. The program becomes more involved, because now S_1, S_2 may also change; and virtual versions for them must be introduced and specified.

3.2 Considerations on specifications of peers' solutions

The example in Section 3.1 shows the main issues in the specification of a peer's solutions under referential DECs. The program with choice operator can be translated into one with standard answer set (or stable model) semantics [18]; and the solutions are in one to one correspondence with the answer sets of the program. Actually, each answer set S corresponds to a solution $r'(S)$ for peer P which coincides with the original, material, global instance on the tables other than R_1, R_2, whereas for the latter the contents are of the form $\{\bar{t} \mid R_i'(\bar{t}) \in S\}, i = 1,2$, resp. The absence of solutions for a peer is captured through the non existence of answer sets for program Π.

Since program Π represents in a compact form all the solutions for a peer, the peer consistent answers from a peer can be obtained by running a query program expressed in terms of the virtually repaired tables, in combination with the specification program Π. For this the combined program is run under the skeptical answer set semantics, for which a system like DLV [14] can be used.

For example, the query $Q(x,z):\ \exists y(R_1(x,y) \wedge R_2(z,y))$ issued to peer P, would be peer consistently answered by running the query program $\mathit{Ans}_Q(x,z) \leftarrow R_1'(x,y), R_2'(x,y)$ together with program Π. Although only (the new versions of) P's relations appear in the query, the program may make P import Q's data.

In the presence of referential DECs, the *choice operator* may have to choose values from the infinite underlying domain, but outside the active domains. There are several options, some of them already considered for CQA: (a) Live with an open infinite domain; repairing existential DECs by picking up elements from it. This, in the presence of cycles, lead to the undecidability of peer consistent query answering (PCQA) [10]. (b) Assign null values without propagation through DECs [2]. In this case, even in the presence of cycles, it is possible to prove that PCQA becomes decidable. (c) Consider an appropriate finite and closed proper superset of the active domains [9]. (d) Introduce fresh constants whenever needed from a separate domain [11]. We do not commit to any of these options here, but this choice and the class of referential ECs (e.g. presence cycles) may determine, e.g. decidability of peer consistent answering [12, 10, 20, 11].

If a peer has local ICs $IC(\mathtt{P})$ to be satisfied, also at query time, then the program that specifies its solutions should take care of its ICs. A simple but radical way of doing this consists in using program denial constraints. If in Section 3.1 we had for peer P the local functional dependency (FD) $\forall x \forall y \forall z(R_1(x,y) \wedge R_1(x,z) \rightarrow y = z)$, then program would include the program constraint $\leftarrow R_1(x,y), R_1(x,z), y \neq z$, having the effect of pruning those solutions that do not satisfy the FD. However, a more flexible (and "robust" [15]) alternative for keeping the local ICs satisfied, consists in having the specification program split in two layers, where the first one builds the solutions, without considering the local ICs, and the second one, repairs the solutions wrt the local ICs, as done with single inconsistent relational databases [2]. A more uniform approach consists in identifying $IC(\mathtt{P})$ with $\Sigma(\mathtt{P},\mathtt{P})$ and considering $(\mathtt{P}, \mathit{same}, \mathtt{P}) \in \mathit{trust}$.

Finally, we should notice that obtaining peer consistent answers has at least the data complexity of consistent query answering, for which some results are known [12, 16, 10]. In the latter case, for common database queries and ICs, Π_2^P-completeness is easily achieved. On the other side, the problem of skeptical query evaluation from the disjunctive programs we are using for P2P systems is also Π_2^P-complete in data complexity [13]. In this sense, the logic programs are not contributing with additional complexity to our problem.

Example 4. (example 1 continued) The answer set programming approach to peer consistent query answering in this case requires the predicates R_1, R_3 to be flexible in the repair process, so their contents have to be specified, however in contrast to the situation in Section 3.1, we get interacting rules for R_1. For this reason the repair process may need to execute several steps until it stabilizes. This requires the use of program rules with annotations as introduced for CQA in the presence of interacting ICs [2].

