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 sematically 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 $\mathcal{P} = \{p_1, \dots, p_n\}$, and a fixed set of built-in predicates $\mathcal{B} = \{e_1, \dots, e_m\}$.

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

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

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 $DB = \{Book(kafka, metamorph, 1915), Book(kafka, meta-morph, 1919)\}$; 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 $DB_2 = \{Book(kafka, meta-morph, 1919)\}$. □

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, $DB' \models_\Sigma Q$.

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. □

---

[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'.

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].
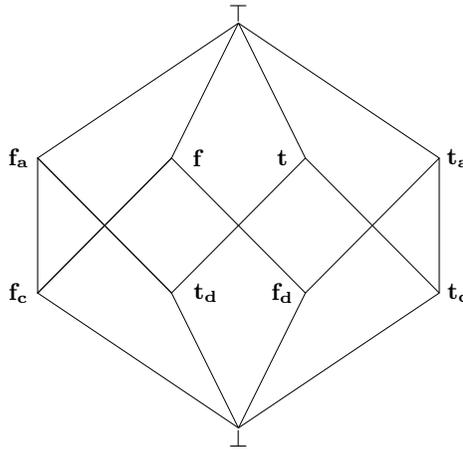
Fig. 1. *Latt* with *constraints values*, *database values* and *advisory values*

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* $DB_{\mathcal{M}} = \{p(\bar{a}) \mid \mathcal{M} \models p(\bar{a}){:}\mathbf{t} \ \vee \ p(\bar{a}){:}\mathbf{t_a}\}$.   □

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)*$:\mathbf{t_d}$.
   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:

3. $Book(x, y, z){:}\mathbf{f_c} \vee Book(x, y, w){:}\mathbf{f_c} \vee (z = w){:}\mathbf{t_c}$.
4. $Book(x, y, z){:}\mathbf{f_c} \vee Book(x, y, z){:}\mathbf{t_c}$, $\neg Book(x, y, z){:}\mathbf{f_c} \vee \neg Book(x, y, z){:}\mathbf{t_c}$.[3]
   These formulas specify that every fact must have one and just one constraint value.

Furthermore

---

[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] 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.

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

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:

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

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 $\{Book(neruda, 20\,love\,poems, 1924)\}$ 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:

$\mathcal{T}(DB) = \{Book(neruda, 20\,love\,poems, 1924) : \mathbf{t_d},\ Author(neruda, chilean) : \mathbf{f_d}, Author(neruda, canadian)\text{:}\mathbf{f_d}, \dots \}$.

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

$\mathcal{M}_1 = \{Book(neruda, 20\,love\,poems, 1924)\text{:}\mathbf{f_a},\ Author(neruda, chilean)\text{:}\mathbf{f}, Author(neruda, canadian)\text{:}\mathbf{f}, \dots \}$
$\mathcal{M}_2 = \{Book(neruda, 20\,love\,poems, 1924)\text{:}\mathbf{t},\ Author(neruda, chilean)\text{:}\mathbf{t_a}, Author(neruda, canadian)\text{:}\mathbf{f}, \dots \}$

---

[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.

$\mathcal{M}_3 = \{Book(neruda, 20\,love\,poems, 1924)\text{:}\mathbf{t},\ Author(neruda, chilean)\text{:}\mathbf{f},$
$\qquad Author(neruda, canadian)\text{:}\mathbf{t_a}, \dots\}.$

We obtain $DB_{\mathcal{M}_1} = \emptyset$, $DB_{\mathcal{M}_2} = \{Book(neruda, 20\,love\,poems, 1924), Author($
$neruda, chilean)\}$ 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:

$$\bigvee_{i=1}^{k} p_i(\bar{x}_i)\text{:}\mathbf{f_c} \vee \bigvee_{i=k+1}^{m} p_i(\bar{x}_i)\text{:}\mathbf{t_c} \vee \exists \bar{z}(\bigwedge_{j=1}^{l} q_j(\bar{y}_j)\text{:}\mathbf{t_c} \wedge \bigwedge_{j=l+1}^{r} q_j(\bar{y}_j)\text{:}\mathbf{f_c}).$$

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 $p(\bar{s})\text{:}\mathbf{f} \vee p(\bar{s})\text{:}\mathbf{f_a}$, and the positive literal $p(\bar{s})$ by the APC formula $p(\bar{s})\text{:}\mathbf{t} \vee p(\bar{s})\text{:}\mathbf{t_a}$. For $p \in \mathcal{B}$, the atom $p(\bar{s})$ is replaced by the APC formula $p(\bar{s})\text{:}\mathbf{t}$.* □

