

Probabilistic Answers over Inconsistent Knowledge Bases

(DISCUSSION PAPER)

Marco Calautti, Nicola Fiorentino, Sergio Greco, Cristian Molinaro, Irina Trubitsyna

DIMES, University of Calabria

{calautti, fiorentino, greco, cmolinaro, trubitsyna}@dimes.unical.it

Abstract. Consistent query answering is a generally accepted approach for querying inconsistent knowledge bases. A consistent answer to a query is a tuple entailed by every repair, where a repair is a consistent database that “minimally” differs from the original (possibly inconsistent) one. This is a somewhat coarse-grained classification of tuples into consistent and non-consistent does not provide much information about the non-consistent tuples (e.g., a tuple entailed by 99 out of 100 repairs might be considered “almost consistent”).

To overcome this limitation, we propose a probabilistic approach to querying inconsistent knowledge bases, which provides more informative query answers by associating a degree of consistency with each query answer by associating a probability to each repair, depending on the changes needed to obtain it.

1 Introduction

There has been a great deal of work on extracting reliable information from inconsistent data, in both the AI and database communities. To deal with this problem, many semantics of query answering have been proposed. Most of them are based on the *consistent query answering* framework [2]. The idea is that a tuple should be considered a consistent answer to a query posed over an inconsistent knowledge base if the tuple is a query answer in every *repair*, where a repair is a consistent database that “minimally” differs from the original one. Different variants of the consistent query answering problem have been investigated depending on the minimality criterion—e.g., see [5]—or the employed repair primitive to restore consistency—e.g., see [3].

Regardless of the specific minimality criterion and repair primitive, all the resulting frameworks share the same drawback, which is a dichotomic classification of tuples in either consistent or non-consistent ones. This can provide very little information to users in many cases, as illustrated in the following example.

Example 1. Consider the knowledge base (D, Σ) , where D contains the following facts:

employee		
bob	cs	nyc
mike	cs	nyc
alice	cs	paris

Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0). This volume is published and copyrighted by its editors. SEBD 2020, June 21-24, 2020, Villasimius, Italy.

and Σ is an ontology consisting of the following equality-generating dependency σ :

$$\text{employee}(E_1, D, C_1), \text{employee}(E_2, D, C_2) \rightarrow C_1 = C_2.$$

As an example, the fact $\text{employee}(\text{bob}, \text{cs}, \text{nyc})$ states that bob is an employee working in the cs department located in nyc. The dependency σ says that every department must be located in a single city. Clearly, the first two facts are conflicting with the last one. There are two repairs for this inconsistent knowledge base, which are as follows (more details on the employed notion of repair will be provided in the following):

$$R_1 = \begin{array}{|c|c|c|} \hline \text{bob} & \text{cs} & \text{nyc} \\ \hline \text{mike} & \text{cs} & \text{nyc} \\ \hline \text{alice} & \text{cs} & \text{nyc} \\ \hline \end{array} \quad R_2 = \begin{array}{|c|c|c|} \hline \text{bob} & \text{cs} & \text{paris} \\ \hline \text{mike} & \text{cs} & \text{paris} \\ \hline \text{alice} & \text{cs} & \text{paris} \\ \hline \end{array}$$

Repair R_1 is obtained from the original database by replacing paris with nyc, while R_2 is obtained by replacing nyc with paris. The query asking for the city of the cs department has no consistent answers. Thus, a user trying to extract this information from the knowledge base is left with no answer altogether, all she/he gets is the empty set. \square

To overcome the limitation illustrated in the example above, we propose a probabilistic approach to querying inconsistent knowledge bases whose aim is to provide more informative query answers than the classical consistent query answering framework. The main idea is to give a “degree of consistency” to every repair, which is then taken into account to assign a degree of consistency to query answers.

Example 2. Consider again the scenario of Example 1. The degree of consistency of R_1 is $2/3$, as nyc is supported by two facts out of three in the original knowledge base, while the degree of consistency of R_2 is $1/3$, as paris is supported by only one fact out of three.

Consider again the query asking for the city of the cs department. Rather than providing no information (like classical consistent query answers do), our query answers are nyc with confidence $2/3$ (as this is entailed by R_1), and paris with confidence $1/3$ (as this is entailed by R_2), which is much more informative than returning nothing. \square

The previous examples illustrate the limitation of providing only certain information and how this might be overcome by assigning a degree of consistency to query answers (notice that consistent query answers are still provided – they have degree 1).