The annotations are constants that are used in an extra argument introduced in each database relation. The annotation $\mathbf{t_d}$ is used to annotate the atoms that are in the original database instance. 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 $\mathbf{f_a}$, with the intended meaning that the atom getting them is advised to be made true, resp. false, in order to restore consistency. This is done when each IC is considered in isolation, but there may be interacting ICs, and we may require an iterative process. In order to achieve this, we use annotations $\mathbf{t}^\star$, $\mathbf{f}^\star$. The former, for example, groups together the annotations $\mathbf{t_d}$ and $\mathbf{t_a}$ for the same atom. 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. 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. All the rules in the program below express these relationships, and in this sense are generic; similar rules will be found in any repair program with annotations. The only specific rules are the last two, that express how to repair the database when a violation of the DECs occurs. The first of the two corresponds to a violation of $\Sigma(\mathtt{P1},\mathtt{P2})$; and the last one, to a violation of $\Sigma(\mathtt{P1},\mathtt{P3})$.

$$
\begin{aligned}
R_1(X,Y,\mathbf{t}^\star) &\leftarrow R_1(X,Y,\mathbf{t_d}).\\
R_1(X,Y,\mathbf{t}^\star) &\leftarrow R_1(X,Y,\mathbf{t_a}).\\
R_1(X,Y,\mathbf{f}^\star) &\leftarrow R_1(X,Y,\mathbf{f_a}).\\
R_1(X,Y,\mathbf{f}^\star) &\leftarrow dom(X), dom(Y), not\ R_1(X,Y,\mathbf{t_d}).\\
R_1(X,Y,\mathbf{t}^{\star\star}) &\leftarrow R_1(X,Y,\mathbf{t_d}), not R_1(X,Y,\mathbf{f_a}).\\
R_1(X,Y,\mathbf{t}^{\star\star}) &\leftarrow R_1(X,Y,\mathbf{t_a}).\\
&\leftarrow R_1(X,Y,\mathbf{t_a}), R_1(X,Y,\mathbf{f_a}).\\
R_2(X,Y,\mathbf{t}^\star) &\leftarrow R_2(X,Y,\mathbf{t_d}).\\
R_2(X,Y,\mathbf{t}^\star) &\leftarrow R_2(X,Y,\mathbf{t_a}).\\
R_2(X,Y,\mathbf{f}^\star) &\leftarrow R_2(X,Y,\mathbf{f_a}).\\
R_2(X,Y,\mathbf{f}^\star) &\leftarrow dom(X), dom(Y), not\ R_2(X,Y,\mathbf{t_d}).\\
R_2(X,Y,\mathbf{t}^{\star\star}) &\leftarrow R_2(X,Y,\mathbf{t_d}), not\ R_2(X,Y,\mathbf{f_a}).\\
R_2(X,Y,\mathbf{t}^{\star\star}) &\leftarrow R_2(X,Y,\mathbf{t_a}).\\
&\leftarrow R_2(X,Y,\mathbf{t_a}), R_2(X,Y,\mathbf{f_a}).\\
R_3(X,Y,\mathbf{t}^\star) &\leftarrow R_3(X,Y,\mathbf{t_d}).\\
R_3(X,Y,\mathbf{t}^\star) &\leftarrow R_3(X,Y,\mathbf{t_a}).\\
R_3(X,Y,\mathbf{f}^\star) &\leftarrow R_3(X,Y,\mathbf{f_a}).\\
R_3(X,Y,\mathbf{f}^\star) &\leftarrow dom(X), dom(Y), not\ R_3(X,Y,\mathbf{t_d}).\\
R_3(X,Y,\mathbf{t}^{\star\star}) &\leftarrow R_3(X,Y,\mathbf{t_d}), not\ R_3(X,Y,\mathbf{f_a}).\\
R_3(X,Y,\mathbf{t}^{\star\star}) &\leftarrow R_3(X,Y,\mathbf{t_a}).\\
&\leftarrow R_3(X,Y,\mathbf{t_a}), R_3(X,Y,\mathbf{f_a}).\\
R_1(X,Y,\mathbf{t_a}) &\leftarrow R_2(X,Y,\mathbf{t}^\star), R_1(X,Y,\mathbf{f}^\star).\\
R_1(X,Y,\mathbf{f_a}) \vee R_3(X,Z,\mathbf{f_a}) &\leftarrow R_1(X,Y,\mathbf{t}^\star), R_3(X,Z,\mathbf{t}^\star), Y \neq Z.
\end{aligned}
$$

