A relational framework for inconsistency-aware query answering

Ousmane Issa, Angela Bonifati, Farouk Toumani

A relational framework for inconsistency-aware query answering

Ousmane Issa
University Clermont Auvergne
Clermont-Ferrand, France
ousmane.issa@uca.fr

Angela Bonifati
Lyon 1 University
Villeurbanne, France
angela.bonifati@univ-lyon1.fr

Farouk Toumani
University Clermont Auvergne
Clermont-Ferrand, France
farouk.toumani@uca.fr

ABSTRACT

We introduce a novel framework for encoding inconsistency into relational tuples and tackling query answering for union of conjunctive queries (UCQs) with respect to a set of denial constraints (DCs). We define a notion of inconsistent tuple with respect to a set of DCs and define four measures of inconsistency degree of an answer tuple of a query. Two of these measures revolve around the minimal number of inconsistent tuples necessary to compute the answer tuples of a UCQ, whereas the other two rely on the maximum number of inconsistent tuples under set- and bag-semantics, respectively. In order to compute these measures of inconsistency degree, we leverage two models of provenance semiring, namely why-provenance and provenance polynomials, which can be computed in polynomial time in the size of the relational instances for UCQs. Hence, these measures of inconsistency degree are also computable in polynomial time in data complexity. We also investigate top-k and bounded query answering by ranking the answer tuples by their inconsistency degrees. We explore both a full materialized approach and a semi-materialized approach for the computation of top-k and bounded query results.

KEYWORDS

Inconsistency, Why-provenance, Query Answering, Conjunctive Queries, Denial Constraints, Ranked Enumeration

ACM Reference Format:
Ousmane Issa, Angela Bonifati, and Farouk Toumani. 2019. A relational framework for inconsistency-aware query answering. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 12 pages.

1 INTRODUCTION

Assessing the quality of data is crucial in many applications, where the quality of raw instances has a non negligible impact on the quality of analytical processes on these instances as well as the trustworthiness of query answering on them. Whereas current work on data curation for relational tuples has addressed the problem of detecting and repairing the violations with respect to a set of constraints [2, 27] along with consistent query answering when multiple repairs are possible[7], very little attention has been paid to leave the database instance as is and quantifiying its degree of inconsistency at different levels of granularity (tuple, sets of tuples, tables). Previous work has focused on quantifying the degree of quality of knowledge bases [13, 19, 26], whereas the relational setting and especially relational query answering have been fairly disregarded.

In our work, we are interested in conceiving novel measures of inconsistency of answer tuples for Unions of Conjunctive Queries (UCQs) over an inconsistent database in the presence of a set of Denial Constraints (DCs) [10]. The inconsistency degree of an answer tuple of a query is determined during query answering by relying on provenance-based information of the input tuples. We first leverage the why-provenance in order to annotate the tuples of a relational instance with their degree of inconsistency with respect to a set of DCs, and then we rely on the provenance polynomials [16] in order to propagate the degrees of inconsistencies from the input tuples to the answer tuples of UCQs. This method is applicable to all types of relational instances that cannot be repaired in situ, such as for instance clinical data, sensitive data and financial data, whose data values cannot be modified for instance due to privacy and ownership restrictions. We exemplify the aforementioned process by means of the following running example.

Example 1. *Consider a relational instance $\mathcal{I}$ in Figure 1 consisting of three relations D, V and S with a corresponding number of denial constraints IC and a query Q_{ex}. In each relation in $\mathcal{I}$, the first column is the patient identifier PID, the second column is the desease reference RefID and the third column is the Date of a given event. Notice that the schema of the three tables is the same solely for illustration purposes and to maximize the number of joins across the tables. In fact, our methods are applicable to relations with an arbitrary schema.*

The denial constraint (C_1) imposes to have any diagnosis for a patient's disease before surgery for the same disease concerning the same patient. The constraint (C_2) and (C_3) establish that a patient cannot be diagnosed a given disease for which he/she has been administered a vaccine on a previous date. [1] *Finally, a conjunctive query Q_{ex} extracts pairs of diseases for which the same patient underwent surgey and was administered a vaccine. The tuples in the relations*

[1]Notice that there are exceptions to the last two constraints when a second shot of a vaccine is somministrated or when the immunization offered by a vaccine did not work properly. These cases would be covered by associating probabilities to the constraints, which we do not consider in our work for the time being.