In this paper, we consider knowledge bases where dependencies are expressed by means of *equality-generating dependencies* (EGDs). Equality-generating dependencies are one of the two major types of data dependencies—the other major type consists of *tuple-generating dependencies* (TGDs)—and can model several kinds of constraints arising in practice, such as functional dependencies and thus key dependencies.

In the presence of EGDs, the two main approaches to restore consistency are to perform fact deletions or fact updates. A drawback of the former is that entire facts are deleted to resolve inconsistency, even if they may still contain “reliable” information. Thus, in this paper we adopt a notion of repair based on fact updates (we will provide a precise definition in the following). Nonetheless, the ideas developed in this paper can be used also when a notion of repair based on fact deletions is adopted.

As we show, providing more informative query answers comes at a price: it is #P-hard. In light of this result, we discuss further developments providing polynomial time algorithms to compute approximate query answers.

Related work. Most inconsistency-tolerant approaches in the literature adopt the classical notion of repair (via fact deletions/insertions) and consistent query answer [16,19,6,18,17], while in this paper we propose a probabilistic generalization where repairs use fact updates, akin to [7,21,11,15], and query answers are probabilistic. Some works on probabilistic query answering on inconsistent knowledge bases exist [1,13,10], but [1] and [13] only consider restricted classes of dependencies, while [10] deals with the classical notion of repair.

2 Preliminaries

Basics. We assume the existence of the following pairwise disjoint (countably infinite) sets: a set *Const* of *constants*, a set *Var* of *variables*, and a set *Null* of *labeled nulls*. Nulls are denoted by the symbol \perp subscripted with natural numbers. A *term* is a constant, variable, or null. We also assume a set of *predicates*, disjoint from the aforementioned sets, with each predicate being associated with an *arity*, which is a non-negative integer.

An *atom* is of the form $p(t_1, \dots, t_n)$, where p is an n -ary predicate and the t_i 's are terms. We write an atom also as $p(\mathbf{t})$, where \mathbf{t} is a sequence of terms. An atom without variables is also called a *fact*. An *instance* is a finite multiset of facts. A *database* is a finite set of facts containing constants only. An instance containing only constants is said to be *complete*. Notice that, as opposed to databases, instances can contain duplicates—as shown in the following, instances are used in intermediate steps during repairs' computation and it is needed to keep duplicates to count the number of modifications being made. We assume the existence of a function db converting a complete instance to a database by eliminating duplicates.

A *homomorphism* is a mapping $h : \text{Const} \cup \text{Var} \cup \text{Null} \rightarrow \text{Const} \cup \text{Var} \cup \text{Null}$ that is the identity on *Const*. Homomorphisms are also applied to atoms and (multi) sets of atoms in the natural fashion, that is, $h(p(t_1, \dots, t_n)) = p(h(t_1), \dots, h(t_n))$, and $h(S) = \{h(A) \mid A \in S\}$ for every (multi) set S of atoms. A *valuation* is a homomorphism ν whose image is *Const*, that is, $\nu(t) \in \text{Const}$ for every $t \in \text{Const} \cup \text{Var} \cup \text{Null}$.

An *equality generating dependency* (EGD) σ is a first-order formula of the form $\forall \mathbf{x} \varphi(\mathbf{x}) \rightarrow x_i = x_j$, where $\varphi(\mathbf{x})$ is a conjunction of atoms (without labeled nulls) whose variables are exactly \mathbf{x} , and x_i and x_j are variables from \mathbf{x} . We call $\varphi(\mathbf{x})$ the *body* of σ , and call $x_i = x_j$ the *head* of σ . We will omit the universal quantification in front of dependencies and assume that all variables are universally quantified.

With a slight abuse of notation, we sometimes treat a conjunction of atoms as the *set* of its atoms. An instance J *satisfies* σ , denoted $J \models \sigma$, if whenever there exists a homomorphism h s.t. $h(\varphi(\mathbf{x})) \subseteq J$, then $h(x_i) = h(x_j)$. An instance J *satisfies* a set Σ of EGDs, denoted $J \models \Sigma$, if $I \models \sigma$ for every $\sigma \in \Sigma$. A *knowledge base* (KB) is a pair

(D, Σ) , where D is a database and Σ is a finite set of EGDs. We say that the knowledge base is *consistent* if $D \models \Sigma$, otherwise it is *inconsistent*.