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:*

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

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:*

$p(\bar{x}, \mathbf{t}^\star) \leftarrow p(\bar{x}, \mathbf{t_d}).$ $p(\bar{x}, \mathbf{t}^\star) \leftarrow p(\bar{x}, \mathbf{t_a}).$
$p(\bar{x}, \mathbf{f}^\star) \leftarrow p(\bar{x}, \mathbf{f_a}).$ $p(\bar{x}, \mathbf{f}^\star) \leftarrow \; not \; p(\bar{x}, \mathbf{t_d}).,$

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:*

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

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

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*$(kafka, metamorph, 1919, \mathbf{t_d})$.   *Book*$(kafka, metamorph, 1919, \mathbf{t_d})$.
Book$(kafka, metamorph, 1915, \mathbf{t_d})$.

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$(x, y, z, \mathbf{f}^\star) \leftarrow Book(x, y, z, \mathbf{f_a})$.   *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:

$\mathcal{M}_1 = \{Book(kafka, metamorph, 1919, \mathbf{t_d}),\ Book(kafka, metamorph, 1919, \mathbf{t^\star}),$
$Book(kafka, metamorph, 1915, \mathbf{t_d}),\ Book(kafka, metamorph, 1915, \mathbf{t^\star}),$
$Book(kafka, metamorph, 1915, \mathbf{f_a}),\ Book(kafka, metamorph, 1915, \mathbf{f^\star}),$
$Eurbook(kafka, metamorph, 1919, \mathbf{t_d}),\ Eurbook(kafka, metamorph, 1919, \mathbf{t^\star})\};$

$\mathcal{M}_2 = \{Book(kafka, metamorph, 1919, \mathbf{t_d}),\ Book(kafka, metamorph, 1919, \mathbf{t^\star}),$
$Book(kafka, metamorph, 1919, \mathbf{f_a}),\ Book(kafka, metamorph, 1919, \mathbf{f^\star}),$
$Book(kafka, metamorph, 1915, \mathbf{t_d}),\ Book(kafka, metamorph, 1915, \mathbf{t^\star}),$
$Eurbook(kafka, metamorph, 1919, \mathbf{t_d}),\ Eurbook(kafka, metamorph, 1919, \mathbf{t^\star}),$
$Eurbook(kafka, metamorph, 1919, \mathbf{f_a}),\ Eurbook(kafka, metamorph, 1919, \mathbf{f^\star})\}.$ □

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:*

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

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 $\quad Q^{\star\star}(y) :$
$\exists z Book(kafka, y, z, \mathbf{t}^{\star\star})$, 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:

   $p(x, \mathbf{f}^\star) \leftarrow \; not \; p(x, \mathbf{t_d})$. $q(x, \mathbf{f}^\star) \leftarrow \; not \; q(x, \mathbf{t_d})$.
3. Annotation rules:

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

   $\leftarrow p(\bar{x}, \mathbf{t_a}), p(\bar{x}, \mathbf{f_a})$. $\leftarrow q(\bar{x}, \mathbf{t_a}), q(\bar{x}, \mathbf{f_a})$.
6. Interpretation rules:

   $p(x, \mathbf{t}^{\star\star}) \leftarrow p(x, \mathbf{t_a})$. $p(x, \mathbf{t}^{\star\star}) \leftarrow p(x, \mathbf{t_d}), \; not \; p(x, \mathbf{f_a})$.
   $p(x, \mathbf{f}^{\star\star}) \leftarrow p(x, \mathbf{f_a})$. $p(x, \mathbf{f}^{\star\star}) \leftarrow \; not \; p(x, \mathbf{t_d}), \; not \; p(x, \mathbf{t_a})$.
   $q(x, \mathbf{t}^{\star\star}) \leftarrow q(x, \mathbf{t_a})$. $q(x, \mathbf{t}^{\star\star}) \leftarrow q(x, \mathbf{t_d}), \; not \; q(x, \mathbf{f_a})$.
   $q(x, \mathbf{f}^{\star\star}) \leftarrow q(x, \mathbf{f_a})$. $q(x, \mathbf{f}^{\star\star}) \leftarrow \; not \; q(x, \mathbf{t_d}), \; not \; q(x, \mathbf{t_a})$.

