Second-Order Specifications and Quantifier Elimination for Consistent Query Answering in Databases
(Abstract)

Leopoldo Bertossi

Universidad Adolfo Ibáñez
Faculty of Engineering and Sciences
and
Millennium Institute for Foundational Research on Data (IMFD)
Santiago, Chile
leopoldo.bertossi@uai.cl

Abstract. Consistent answers to a query from a possibly inconsistent database are answers that are simultaneously retrieved from every possible repair of the database. Repairs are consistent instances that minimally differ from the original inconsistent instance. It has been shown before that database repairs can be specified as the stable models of a disjunctive logic program. We show how to use the repair programs to transform the problem of consistent query answering into a problem of reasoning w.r.t. a theory written in second-order predicate logic. We show how a first-order theory can be obtained instead by applying second-order quantifier elimination techniques.

1 Introduction

Integrity constraints (ICs) on databases are expected to be satisfied by the instances of the given schema $\mathcal{S}$. If an instance does not satisfy the ICs, it is said to be *inconsistent*, and becomes only partially semantically correct. *Consistent query answering* (CQA) attempts to characterize and compute answers to a query that are consistent with respect to (w.r.t.) a given set of ICs [5, 8, 15, 11]. Informally, a tuple of constants $\bar{t}$ is a consistent answer from an instance D to a query $\mathcal{Q}(\bar{x})$ w.r.t. a set of ICs IC if $\bar{t}$ can be obtained as a usual answer to $\mathcal{Q}$ from every *repair* of D. Here, a *repair* is a consistent instance for the schema $\mathcal{S}$ that differs from D by a minimal set of database atoms under set inclusion [5].

It has been shown [6, 13] that repairs can be specified as the *stable models of a disjunctive logic program* [22] Π, by a so-called *repair program*. In this way, CQA becomes a problem of reasoning with program Π. Logic programs with stable model semantics are also called *answer-set programs* [22], and their stable models are also called answer sets. Answer-set programming has become a powerful paradigm and tool for the specification and solution of hard combinatorial problems [12].

Ideally, consistent answers to a query $\mathcal{Q}$ from a database instance D should be obtained by posing a new query $\mathcal{Q}'$ to D, as an ordinary query that is, hopefully, easy to evaluate against D. This is the case, for example, when $\mathcal{Q}'$ is a query expressed in the first-order (FO) language $L(\mathcal{S})$. Some classes of queries and ICs with this property have been already identified [5, 14, 21]; and many more by Wijsen in a series of papers on conjunctive queries and *key constraints* [1]. C.f. [36] and [37] for excellent surveys, and [23] for more recent results and references.

The main result for CQA for conjunctive queries (CQs) under key constraints (KCs), tells us that one can syntactically classify and decide CQs in terms of their data complexity for CQA.[1] A trichotomy appears: a CQ can be FO-rewritable, or in *PTIME* (L-complete), or *coNP*-complete. There are queries for these three classes. For the first class, the rewriting can be computed, in which case, it is possible to compute the consistent answers in polynomial time. It is worth emphasizing that there are CQs for which CQA can be done in polynomial time, but provably not via FO-rewriting [35, 34]. This opens the question about the right logical language for a rewriting, if any.

At the other extreme, repair programs provide a general mechanism for computing consistent answers. 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 [16, 14]. Apart from providing the right expressive power and complexity for dealing with repairs and CQA, the semantics of answer-set programming is a non-monotonic, non-classical logical semantics, which is particularly suitable for applications in databases, through the implicit use of the *closed-world assumption* [32], and the minimality of models under set inclusion. This last feature is useful in relation to the minimality of database repairs.

In those cases where a FO rewriting for CQA is possible, one can transform the problem of CQA into one of reasoning in classical predicate logic, because the original database can be "logically reconstructed" as a FO theory [32]. 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. This theory can be written in second-order or first-order predicate logic. We start by trying to achieve the former, by providing specifications of database repairs in second-order (SO) predicate logic. They are obtained by applying recent results on the specification in SO logic of the stable models of a logic program [19, 20] -in our case, a repair program- and older results on their characterization as the models of a circumscriptive theory [29] for the case of disjunctive stratified programs [30, 31]. This circumscription can be specified in SO predicate logic [25, 33].