These are the facts of the program: $R_1(a,b,\mathbf{t_d})$. $R_1(s,t,\mathbf{t_d})$. $R_2(c,d,\mathbf{t_d})$. $R_2(a,e,\mathbf{t_d}).R_2(t,h,\mathbf{t_d})$. $R_3(a,f,\mathbf{t_d})$. $R_3(s,u,\mathbf{t_d})$. $R_3(t,u,\mathbf{t_d})$. $dom(a)$. $dom(b)$. $dom(s)$. $dom(t)$. $dom(c)$. $dom(d)$. $dom(e)$. $dom(f)$. $dom(u)$. $dom(h)$.

The non domain atoms say that the original tables R_1, R_2, R_3 contain the tuples $\{(a,b),(s,t)\}$, $\{(c,d),(a,e),(t,h)\}$, $\{(a,f),(s,u),(t,u)\}$. The use of annotations allows us to give up using virtual versions R_1', R_3' for R_1, R_2, because the final contents for the latter will be read off from those atoms that become annotated with $\mathbf{t}^{\star\star}$. □

4 Extensions

4.1 Optimizations

It is possible to perform some optimizations on the solution specification program, to make its evaluation simpler. Disjunctive programs under the stable model semantics are more complex than non disjunctive programs [13]. However, it is known that a disjunctive program can be transformed into a non disjunctive program if the program is head-cycle free (HCF) [3,23]. Intuitively speaking, a disjunctive program is HCF if there are no cycles involving two literals in the head of a same rule, where a link is established from a literal to another if the former appears positive in the body of a rule, and the latter appears in the head of the same rule. These considerations about HCF programs hold for programs that do not contain the choice operator, i.e. they might not automatically apply to our programs that specifies the solutions for a peer under referential constraints. However, it is possible to prove that a disjunctive choice program Π is HCF when the program obtained from Π by removing its choice goals is HCF.

Example 5. Consider the choice program Π presented in Section 3.1. If the choice operator is eliminated from rule (9), we are left with the rule

$$\neg R_1'(x,y) \vee R_2'(x,w) \leftarrow R_1(x,y), S_1(z,y),\ not\ aux_1(x,z), S_2(z,w).$$

The resulting program is HCF and then rule (9) can be replaced by two rules:

$$
\begin{aligned}
\neg R_1'(x,y) \leftarrow{} & R_1(x,y), S_1(z,y),\ not\ aux_1(x,z), S_2(z,w),\ not\ R_2'(x,w),\\
& choice((x,z),w).\\
R_2'(x,w) \leftarrow{} & R_1(x,y), S_1(z,y),\ not\ aux_1(x,z), S_2(z,w),\ not\ \neg R_1'(x,y),\\
& choice((x,z),w).
\end{aligned}
$$

□

4.2 A LAV approach

There are some clear connections between P2P query answering and virtual integration of data sources by means of mediator based systems [20,27]. There are basically two approaches to the latter problem. According to *global-as-view* (GAV), each virtual table at the mediator (global) level is expressed as a view of the collection of relations in the data sources. According to *local-as-view* (LAV), relations in the (local) data sources as expressed as views of the virtual global relations. GAV is more natural and simpler for query evaluation than LAV, but LAV is simpler to deal with when sources leave and enter the integration system. GLAV is a mixture of the two approaches (see [22] for a survey).

The logic programming-based approach proposed in Section 3.1 can be assimilated to the GAV approach, because tables in the solutions are specified as views over peer's schemas. However, a LAV approach could also be attempted. In this case, we also introduce virtual, global versions of S_1, S_2. The relations in the sources have to be defined as views of the virtual relations in a solution, actually, through the following specification of a virtual integration system [19]:

View definitions	label	source
$R_1(x,y) \leftarrow R_1'(x,y)$	*closed*	r_1
$R_2(x,y) \leftarrow R_2'(x,y)$	*open*	r_2
$S_1(x,y) \leftarrow S_1'(x,y)$	*clopen*	s_1
$S_2(x,y) \leftarrow S_2'(x,y)$	*clopen*	s_2