Acyclic EGDs. An *argument* of a set of EGDs Σ is an expression of the form $p[i]$, where p is an n -ary predicate appearing in Σ and $1 \leq i \leq n$. The *argument graph* of Σ is a directed graph $G_\Sigma = (V, E)$, where V is the set of all arguments of Σ , and E contains a directed edge from $p[i]$ to $q[j]$ labeled σ iff there is an EGD $\sigma \in \Sigma$ such that:

- the body of σ contains an atom $p(t_1, \dots, t_n)$ such that either t_i is a constant or t_i is a variable occurring more than once in the body of σ , and
- the body of σ contains an atom $q(u_1, \dots, u_m)$ such that u_j is a variable also appearing in the head of σ .

The *dependency graph* of Σ is a directed graph $\Gamma_\Sigma = (\Sigma, \Omega)$, where Ω is the set $\{(\sigma_1, \sigma_2) \mid G_\Sigma = (V, E) \wedge (p([i], q[j], \sigma_1), (q[j], r[k], \sigma_2)) \in E\}$. We say that Σ is *acyclic* if its dependency graph is acyclic.

In the rest of the paper we consider acyclic sets of EGDs only. Thus, from now on, Σ is understood to be acyclic for every knowledge base (D, Σ) . For acyclic sets of EGDs we can define a partial order $(\Sigma, <)$ where EGDs in Σ are ordered by reachability in Γ_Σ . The result of evaluating a query Q over a database D is denoted $Q(D)$.

Repairs. The notion of repair used in this paper is based on updating facts and assumes that conflicting facts denote an *attribute-level uncertainty* in the data [12,7,21,4,11,15]. Alternative approaches based on fact deletions assume that conflicting facts denote a *tuple-level uncertainty* [2,20,10]. The computation of repairs consists of a chase-like procedure that acts as follows: whenever an EGD $\varphi(\mathbf{x}) \rightarrow x_i = x_j$ is not satisfied by a database D , i.e. there exists an homomorphism h such that $D \models h(\varphi(\mathbf{x}))$ and $h(x_i) \neq h(x_j)$, we have to enforce it by making the two values equal, that is either $h(x_i)$ replaces $h(x_j)$ or vice versa. A repair is obtained through an exhaustive application of this repair step. Note that, although we follow the partial order $(\Sigma, <)$ to choose the EGD to enforce, there is a non-deterministic choice to be made when selecting an EGD and when updating values; this may lead to multiple repairs.

Example 3. Consider the set of EGDs

$$\Sigma = \{\sigma_1: \text{employee}(X, Y_1, Z_1) \wedge \text{employee}(X, Y_2, Z_2) \rightarrow Y_1 = Y_2, \\ \sigma_2: \text{employee}(X_1, Y, Z_1) \wedge \text{employee}(X_2, Y, Z_2) \rightarrow Z_1 = Z_2\}$$

and the database D :

bob	cs	rome
bob	math	rome
alice	math	nyc

By enforcing σ_1 into the first two facts, either cs or math can be chosen as bob's department. If the latter is chosen, then the instance D' below is obtained.

Suppose now that σ_2 is enforced into the last two facts of D' . Then, either rome or nyc can be chosen as math's city. If the former is chosen, we obtain the instance D'' :

$$D' = \begin{array}{|c|c|c|} \hline \text{bob} & \text{math} & \text{rome} \\ \hline \text{bob} & \text{math} & \text{rome} \\ \hline \text{alice} & \text{math} & \text{nyc} \\ \hline \end{array} \quad D'' = \begin{array}{|c|c|c|} \hline \text{bob} & \text{math} & \text{rome} \\ \hline \text{bob} & \text{math} & \text{rome} \\ \hline \text{alice} & \text{math} & \text{rome} \\ \hline \end{array}$$

No further dependency enforcement is applicable at this point and thus the database derived from D'' by eliminating duplicates is a repair. \square

The previous example informally illustrated the basic idea of the repair strategy we adopt. In the next section, we formally define it and show how to associate probabilities to repairs on the basis of the modifications that yielded them.

Notice that it might be possible to restore consistency in other different ways. For instance, in the first step of the example above, one may modify the employee names. However, we do not consider this option because it is unclear which (different) values should be assigned (any constant in Const is a candidate value). For instance, bob in the first fact might be replaced with rome, but this is somewhat arbitrary and indeed does not make much sense. In contrast, our repair strategy chooses candidate values that are somehow “justified” by the content of the database (e.g., in the example above, bob works for either the cs or the math department). Moreover, when EGDs are key dependencies, the aforementioned way of restoring consistency may lead to the introduction of entities that are not meaningful. Indeed, our choice has been made by different approaches relying on value updates—e.g., [7]. For special classes of EGDs, the repair strategy based on updating values coincides with the repair strategy based on fact deletions.

