From Database Repair Programs to Consistent Query Answering in Classical Logic (extended abstract)

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

Abstract. Consistent answers to a query from an inconsistent database are answers that can be simultaneously retrieved from every possible repair; and repairs are consistent instances that minimally differ from the original instance. Database repairs can be specified as the stable models of a disjunctive logic program. In this paper we show how to use the repair programs to transform the problem of consistent query answering into a problem of reasoning wrt a concrete theory written in second-order predicate logic. It also investigated how a first-order theory can be obtained instead, by applying second-order quantifier elimination techniques.

1 Introduction

Integrity constraints (ICs) are conditions that come with a relational schema $\mathcal{S}$, and should be satisfied by the instances of $\mathcal{S}$. In this way, database instances stay in correspondence with the outside reality they intend to model. If an instance D of $\mathcal{S}$ does not satisfy the ICs, it is said to be inconsistent. For several reasons a database instance may become inconsistent, and in consequence, it is only partially semantically correct.

Consistent query answering (CQA) in databases is about characterizing and computing answers to a query that are consistent wrt to a given set of integrity constraints. The database instance being queried may be inconsistent as a whole. However, via CQA only locally consistent information is extracted from the database. These problems have been investigated by the database community at least since the notion of consistent query answer was explicitly introduced in [4]. (Cf. [10, 17] for recent surveys of CQA.)

Informally, a tuple of constants $\bar{t}$ is a consistent answer to a query $\mathcal{Q}(\bar{x})$ from D wrt a set of ICs IC if $\bar{t}$ can be obtained as a usual answer to $\mathcal{Q}$ from every *repair* of D wrt *IC*, where a repair is a consistent instance of the schema $\mathcal{S}$ that differs from D by a minimal set of database atoms under set inclusion [4].

In [6] it was shown how repairs of a database D wrt a set of ICs can be specified as the stable models of a disjunctive Datalog program Π [27, 38, 21], a so-called *repair program*, whose set of facts corresponds to the original instance D. In this way, obtaining consistent answers becomes reasoning over the class of stable models of Π.

Example 1. Schema $\mathcal{S}$ contains a predicate $P(X,Y)$ and the functional dependency (FD), $X \rightarrow Y$, of attribute Y upon attribute X. It can be expressed in the first-order (FO) language $L(\mathcal{S})$ associated to $\mathcal{S}$, as the sentence $\forall x \forall y \forall z(P(x,y) \wedge P(x,z) \rightarrow y = z)$. $D = \{P(a,b), P(a,c), P(d,e)\}$ is inconsistent, and has two repairs: $D_1 = \{P(a,b), P(d,e)\}$ and $D_2 = \{P(a,c), P(d,e)\}$. The only consistent answer to the query $\mathcal{Q}_1(y): \exists x P(x,y)$ is (e), whereas those to $\mathcal{Q}_2(x): \exists y P(x,y)$ are $(a), (d)$.

The repairs can be specified by a logic program that contains, among other rules, a main rule that takes care of restoring consistency: $P(x,y,\mathbf{f}) \vee P(x,z,\mathbf{f}) \leftarrow P(x,y),$

⋆ Faculty member of the IBM Center for Advanced Studies, Toronto Lab.

$P(x,z), y \neq z$. It specifies that whenever the FD is violated by two tuples, which is captured by the body, then, as captured by the head, one (and only one if possible) of the tuples has to be deleted (made false, as indicated by the annotation constant $\mathbf{f}$).

Repair programs can always be used for CQA. However, as shown in [4], it is sometimes possible to obtain CQA by posing a new query to the inconsistent database. For example, the consistent answers to the query $\mathcal{Q}_3 : P(x,y)$ can be obtained by rewriting $\mathcal{Q}_3$ into $\mathcal{Q}_3' : P(x,y) \wedge \neg\exists z(P(x,z) \wedge z \neq y)$, and posing it to D, obtaining (d,e). ■
Ideally, consistent answers to $\mathcal{Q}(\bar{x}) \in L(\mathcal{S})$ from D should be obtained by posing a new query $\mathcal{Q}'(\bar{x}) \in L(\mathcal{S})$ to D, as an ordinary query: $D \models \mathcal{Q}'(\bar{x})$?. This can be done in polynomial time in $|D|$. Classes of queries and ICs with this property have been identified [4, 16, 26, 40]. Unfortunately, FO query rewriting has limited applicability: Even for conjunctive queries and FDs, CQA can be $coNP$-complete (in data) [16, 26].

Repair programs provide a general mechanism for CQA. Actually, the data complexity of CQA can be as high as the data complexity of cautious query evaluation from disjunctive logic programs under the stable model semantics, namely Π_2^P-complete [18, 16]. However, repair programs may be expensive for queries that can be answered more efficiently. It turns out that the complexity landscape between FO rewritable cases and Π_2^P-completeness for CQA is still not quite clear.

Those cases with FO rewritable CQA transform the problem into reasoning in classical predicate logic, because the original database can be "logically reconstructed" as a FO theory [39]. In this work we investigate how repair programs can be used to generate a theory written in classical logic from which CQA can be captured as logical entailment. We provide concrete specifications of database repairs in second-order (SO) classical logic. They are obtained by applying recent results on the specification in SO logic of the stable models of a logic program [23], and on their characterization as the models of a circumscription theory [36] in the stratified cases [37, 38]. Circumscription can be specified in SO classical logic [30].

In the case of FDs, we apply techniques for SO quantifier elimination introduced in [19], obtaining a FO specification of the database repairs. This transforms the problem of CQA into a problem of logical reasoning in FO logic. We illustrate by means of an example how to obtain a FO rewriting for CQA from this specification. In this work we concentrate mostly on FDs. Most of the complexity results in CQA have been obtained for FDs, but their complexity is not fully understood yet. We expect that the kind of results obtained in this work will help shed more light on this picture, in particular with respect to rewritability for CQA. These applications and others, like a better understanding of "the logic of CQA", are still to be developed.

In the Appendix we provide some basic notions related to disjunctive logic programs, stable model semantics, stratified disjunctive programs, and circumscription. We refer to [11] for the extended presentation of this work.

2 The Framework

The relational schema $\mathcal{S}$ contains a possible infinite domain $\mathcal{U}$ without nulls. $\mathcal{S}$ determines a language $L(\mathcal{S})$ of FO predicate logic, and ICs will be universal sentences in $L(\mathcal{S})$. (Cf. [12] for extensions to existential ICs and instances with nulls.) An instance D for $\mathcal{S}$ is a finite set of ground atoms of the form $R(\bar{a})$, with $R \in \mathcal{S}$ and $\bar{a}$ is a tuple of constants in $\mathcal{U}$.[1] D can be seen as a Herbrand structure [32] for interpreting $L(\mathcal{S})$,

[1] When we write something like $R \in \mathcal{S}$, we understand that R is a database predicate, not a built-in. For a tuple of constants $\bar{a} = (a_1, \ldots, a_k)$, $\bar{a} \in \mathcal{U}$ denotes $a_i \in \mathcal{U}$ for $i = 1, \ldots, k$.

namely $\langle \mathcal{U}, (R^D)_{R \in \mathcal{S}}, (u)_{u \in \mathcal{U}} \rangle$, with $R^D = \{R(\bar{a}) \mid R(\bar{a}) \in D\}$. The database D can also be logically reconstructed as a first-order sentence $\mathcal{R}(D)$, as done by Reiter in [39].