$Diagnosis(D)$

PID	RefD	Date	
P01	R1	2	t_1
P02	R7	4	t_2
P03	R7	10	t_3
P04	R1	3	t_4
P04	R4	8	t_5
P02	R4	7	t_6
P01	R2	5	t_7
P08	R7	4	t_8
P10	R1	5	t_9

$Vaccination(V)$

PID	RefD	Date	
P01	R2	3	t_{10}
P02	R4	5	t_{11}
P03	R4	6	t_{12}
P04	R4	7	t_{13}
P02	R5	10	t_{14}
P08	R5	7	t_{15}
P10	R3	8	t_{16}

$Surgery(S)$

PID	RefD	Date	
P01	R1	1	t_{17}
P02	R7	3	t_{18}
P03	R7	9	t_{19}
P04	R1	4	t_{20}
P08	R7	5	t_{21}

The query (Q_{ex}) is the following

$Q(y, u) \leftarrow D(x, y, z),\ S(x, y, z'),\ V(x, u, v)$

The set of denial constraints (DCs) are :

$C_1 : \leftarrow\ D(x, y, z) \wedge S(x, y, u) \wedge z > u$

$C_2 : \leftarrow\ D(x,' R2', y) \wedge V(x,' R2', z) \wedge y > z$

$C_3 : \leftarrow\ D(x,' R4', y) \wedge V(x,' R4', z) \wedge y > z$

Figure 1: A hospital database hdb with a set of denial constraints (DCs) and a query Q_{ex}

y	u	Provenance
R1	R2	$[t_1,\ t_{10},\ t_{17}]$
R7	R4	$[t_2,\ t_{18},\ t_{11}]$ or $[t_2,\ t_{12},\ t_{19}]$
R1	R4	$[t_4,\ t_{13},\ t_{20}]$
R7	R5	$[t_2,\ t_{18},\ t_{14}]$ or $[t_8,\ t_{15},\ t_{21}]$

Table 1: Answer tuples of query Q_{ex} with their provenance information

highlighted in red are those that violate one or more constraints (C_1), (C_2) or (C_3).

Before evaluating the query Q_{ex}, we annotate each tuple in the databases instance I with a unique identifier. When applying the why-provenance [16] to the tuples augmented with their identifiers, and for each answer tuple of query Q_{ex}, the corresponding possible derivations in terms of tuple identifiers are shown (see Table 1).

We can notice that each answer tuple contains red-flagged tuple identifiers. By counting the number of red-flagged identifiers, we can compute the inconsistency degree of the tuple. For instance, the answer tuple $\{R_1, R_2\}$ has inconstency degree equal to 3, whereas the answer tuple $\{R_7, R_4\}$ will have inconsistency degree equal to 3 or 2, depending on whether we favor a greater or smaller number of derivating inconsistent tuples, and so on. We can also obtain duplicates in the provenance column that might be taken into account in the counting or not. These considerations led us to precisely define four measures of inconsistency degrees unser set- and bag-semantics and to provide practical methods to compute them starting from plain database instances. We tackle these questions in the remainder of the paper.

Main Contributions. In this paper, we make the following contributions:

(1) We conceive four novel measures of inconsistency degrees of answer tuples for Unions of Conjunctive Queries (UCQs) over an inconsistent database in the presence of a set of Denial Constraints (DCs). We consider the Max and Min number of derivation tuples and the set- and bag-semantics in order to define these four measures.

(2) We leverage why-provenance in order to identify the inconsistent tuples of a given database w.r.t. a set of *DC*, and then we exploit provenance polynomials to propagate the inconsistency degree of tuples in the original database instance to the answer tuples of UCQs in the presence of DCs, which is to the best of our knowledge a novel and quite promising usage of provenance.

(3) We define threshold query answering and top-k query answering consisting of computing the answer tuples that satisfy a given threshold of inconsistency degree and/or enumerating those that exhibit a given value (k) of inconsistency degree.