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

   $r(x, \mathbf{f}^\star) \leftarrow not \; r(x, \mathbf{t_d})$. $r(x, \mathbf{f}^\star) \leftarrow r(x, \mathbf{f_a})$. $r(x, \mathbf{t}^\star) \leftarrow r(x, \mathbf{t_a})$.
   $r(X, \mathbf{t}^\star) \; \leftarrow \; r(X, \mathbf{t_d})$. $r(x, \mathbf{t}^{\star\star}) \; \leftarrow \; r(x, \mathbf{t_a})$. $r(x, \mathbf{t}^{\star\star}) \; \leftarrow$
   $r(x, \mathbf{t_d}), not \; r(x, \mathbf{f_a})$.
   $r(x, \mathbf{f}^{\star\star}) \leftarrow r(x, \mathbf{f_a})$. $r(x, \mathbf{f}^{\star\star}) \leftarrow not \; r(x, \mathbf{t_d}), not \; r(x, \mathbf{t_a})$.
   $q(x, \mathbf{f_a}) \vee r(x, \mathbf{t_a}) \leftarrow q(x, \mathbf{t}^\star), r(x, \mathbf{f}^\star)$. $\leftarrow r(x, \mathbf{t_a}), r(x, \mathbf{f_a})$.

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:

$$p(\bar{x}, \mathbf{f_a}) \vee q(\bar{x}', null, \mathbf{t_a}) \leftarrow p(\bar{x}, \mathbf{t^\star}), \; not \; aux(\bar{x}'), \; not \; q(\bar{x}', null, \mathbf{t_d}). \quad (5)$$
$$aux(\bar{x}') \leftarrow q(\bar{x}', y, \mathbf{t_d}), \; not \; q(\bar{x}', y, \mathbf{f_a}). \quad (6)$$
$$aux(\bar{x}') \leftarrow q(\bar{x}', y, \mathbf{t_a}). \quad (7)$$

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 $p(\bar{x}) \to \exists y q(\bar{x}, y)$, 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 $\forall \bar{x}(p(\bar{x}) \to \exists y q(\bar{x}', y))$, 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 $\forall \bar{x}(p(\bar{x}) \to \exists y q(\bar{x}', y)) \in IC_R$, 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 $\forall xy(q(x, y) \to r(x, y))$ together with the referential integrity constraints $\forall x(p(x) \to \exists y q(x, y))$ and $\forall x(s(x) \to \exists y r(x, y))$ 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]

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).
```

---

[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.

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 *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 $sh(\Pi)$ 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 $p_i(\bar{x}_i) \leftarrow \bigwedge_{j=1}^{m} q_j(\bar{y}_j) \wedge \bigwedge_{k \neq i} \textit{not } p_k(\bar{x}_k)$, $i = 1, ..., n$. 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].
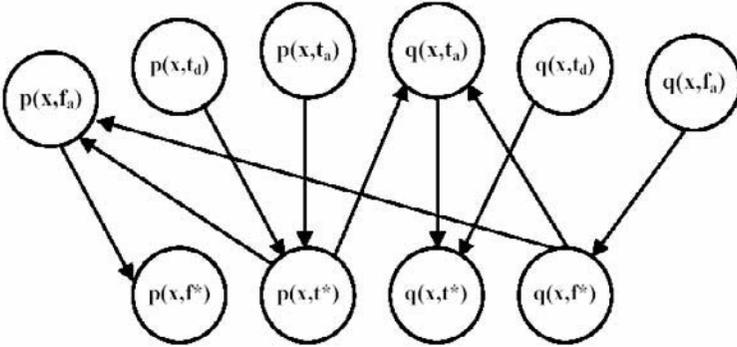
The repair program is HCF because, as it can be seen from the (relevant part of the) dependency graph, there is no cycle involving both $p(x, \mathbf{f_a})$ and $q(x, \mathbf{t_a})$), the atoms that appear in the only disjunctive head.

The non disjunctive version of the program has the disjunctive clause replaced by the two definite clauses $p(x, \mathbf{f_a}) \leftarrow p(x, \mathbf{t}^\star), q(x, \mathbf{f}^\star), \textit{not } q(x, \mathbf{t_a})$, and $q(x, \mathbf{t_a}) \leftarrow p(x, \mathbf{t}^\star), q(x, \mathbf{f}^\star), \textit{not } p(x, \mathbf{f_a})$. 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

$$q_1(\bar{t}_1) \vee \cdots \vee q_n(\bar{t}_n) \leftarrow p_1(\bar{s}_1) \wedge \cdots \wedge p_m(\bar{s}_m). \tag{9}$$

(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 17. 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 18. 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 19. 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 20. 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
f⋆⋆ or **f⋆** 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 21. (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. $in(a) \leftarrow in(a_1), \dots, in(a_k), out(b_1), \dots, out(b_m)$ has the intended procedural meaning of inserting the database atom a whenever $a_1, \dots, a_k$ are in the database, but not $b_1, \dots, b_m$. 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.

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.