In order to achieve a FO specification, for some cases related to queries and KCs, we apply techniques for SO quantifier elimination that have been introduced in [17]. In this way it is possible to obtain a FO specification of the database repairs. This transforms 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 under a KC. We illustrate the SO quantifier elimination technique. Generalizing the methodology to more general cases is left for future investigation. C.f. Section 5), where we also discuss the possibility

[1] As usual in databases, all the complexity results in this paper are about *data complexity*, i.e. in terms of the size of the database instance.

of obtaining rewritings in fixed-point logic, when it is provably the case that no FO rewriting exists. *This paper is an excerpt from* [9]. *In* [10] *one can find an extended and updated version of both the latter and this paper, containing all the details and much more.*

2 Database Repairs and Repair Programs

Consider a database instance D and a set of *integrity constraints* (ICs), that is a set Σ of sentences in the first-order language of predicate logic associated to the database schema. The database may not satisfy Σ in which case we say that D is *inconsistent*. The database can be *repaired* by inserting or deleting full tuples into/from D, in such a way that the resulting instance becomes consistent. A (minimal) *repair* of D is an instance D' that satisfies Σ and minimally differs from D under set inclusion, i.e. $D \Delta D'$, the symmetric set difference, is minimal under set inclusion [5, 11]. For *monotone ICs* (they are never violated by tuple deletions), like the ones we will consider below, the repairs are always maximal-subsets (subinstances) of D. The repairs of an inconsistent database can be specified by means of answer-set programs (c.f. [11] for details and references). Those are the *repair programs*.

Repair programs use *annotation constants* in an extra argument for each of the database predicates. More precisely, for each n-ary $P \in \mathcal{S}$, we make a copy $P_{\_}$, which is $(n+1)$-ary [13]. Here, we need only the following annotations, with its intended semantics: (a) $\mathbf{f}$ in atoms $P_{\_}(\bar{a}, \mathbf{f})$, meaning "made false (deleted)", (b) $\mathbf{t}^{\star\star}$ in atoms $P_{\_}(\bar{a}, \mathbf{t}^{\star\star})$, meaning "true in repair".

Example 1. The relational schema $\mathcal{S}$ contains predicate $P(X, Y)$, and the functional dependency (FD) $X \rightarrow Y$, actually a KC, stating that the first attribute functionally determines the second. It can be expressed as the first-order (FO) sentence

$$FD\colon\ \forall x \forall y \forall z (P(x, y) \wedge P(x, z) \rightarrow y = z).$$

The database instance $D = \{P(a, b), P(a, c), P(d, e)\}$ is inconsistent since the first two tuples jointly violate the FD. We have two repairs: $D_1 = \{P(a, b), P(d, e)\}$ and $D_2 = \{P(a, c), P(d, e)\}$. The query $\mathcal{Q}_1(y)\colon \exists x P(x, y)$ has the consistent answer (e), whereas the query $\mathcal{Q}_2(x)\colon \exists y P(x, y)$ has $(a), (d)$ as consistent answers. They are standard answers *from both* repairs. These repairs can be specified as the stable models of the following repair program $\Pi(D, FD)$:

1. Original database facts: $P(a, b), P(a, c), P(d, e)$.
2. The repair rule: $P_{\_}(x, y, \mathbf{f}) \vee P_{\_}(x, z, \mathbf{f}) \leftarrow P(x, y), P(x, z), y \neq z$.

It specifies that whenever the FD is violated, as captured by the rule body (the RHS), then one (and only one if possible) of the two tuples involved in the violation has to be made false (deleted), as captured by the disjunctive rule head (LHS).

3. Annotations constant $\mathbf{t}^{\star\star}$ is used to read off the atoms in a repair, saying that whichever atom was in the original instance and not deleted stays in the repair:

$$P_{\_}(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{x}),\ \mathit{not}\ P_{\_}(\bar{x}, \mathbf{f}).$$

For simplicity, and from now on, we use new predicates $P_f(\_, \_)$ for $P_{\_}(\_, \_, \mathbf{f})$, $P_{\star\star}(\_, \_)$ for $P_{\_}(\_, \_, \mathbf{t}^{\star\star})$. The repairs are in one-to-one correspondence with the restric-

tion of the stable models to the predicates of the form $P_{\star\star}$ [13]. In this example, they are: $D_1 = \{P_{\star\star}(a,b), P_{\star\star}(d,e)\}$ and $D_2 = \{P_{\star\star}(a,c), P_{\star\star}(d,e)\}$. ■

In order to obtain the consistent answers to a FO query $\mathcal{Q}$, a query program $\Pi^{\mathcal{Q}}$, containing a query-answer predicate $Ans^{\mathcal{Q}}$, is combined with the repair program $\Pi(D, FD)$. Next, as is common with ASPs, we can use the *cautious* entailment semantics from ASP, denoted $\models_{cs}$, which means that the right-hand side is true in all the stable models of the program on the left-hand side.