(4) We propose a preliminary solution consisting in two practical algorithms to allow inconsistency-aware query answering: the **Full-materialized Algorithm** being our baseline and allowing polynomial-time computation of inconsistent-aware query answering. This approach is costly in terms of space since all the possible solutions need to be computed and stored beforehand; the **Semi-materialized Algorithm**, that allows to check on the fly the two variants of the threshold problems (less than k problem measuring the minimum number of inconsistent tuples and greater than k problem on the maximum number, respectively) whether the answer tuples are in the result or not, thus incurring less space; when applied to top-k query answering the latter algorithm allows to leverage ranked enumeration and to retrieve directly the top-k answers of a conjunctive query with polynomial delay. This second approach applies for a restricted class of queries having a bounded tree decompositions.

The remainder of this paper is organized as follows. Section 2 discusses related works. Section 3 introduces basic notions. In this section definitions of database instance, conjunctive query (CQ), union of CQ, denial constraint (DC), *K-instance* and *why-provenance* are given. In section 4 we formally define inconsistency degree of each tuple in answers of query. Section 5 defines some underling filtering problems according to inconsistency degree. And in section 6, we conclude this paper.

2 RELATED WORK

Central to data quality is data consistency, a significant data quality dimension extensively investigated over the past decades. In the database field, data consistency is captured by means of integrity constraints as violations of constraints reflect a lack of consistency of the data stored in a database. In this setting, starting from the pioneering work of [2], there has been a wealth of research on the problem of *consistent query answering in inconsistent databases*

[4, 24, 27]. Most of the existing works make a distinction between consistent and inconsistent answers based on the notion of a *database repair* (see [6, 7] for a survey). A repair is a database instance obtained by applying some *minimal changes* on the initial database instance in order to make it compliant with a given set of integrity constraints. Usually, there exist several possible repairs of a given database and for such a reason, following the *certain answers* semantics, consistent answers to a given query are defined as being the intersection of the query answers on all possible repairs of the initial database instance. The problem of *consistent query answering* has been investigated in several settings depending on: *(i)* the definition of repairs, where a repair is defined in terms of the sets of inserted and/or deleted tuples [6, 7], or updates [3, 20, 27] and *(ii)* the considered integrity constraints and the query language. The latter correspond to varied combinations of queries and constraints such as first-order queries and binary universal constraints or single functional dependencies [2], union of conjunctive queries and key and inclusion dependencies [8], first-order queries and denial constraints [9], to mention a few. More similarly to our work, recent approaches studied a counting variant of consistent query answering [24], i.e., *counting how many repairs satisfy a given query q* in the attempt of defining a metric for consistent query answering. The work in [24] establishes a dichotomy for the #-CERTAINTY(q) problem, where q is a boolean conjunctive query possibly with self-joins and with single-attribute primary-key (PK) constraints. The dichotomy does not apply to relations with composite PKs or to UCQs, thus it would be not usable in practice. In our work, we depart from the notion of repair, which might not be easy to compute in some circumstances with data that cannot be directly modified (due to data protection rules or other reasons). *In view of this, we annotate the query results at the tuple level with an inconsistency degree, i.e. by associating to each tuple the number of violations of the involved constraints. We show that instead of computing all possible repairs and counting them, we can leverage provenance to actually count the number of violations of a large spectrum of constraints (up to Denial Constraints, and thus well beyond PKs).*

Reasoning in Knowledge Bases (KBs) in the presence of inconsistency is also widely studied in the Artificial Intelligence community, where it has been shown that an arbitrary consequent can be entailed from an inconsistent set of premises (see the survey by Lang et al. [21]). Similarly, the role of inconsistency tolerance in KBs has been introduced in [5]. *They are more concerned about the co-existence of consistency and inconsistency as separate axioms when doing reasoning in KBs rather than using inconsistency measures to yield inconsistency-aware query answering, as we do in our work.*

Another line of research deals with the the definition of inconsistency measures in KBs [13, 19]. Numerous measures of inconsistency in KBs have been proposed based on various paradigms such as information theory [22], possibility theory [11] or quasi-classical logic [19], to mention a few. The ultimate goal is, however, to provide measures to quantify the inconsistency of a KBs in order, for example, to allow comparison of KBs based on their inconsistency degrees. Extensions of such an approach to the database field has been recently investigated in [6] where inconsistency measures based on various classes of repair semantics are studied. *However, their notion of inconsistency measures (despite the name) relies on how complex it is to restore the consistency of a database, thus on the class of repairs admitted by the database. As such, it is quite different from our notion of inconsistency degree in which repairs are not taken into account.*