Example 2. If $\mathcal{S}$ has domain $\mathcal{U} = \{a, b, c, d, e, f, g\}$ and predicate $P(\cdot, \cdot)$, then $D = \{P(a, b), P(a, c), P(d, e)\}$ is an instance for $\mathcal{S}$. In this case, $\mathcal{R}(D)$ is the conjunction of the following sentences: (a) Domain Closure Axiom (DCA): $\forall x(x = a \vee x = b \vee x = c \vee x = d \vee x = e \vee x = f \vee x = g)$. (b) Unique Names Axiom (UNA): $(a \neq b \wedge \cdots \wedge f \neq g)$. (c) Predicate Completion Axiom (PCA): $\forall x \forall y(P(x, y) \equiv (x = a \wedge y = b) \vee (x = a \wedge y = c) \vee (x = d \wedge y = e))$. The theory $\mathcal{R}(D)$ is categorical, i.e. D is essentially its only model. ■

In the previous example, the domain is finite, which makes it possible to use a domain closure axiom. If the domain $\mathcal{U}$ is infinite, the domain closure axiom (DC) is applied to the active domain, $Ac(D)$, of the database, i.e. to the set of constants appearing in the relations of the database instance. Since the extensions of the predicates are always finite, we can always build a DCA and PCAs. For static databases and CQA wrt universal ICs, the active domain suffices to restore consistency and define repairs. If we restrict ourselves to Herbrand structures, we do not need the DCA or the UNA.

2.1 Database repairs

A *repair* of D wrt a set *IC* of ICs is an instance D' over $\mathcal{S}$ that satisfies IC, i.e. $D' \models IC$, and makes the symmetric set-difference $\Delta(D, D')$ minimal wrt set inclusion [4]. $Rep(D, IC)$ denotes the set of repairs of D wrt *IC*.

Given D and *IC*, a disjunctive logic *repair program* $\Pi(D, IC)$ with stable model semantics [27] can be used to specify $Rep(D, IC)$. More precisely, (all and only) the repairs of D can be read-off from the stable models of $\Pi(D, IC)$. Because of their simplicity and scope, we will use the repair programs introduced in [6] in their slightly modified version in [13]. Earlier forms of repair programs can also be found in [5, 28]. Repair programs use annotation constants in an extra argument of each of the database pre-
dicates. More precisely, for each n-ary $P \in \mathcal{S}$, we make a copy $P\_$, which is $(n+1)$-ary. The intended semantics of the annotations is indicated in the following table.

Annotation	Atom	The tuple $P(\bar{a})$ is:
$\mathbf{t}$	$P\_(\bar{a}, \mathbf{t})$	made true/inserted
$\mathbf{f}$	$P\_(\bar{a}, \mathbf{f})$	made false/deleted
$\mathbf{t}^\star$	$P\_(\bar{a}, \mathbf{t}^\star)$	true or made true
$\mathbf{t}^{\star\star}$	$P\_(\bar{a}, \mathbf{t}^{\star\star})$	true in the repair

Example 3. Consider $IC: \ \forall xy(P(x, y) \rightarrow Q(x, y))$; and the inconsistent database instance $D = \{P(c, l), P(d, m), Q(d, m), Q(e, k)\}$. The repair program $\Pi(D, IC)$ has the following rules (and facts):

1. Original database facts: $P(c, l)$, etc.
2. Whatever was true or becomes true, is annotated with $\mathbf{t}^\star$:
   $P\_(\bar{x}, \mathbf{t}^\star) \leftarrow P(\bar{x})$. $\quad P\_(\bar{x}, \mathbf{t}^\star) \leftarrow P\_(\bar{x}, \mathbf{t})$. $\quad$ (the same for Q)
3. There may be interacting ICs (not here), and the repair process may take several steps, changes could trigger other changes:
   $P\_(\bar{x}, \mathbf{f}) \ \vee \ Q\_(\bar{x}, \mathbf{t}) \ \leftarrow \ P\_(\bar{x}, \mathbf{t}^\star), Q\_(\bar{x}, \mathbf{f})$.
   $P\_(\bar{x}, \mathbf{f}) \ \vee \ Q\_(\bar{x}, \mathbf{t}) \ \leftarrow \ P\_(\bar{x}, \mathbf{t}^\star), \ \mathit{not}\ Q(\bar{x})$.
   Two rules per IC that say how to repair the satisfaction of the IC (cf. the head) in case of a violation (cf. the body). Passing to annotation $\mathbf{t}^\star$ allows to keep repairing the database wrt to all the ICs until the process stabilizes.
4. Program constraints: $\quad \leftarrow P\_(\bar{x}, \mathbf{t}), P\_(\bar{x}, \mathbf{f})$. $\quad$ (similarly for Q)

5. Annotations constants $\mathbf{t}^{\star\star}$ are used to read off the atoms in a repair:
   $P\_(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow P\_(\bar{x}, \mathbf{t})$. $P\_(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow P\_(\bar{x}, \mathbf{d}),\ not\ P\_(\bar{x}, \mathbf{f})$. (similarly for Q)

The *program constraints* in 4. are used to filter out *incoherent models*, with some both inserted and deleted tuple. In this example, we do not need them, because this never happens. However, they may be necessary when there are interacting ICs [14]. ■

General repair programs can be found in [11]. From now on, we use, for simplicity, $P_f(\_,\_)$ for $P\_(\_,\_,\mathbf{f})$, $P_{\star\star}(\_,\_)$ for $P\_(\_,\_,\mathbf{t}^{\star\star})$, etc. That is, annotations are "predicated" using new predicates. Repairs are in one-to-one correspondence with the restriction of the stable models to their atoms that use predicates of the form $P_{\star\star}$ [12].

2.2 Queries and consistent answers

A tuple $\bar{a} \in \mathcal{U}$ is a *consistent answer* to a query $\mathcal{Q}(\bar{x})$ from D wrt IC, denoted $D \models_c \mathcal{Q}(\bar{a})$, iff $D' \models \mathcal{Q}(\bar{a})$ for every $D' \in Rep(D, IC)$ [4]. In this paper, $\mathcal{Q}$ will usually be a safe query [1] in $L(\mathcal{S})$, e.g. a conjunctive query with built-ins. In order to pose this query to the (models of the) repair program, i.e. to the repairs, it has to be reformulated as a query $\mathcal{Q}^{\star\star}$ that is obtained from Q by replacing each database predicate P by $P_{\star\star}$. For example, for $\mathcal{Q}(y):\ \exists x(P(x,y) \wedge \neg Q(x,y) \wedge x \neq y)$, $\mathcal{Q}^{\star\star}(y)$ is $\exists x(P_{\star\star}(x,y) \wedge \neg Q_{\star\star}(x,y) \wedge x \neq y)$. The query $\mathcal{Q}$ could also be given as a (safe) Datalog query (or in any of its extensions) [1]. In this case, $\mathcal{Q}^{\star\star}$ is obtained from $\mathcal{Q}$ by replacing every extensional predicate P by $P_{\star\star}$.