The extension of the answer predicate $Ans^{\mathcal{Q}}$ in the intersection of all stable models of $\Pi := \Pi(D, FD) \cup \Pi^{\mathcal{Q}}$ contains exactly the consistent answers. That is, $\bar{a}$ is a consistent answer to $\mathcal{Q}$, denoted, $D \models_c \mathcal{Q}(\bar{a})$, iff $\Pi(D, FD) \cup \Pi^{\mathcal{Q}} \models_{cs} Ans^{\mathcal{Q}}(\bar{a})$. In general, $\Pi^{\mathcal{Q}}$ will be a (stratified) non-recursive and normal Datalog^{not} query $\Pi^{\mathcal{Q}}$ with answer predicate $Ans^{\mathcal{Q}}(\bar{x})$ appearing only in rule heads [1, 27].

Example 2. (ex. 1 cont.) A possible query is $\mathcal{Q}(x,y) : \ P(x,y)$, which can be represented by the simple query program $\Pi^{\mathcal{Q}}$: $Ans(x,y) \leftarrow P_{\star\star}(x,y)$. This program is combined with 1.-3. above, and the consistent answers to $\mathcal{Q}$ are those tuples $\bar{a}$, such that $\Pi^{\mathcal{Q}} \cup \Pi(D, FD) \models_{cs} Ans(\bar{a})$, obtaining the only consistent answer is (d,e). ■

3 Second-Order Specification of Repairs

In [19, 20], the stable model semantics of logic programs introduced in [22] is reobtained via an explicit specification in classical SO predicate logic that is based on circumscription. First, the program Π is transformed into (or seen as) a FO sentence $\psi(\Pi)$. Next, the latter is transformed into a SO sentence $\Phi(\Pi)$. Here, $\psi(\Pi)$ 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 predicates $P_1, ..., P_n$ in ψ that are going to be circumscribed,[2] 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 $Q \in \{\forall, \exists\}$. Notice that we assume there is no explicit logical negation in formulas. Instead, a formula of the form $\neg\chi$ is assumed to be represented as $(\chi \rightarrow \bot)$, with $\bot$ standing for an always false propositional formula.

The Herbrand models of the SO sentence $\Phi(\Pi)$ associated to $\psi(\Pi)$ correspond to the stable models of the original program Π [19]. We can see that $\Phi(\Pi)$ is similar to a parallel circumscription of the predicates in program Π w.r.t. the FO sentence $\psi(\Pi)$

[2] In circumscription, some predicate may be minimized, others may stay flexible (or variable) to accommodate to the minimization of others, and some may stay fixed [25].

associated to Π [29, 26]. In principle, the transformation rule (e) above could make formula $\Phi(\Pi)$ differ from a circumscription.

Now, let D be a relational database, Π^r the repair program without the database facts. From now on $\Pi = D \cup \Pi^r \cup \Pi^{\mathcal{Q}}$. Π^r depends only on the integrity constraints, and includes definitions for the annotation predicates. The only predicates shared by Π^r and $\Pi^{\mathcal{Q}}$ are of the form $P_{\star\star}$, which appear only in the rule bodies of $\Pi^{\mathcal{Q}}$. These predicates produce a *splitting* of the combined program [28], which allows us to analyze separately Π^r and $\Pi^{\mathcal{Q}}$. The latter can be translated into classical logic by predicate completion, or a prioritized circumscription [30]. If the query is FO, we can use query itself.

Example 3. (ex. 2 cont.) Leaving aside many details that can be found in [10], we obtain the following SO formula $\Phi(\Pi)$ that captures the stable models of the original program:

$$\forall xy(P(x,y) \equiv (x = a \wedge y = b) \vee (x = a \wedge y = c) \vee (x = d \wedge y = e)) \ \wedge \quad (1)$$

$$\forall xy(P_{\star\star}(x,y) \equiv Ans(x,y)) \ \wedge \quad (2)$$

$$\forall xy((P(x,y) \wedge \neg P_f(x,y)) \equiv P_{\star\star}(x,y)) \ \wedge \quad (3)$$

$$\forall xyz(P(x,y) \wedge P(x,z) \wedge y \neq z \rightarrow (P_f(x,y) \vee P_f(x,z))) \ \wedge \quad (4)$$

$$\neg\exists U_f((U_f < P_f) \wedge \forall xyz(P(x,y) \wedge P(x,z) \wedge y \neq z \rightarrow (U_f(x,y) \vee U_f(x,z))). \quad (5)$$

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 minimizations of the predicates $P, P_{\star\star}$ and Ans are expressed as their predicate completion. Predicate P_f is minimized via (5).