Here the r_i, s_j are the original material extensions of relations R_i, S_j. Labels are assigned to the sources on the basis of the view definitions in the first column, the IC (3) and the trust relationships; in the latter case, by the fact that R_1, R_2 can change, but not S_1, S_2. More precisely, the label in the first row corresponds to the fact that (3) can be satisfied by deleting tuples from R_1, then the contents of the view defined in there must be contained in the original relation r_1 (the material source). The label in the second row indicates that we can insert tuples into R_2 to satisfy the constraint, and then, the extension of the solution contains the original source r_2. Since, S_1, S_2 do not change, they are declared as both closed and open, i.e. clopen.

If a query is posed to peer P, it has to be first formulated in terms of R_1', R_2', and then it can be peer consistently answered by querying the integration system subject to the global IC: $\forall xyz \exists w(R_1'(x,y) \wedge S_1'(z,y) \rightarrow R_2'(x,w) \wedge S_2'(z,w))$. A methodology that is similar to the one applied for consistently querying virtual data integration systems under LAV can be used. In [5,8] methodologies for open sources are presented, and in [4] the mixed case with both open, closed and clopen sources is treated. However, there are differences with the P2P scenario; and the methodologies need to be adjusted as discussed below.

The methodology presented in [4] for CQA in virtual data integration is based on a three-layered answer set programming specification of the repairs of the system: a first layer specifies the contents of the global relations in the minimal legal instances (to this layer only open and clopen sources contribute), a second layer consisting of program denial constraints that prunes the models that violate the closure condition for the closed sources; and a third layer specifying the minimal repairs of the legal instances [5] left by the other layers wrt the global ICs. For CQA, repairs are allowed to violate the original labels.

In our P2P scenario, we want, first of all, to consider only the legal instances that satisfy the mapping in the table and that, in the case of closed sources, include the maximum amount of tuples from the sources (the virtual relations must be kept as close as possible to their original, material versions). For the kind of mappings that we have in the table, this can be achieved by using exactly the same kind of specifications presented in in [4] for the mixed case, *but* considering the closed sources as clopen. In doing so, they will contribute to the program with both rules that import their contents into the system (maximizing the set of tuples in the global relation) and denial program constraints. Now, the trust relation also makes a difference. In order for the virtual relations to satisfy the original labels, that in their turn capture the trust relationships, the rules that repair the chosen legal instances will consider only tuple deletions (insertions) for the virtual global relations corresponding to the closed (resp. open) sources. For clopen sources the rules can neither add nor delete tuples.[4] This methodology can handle universal and simple referential DECs (no cycles and single atom consequents, conditions that are imposed by the repair layer of the program), which covers a broad class of DECs. The DEC in (3) does not fall in this class, but the repair layer can be adjusted in order to generate the solutions for P. The corresponding program is given in the appendix.

4.3 Beyond direct solutions

It is natural to consider *transitive* DECs when a peer A, that is being queried, gets data from another peer B, who in its turn -and without A possibly knowing- gets data from a third peer C to answer A's request. Most likely there won't be any explicit DEC from A to C capturing this transitive exchange; and we do not want to derive them (c.f. [20]).

In order to attack peer consistent query answering in this more complex scenario, it becomes necessary to integrate the local solutions, what can be achieved by integrating the "local" specification programs. In this case, we prefer to define the global solutions directly from the the stable models of the combined program obtained from the specification of direct interactions. This is more natural and simpler than extending to the global case the definition of solution for the direct case.[5] Of course, there might be no solutions, what is reflected in the absence of stable models for the program. A problematic case appears when there are implicit cyclic dependencies [20].