Top-k query processing has been extensively studied in the database literature (see [17] for a survey). The basic idea is to associate to a tuple a score and to use this score in the algebraic plan and consequent query processing in order to return the top-k results of a given query [1, 18]. Our techniques are based on provenance and are not considering specific choices of query plans and query processing decisions. We believe that this line of research could be interesting in future steps of our work. We use recent work [14] that enumerates the top-k answers ordered by their score in polynomial delay.

3 PRELIMINARIES

In this section, we introduce some basic notions used in this paper.

3.1 Basic notions

A database schema $\mathcal{S}$ is a finite set of relation symbols or predicates $\{R_1, \dots, R_n\}$. We denote the arity of a relational symbol R_i as $arity(R_i)$. Let $\mathcal{D}$ be an infinite set of constants called domain. A database instance $\mathcal{I}$ over a schema $\mathcal{S}$ is a finite set of relations $\mathcal{I}(R_i) \subseteq \mathcal{D}^{arity(R_i)}$ for every relational symbol $R_i \in \mathcal{S}$. Let Γ be an infinite set of identifiers distinct from the domain $\mathcal{D}$. We denote by id a function from a database instance $\mathcal{I}$ to Γ that associates to each tuple $t \in \mathcal{I}(R_i)$ an unique idenfier $id(t)$ from the set Γ.

Union of Conjunctive Queries (UCQ). A built-in atom is an atom under the form $(x\ op\ y)$ where $op \in \{=, \neq, \geq, \leq, <, >\}$, with x and y being either variables or constants. A built-in conjunctive formula is a conjunction of built-in atoms. A query is a mapping from a database $\mathcal{I}$ over a schema $\mathcal{S}$ to relations over a new schema $\mathcal{O}$, called output schema. The class of conjunctive queries (CQ) is defined as a set of queries of the following form:

$$Q(u) \leftarrow R_1(u_1), ..., R_n(u_n), \phi(u_1, \dots, u_n) \tag{1}$$

where each R_i is a relation symbol in the database schema $\mathcal{S}$ and Q is a relation symbol in the output schema $\mathcal{O}$. u is a tuple of distinguished varaibales or constants and each u_i is a tuple of variables and constants having the same arity as R_i. The formula $\phi(u_1, \dots, u_n)$ is a built-in conjunctive formula over the variables appearing in the u_is and constants from the domain $\mathcal{D}$. The head of query Q is $head(Q) \overset{\text{def}}{=} Q(u)$. The body of Q is $body(Q) \overset{\text{def}}{=} R_1(u_1), ..., R_n(u_n)$. All variables in $head(Q)$ appear in $body(Q)$. We denote by $Vars(Q)$ all variables in Q. When u is empty, we say that Q is a boolean conjunctive query. An union of conjunctive queries ($\mathcal{UCQ}$) is a set of conjunctive queries with same arity of heads. A query Q is a full CQ if all variables in Q appear also in $head(Q)$. A query Q is a self-join free CQ if in the body of Q there is no two atoms with the same predicate name. A natural join CQ is a full CQ where in each atom in $body(Q)$ there is no constant and there is no variable with more than one occurrence.

Denial contraints. A denial constraint is a formula of the form:

$$\leftarrow R_1(u_1) \wedge ... \wedge R_n(u_n) \wedge \phi(u_1, \dots, u_n) \tag{2}$$

where the $R_i(u_i)$ and $\phi(u_1, \dots, u_n)$ are defined as previously.

A database instance $\mathcal{I}$ is consistent w.r.t. a denial constraint C iff $\mathcal{I} \models C$. This occurs if the body of the denial constraint, which can be viewed as a boolean conjunctive query, retruns $false$ when evaluated over the instance $\mathcal{I}$.