We obtain the SO sentence for program Π as a parallel circumscription of the predicates in the repair program seen as a FO sentence. The circumscription actually becomes a *prioritized circumscription* [25] given the stratified nature of the repair program: first the database predicate is minimized, next P_f, next $P_{\star\star}$, and finally Ans. ■

Generalizations of the result in the previous example can be found in [10]. In the following we concentrate on the problem of possibly turning this SO reasoning problem into one at the FO level.

4 From Second-Order to First-Order CQA

In this section we discuss the possibility of obtaining a FO rewriting of the original query as posed to the repair program. We do this through the analysis of the SO sentence obtained in Example 3, concentrating on the SO sentence (5). *In the rest of this section many details are missing. They can be all found in* [10]*, the extended version of this work.*

Sentence (5) can be expressed as

$$\neg\exists U_f((U_f < P_f) \wedge \forall xyz(\kappa(x,y,z) \rightarrow (U_f(x,y) \vee U_f(x,z))), \quad (6)$$

where $\kappa(x,y,z)$ is the formula $P(x,y) \wedge P(x,z) \wedge y \neq z$. For simplicity, we use U instead of U_f. We will apply to (6) the SO quantifier elimination techniques in [17]. The negation of (6) turns out to be -after several steps [10]- logically equivalent to:

$$\exists st\exists U(\ \forall xyz(\neg\kappa(x,y,z) \vee U(x,y) \vee U(x,z)) \wedge \quad (7)$$
$$\forall uv(\neg U(u,v) \vee P_f(u,v)) \wedge (P_f(s,t) \wedge \neg U(s,t))).$$

The first conjunct in (7), with $w = \vee(y, z)$ standing for $(w = y \vee w = z)$, can be equivalently written as

$$\exists f \forall r(\forall x_1 y_1 z_1(\neg\kappa(x_1, y_1, z_1) \vee f(x_1, y_1, z_1) = \vee(y_1, z_1)) \wedge \forall xyz(\neg\kappa(x, y, z) \vee r \neq f(x, y, z) \vee U(x, r))),$$

where $\exists f$ is a quantification over functions. Formula (7) becomes:

$$\exists st \exists f \exists U \forall x \forall r((\forall x_1 y_1 z_1(\neg\kappa(x_1, y_1, z_1) \vee f(x_1, y_1, z_1) = \vee(y_1, z_1)) \wedge \forall yz(\neg\kappa(x, y, z) \vee r \neq f(x, y, z) \vee U(x, r))) \wedge \forall uv(\neg U(u, v) \vee P_f(u, v)) \wedge (P_f(s, t) \wedge \neg U(s, t))).$$

We are ready to apply Ackermann's Lemma [2, 3], with the last formula written as:

$$\exists st \exists f \exists U \forall x \forall r((A(x, r) \vee U(x, r)) \wedge B(\frac{U}{\neg U}), \tag{8}$$

where $B(\frac{U}{\neg U})$ is formula B with predicate U replaced by $\neg U$; and formulas A, B are: $A(x, r)\colon\ \forall yz(\forall yz(\neg\kappa(x, y, z) \vee r \neq f(x, y, z))$; and $B(U)\colon\ \forall x_1 y_1 z_1(\neg\kappa(x_1, y_1, z_1) \vee f(x_1, y_1, z_1) = \vee(y_1, z_1)) \ \wedge\ \forall uv(U(u, v) \vee P_f(u, v)) \wedge (P_f(s, t) \wedge U(s, t)))$.