Repair programs can be used to obtain consistent answers to $\mathcal{Q}$ as cautions (or skeptical) answers from the combined program consisting of the repair program and a query program $\Pi^{\mathcal{Q}}$. Given a FO query $\mathcal{Q}(\bar{x})$, $\mathcal{Q}^{\star\star}(\bar{x})$ is rewritten as a Datalog query $\Pi^{\mathcal{Q}}$, possibly containing weak negation, not. $\Pi^{\mathcal{Q}}$ contains a predicate, $Ans^{\mathcal{Q}}(\bar{x})$, appearing only in heads, to collect the query answers. If $\mathcal{Q}$ is given directly as a Datalog program with negation, then Π^{Q} is simply $\mathcal{Q}^{\star\star}$. According to the usual conventions, we will assume that such Datalog queries $\mathcal{Q}$ are stratified normal programs, most usually, a non-recursive Datalog^{not} query [1], that is obtained as a translation of a FO query. It holds:

$$D \models_c \mathcal{Q}(\bar{a}) \iff D' \models \mathcal{Q}(\bar{a}),\ \text{for every } D' \in Rep(D, IC) \qquad (1)$$

$$\iff \Pi(D, IC) \cup \Pi^{\mathcal{Q}} \models_{cs} Ans^{\mathcal{Q}}(\bar{a}), \qquad (2)$$

where $\models_{cs}$ stands for *cautious*, i.e. being true in all stable models. If on the LHS of (1) $\mathcal{Q}$ is already a Datalog program, $D' \models \mathcal{Q}(\bar{a})$ means that $\bar{a}$ is an answer to the Datalog query when using D' as the underlying extensional database of program facts.

2.3 Functional dependencies

For some classes of FDs and conjunctive queries there are efficient algorithms for CQA based on FO query rewriting [4, 16, 26, 40]. In [25, 40] there are examples of conjunctive queries for which CQA wrt certain FDs is in $PTIME$, but there is no consistent FO rewriting of the query. FDs are particular cases of *denial constraints*, i.e. sentences of the form $\bar{\forall}\neg(A_1 \wedge \cdots \wedge A_m)$, where the A_i are database or built-in atoms, and $\bar{\forall}$ denotes the universal closure of the formula.

In [7], it is proved that for certain classes of ICs, that include all denial constraints, the repair programs become *head-cycle free* (HCF). For this class of programs cautious query evaluation becomes $coNP$-complete [8, 18]. It follows that CQA of conjunctive queries wrt functional dependencies belongs $coNP$ [7]. For conjunctive queries and certain functional dependencies (actually, a single key dependency suffices), CQA turns out to be $coNP$-complete [16, 26, 40], matching the upper bound provided by the repair program.

Example 4. (example 1 continued) The repair program $\Pi(D, IC)$ is:

$$P_f(x,y) \vee P_f(x,z) \leftarrow P(x,y), P(x,z), y \neq z. \tag{3}$$

$$P_{\star\star}(x,y) \leftarrow P(x,y), not\ P_f(x,y). \tag{4}$$

$$P(a,b).\ \ P(a,c).\ \ P(d,e). \tag{5}$$

The first rule solves conflicts between two tuples by deleting one of them from the database. The second rule collects the tuples that remain after all conflicts have been solved. For FDs we do not need the annotation $\mathbf{t}$ or program constraints, because inconsistencies are resolved by deletions. The repairs are obtained as restrictions of the two stable models to predicate $P_{\star\star}$: $D_1 = \{P_{\star\star}(a,b), P_{\star\star}(d,e)\}$ and $D_2 = \{P_{\star\star}(a,c), P_{\star\star}(d,e)\}$. In the former, the tuple $P(a,c)$ is deleted from the database; in the latter, the tuple $P(a,b)$.

The query $\mathcal{Q}(x,y)\!: P(x,y)$ can be represented as the program $\Pi^{\mathcal{Q}}$: $Ans(x,y) \leftarrow P_{\star\star}(x,y)$. If Π is the program consisting of this query plus (3)-(5), the consistent answers to query $\mathcal{Q}$ are those tuples $\bar{a}$, such that $\Pi \models_{cs} Ans(\bar{a})$. ■

We can see that repair programs for FDs are stratified disjunctive programs [37]. They are also HCF programs, which makes it possible to translate them into equivalent normal (non-disjunctive) programs [8, 18]. However, they are not stratified as normal programs.

3 SO Specification of Repairs

In [23], the stable model semantics of logic programs introduced in [27] is reobtained via a concrete and explicit form of specification in classical SO predicate logic: First, the program Π is transformed into (or seen as) a FO sentence $\psi(\Pi)$. Next, the latter is transformed into a SO sentence $\Phi(\Pi)$, the *stable sentence* of the program.

$\psi(\Pi)$ is obtained from Π as follows: (a) Replace every comma by $\wedge$, and every not by $\neg$. (b) Turn every rule $Head \leftarrow Body$ into the formula $Body \rightarrow Head$. (c) Form the conjunction of the universal closures of those formulas.

Now, given a FO sentence ψ (e.g. the $\psi(\Pi)$ above), a SO sentence Φ is defined as $\psi \wedge \neg\exists\bar{X}((\bar{X} < \bar{P}) \wedge \psi^\circ(\bar{X}))$, where $\bar{P}$ is the list of all non-logical predicates $P_1, ..., P_n$ in ψ, and $\bar{X}$ is a list of distinct predicate variables $X^{P_1}, ..., X^{P_n}$, with P_i and X^{P_i} of the same arity. Here, $(\bar{X} < \bar{P})$ means $(\bar{X} \leq \bar{P}) \wedge (\bar{X} \neq \bar{P})$, i.e. $\bigwedge_i^n \forall\bar{x}(X^{P_i}(\bar{x}) \rightarrow P_i(\bar{x})) \wedge \bigvee_i^n (X^{P_i} \neq P_i)$. $X^{P_i} \neq P_i$ stands for $\exists\bar{x}_i(P_i(\bar{x}_i) \wedge \neg X^{P_i}(\bar{x}_i))$.

$\psi^\circ(\bar{X})$ is defined recursively as follows: (a) $P_i(t_1, ..., t_m)^\circ := X^{P_i}(t_1, ..., t_m)$. (b) $(t_1 = t_2)^\circ := (t_1 = t_2)$. (c) $\bot^\circ{:=}\bot$. (d) $(F \odot G)^\circ := (F^\circ \odot G^\circ)$ for $\odot \in \{\wedge, \vee\}$. (e) $(F \rightarrow G)^\circ := (F^\circ \rightarrow G^\circ) \wedge (F \rightarrow G)$. (f) $(QxF)^\circ := QxF^\circ$ for $Q \in \{\forall, \exists\}$.

The Herbrand models of the SO sentence $\Phi(\Pi)$ associated to $\psi(\Pi)$ correspond to the stable models of the original program Π [23]. We can see that $\Phi(\Pi)$ is similar to a parallel circumscription of the predicates in program Π wrt the FO sentence $\psi(\Pi)$ associated to Π [36, 31]. In principle, the transformation rule (e) above could make formula $\Phi(\Pi)$ differ from a circumscription.

Now, let Π^r be the repair program without the database facts, and $\mathcal{Q}(\bar{x})$ a query represented by a non-recursive normal Datalog^{not} query $\Pi^{\mathcal{Q}}$ with answer predicate $Ans^{\mathcal{Q}}(\bar{x})$. From now on,

$$\Pi = D \cup \Pi^r \cup \Pi^{\mathcal{Q}} \tag{6}$$

denotes the program for consistently answering $\mathcal{Q}$. That is, $\Pi = \Pi(D, IC) \cup \Pi^{\mathcal{Q}}$. Notice that Π^r depends only on the ICs, and it includes definitions for the annotation predicates. The only predicates shared by Π^r and $\Pi^{\mathcal{Q}}$ are those of the form $P_{\star\star}$, with $P \in \mathcal{S}$, and they appear only in bodies in $\Pi^{\mathcal{Q}}$. These predicates produce a *splitting* of the combined program, whose stable models are obtained as extensions of the stable models for $\Pi(D, IC)$ [34]. In Example 4, Π^r is formed by rules (3) and (4); D is the set of facts in (5); and $\Pi^{\mathcal{Q}}$ is $Ans(x, y) \leftarrow P_{\star\star}(x, y)$.