3 Probabilistic Repairs and Query Answers

One problem of the classical notion of consistent query answer is that query answers can provide little information in the presence of conflicting information. The alternative approach proposed in this paper is to compute probabilistic answers by taking into account in to what extent information is updated to restore consistency.

In this section, we define probabilistic repairs and probabilistic query answers. In the next section, we show how to compute a compact representation of all probabilistic repairs. In both cases, we will use *probabilistic instances*, which are used to represent a single probabilistic repair in this section and are used to compactly represent all probabilistic repairs in the next section. Probabilistic instances are instances augmented with a set of “probability assignments”—intuitively, expressions stating conditions on nulls and probabilities on the values they can take.

More formally, let \mathcal{C} be the set of all expressions, called *conditions*, that can be built using the standard logical connectives \wedge , \vee , \neg , and expressions of the form $t_i = t_j$, true, and false, where $t_i, t_j \in \text{Const} \cup \text{Null}$. We will also use $t_i \neq t_j$ as a shorthand for $\neg(t_i = t_j)$. A homomorphism h *satisfies* a condition φ , denoted $h \models \varphi$, if $h(\varphi)$ is true.

A *probability assignment* is an expression of the form $P(\varphi_1 \mid \varphi_2) = p$, where φ_1, φ_2 are conditions and $p \in [0, 1]$.

A homomorphism h *satisfies* a probability assignment $P(\varphi_1 \mid \varphi_2) = p$ if the following holds true: if $h \models \varphi_2$ then $h \models \varphi_1$. Also, h *satisfies* a set Φ of probability assignments, denoted $h \models \Phi$, if h satisfies every probability assignment in Φ . For ease of presentation, a probability assignment of the form $P(\varphi_1 \mid \text{true}) = p$ will be simply written as $P(\varphi_1) = p$.

A *probabilistic instance* (PI) is a pair $K = (J, \phi)$, where J is an instance and ϕ is a finite set of probability assignments. For any valuation ν , we define

$$\text{Pr}(\nu, K) = \prod \{p \mid P(\varphi_1 \mid \varphi_2) = p \in \Phi \text{ and } \nu \models \varphi_2\}.$$

The semantics of K is given by the set of its *probabilistic worlds*, that is, the set of pairs (instance, probability) defined as follows:

$$pw(K) = \{(\nu(J), Pr(\nu, K)) \mid \nu \text{ is a valuation s.t. } \nu \models \Phi \wedge Pr(\nu, K) > 0\}.$$

The probabilistic repairs of an inconsistent knowledge base (D, Σ) are computed starting from the probabilistic instance (D, \emptyset) and iteratively enforcing EGDs (following a topological sorting of Γ_Σ). During the repair process we keep track of which values have been updated and how many modifications have been applied using labeled nulls and probability assignments. The enforcement of an EGD, which we call probabilistic repair step, is informally shown in the next example.

Example 4. Consider the knowledge base (D, Σ) , where D and Σ are from Example 3. Starting from the probabilistic instance (D, \emptyset) , by enforcing σ_1 into the first two facts, either cs or math can be chosen as bob's department. Therefore we obtain the following two (alternative) probabilistic instances K_1 (left) and K_2 (right):

bob	\perp_1	rome
bob	\perp_1	rome
alice	math	nyc

$P(\perp_1 = \text{cs}) = 1/2$

bob	\perp_1	rome
bob	\perp_1	rome
alice	math	nyc

$P(\perp_1 = \text{math}) = 1/2$

The only probabilistic world of K_1 is $(D_1, 1/2)$, while the only probabilistic world of K_2 is $(D_2, 1/2)$, where D_1 and D_2 are as follows (and are obtained by replacing \perp_1 with cs and math, respectively):

$D_1 =$

bob	cs	rome
bob	cs	rome
alice	math	nyc

$D_2 =$

bob	math	rome
bob	math	rome
alice	math	nyc

As $D_1 \models \Sigma$, $(db(D_1), 1/2)$ is a probabilistic repair. On the other hand, $D_2 \not\models \Sigma$, and thus we need to keep applying pr-steps over K_2 . By enforcing σ_2 over the first and third facts of K_2 we can get the following two (alternative) probabilistic instances K_3 (left) and K_4 (right):

bob	\perp_1	\perp_2
bob	\perp_1	rome
alice	math	\perp_2

$P(\perp_1 = \text{math}) = 1/2$
 $P(\perp_2 = \text{rome}) = 1/2$

bob	\perp_1	\perp_2
bob	\perp_1	rome
alice	math	\perp_2

$P(\perp_1 = \text{math}) = 1/2$
 $P(\perp_2 = \text{nyc}) = 1/2$

The only probabilistic world of K_3 is $(D_3, 1/4)$, while the only probabilistic world of K_4 is $(D_4, 1/4)$, where D_3 and D_4 are as follows:

$D_3 =$

bob	math	rome
bob	math	rome
alice	math	rome

$D_4 =$

bob	math	nyc
bob	math	rome
alice	math	nyc

In the rest of the paper, for every probabilistic instances (J, ϕ) , we report the probability assignments in ϕ under J .

Since $D_3 \models \Sigma$, $(\text{db}(D_3), 1/4)$ is a probabilistic repair. On the other hand, $D_4 \not\models \Sigma$, and thus further pr-steps need to be applied to K_4 . By enforcing σ_2 over the first two facts of K_4 we can get the following two (alternative) probabilistic instances K_5 (left) and K_6 (right):

<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px;">bob</td><td style="padding: 2px;">\perp_1</td><td style="padding: 2px;">\perp_3</td></tr> <tr><td style="padding: 2px;">bob</td><td style="padding: 2px;">\perp_1</td><td style="padding: 2px;">\perp_3</td></tr> <tr><td style="padding: 2px;">alice</td><td style="padding: 2px;">math</td><td style="padding: 2px;">\perp_3</td></tr> </table>	bob	\perp_1	\perp_3	bob	\perp_1	\perp_3	alice	math	\perp_3	<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px;">bob</td><td style="padding: 2px;">\perp_1</td><td style="padding: 2px;">\perp_3</td></tr> <tr><td style="padding: 2px;">bob</td><td style="padding: 2px;">\perp_1</td><td style="padding: 2px;">\perp_3</td></tr> <tr><td style="padding: 2px;">alice</td><td style="padding: 2px;">math</td><td style="padding: 2px;">\perp_3</td></tr> </table>	bob	\perp_1	\perp_3	bob	\perp_1	\perp_3	alice	math	\perp_3
bob	\perp_1	\perp_3																	
bob	\perp_1	\perp_3																	
alice	math	\perp_3																	
bob	\perp_1	\perp_3																	
bob	\perp_1	\perp_3																	
alice	math	\perp_3																	
$P(\perp_1 = \text{math}) = 1/2$	$P(\perp_1 = \text{math}) = 1/2$																		
$P(\perp_2 = \text{nyc}) = 1/2$	$P(\perp_2 = \text{nyc}) = 1/2$																		
$P(\perp_3 = \text{rome}) = 1/3$	$P(\perp_3 = \perp_2) = 2/3$																		

The only probabilistic world of K_5 is $(D_5, 1/12)$, while the only probabilistic world of K_6 is $(D_6, 1/6)$, where D_5 and D_6 are as follows:

$D_5 =$	<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px;">bob</td><td style="padding: 2px;">math</td><td style="padding: 2px;">rome</td></tr> <tr><td style="padding: 2px;">bob</td><td style="padding: 2px;">math</td><td style="padding: 2px;">rome</td></tr> <tr><td style="padding: 2px;">alice</td><td style="padding: 2px;">math</td><td style="padding: 2px;">rome</td></tr> </table>	bob	math	rome	bob	math	rome	alice	math	rome	$D_6 =$	<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px;">bob</td><td style="padding: 2px;">math</td><td style="padding: 2px;">nyc</td></tr> <tr><td style="padding: 2px;">bob</td><td style="padding: 2px;">math</td><td style="padding: 2px;">nyc</td></tr> <tr><td style="padding: 2px;">alice</td><td style="padding: 2px;">math</td><td style="padding: 2px;">nyc</td></tr> </table>	bob	math	nyc	bob	math	nyc	alice	math	nyc
bob	math	rome																			
bob	math	rome																			
alice	math	rome																			
bob	math	nyc																			
bob	math	nyc																			
alice	math	nyc																			

Since $D_5 \models \Sigma$ and $D_6 \models \Sigma$, both $(\text{db}(D_5), 1/12)$ and $(\text{db}(D_6), 1/6)$ are probabilistic repairs, and no further pr-step need to be applied.