Example 6. (example in Section 3.1 continued) Let us consider another peer C with a relation $U(\cdot,\cdot)$. The following exchange constraint $\Sigma(\mathtt{Q},\mathtt{C})\colon\ \forall x \forall y(U(x,y) \rightarrow S_1(x,y))$ exists from Q to C and $(\mathtt{Q}, less, \mathtt{C}) \in trust$, meaning that Q trusts C's data more than its own. When P requests data from Q, the latter will request data from C's relation U. Now, consider the peer instances: $r_1 = \{(a,b)\}, s_1 = \{\}$, $r_2 = \{\}$, $s_2 = \{(c,e),(c,f)\}$ and $u = \{(c,b)\}$. If we analyze each peer locally, the solution for Q would contain the tuple $S_1(c,b)$ added; and P would have only one solution, corresponding to the original instances, because the DEC is satisfied without making any changes. When considering them globally, the tuple that is locally added into Q requires tuples to be added and/or deleted into/from P in order to satisfy the DEC. The combined program that specifies the global solutions consists of rules (4), (5),(7), (8) plus

$$\neg R_1'(x,y) \leftarrow R_1(x,y), S_1'(z,y),\ not\ aux_1(x,z),\ not\ aux_2(z) \quad (10)$$

$$\neg R_1'(x,y) \vee R_2'(x,w) \leftarrow R_1(x,y), S_1'(z,y),\ not\ aux_1(x,z), S_2(z,w),$$

[4] This preference criterion for a subclass of the repairs is similar to the *loosely-sound semantic* for integration of open sources under GAV [21].

[5] The approaches to P2P data exchange semantics in [11,15] also appeal to this kind of 2-step process, however in a framework based on epistemic logic.

$$
\begin{aligned}
& choice((x,z),w) && (11)\\
S_1'(x,y) &\leftarrow S_1(x,y),\ \ not\ \neg S_1'(x,y) && (12)\\
S_1'(x,y) &\leftarrow U(x,y),\ \ not\ S_1(x,y). && (13)
\end{aligned}
$$

Rules (10) and (11) replace (6), (9), resp. (12) is a persistence rule for S_1 (so as (4), (5) for R_1, R_2, resp.). (13) enforces the satisfaction of $\Sigma(\mathtt{Q},\mathtt{C})$. The solutions obtained from the stable models of the program are the expected ones: $r' = \{S_2(c,e), S_2(c,f), U(c,b), S_1'(c,b), R_2'(a,f), R_1'(a,b)\}$, $r'' = \{S_2(c,e), S_2(c,f), U(c,b), S_1'(c,b)\}$, $r''' = \{S_2(c,e), S_2(c,f), U(c,b), S_1'(c,b),\ R_2'(a,e), R_1'(a,b)\}$. □

5 Discussion and Conclusions

We have presented a logical framework that provides *semantics* and *specifications* for peer consistent query answering (PCQA). In principle it is possible to compute answers from those specifications (and the data available). However, as future work, the most urgent line of research on PCQA consists in "translating" the specifications into concrete algorithms to query the peers' databases and integrate their answers.

At the answer set programming level, it becomes necessary to derive *specialized specifications*, that are easier to handle and compute for particular classes of DECs and queries; and from the latter also *specialized algorithms* for PCQA as indicated above.

The *specifications* themselves have to be optimized (as logic programs); and also the computations of/under the answer set semantics. In particular, it becomes necessary to avoid extra complexity in cases where complexity of PCQA is lower that general data complexity of disjunctive answer set programming [13] (although the latter is not higher that the *general* data complexity of peer consistent query answering [12, 10]). Finally, the *interaction* between the logic programming system and the data sources has to be optimized. Relevant research in this direction has been reported in [?].

The solution-based semantics we have presented might be considered too strict in the sense that peer consistent answers are sanctioned as such wrt *all* the possible solutions. It is imaginable in this context that a *brave* or *possible* semantics could be adopted: an answer is acceptable as long as it is valid in *some* solution. Adopting this approach would make computations easier; but our general logical specifications would not change.

The logical specifications we provided can be easily changed to adopt other preference criterion on forms of repair or on solutions. This can be achieved, in general, by explicitly modifying the specification programs accordingly. However, some it is thinkable that some general preference policies on answer sets could be specified at a meta program level [?].

Acknowledgements: Research funded by NSERC Grant 250279-02 and a grant from the CITO/IBM-CAS Student Internship Program. L. Bertossi is Faculty Fellow of the IBM Center for Advanced Studies, Toronto Lab. We are grateful for stimulating conversations with Ariel Fuxman, who in particular proposed to consider the example in Section 3.1; and Pablo Barcelo.