The just mentioned splitting of Π allows us to analyze separately Π^r and $\Pi^{\mathcal{Q}}$. Since the latter is a non-recursive normal program, it is stratified, and its only stable model (over a give extension for its extensional predicates) can be obtained by predicate completion, or a prioritized circumscription [37]. Actually, if the query is given directly as FO query, we can use instead of the completion (or circumscription) of its associated program, the FO query itself. In consequence, in the rest of this section we concentrate mostly on the facts-free repair program Π^r.

In the following, we will omit the program constraints from the repair programs, because their transformation via the SO sentence of the program is straightforward: We obtain as a conjunct of the SO sentence, the sentence $\forall \bar{x} \neg (P_t(\bar{x}) \wedge P_f(\bar{x}))$ [24, Prop. 2] for a program constraint of the form $\leftarrow P_t(\bar{x}), P_f(\bar{x})$.

Example 5. (example 4 continued) We first obtain the FO sentence $\psi(\Pi)$:

$$\begin{aligned} P(a,b) \wedge P(a,c) \wedge P(d,e)\ \wedge \\ \forall xyz(((P(x,y) \wedge P(x,z) \wedge y \neq z) \rightarrow (P_f(x,y) \vee P_f(x,z)))\ \wedge \\ \forall xy((P(x,y) \wedge \neg P_f(x,y)) \rightarrow P_{\star\star}(x,y))\ \wedge\ \forall xy(P_{\star\star}(x,y) \rightarrow Ans(x,y)). \end{aligned} \tag{7}$$

The second-order formula $\Phi(\Pi)$ that captures the stable models of the original program is the conjunction of (7) and (with $<$ below being the "parallel" pre-order [30])

$$\begin{aligned} & \neg \exists X^P X^P_f X^P_{\star\star} X^{Ans}\ [\ (X^P, X^P_f, X^P_{\star\star}, X^{Ans}) < (P, P_f, P_{\star\star}, Ans)\ \wedge \\ & X^P(a,b) \wedge X^P(a,c) \wedge X^P(d,e)\ \wedge \\ & \forall xyz(X^P(x,y) \wedge X^P(x,z) \wedge y \neq z \rightarrow X^P_f(x,y) \vee X^P_f(x,z))\ \ \wedge \\ & \forall xyz(P(x,y) \wedge P(x,z) \wedge y \neq z \rightarrow P_f(x,y) \vee P_f(x,z))\ \ \wedge && (8) \\ & \forall xy(X^P(x,y) \wedge (\neg P_f(x,y))^{\circ} \rightarrow X^P_{\star\star}(x,y))\ \ \wedge && (9) \\ & \forall xy(P(x,y) \wedge \neg P_f(x,y)) \rightarrow P_{\star\star}(x,y))\ \ \wedge && (10) \\ & \forall xy(X^P_{\star\star}(x,y) \rightarrow X^{Ans}(x,y))\ \ \wedge \\ & \forall xy(P_{\star\star}(x,y) \rightarrow Ans(x,y))]. && (11) \end{aligned}$$

From this sentence, the conjuncts (8), (10) and (11), that already appear in (7), can be eliminated. The formula $(\neg P_f(x,y))^{\circ}$ in (9) has to be expressed as $(P_f(x,y) \rightarrow \bot)^{\circ}$. It turns out that, being the $\circ$-transformation of a negative formula, it can be replaced by its original version without predicate variables, i.e. by $\neg P_f(x,y)$ [23, Prop. 2]. We obtain that $\Phi(\Pi)$ is logically equivalent to the conjunction of the UNA and DCA[2] and

[2] From now on, unless stated otherwise, the UNA and DCA will be always implicitly considered.

(modulo some standard simplification techniques for SO quantifiers [30, 31]):

$$\forall xy(P(x,y) \equiv (x = a \wedge y = b) \vee (x = a \wedge y = c) \vee (x = d \wedge y = e)) \;\wedge \quad (12)$$
$$\forall xy(P_{\star\star}(x,y) \equiv Ans(x,y)) \;\wedge \quad (13)$$
$$\forall xy((P(x,y) \wedge \neg P_f(x,y)) \equiv P_{\star\star}(x,y)) \;\wedge \quad (14)$$
$$\forall xyz(P(x,y) \wedge P(x,z) \wedge y \neq z \rightarrow (P_f(x,y) \vee P_f(x,z))) \;\;\wedge \quad (15)$$
$$\neg\exists U_f((U_f < P_f) \wedge \forall xyz(P(x,y) \wedge P(x,z) \wedge y \neq z \rightarrow (U_f(x,y) \vee U_f(x,z))).\quad (16)$$

Here, $U_f < P_f$ stands for the formula $\forall xy(U_f(x,y) \rightarrow P_f(x,y)) \wedge \exists xy(P_f(x,y) \wedge \neg U_f(x,y))$. In this sentence, the minimization of predicates $P, P_{\star\star}$ and Ans are expressed by their predicate completions. Predicate P_f is minimized via (16). ■

In this example we have obtained the SO sentence for program Π as a parallel circumscription of the predicates in the repair program seen as a FO sentence. Actually, the circumscription becomes a *prioritized circumscription* [30] given the stratified nature of the repair program: first the database predicate is minimized, next P_f, next $P_{\star\star}$, and finally Ans. As we state in Proposition 1, repair programs in their predicated-annotation version and without their program constraints become *stratified disjunctive Datalog programs* [21, 37].[3]

Proposition 1. [15] For universal integrity constraints, repairs programs without their program constraints are stratified, and the upwards stratification is as follows: 0. Extensional database predicates $P \in \mathcal{S}$; 1. Predicates of the form $P_f, P_t, P_\star$; and 2. Predicates of the form $P_{\star\star}$. ■

If a stratified query program is run on top of the repair program, the combined program becomes stratified, with the stratification of the query on top of the one of the repair program. It is worth noticing that the data complexity of cautious query evaluation from disjunctive logic programs with stratified negation is the same as for disjunctive logic programs with unstratified negation and stable model semantics, namely Π^P_2-complete [21] The stable models of the combined (stratified and disjunctive) program Π coincide with the *perfect models* of the program [38], and the latter can be obtained as the (Herbrand) models of a prioritized circumscription that follows the stratification of the program [37]. Program constraints can be added at the end, after producing a circumscription (or the SO stable sentence of Π). In consequence, we obtain the following

Proposition 2. For a set of universal ICs, the SO sentence Φ associated to a repair program $\Pi(IC, D)$ is logically equivalent to

$$\mathcal{R}(D) \wedge \bigwedge_{P \in \mathcal{S}} \forall \bar{x}((P(\bar{x}) \vee P_t(\bar{x})) \equiv P_\star(\bar{x})) \wedge \bigwedge_{P \in \mathcal{S}} \forall \bar{x}(P_\star(\bar{x}) \wedge \neg P_f(\bar{x}) \equiv P_{\star\star}(\bar{x}))$$
$$\wedge \bigwedge_{P \in \mathcal{S}} \forall \bar{x} \neg (P_t(\bar{x}) \wedge P_f(\bar{x})) \wedge \mathit{Circ}(\Theta; \{P_t, P_f \mid P \in \mathcal{S}\}; \{P_\star \mid P \in \mathcal{S}\}).\quad (17)$$

Here, the last conjunct is the *parallel circumscription* [30] of the predicates in the second argument (with variable $P_\star$ predicates) wrt the theory Θ obtained from the conjunction rules in the repair program that are relevant to compute the P_t, P_f's, seen as FO sentences.[4] ■

[3] Program constraints spoil the stratification, because they have to be replaced by rules of the form $p \leftarrow P_t(\bar{x}), P_f(\bar{x}),\ not\ p$.