Notation over Polynomials. A monomial M_o over $\mathbb{N}$ and a finite set of variables X is defined by $M_o(X) = a \times x_1^{m_1} \times ... \times x_n^{m_n}$ with $a, m_1, ..., m_n \in \mathbb{N}$ and $x_1, ..., x_n \in X$. M_o is a non null monomial ($M \neq 0$) if $a \neq 0$ otherwise it is a null monomial. The weight of a variable x_i w.r.t a monomial $M_o = a \times x_1^{m_1} \times ... \times x_n^{m_n}$, denoted by $W(M_o, x)$, is equal to m_i, the exponent of the variable x_i in M_o. The weight of a non null M_o, denoted by $W(M_o)$, is defined as the sum of the weights of its variables, i.e.:

$$W(M_o) = \sum_{i=0}^{n} W(M_o, x)$$

The set of variables of M_o, denoted by $Var(M_o)$, is the set of variables in M_o with non null weight. A polynomial P over $\mathbb{N}$ and a finite set of variables X is a finite sum of monomials over X. The set of variables of P, denoted by $Var(P)$, is the union of sets of variables of its monomials. Given a polynomial P, we use in the sequel following notation:

(1) $\mathcal{M}(P)$ to denote the set of monomials of P.
(2) $\mathcal{V}^{mi}(P)$ is the number of variables that appear in the smallest, w.r.t. the number of variables, monomial of P, i.e.,

$$\mathcal{V}^{mi}(P) = \underset{M_o \in \mathcal{M}(P)}{Min} |Var(M_o)|$$

(3) $\mathcal{V}^{ma}(P)$ is the number of variables that appear in the largest, w.r.t. the number of variables, monomial of P, i.e.,

$$\mathcal{V}^{ma}(P) = \underset{M_o \in \mathcal{M}(P)}{Max} |Var(M_o)|$$

(4) $\mathcal{W}^{mi}(P)$ is the minimal weight among the weights of non null monomials of P.

$$\mathcal{W}^{mi}(P) = \underset{M_o \in \mathcal{M}(P)}{Min} W(M_o)$$

(5) $\mathcal{W}^{ma}(P)$ is the highest weight among the weights of non null monomials of P. This is also the degree of P.

$$\mathcal{W}^{ma}(P) = \underset{M_o \in \mathcal{M}(P)}{Max} W(M_o)$$

Example 2. *Let P be the following polynomial over variables $\{x, y, z\}$*

$$P(x, y, z) = x^2 \times y^3 + x \times z \times y + 2 \times z^2$$

The different notions over P are

- $\mathcal{M}(P) = \{x^2 \times y^3,\ x \times z \times y,\ 2 \times z^2\}$
- $\mathcal{V}^{mi}(P) = 1$, *since the monomial with minimal number of variables is $2 \times z^2$ thus containing only one variable.*
- $\mathcal{V}^{ma}(P) = 3$, *since the monomial with maximum number of variables is $x \times z \times y$, hence containing 3 variables.*
- $\mathcal{W}^{mi}(P) = 2$, *since the monomial with minimum degree is $2 \times z^2$ and its degree is 2*
- $\mathcal{W}^{ma}(P) = 5$, *since the degree of P is 5*

A Γ-relation S'

PID	*RefD*	*Date*	
P01	R1	1	t_{17}
P02	R7	3	t_{18}
P03	R7	9	t_{19}
P04	R1	4	t_{20}
P08	R7	5	t_{21}

(a)

$Q(S')$

x	y	
P01	R1	$t_{17}^2 \times t_{18} + t_{17}^2 \times t_{19} + t_{17}^2 \times t_{21}$
P02	R7	$t_{18}^2 \times t_{17} + t_{18}^2 \times t_{21}$

(b)

Figure 2: (a) Example of a Γ-*relation* and (b) Answers of query Q over the Γ-*relation*.

3.2 Provenance Semirings

We recall below the provenance semirings framework [16] enabling to capture a wide range of provenance models at different levels of granularity [12, 15, 16]. This framework is based on a general data model extending the relational model to the so-called *K-relations* in which tuples are assigned annotations from a given semiring.

Notion of K-relations. An algebraic strucutre $(K, \oplus, \otimes, 0, 1)$ with binary operations sum $\oplus$ and product $\otimes$ and constants 0 and 1 is a (commutative) semiring iff $(K, \oplus, 0)$ and $(K, \otimes, 1)$ are commutative monoids with identities 0 and 1 respectively, $\otimes$ is distributive over $\oplus$ and $0 \otimes a = a \otimes 0 = 0$ holds $\forall a \in K$. Recall that a monoid is a set equipped with a single binary operation which is associative and has an identity element. In addition, the set is closed under the binary operation (i.e., the operation on any two elements in the set leads to a result which belongs to the set).