References

1. Arenas, M., Bertossi, L., Chomicki, J. Consistent Query Answers in Inconsistent Databases. Proc. ACM Symposium on Principles of Database Systems (PODS 99), 1999, pp. 68–79.
2. Barcelo, P., Bertossi, Bravo, L. Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. In *Semantics in Databases*, Springer LNCS 2582, 2003, pp. 1–27.
3. Ben-Eliyahu, R. and Dechter, R. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics in Artificial Intelligence*, 1994, 12:53-87.
4. Bertossi, L. and Bravo, L. Consistent Query Answers in Virtual Data Integration Sistems. To appear as a book chapter in 'Inconsistency Tolerance in Knowledgebases, Databases and Software Specifications'. Springer.
5. Bertossi, L., Chomicki, J., Cortes, A. and Gutierrez, C. Consistent Answers from Integrated Data Sources. In Proc. Flexible Query Answering Systems (FQAS 02), Springer LNAI 2522, 2002, pp. 71–85.
6. Bertossi, L.; and Chomicki, J. Query Answering in Inconsistent Databases. In *Logics for Emerging Applications of Databases*, J. Chomicki, G. Saake and R. van der Meyden (eds.), Springer, 2003.
7. Bertossi, L., Schwind, C. Database Repairs and Analytic Tableaux. *Annals of Mathematics and Artificial Intelligence*, 2004, 40(1-2): 5-35.
8. Bravo, L. and Bertossi, L. Logic Programs for Consistently Querying Data Sources In Proc. of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI 03), Morgan Kaufmann, 2003, pp. 10–15.
9. 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.*
10. Cali, A., Lembo, D. and Rosati, R. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In Proc. ACM PODS 03, 2003, pp. 260-271.
11. Calvanese, D., Damaggio, E., De Giacomo, G., Lenzerini, M. and Rosati, R. Semantic Data Integration in P2P Systems. In Proc. International Workshop On Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 03), Springer LNCS 2944, 2004.
12. Chomicki, J. and Marcinkowski, J. Minimal-Change Referential Integrity Maintenance Using Tuple Deletions. arXiv.org paper cs.DB/0212004. To appear in *Information and Computation.*
13. Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. Complexity And Expressive Power Of Logic Programming. *ACM Computer Surveys.* 2001, 33(3), 374-425.
14. Eiter, T., Faber, W.; Leone, N., Pfeifer, G. Declarative Problem-Solving in DLV. In *Logic-Based Artificial Intelligence.* J. Minker (ed.), Kluwer, 2000, pp. 79-103.
15. 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.
16. Fuxman, A. and Miller, R.J. Towards Inconsistency Management in Data Integration Systems. In on-line Proceedings of the IJCAI-03 Workshop on Information Integration on the Web.
17. Gelfond, M. and Lifschitz, V. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 1991, 9:365–385.
18. Giannotti, F.; Pedreschi, D.; Sacca, D., Zaniolo, C. Non-Determinism in Deductive Databases. In Proc. DOOD, Springer LNCS 566, 1991, pp. 129–146.
19. Grahne, G. and Mendelzon, A. Tableau Techniques for Querying Information Sources through Global Schemas. In Proc. International Conference on Database Theory (ICDT 99), Springer LNCS 1540, 1999, pp. 332–347.
20. Halevy, A.Y., Ives, Z.G., Suciu, D. and Tatarinov, I. Schema Mediation in Peer Data Management Systems. Proceedings International Conference on Data Engineering (ICDE 03), 2003, pp. 505-518.
21. Lembo, D., Lenzerini, M. and Rosati, R. Source Inconsistency and Incompleteness in Data Integration. In Proc. International Workshop Knowledge Representation meets Databases (KRDB 02), 2002.
22. Lenzerini, M. Data Integration: A Theoretical Perspective. In Proc. ACM Symposium on Principles of Database Systems (PODS 02), 2002, pp. 233-246.
23. Leone, N., Rullo, P. and Scarcello, F. Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics, and Computation. *Information and Computation*, 1997, 135(2):69-112.
24. 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.*
25. Levy, A. Logic-Based Techniques in Data Integration. In *Logic Based Artificial Intelligence*, J. Minker (ed.), Kluwer, 2000, pp. 575-595.
26. Lifschitz, V. Computing Circumscription. Proc. AAAI 1987.
27. Tatarinov, I. *et al.* The Piazza Peer Data Management Project. *SIGMOD Record*, 2003, 32(3):47-52.