[4] They are rules 1.- 3. in Example 3.

This result has been obtained from the stratification of the repair programs. However, it is possible to obtain the same result by simplifying the SO sentence associated to it, as done in Example 5. Notice that the more involved repair program in Example 3 already contains the relevant features of a general repair program for universal ICs, namely the negations in the rule bodies affect only base predicates and the predicates $P_\star$ in the definitions of the $P_{\star\star}$ [35]. In any case, we obtain a SO specification of the program for CQA Π in (6). Combining with (1), we obtain

$$D \models_c \mathcal{Q}(\bar{a}) \iff \Phi(\Pi) \models Ans^{\mathcal{Q}}(\bar{a}), \quad (18)$$

where $\Phi(\Pi)$ is the SO sentence which captures the stable models of Π.[5] Actually, $\Phi(\Pi)$ can be decomposed as the conjunction of three formulas:

Proposition 3. Let Φ be the SO sentence for the program Π in (6) for CQA. It holds:

$$D \models_c \mathcal{Q}(\bar{a}) \iff \{\mathcal{R}(D), \Phi(\Pi^r), \Phi(\Pi^{\mathcal{Q}})\} \models Ans(\bar{a}).$$

Here, $\Phi(\Pi^r)$ is a SO sentence that specifies the repairs for fixed extensional predicates, and $\Phi(\Pi^{\mathcal{Q}})\}$ a SO sentence that specifies the models of the query, in particular predicate $Ans^{\mathcal{Q}}$, for fixed predicates $P_{\star\star}$. ■

Example 6. (example 5 continued) $\mathcal{R}(D)$ is captured by the DCA, UNA plus (12); $\Phi(\Pi^r)$ by (14)-(16); and $\Phi(\Pi^{\mathcal{Q}})$ by (13). Actually, what we have obtained is that for consistent answers (t_1, t_2), it holds

$$\Psi \wedge \forall x \forall y (Ans(x,y) \equiv P_{\star\star}(x,y)) \models Ans(t_1, t_2), \quad (19)$$

where Ψ is the SO sentence that is the conjunction of (12), (14)-(16). ■

We have transformed CQA into a problem of reasoning in classical SO predicate logic. Most commonly the query $\mathcal{Q}$ will be given as a FO query or as a safe and non-recursive Datalog^{not} program. In these cases, $\Phi(\Pi^{\mathcal{Q}})$ is obtained by predicate completion and will contain as a conjunct an explicit definition of predicate $Ans^{\mathcal{Q}}$. The definition of $Ans^{\mathcal{Q}}$ will be of the form $\forall \bar{x}(Ans^{\mathcal{Q}}(\bar{x}) \equiv \Psi(\bar{x}))$, where $\Psi(\bar{x})$ is a FO formula containing only predicates of the form $P_{\star\star}$, with $P \in S$, plus possibly some built-ins and auxiliary predicates. For example, in (19) we have an explicit definition of Ans.

4 Scaling-Down Repair Programs for CQA under FDs

We discuss in this section the possibility of using a program for CQA Π of the form (6) to obtain a FO theory from which to do CQA as classical entailment. In particular, exploring the possibility of obtaining a FO rewriting of the original query. The idea is to do it through the analysis of the SO sentence associated to the program. In order to explore the potentials of this approach, we restrict ourselves to the case of FDs, the most studied case in the literature wrt complexity of CQA [16, 26, 40].

We start with a schema with a predicate $P(X,Y)$, with the $FD: X \rightarrow Y$, as in Example 1. In this case, the repair program $\Pi(D, FD)$ is associated to the circumscription of P_f given by the conjunction of (12), (14)-(16). We concentrate on the last conjunct, (16), which can be expressed as

$$\neg \exists U_f((U_f < P_f) \wedge \forall xyz(\kappa(x,y,z) \rightarrow (U_f(x,y) \vee U_f(x,z))), \quad (20)$$

[5] If we omit the DCA and UNA axioms, on the RHS the logical consequence is relative to Herbrand models.

where $\kappa(x,y,z)$ is the formula $P(x,y) \wedge P(x,z) \wedge y \neq z$, that captures the inconsistencies wrt *FD*.

We will apply to (20) the techniques for elimination of SO quantifiers developed in [19] on the basis of Ackerman's Lemma [2, 3]. First of all, we express (20) as an equivalent universally quantified formula (for simplicity, we use U instead of U_f):

$$\forall U(\forall xyz(\kappa(x,y,z) \rightarrow U(x,y) \vee U(x,z)) \wedge U \leq P_f \ \rightarrow \ P_f \leq U). \tag{21}$$

Its negation produces the existentially quantified formula

$$\exists U(\forall xyz(\kappa(x,y,z) \rightarrow U(x,y) \vee U(x,z)) \wedge U \leq P_f \wedge \neg P_f \leq U). \tag{22}$$

We obtain the following logically equivalent formulas

$$\begin{aligned} \exists U(\ & \forall xyz(\neg\kappa(x,y,z) \vee U(x,y) \vee U(x,z)) \wedge \forall uv(\neg U(u,v) \vee P_f(u,v)) \\ & \qquad\qquad \wedge \exists st(P_f(s,t) \wedge \neg U(s,t))). \\ \exists st \exists U(\ & \forall xyz(\neg\kappa(x,y,z) \vee U(x,y) \vee U(x,z)) \wedge \\ & \qquad \forall uv(\neg U(u,v) \vee P_f(u,v)) \wedge (P_f(s,t) \wedge \neg U(s,t))). \end{aligned} \tag{23}$$

The first conjunct in (23), with $w = \vee(y,z)$ standing for $(w = y \vee w = z)$, can be written as (cf. [11] for all the details): $\forall xyz(\neg\kappa(x,y,z) \vee \exists w(w = \vee(y,z) \wedge U(x,w)))$. Equivalently, $\exists f \forall r(\forall x_1y_1z_1(\neg\kappa(x_1,y_1,z_1) \vee f(x_1,y_1,z_1) = \vee(y_1,z_1)) \wedge$
$\forall xyz(\neg\kappa(x,y,z) \vee r \neq f(x,y,z) \vee U(x,r)))$.