Thus, the probabilistic repairs are $(\text{db}(D_1), 1/2)$, $(\text{db}(D_3), 1/4)$, $(\text{db}(D_5), 1/12)$, and $(\text{db}(D_6), 1/6)$. Notice that the sum of the probabilities of the probabilistic repairs is 1. Also, notice that D_3 and D_5 coincide. They might be replaced by a single probabilistic repair $(D', 1/4 + 1/12)$, where $D' = D_3 = D_5$. However, this does not affect query answering, that is, the probabilistic query answers are the same in both cases. \square

4 Further developments

In light of previous result, in the full version of this paper we develop a polynomial time algorithm to compute approximate query answers where point probability are approximated by probability intervals. Thus, rather than providing query answers of the form (t, p) , where t is a tuple and p is its probability, our approximation algorithm gives query answers of the form $(t, [\ell, u])$ guaranteeing that $p \in [\ell, u]$.

In order to do that, we define *probabilistic universal repairs*, i.e., compact representations of all repairs along with their degrees of consistency. This is one of the main tools used by our approximation algorithm, which works as follows: (i) it computes a universal probabilistic repair, (ii) it evaluates the given query over the universal probabilistic repair so as to produce a set of tuples associated with conditions (which keep track of how each tuple is derived), and (iii) it combines tuples' conditions with probability assignments of the universal probabilistic repair to derive the approximate intervals. Further developments should consider more general frameworks with TGDs [10] and logic rules [14,8,9].

References

1. Periklis Andritsos, Ariel Fuxman, and Renée J. Miller. Clean answers over dirty databases: A probabilistic approach. In *ICDE*, page 30, 2006.

2. Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.
3. Leopoldo E. Bertossi. *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.
4. Leopoldo E. Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Inf. Syst.*, 33(4-5):407–434, 2008.
5. Meghyn Bienvenu, Camille Bourgaux, and Francois Goasdoué. Querying inconsistent description logic knowledge bases under preferred repair semantics. In *AAAI*, pages 996–1002, 2014.
6. Meghyn Bienvenu and Riccardo Rosati. Tractable approximations of consistent query answering for robust ontology-based data access. In *IJCAI*, pages 775–781, 2013.
7. Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154, 2005.
8. Marco Calautti, Sergio Greco, Cristian Molinaro, and Irina Trubitsyna. Checking termination of logic programs with function symbols through linear constraints. In *RuleML*, volume 8620 of *LNCSE*, pages 97–111. Springer, 2014.
9. Marco Calautti, Sergio Greco, Cristian Molinaro, and Irina Trubitsyna. Logic program termination analysis using atom sizes. In *IJCAI*, pages 2833–2839. AAAI Press, 2015.
10. Marco Calautti, Leonid Libkin, and Andreas Pieris. An operational approach to consistent query answering. In *PODS*, 2018.
11. Sergio Flesca, Filippo Furfaro, and Francesco Parisi. Querying and repairing inconsistent numerical databases. *ACM Trans. Database Syst.*, 35(2):14:1–14:50, 2010.
12. Enrico Franconi, Antonio Laureti Palma, Nicola Leone, Simona Perri, and Francesco Scarcello. Census data repair: a challenging application of disjunctive logic programming. In *LPAR*, pages 561–578, 2001.
13. Sergio Greco and Cristian Molinaro. Probabilistic query answering over inconsistent databases. *Ann. Math. Artif. Intell.*, 64(2-3):185–207, 2012.
14. Sergio Greco, Cristian Molinaro, and Irina Trubitsyna. Logic programming with function symbols: Checking termination of bottom-up evaluation through program adornments. *Theory Pract. Log. Program.*, 13(4-5):737–752, 2013.
15. Sergio Greco, Cristian Molinaro, and Irina Trubitsyna. Computing approximate query answers over inconsistent knowledge bases. In *IJCAI*, pages 1838–1846, 2018.
16. Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Inconsistency-tolerant query answering in ontology-based data access. *J. Web Sem.*, 33:3–29, 2015.
17. Thomas Lukasiewicz, Enrico Malizia, and Cristian Molinaro. Complexity of approximate query answering under inconsistency in datalog \pm . In *IJCAI*, pages 1921–1927, 2018.
18. Thomas Lukasiewicz, Maria Vanina Martinez, Andreas Pieris, and Gerardo I. Simari. From classical to consistent query answering under existential rules. In *AAAI*, pages 1546–1552, 2015.
19. Riccardo Rosati. On the complexity of dealing with inconsistency in description logic ontologies. In *IJCAI*, pages 1057–1062, 2011.
20. Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.
21. Jef Wijsen. Database repairing using updates. *ACM Trans. Database Syst.*, 30(3):722–768, 2005.