Formula B is positive in U, then the whole subformula in (8) starting with $\exists U$ can be equivalently replaced by $B(\frac{U}{A(x,r)})$ [17, lemma 1], getting rid of the SO variable U, and obtaining:

$$\exists st \exists f \forall xyz((\neg\kappa(x, y, z) \vee f(x, y, z) = \vee(y, z)) \wedge (\neg\kappa(x, y, z) \vee P_f(u, f(x, y, z))) \wedge (P_f(s, t) \wedge (x \neq s \vee \neg\kappa(x, y, z) \vee t \neq f(x, y, z)))).$$

Unskolemizing, getting rid of function variable f, we obtain

$$\exists st \forall xyz \exists w((\neg\kappa(x, y, z) \vee w = \vee(y, z)) \wedge (\neg\kappa(x, y, z) \vee P_f(u, w)) \wedge (P_f(s, t) \wedge (x \neq s \vee \neg\kappa(x, y, z) \vee t \neq w))),$$

which is equivalent to the negation of (6). Negating again, we obtain a formula equivalent to (6):

$$\forall st(P_f(s, t) \rightarrow \exists xyz(\kappa(x, y, z) \wedge \forall w[(w \neq y \wedge w \neq z) \vee \neg P_f(x, w) \vee (x = s \wedge t = w)]).$$

The formula in the square bracket inside 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:

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

Summing up, the SO sentence for the repair program $\Pi(D, IC)$ is logically equivalent to a FO sentence, ψ, that is the conjunction of (1), (3), (4), and

$$\forall st(P_f(s, t) \rightarrow \exists z(\kappa(s, t, z) \wedge \neg P_f(s, z))), \tag{9}$$

which says, 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.

Coming back to CQA, for consistent answers $\bar{t}$, we now have classical FO entailment:

$$\psi \wedge \forall \bar{x}(Ans^{\mathcal{Q}}(\bar{x}) \equiv \chi(\bar{x})) \ \models\ Ans^{\mathcal{Q}}(\bar{t}), \tag{10}$$

where χ is the FO definition of $Ans^{\mathcal{Q}}$ in terms of the $P_{\star\star}$ predicate. This is not FO query rewriting in the sense of obtaining a FO query to be posed to the original database.

However, and for example, it is not difficult to show [10] that for the query $\mathcal{Q}: P(x, y)$, and any consistent answer $\langle t_1, t_2 \rangle$, this is equivalent to having:

$$D \models P(t_1, t_2) \wedge \neg \exists z (P(t_1, z) \wedge z \neq t_2). \tag{11}$$

The query rewriting on the RHS of in (11) is one of those obtained in [5] using a completely different and more general resolution-based rewriting methodology.

5 Towards Fixed-Point Logic

As described in Section 1, there are syntactic classes of CQs for which consistent query answering can be done in polynomial time in data complexity. For one class, this can be done via FO query rewriting. For a different class, its queries provably do not admit a first-order rewriting. Even more, one can decide if a CQ falls in this case or not [36, 37].

For example, the Boolean conjunctive query $\mathcal{Q}: \exists x \exists y (R(x, y) \wedge S(y, x))$, with the first attributes of R and S as keys for them, is a query in the second class in that it can be consistently answered in polynomial time, but no FO rewriting for it exists. Results of this kind are established in [35, 34] by means of the notions of *Hanf-locality* and *Ehrenfeucht-Fraïssé games* for FO-logic [24].

This opens the ground for investigating two problems:

1. Apply the second-order quantifier elimination technique in [17], that we applied in this work, with the purpose of recovering the FO rewritings for the whole class of queries that admit FO consistent rewritings (as determined by Wijsen [35]).
2. Identify and obtain logical languages that can be used for rewriting the queries in the second class, in such a way that query answering for the rewritten query can be done in polynomial time.

For the second problem, it would be interesting to see if second-order quantifier elimination could be applied to second-order specification of Section 3, in such a way that the resulting query is expressed, not in FO logic, but in fixed-point logic, which would lead to a polynomial-time answer [24]. Actually, in [18], the authors have been able to eliminate second-order quantifiers, obtaining fixed-point formulas. It is worth investigating if this is a way to obtain polynomial-time, logical, but non-FO, rewritings for CQA. This undertaking is not a priori impossible. The existence of non-FO rewritable but PTIME-complete queries (in data) already identified [35, 23] is in principle compatible with the PTIME-completeness of fixed-point logic (in data) [16].

Acknowledgements: Useful comments from anonymous reviewers for a previous and the submitted version of this paper are much appreciated. Leopoldo Bertossi has been partially funded by the ANID - Millennium Science Initiative Program - Code ICN17-002.