Here, $\exists f$ is a quantification over functions. Thus, formula (23) becomes

$$\begin{aligned} & \exists st \exists f \exists U \forall x \forall r((\forall x_1y_1z_1(\neg\kappa(x_1,y_1,z_1) \vee f(x_1,y_1,z_1) = \vee(y_1,z_1)) \wedge \\ & \qquad \forall yz(\neg\kappa(x,y,z) \vee r \neq f(x,y,z) \vee U(x,r))) \wedge \\ & \qquad\qquad \forall uv(\neg U(u,v) \vee P_f(u,v)) \wedge (P_f(s,t) \wedge \neg U(s,t))). \end{aligned}$$

Now we are ready to apply Ackermann's lemma. The last formula can be written as

$$\exists st \exists f \exists U \forall x \forall r((A(x,r) \vee U(x,r)) \wedge B(\neg U \mapsto U)). \tag{24}$$

$B(\neg U \mapsto U)$ denotes the formula B where predicate U has been replaced by $\neg U$. Here, formulas A, B are as follows

$$\begin{aligned} A(x,r): \ & \forall yz(\forall yz(\neg\kappa(x,y,z) \vee r \neq f(x,y,z)). \\ B(U): \ & \forall x_1y_1z_1(\neg\kappa(x_1,y_1,z_1) \vee f(x_1,y_1,z_1) = \vee(y_1,z_1)) \wedge \\ & \qquad \forall uv(U(u,v) \vee P_f(u,v)) \wedge (P_f(s,t) \wedge U(s,t))). \end{aligned}$$

Formula B is positive in U. In consequence, the whole subformula in (24) starting with $\exists U$ can be equivalently replaced by $B(A(x,r) \mapsto U)$ [19, lemma 1], getting rid of the SO variable U, and thus obtaining (modulo simple syntactic steps)

$$\begin{aligned} & \exists st \exists f \forall xyz((\neg\kappa(x,y,z) \vee f(x,y,z) = \vee(y,z)) \wedge (\neg\kappa(x,y,z) \vee P_f(u, f(x,y,z))) \wedge \\ & \qquad (P_f(s,t) \wedge (x \neq s \vee \neg\kappa(x,y,z) \vee t \neq f(x,y,z)))). \end{aligned}$$

Now we unskolemize, getting rid of the function variable f, obtaining

$$\begin{aligned} & \exists st \forall xyz \exists w((\neg\kappa(x,y,z) \vee w = \vee(y,z)) \wedge (\neg\kappa(x,y,z) \vee P_f(u,w)) \wedge \\ & \qquad (P_f(s,t) \wedge (x \neq s \vee \neg\kappa(x,y,z) \vee t \neq w))). \end{aligned}$$

This formula is logically equivalent to the negation of (21). Negating again, we obtain a formula that is logically equivalent to (21), namely

$$\begin{aligned} & \forall st(P_f(s,t) \rightarrow \exists xyz(\kappa(x,y,z) \wedge \forall w[(w \neq y \wedge w \neq z) \vee \\ & \qquad \neg P_f(x,w) \vee (x = s \wedge t = w)]). \end{aligned}$$

The subformula inside the square brackets can be equivalently replaced by

$$((w = y \vee w = z) \wedge P_f(x,w)) \rightarrow (s = x \wedge t = w).$$

So, we obtain $\forall st(P_f(s,t) \rightarrow \exists xyz(\kappa(x,y,z) \wedge (P_f(x,y) \rightarrow s = x \wedge t = y) \wedge$
$(P_f(x,z) \rightarrow s = x \wedge t = z)))$.

Due to the definition of $\kappa(x,y,z)$, it must hold $y \neq z$. In consequence, we obtain

$\forall st(P_f(s,t) \rightarrow \exists z(\kappa(s,t,z) \wedge \neg P_f(s,z)))$.

Proposition 4. Let FD be $\forall xyz(P(x,y) \wedge P(x,z) \rightarrow y = z)$. The SO sentence for the repair program $\Pi(D, FD)$ is logically equivalent to a FO sentence, namely to the conjunction of (12), (14) (i.e. the completions of the predicates $P, P_{\star\star}$, resp.), (15), and

$$\forall st(P_f(s,t) \rightarrow \exists z(\kappa(s,t,z) \wedge \neg P_f(s,z))), \tag{25}$$

where $\kappa(x,y,z)$ is the formula that captures a violation of the FD, i.e. $(P(x,y) \wedge P(x,z) \wedge y \neq z)$. ■

This is saying, in particular, that whenever there is a conflict between two tuples, one of them must be deleted, and for every deleted tuple due to a violation, there must be a tuple with the same key value that has not been deleted. Thus, not all mutually conflicting tuples can be deleted.

Now, reconsidering CQA, if we have a query $\mathcal{Q}$, we can obtain the consistent answers $\bar{a}$ as entailments in classical predicate logic:

$$\psi \wedge \forall \bar{x}(Ans^{\mathcal{Q}}(\bar{x})) \equiv \chi(\bar{x})) \ \models \ Ans^{\mathcal{Q}}(\bar{a}), \tag{26}$$

where ψ is the FO sentence that is the conjunction of (12), (14), (15) and (25); and χ is the FO definition of $Ans^{\mathcal{Q}}$ in terms of $P_{\star\star}$.

For example, for $\mathcal{Q}: \ P(x,y)$, we have, instead of (19):

$$\psi \wedge \forall x \forall y(Ans(x,y) \equiv P_{\star\star}(x,y)) \ \models \ Ans(t_1,t_2).$$

From here we obtain, using (14), that (t_1,t_2) is a consistent answer iff $\psi \models P_{\star\star}(t_1,t_2)$ iff $\psi \models (P(t_1,t_2) \wedge \neg P_f(t_1,t_2))$. That is,

$$\{\mathcal{R}(D), \forall xyz(\kappa(x,y,z) \rightarrow (P_f(x,y) \vee P_f(x,z))),$$
$$\forall xy(P_f(x,y) \rightarrow \exists z(\kappa(x,y,z) \wedge \neg P_f(x,z)))\} \models P(t_1,t_2) \wedge \neg P_f(t_1,t_2). \tag{27}$$

This requires $P(t_1,t_2)$ to hold in $\mathcal{R}(D)$, and the negation of $\neg P_f(t_1,t_2)$ to be inconsistent with the theory on the LHS of (27). This happens iff $\forall z \neg\kappa(t_1,t_2,z)$ follows from $\mathcal{R}(D)$. In consequence, (t_1,t_2) is a consistent answer iff $\mathcal{R}(D) \ \models \ P(t_1,t_2) \wedge \forall z \neg \kappa(t_1,t_2,z)$, which is equivalent to

$$D \models P(t_1,t_2) \wedge \neg \exists z(P(t_1,z) \wedge z \neq t_2). \tag{28}$$

The rewriting in (28), already presented in Example 1, is one of those obtained in [4] using a more general rewriting methodology for queries that are quantifier-free conjunctions of database literals and classes of ICs that include FDs. The technique in [4] is not based on explicit specification of repairs. Actually, it relies on an iteration of resolution steps between ICs and intermediate queries, and is not defined for queries or ICs with existential quantifiers. Rewriting (28) is also a particular case of a result in [16, theo. 3.2] on FO rewritability of CQA for conjunctive queries without free variables.[6]

Notice that (26), in spite of being expressed as entailment in FO logic, does not necessarily allow us to obtain a FO rewriting to consistently answering query $\mathcal{Q}(\bar{x})$.

[6] That result can be applied with our query $\mathcal{Q}(x,y) \ : \ P(x,y)$, by transforming it first into $\exists x \exists y(P(x,y) \wedge x = t_1 \wedge y = t_2)$, with generic, symbolic constants t_1, t_2, as above.

A FO rewriting, and the subsequent polynomial-time data complexity, are guaranteed when we obtain a condition of the form $D \models \varphi(\bar{t})$ for consistent answers $\bar{t}$, and φ is a FO formula expressed in terms of the database predicates in D. This is different from we could naively obtain from (26), namely a sentence containing possibly complex and implicit view definitions, like the derived definition of P_f above. A finer analysis from (26) is required in order to obtain a FO rewriting, whenever possible.

The particular case considered in Proposition 4 has all the features of the case of FDs most studied in the literature, namely where there is one FD per database predicate [16, 26, 40]. Under this assumption, if we have a class of FDs involving different predicates, we can treat each of the FDs separately, because there is no interaction between them. So, each predicate P_f can be circumscribed independently from the others, obtaining results similar to those for the particular case.