6 Appendix

The following answer set program specifies the solutions for the example in Section 3.1 following a LAV approach to P2P data exchange (see Section 4.2). Assume that the peers have the following instances: $r_1 = \{(a,b)\}, s_1 = \{(c,b)\}$, $r_2 = \{\}$ and $s_2 = \{(c,e),(c,f)\}$. Then, the facts of the program are: $R_1(a,b)$, $S_1(c,b), S_2(c,e), S_2(c,f)$. The layer that specifies the preferred legal instances contains the following rules:

$$
\begin{aligned}
R_1'(X,Y,\mathbf{t_d}) &\leftarrow R_1(X,Y).\\
S_1'(X,Y,\mathbf{t_d}) &\leftarrow S_1(X,Y).\\
R_2'(X,Y,\mathbf{t_d}) &\leftarrow R_2(X,Y).\\
S_2'(X,Y,\mathbf{t_d}) &\leftarrow S_2(X,Y).\\
&\leftarrow R_1'(X,Y,\mathbf{t_d}), R_1(X,Y).\\
&\leftarrow S_1'(X,Y,\mathbf{t_d}), S_1(X,Y).\\
&\leftarrow S_2'(X,Y,\mathbf{t_d}), S_2(X,Y).
\end{aligned}
$$

The layer that specifies the repairs of the legal instances contains the following rules. The annotation constants in the third arguments in the relations are used as auxiliary elements in the repairs process [2, 6]. The choice operator has been unfolded, producing the stable version of the choice program.

$$
\begin{aligned}
R_1'(X,Y,\mathbf{t^{\star\star}}) &\leftarrow R_1'(X,Y,\mathbf{t_d}),\ \ not\ R_1'(X,Y,\mathbf{f_a}).\\
R_1'(X,Y,\mathbf{t^{\star\star}}) &\leftarrow R_1'(X,Y,\mathbf{t_a}).\\
&\leftarrow R_1'(X,Y,\mathbf{t_a}), R_1'(X,Y,\mathbf{f_a}).\\
S_1'(X,Y,\mathbf{t^{\star\star}}) &\leftarrow S_1'(X,Y,\mathbf{t_d}),\ \ not\ S_1'(X,Y,\mathbf{f_a}).\\
S_1'(X,Y,\mathbf{t^{\star\star}}) &\leftarrow S_1'(X,Y,\mathbf{t_a}).\\
&\leftarrow S_1'(X,Y,\mathbf{t_a}), S_1'(X,Y,\mathbf{f_a}).\\
R_2'(X,Y,\mathbf{t^{\star\star}}) &\leftarrow R_2'(X,Y,\mathbf{t_d}),\ \ not\ R_2'(X,Y,\mathbf{f_a}).\\
R_2'(X,Y,\mathbf{t^{\star\star}}) &\leftarrow R_2'(X,Y,\mathbf{t_a}).\\
&\leftarrow R_2'(X,Y,\mathbf{t_a}), R_2'(X,Y,\mathbf{f_a}).\\
S_2'(X,Y,\mathbf{t^{\star\star}}) &\leftarrow S_2'(X,Y,\mathbf{t_d}),\ \ not\ S_2'(X,Y,\mathbf{f_a}).\\
S_2'(X,Y,\mathbf{t^{\star\star}}) &\leftarrow S_2'(X,Y,\mathbf{t_a}).\\
&\leftarrow S_2'(X,Y,\mathbf{t_a}), S_2'(X,Y,\mathbf{f_a}).\\
R_1'(X,X,\mathbf{f_a}) &\leftarrow R_1'(X,Y,\mathbf{t_d}), S_1'(Z,Y,\mathbf{t_d}),\ \ not\ aux_1(X,Z),\\
&\qquad not\ aux_2(Z).\\
aux_1(X,Z) &\leftarrow R_2'(X,U,\mathbf{t_d}), S_2'(Z,U,\mathbf{t_d}).\\
aux_2(Z) &\leftarrow S_2'(Z,W,\mathbf{t_d}).\\
R_1'(X,Y,\mathbf{f_a}) \vee R_2'(X,W,\mathbf{t_a}) &\leftarrow R_1'(X,Y,\mathbf{t_d}), S_1'(Z,Y,\mathbf{t_d}),\ \ not\ aux_1(X,Z),\\
&\qquad S_2'(Z,W,\mathbf{t_d}), chosen(X,Z,W).\\
chosen(X,Z,W) &\leftarrow R_1'(X,Y,\mathbf{t_d}), S_1'(Z,Y,\mathbf{t_d}),\ \ not\ aux_1(X,Z),\\
&\qquad S_2'(Z,W,\mathbf{t_d}),\ \ not\ diffchoice(X,Z,W).\\
diffchoice(X,Z,W) &\leftarrow chosen(X,Z,U), S_2'(Z,W,\mathbf{t_d}), U \neq W.
\end{aligned}
$$