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] Alviano, M., Morak, M. and Pieris, A. Stable Model Semantics for Tuple-Generating Dependencies Revisited. Proc. PODS, 2017, pp. 377-388.
[5] 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.
[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] Bertossi, L. and Schwind, C. Database Repairs and Analytic Tableaux. *Annals of Mathematics and Artificial Intelligence*, 2004, 40(1-2):5-35.
[8] Bertossi, L. Consistent Query Answering in Databases. In *ACM Sigmod Record*, June 2006, 35(2):68-76.
[9] Bertossi, L. From Database Repair Programs to Consistent Query Answering in Classical Logic (extended abstract). Proc. Alberto Mendelzon International Workshop on Foundations of Data Managemente (AMW), 2009. CEUR Workshop Proceedings, Vol. 450, 2009.
[10] Bertossi, L. Second-Order Specifications and Quantifier Elimination for Consistent Query Answering in Databases. Posted as Corr arXiv Paper 2108.08423, 2021.
[11] Bertossi, L. *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2011.
[12] Brewka, G., Eiter, T. and Truszczynski, M. Answer Set Programming at a Glance. *Communications of the ACM*, 2011, 54(12):92-103.
[13] Caniupan-Marileo, M. and Bertossi, L. The Consistency Extractor System: Answer Set Programs for Consistent Query Answering in Databases. *Data and Knowledge Engineering*, 2010, 69(6):545-572.
[14] Chomicki, J. and Marcinkowski, J. Minimal-Change Integrity Maintenance using Tuple Deletions. *Information and Computation*, 2005, 197(1-2):90-121.
[15] Chomicki, J. Consistent Query Answering: Five Easy Pieces. *Proc. International Conference on Database Theory*, Springer LNCS 4353, 2007, pp. 1-17.
[16] Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 2001, 33(3):374-425.
[17] Doherty, P., Lukaszewicz, W. and Szalas, A. Computing Circumscription Revisited. A Reduction Algorithm. *Journal of Automated Reasoning*, 1997, 18(3):297-336.
[18] Doherty, P., Lukaszewicz, W. and Szalas, A. A Reduction Result for Circumscribed Semi-Horn Formulas. *Fundamenta Informaticae*, 1996, 28(3-4):261-271.
[19] 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.
[20] Ferraris, P., Lee, J. and Lifschitz, V. Stable Models and Circumscription. *Artificial Intelligence*, 2011, 175(1):236-263.
[21] Fuxman, A. and Miller, R. First-Order Query Rewriting for Inconsistent Databases. *J. Computer and Systems Sciences*, 2007, 73(4):610-635.
[22] Gelfond, M., Lifschitz, V. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 1991, 9(3/4):365-385.
[23] Koutris, P. and Wijsen, J. First-Order Rewritability in Consistent Query Answering with Respect to Multiple Keys. Proc. PODS 2020, pp. 113-129.
[24] Libkin, L. *Elements of Finite Model Theory.* Springer, 2004.
[25] Lifschitz, V. Computing Circumscription. *Proc. International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1985, pp. 121-127.
[26] Lifschitz, V. Circumscription. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, Vol. 3. Oxford University Press, 1994, pp.297-352.
[27] Lloyd, J.W. *Foundations of Logic Programming*. Springer Verlag, 1987.

[28] Lifschitz, V. and Turner, H. Splitting a Logic Program. *Proc. International Conference on Logic Programming*, MIT Press, 1994, pp. 23-37.

[29] McCarthy, J. Circumscription - A Form of Non-Monotonic Reasoning. *Artificial Intelligence*, 1980, 13(1-2):27-39.

[30] 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.

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

[32] 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.

[33] Van Hermelen, F., Lifschitz, V. and Porter, B. (eds.) *Handbook of Knowledge Representation*. Elsevier, 2008.

[34] Wijsen, J. A Remark on the Complexity of Consistent Conjunctive Query Answering under Primary Key Violations. *Information Processing Letters*, 2010, 110:950-955.

[35] Wijsen, J. On the Consistent Rewriting Of Conjunctive Queries under Primary Key Constraints. *Information Systems*, 2009, 34:578-601.

[36] Wijsen, J. A Survey of the Data Complexity of Consistent Query Answering under Key Constraints. Proc. FoIKS 2014, LNCS 8367, pp. 62-78.

[37] Wijsen, J. Foundations of Query Answering on Inconsistent Databases. *SIGMOD Record*, 2019, 48(3):6-16.