5 Conclusions

Repair programs for CQA have been well studied in the literature. They specify the database repairs as their stable models. On their basis, and using available implementations for the disjunctive stable model semantics for logic programs, we have the most general mechanism for CQA [14]. As expected, given the nature of CQA, its semantics is non-monotonic, and its logic is non-classical. In this work we have presented the first steps of an ongoing research program that aims to take advantage of specifications of database repairs in classical logic, from which CQA can be done as logical entailment.

That stable models, and in particular database repairs, can be specified in SO logic can be obtained from complexity-theoretic results. The decision problem of stable model checking (SMC) consists in deciding if, for a fixed program, a certain finite input set of atoms is a stable model of the program. The repair checking problem (RC) consists, for a fixed set of ICs IC, if D' if a repair of D wrt to *IC*. Here, D, D' are inputs to the problem. Both SMC and RC are $coNP$-complete (cf. [21] and [16], resp.). Since by Fagin's theorem (cf. [22] and [29, chapter 9]), universal SO logic captures the class $coNP$, there is a a universal SO sentence that specifies the repairs. For the same reason, the stable models of a fixed program can be specified in universal SO classical logic. (Cf. also [20] for applications of such representation results.)

In this work we have shown concrete specifications of repairs in SO classical logic. They have been obtained from the results in [23], that presents a characterization of the stable models as models of a theory in SO predicate logic. However, due to the nature of repair programs, we are able to provide a circumscriptive SO characterization of them. A first and preliminary circumscriptive approach to the specification of database repair was presented in [9].

Furthermore, we have shown, starting from the SO specification of stable models in [23], that, in the case of repair programs wrt functional dependencies, it is possible to obtain a specification in first-order classical logic. The FO theory can be obtained from the circumscriptive theory by newer quantifier elimination techniques that have their origin in the work of Herbrand on decidable classes for the decision problem. In particular, we have shown that it is possible to obtain FO rewritings for CQA of the kind presented in [4].

Many problems are open for ongoing and future research. For example, and most prominently, the natural question is as to whether the combination of a repair program and a query program can be used, through their transformation, to obtain more efficient algorithms that the standard way of evaluating disjunctive logic programs under the stable model semantics. We know that, in the worst cases of CQA, this is not possible, but it should be possible for easier classes of queries and ICs.

More specifically, the following are natural problems to consider: (a) Identification of classes of ICs and queries for which repair programs can be automatically "simplified" into queries of lower complexity. In particular, reobtaining previously identified classes, and identifying new ones. (b) More generally, we would like to obtain new complexity results for CQA. (c) Shed more light on those cases, possibly classes, where CQA can be done in polynomial time, but not via FO rewriting.

Furthermore, the "logic" of CQA is not fully understood yet. We should be able to better understand the logic of CQA through the analysis of repair programs. However, their version in classical logic as presented in this work seems more appropriate for this task. For example, we would like to obtain results about compositionality of CQA, i.e. computing consisting answers to queries on the bases of consistent answers to subqueries or auxiliary views. Techniques of this kind are important for the practice of CQA. We know how to logically manipulate and transform a specification written in classical FO or SO logic, which is not necessarily the case for logic programs. It seems to be easier to (meta)reason about the specification if it is written in classical logical than written as a logic program, which is mainly designed to compute from it.

Also dynamic aspects of CQA have been largely neglected (cf. [33] for some initial results). Computational complexity results and incremental algorithms for CQA are still missing. Results on updates of logic programs and/or theories in classical logic might be used in this direction.

Acknowledgements: Useful comments from anonymous reviewers are much appreciated.

References

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

[2] Ackermann, W. Untersuchungen über das Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 1935, 110:390-413.

[3] Ackermann, W. *Solvable cases of the Decision Problem*. North-Holland Pub. Co., 1954.

[4] Arenas, M., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. *Proc. ACM Symposium on Principles of Database Systems*, ACM Press, 1999, pp. 68-79.

[5] Arenas, M., Bertossi, L. and Chomicki, J. Answer Sets for Consistent Query Answering in Inconsistent Databases. *Theory and Practice of Logic Programming*, 2003, 3(45):393-424.

[6] Barcelo, P. and Bertossi, L. Logic Programs for Querying Inconsistent Databases. *Proc. Practical Aspects of Declarative Languages*, 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. In *Semantics of Databases*, Springer LNCS 2582, 2003, pp. 1-27.

[8] Ben-Eliyahu, R. and Dechter, R. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 1994, 12(1-2):53-87.

[9] Bertossi, L. and Schwind, C. Database Repairs and Analytic Tableaux. *Annals of Mathematics and Artificial Intelligence*, 2004, 40(1-2):5-35.

[10] Bertossi, L. Consistent Query Answering in Databases. In *ACM Sigmod Record*, June 2006, 35(2):68-76.

[11] Bertossi, L. From Database Repair Programs to Consistent Query Answering in Classical Logic. Extended version: http://www.scs.carleton.ca/~bertossi/papers/FrRP2CQAext2.pdf

[12] Bravo, L., Bertossi, L. Semantically Correct Query Answers in the Presence of Null Values. *Proc. EDBT WS on Inconsistency and Incompleteness in Databases*, Springer LNCS 4254, 2006, pp. 336-357.
[13] Caniupan, M., Bertossi, L. Optimizing Repair Programs for Consistent Query Answering. *Proc. International Conference of the Chilean Computer Science Society*, IEEE Computer Society Press, 2005, pp. 3-12.
[14] Caniupan, M., Bertossi, L. The Consistency Extractor System: Querying Inconsistent Databases using Answer Set Programs. *Proc. of the Scalable Uncertainty Management Conference*, Springer LNCS 4772, 2007, pp. 74-88.
[15] Caniupan, M. and Bertossi, L. The Consistency Extractor System: Answer Set Programs for Consistent Query Answering in Databases. Journal submission, 2009.
[16] Chomicki, J. and Marcinkowski, J. Minimal-Change Integrity Maintenance using Tuple Deletions. *Information and Computation*, 2005, 197(1-2):90121.
[17] Chomicki, J. Consistent Query Answering: Five Easy Pieces. *Proc. International Conference on Database Theory*, Springer LNCS 4353, 2007, pp. 1-17.
[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] Doherty, P., Lukaszewicz, W. and Szalas, A. Computing Circumscription Revisited. A Reduction Algorithm. *Journal of Automated Reasoning*, 1997, 18(3):297-336.
[20] Eiter, T. and Gottlob, G. Expressiveness of Stable Model Semantics for Disjunctive Logic Programs with Functions. *Journal of Logic Programming*, 1997, 33(2):167-178.
[21] Eiter, T., Gottlob, G. and Mannila, H. Disjunctive Datalog. *ACM Transactions on Database Systems*, 1997, 22(3):364-418.
[22] Fagin, R. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In *Complexity of Computation*, R. Karp (ed.), SIAM-AMS Proceedings 7, 1974, pp. 43-73.
[23] Ferraris, P., Lee, J. and Lifschitz, V. A New Perspective on Stable Models. In *Proc. International Joint Conference on Artificial Intelligence*, 2007, pp. 372-379.
[24] Ferraris, P., Lee, J. and Lifschitz, V. Stable Models and Circumscription. *Artificial Intelligence* (to appear). http://www.cs.utexas.edu/users/vl/papers/smcirc.ps
[25] Fuxman, A. and Miller, R. Towards Inconsistency Management in Data Integration Systems. *Proc. Workshop on Information Integration on the Web (IIWeb)*, 2003.
[26] Fuxman, A. and Miller, R. First-Order Query Rewriting for Inconsistent Databases. *J. Computer and Systems Sciences*, 2007, 73(4):610-635.
[27] Gelfond, M., Lifschitz, V. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 1991, 9(3/4):365-385.
[28] 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):13891408.
[29] Libkin, L. *Elements of Finite Model Theory.* Springer, 2004.
[30] Lifschitz, V. Computing Circumscription. *Proc. International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1985, pp. 121-127.
[31] Lifschitz, V. Circumscription. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, Vol. 3. Oxford University Press, 1994, pp.297-352.
[32] Lloyd, J.W. *Foundations of Logic Programming*. Springer Verlag, 1987.
[33] Lopatenko, A. and Bertossi, L. Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics. *Proc. International Conference of Database Theory*, Springer LNCS 4353, 2007, pp. 179-193.
[34] Lifschitz, V. and Turner, H. Splitting a Logic Program. *Proc. International Conference on Logic Programming*, MIT Press, 1994, pp. 23-37.
[35] Lifschitz, V. Twelve Definitions of Stable Model. *Proceedings International Conference on Logic Programming*. Springer LNCS 5366, 2008, pp. 37-51.
[36] McCarthy, J. Circumscription - A Form of Non-Monotonic Reasoning. *Artificial Intelligence*, 1980, 13(1-2):27-39.