Running this program with DLV, we obtain the following stable models of the program, where td, ta, fa, tss stand for $\mathbf{t_d}, \mathbf{t_a}, \mathbf{f_a}, \mathbf{t^{\star\star}}$, resp.:

M_1= $\{R_1(a,b),\ S_1(c,b),\ S_2(c,e),\ S_2(c,f),\ R_1'(a,b,td),\ S_1'(c,b,td),\ S_2'(c,e,td),$
$S_2'(c,f,td),\ aux_2(c),\ S_1'(c,b,tss),\ S_2'(c,e,tss),\ S_2'(c,f,tss),\ R_1'(a,b,tss),$
$diffchoice(a,c,e),\ chosen(a,c,f),\ R_2'(a,f,ta),\ R_2'(a,f,tss)\}$
M_2= $\{R_1(a,b),\ S_1(c,b),\ S_2(c,e),\ S_2(c,f),\ R_1'(a,b,td),\ S_1'(c,b,td),\ S_2'(c,e,td),$
$S_2'(c,f,td),\ aux_2(c),\ S_1'(c,b,tss),\ S_2'(c,e,tss),\ S_2'(c,f,tss),\ R_1'(a,b,fa),$
$diffchoice(a,c,e),\ chosen(a,c,f)\}$
M_3= $\{R_1(a,b),\ S_1(c,b),\ S_2(c,e),\ S_2(c,f),\ R_1'(a,b,td),\ S_1'(c,b,td),\ S_2'(c,e,td),$
$S_2'(c,f,td),\ aux_2(c),\ S_1'(c,b,tss),\ S_2'(c,e,tss),\ S_2'(c,f,tss),\ R_1'(a,b,tss),$
$chosen(a,c,e),\ diffchoice(a,c,f),\ R_2'(a,e,ta),\ R_2'(a,e,tss)\}$
M_4= $\{R_1(a,b),\ S_1(c,b),\ S_2(c,e),\ S_2(c,f),\ R_1'(a,b,td),\ S_1'(c,b,td),\ S_2'(c,e,td),$
$S_2'(c,f,td),\ aux_2(c),\ S_1'(c,b,tss),\ S_2'(c,e,tss),\ S_2'(c,f,tss),\ R_1'(a,b,fa),$
$chosen(a,c,e),\ diffchoice(a,c,f)\}$,

which correspond to the following solutions (they can be obtained by selecting only the tuples with annotation $\mathbf{t}^{\star\star}$):

$$\begin{aligned}
\bar{r}^1 &= \{S'_1(c,b),\, S'_2(c,e),\, S'_2(c,f),\, R'_1(a,b),\, R'_2(a,f)\},\\
\bar{r}^2 &= \{S'_1(c,b),\, S'_2(c,e),\, S'_2(c,f)\},\\
\bar{r}^3 &= \{S'_1(c,b),\, S'_2(c,e),\, S'_2(c,f),\, R'_1(a,b),\, R'_2(a,e)\},\\
\bar{r}^4 &= \{S'_1(c,b),\, S'_2(c,e),\, S'_2(c,f)\}.
\end{aligned}$$