[37] Przymusinski, T. On the Declarative Semantics of Deductive Databases and Logic Programs. In *Foundations of Deductive Databases and Logic Programming*, J. Minker (ed.), Morgan Kaufmann Publishers Inc., 1988, pp. 193-216.
[38] Przymusinski, T. Stable Semantics for Disjunctive Programs. *New Generation Computing*, 1991, 9(3/4):401-424.
[39] Reiter, R. Towards a Logical Reconstruction of Relational Database Theory. In *On Conceptual Modelling*, M.L. Brodie, J. Mylopoulos and J.W. Schmidt (eds.), Springer, 1984, pp. 191-233.
[40] Wijsen, J. On the Consistent Rewriting of Conjunctive Queries Under Primary Key Constraints. *Proc. International Symposium on Database Programming Languages*, Springer LNCS 4797, 2007, pp. 112-126.

Appendix: Basic Notions

Disjunctive logic programs

We consider disjunctive Datalog programs Π [21] with a finite number of rules of the form

$$A_1 \vee \ldots A_n \leftarrow P_1, \ldots, P_m,\ not\ N_1, \ldots,\ not\ N_k,$$

with $0 \leq n, m, k$, and the A_i, P_j, N_s are positive FO atoms. The terms in these atoms are constants or variables. The variables in the A_i, N_s appear all among those in the P_j. The constants in the program Π form the (finite) Herbrand universe U of the program. The ground version of program Π, $gr(\Pi)$, is obtained by instantiating the variables in Π in all possible combinations using values from U. The Herbrand base HB of Π consists of all the possible atomic sentences obtained by instantiating the predicates in Π in U. A subset M of HB is a model of Π it is satisfies $gr(\Pi)$, that is: For every ground rule $A_1 \vee \ldots A_n \leftarrow P_1, \ldots, P_m,\ not\ N_1, \ldots,\ not\ N_k$ of $gr(\Pi)$, if $\{P_1, \ldots, P_m\} \subseteq M$ and $\{N_1, \ldots, N_k\} \cap M = \emptyset$, then $\{A_1, \ldots, A_n\} \cap M \neq \emptyset$. M is a minimal model of Π if it is a model of Π, and Π has no model that is properly contained in M. $MM(\Pi)$ denotes the class of minimal models of Π.

Now, take $S \subseteq HB(\Pi)$, and transform $gr(\Pi)$ into a new, positive program $gr(\Pi)\!\downarrow$ (i.e. without not), as follows: Delete every rule $A_1 \vee \ldots A_n \leftarrow P_1, \ldots, P_m,\ not\ N_1, \ldots,\ not\ N_k$ for which $\{N_1, \ldots, N_k\} \cap S \neq \emptyset$. Next, transform each remaining rule $A_1 \vee \ldots A_n \leftarrow P_1, \ldots, P_m,\ not\ N_1, \ldots,\ not\ N_k$ into $A_1 \vee \ldots A_n \leftarrow P_1, \ldots, P_m$. Now, S is a stable model of Π if $S \in MM(gr(\Pi)\!\downarrow)$.

A disjunctive Datalog program is stratified if its set of predicates $\mathcal{P}$ can be partitioned into a sequence $\mathcal{P}_1, \ldots, \mathcal{P}_k$ in such a way that, for every $P \in \mathcal{P}$:

1. If $P \in \mathcal{P}_i$ and predicate Q appears in a head of a rule with P, then $Q \in \mathcal{P}_i$.
2. If $P \in \mathcal{P}_i$ and Q appears positively in the body of a rule that has P in the head, then $Q \in \mathcal{P}_j$, with $j \leq i$.
3. If $P \in \mathcal{P}_i$ and Q appears negatively in the body of a rule that has P in the head, then $Q \in \mathcal{P}_j$, with $j < i$.

If a program is stratified, then its stable models can be computed bottom-up by propagating data upwards from the underlying extensional database, and making sure to minimize the selection of true atoms from the disjunctive heads. Since the latter introduce a form of non-determinism, a program may have several stable models.

Circumscription

Let $\bar{P}, \bar{Q}$ be disjoint tuples of FO predicates. The circumscription of $\bar{P}$ wrt $\preceq$ in the FO sentence $\Sigma(\bar{P}, \bar{Q})$ with variable $\bar{Q}$ can be expressed by means of the SO sentence [30] $Circ(\Sigma(\bar{P}, \bar{Q}); \bar{P}; \bar{Q})$: $\Sigma(\bar{P}, \bar{Q}) \wedge \neg \exists \bar{X} \bar{Y} (\Sigma(\bar{X}, \bar{Y}) \wedge \bar{X} \preceq \bar{P} \wedge \bar{X} \neq \bar{P})$, where $\bar{X}, \bar{Y}$ are tuples of SO variables that replace $\bar{P}$, resp. $\bar{Q}$ in $\Sigma(\bar{P}, \bar{Q})$, producing $\Sigma(\bar{X}, \bar{Y})$.

Here, $\preceq$ stands for a FO definable pre-order relation (reflexive and transitive) between tuples of predicate extensions. All the other predicates in $\Sigma(\bar{P}, \bar{Q})$ are left untouched and they are kept fixed during the minimization of those in $\bar{P}$, while those in $\bar{Q}$ become flexible. By appropriately choosing the relation $\preceq$, different forms of circumscription can be captured. Prioritized circumscription is based on a prioritized partial order relation between tuples $\bar{S} = (S_1, \ldots, S_m)$, and $\bar{T} = (T_1, \ldots, T_m)$ of similar predicates (i.e. same length and corresponding arities). It can be defined by $\bar{S} \preceq^{pri} \bar{T} \ \equiv \ \bigwedge_{i=1}^{m} (\bigwedge_{j=1}^{i-1} S_i = T_i \rightarrow S_i \leq T_i)$. Here, $\leq$ stands for the subset relation. The parallel circumscription of the predicates in $\bar{P}$ can be obtained by means of the relation: $\bar{S} \preceq^{par} \bar{T} \ \equiv \ \bigwedge_{i=1}^{m} S_i \leq T_i$.