Example 3. *The following algebraic structures are commutative semirings:*

- $(\mathbb{N}, +, \times, 0, 1)$, *where $\mathbb{N}$ is the set of natural numbers and $+, \times$ are respectively addition and product on natural numbers.*
- $(\mathbb{B}, \vee, \wedge, false, true)$, *where $\mathbb{B} = \{false, true\}$ and $\vee, \wedge$ are respectively, the logical disjunction and logical conjunction.*
- $(\mathbb{N}[X], +, \times, 0, 1)$ *is the provenance polynomials semiring [15], where X is a set of variables and $\mathbb{N}[X]$ denotes the set of polynomials with variables from X and coefficients from $\mathbb{N}$ and $+, \times$ are respectively addition and product on natural numbers.*

An n-ary K-*relation* is a function $R : \mathcal{D}^n \to K$ such that its support, defined by $supp(R) \overset{\text{def}}{=} \{t : t \in \mathcal{D}^n, R(t) \neq 0\}$, is finite. Hence, a K-*relation* is an extension of the classical notion of relation to allow tuples to be annotated by semiring annotations. Let R be an n-ary K-*relation* and let $t \in domain^n$, the value $R(t) \in K$ assigned to the tuple t by the K-relation R is called the annotation of t in R.

Example 4. *Figure 2(a) shows an example of an Γ-relation S',i.e., a K-relation where K is the set Γ of tuple identifiers. This Γ-relation annotates each tuple of the relation Surgery of Figure 1 with its identifier.*

A K-*instance* is a mapping from relations symbols in a database schema $\mathcal{S}$ to K-*relations* (i.e, a finite set of K-*relations* over $\mathcal{S}$). If

$\mathcal{I}$ is a K-instance over a database schema $\mathcal{S}$ and $R_i \in \mathcal{S}$ is a relation symbol in $\mathcal{S}$, we denote by $\mathcal{I}(R_i)$ the K-relation corresponding to the value of R_i in $\mathcal{I}$.

Union of conjunctive queries on K-instances [15]. Let Q be a CQ and $Vars(Q)$ the set of variables that occur in Q. A valuation of Q over a domain $\mathcal{D}$ is a function $v : Vars(Q) \to \mathcal{D}$, extended to be the identity on constants.

Valuations are extended to tuples as follows: if $t = (t_1, ..., t_n)$ then $v(t) = (v(t_1), ..., v(t_n))$. Let Q be a conjunctive query,

$$Q(u) \leftarrow R_1(u_1), ..., R_n(u_n), \phi(u_1, \dots, u_n)$$

and let M be a K-*instance* over the same schema than Q, with $(K, \oplus, \otimes, 0, 1)$ a semiring. The result of evaluating Q over the K-*instance* M, using the semiring $(K, \oplus, \otimes, 0, 1)$, is the K-*relation* $Q(M)$ defined as follows:

$$Q(M) \overset{\text{def}}{=} \{\ (t, \sum_{v\ s.t\ cond_t^v} prod_v^q(M)\ )\ \}$$

where:

- $cond_t^v \overset{\text{def}}{=} (v(u) = t\ \wedge\ \phi(v(u_1), \dots, v(u_n))$, i.e., v is a valuation that maps u to t and ϕ to $true$, and
- $prod_v^q(M) \overset{\text{def}}{=} \Pi_{i=1}^n R_i(v(u_i))$, computes an annotation $\neq 0$ for each tuple t which is in the answer of the query Q over the K-*instance* M. The tuples $R_i(v(u_i)$ that are involved in the computation of the answer t are joined using the $\otimes$ operator. Since there could exist different ways to compute the same answer t, the complete annotation of t is obtained by combining the alternative ways to derive a tuple t using the $\oplus$ operator. Hence, the provenance of an answer t of the query Q over a K-*instance* M is computed as follows:

$$Q(M)(t) = \sum_{v\ s.t\ cond_t^v} prod_v^q(M)$$

The computation of the answers of an **UCQ** query $Q = (Q_1, ..., Q_m)$ over a K-*instance* M is obtained by merging the annotations of the answers of each query Q_I over M using the $\oplus$ operator, i.e.:

$$Q(M) \overset{\text{def}}{=} \{\ (t, \sum_{i=1}^{m} Q_i(M)(t)\ )\ \}$$

Example 5. *Consider the provenance polynomials semiring $(\mathbb{N}[\Gamma], +, \times, 0, 1)$, with variables from the set of tuples identifiers Γ, and the Γ-relation S' of of Figure 2(a) which annotates the tuple of the relation Surgery with their corresponding identifiers.*

Consider the following example query Q_{ex2}:

$$Q_{ex2}(x, y) \leftarrow S(x, y, z_1), S(x, y, z_2), S(x_1, y_1, z_3), y \neq y_1$$

The answers of the query Q_{ex2} over the Γ-relation S' is given in Figure 2(b). Note that, the answers of Q_{ex2} are annotated with polynomials from $\mathbb{N}[\Gamma]$. For example, the tuple $(P01, R1)$ is an answer of Q_{ex2} which is annotated with the polynomial $P = t_{17}^2 \times t_{18} + t_{17}^2 \times t_{19} + t_{17}^2 \times t_{21}$. Each monomial in P gives a way to compute the answer $(P01, R1)$ by the query Q_{ex2}. Hence, P conveys the information that the answer $(P01, R1)$ is computed by the query Q_{ex2} in three different ways: (i) using the tuple t_{17} twice together with the tuple t_{18}, or (ii) using the tuple t_{17} twice together with t_{19}, or (iii) using the tuple t_{17} twice together with t_{21}.

4 QUANTIFYING INCONSISTENCY DEGREES OF QUERY ANSWERS

In this section, we focus on the problem of quantifying the inconsistency degrees of query answers. Let $\mathcal{I}$ be an instance over a database schema $\mathcal{S}$ and let *DC* be a set of denial contraints over $\mathcal{S}$. We proceed in three steps in order to achieve our goal:

- *Identifying inconsistent tuples:* We first start by identifying the inconsistent tuples of an instance $\mathcal{I}$ over $\mathcal{S}$ w.r.t a set of denials constraints *DC*. To achieve this task, we turn the set *DC* of denial contraints into a boolean **UCQ** Q^{DC} and then we use the why-provenance (also known as lineage of Q^{DC}) to compute the set of inconsistent tuples of $\mathcal{I}$ w.r.t. to *DC*. The set of inconsistent tuples of an instance $\mathcal{I}$ w.r.t. a set of denial constraints *DC* is denoted by $IncT(\mathcal{I}, DC)$.
- *Annotating the initial database instance:* Using the set $IncT(\mathcal{I}, DC)$ of inconsistent tuples, we convert the instance $\mathcal{I}$ into a K-*instance* by annotating each consistent tuple in $\mathcal{I}$ with the value 1 and each inconsistent tuple $t \in IncT(\mathcal{I}, DC)$ with its identifier $id(t)$.
- *Defining inconsistency degrees of query answers:* Then, given an **UCQ** Q over the instance $\mathcal{I}$, we use provenance polynomials semiring to annotate the query answers. The latter provenance is the most informative form of provenance annotation [15] and hence is exploited in our setting in order to define different inconsistency degrees for query answers.

We shall detail in the sequel the proposed three-step approach. The first two steps are described in the Algorithm 1.

4.1 Identifying inconsistent tuples

Let $\mathcal{I}$ be an instance over a database schema $\mathcal{S}$ and let *DC* be a set of denial contraints over $\mathcal{S}$. We first convert the set *DC* into a boolean query Q^{DC} as follows. For each constraint $C \in DC$ of the form:

$$\leftarrow R_1(u_1) \wedge ... \wedge R_n(u_n) \wedge \phi(u_1, \dots, u_n)$$

we generate a boolean conjunctive query Q^{DC}:

$$Q^{DC}() \leftarrow R_1(u_1) \wedge ... \wedge R_n(u_n) \wedge \phi(u_1, \dots, u_n)$$

Hence, the obtained query Q^{DC} from the set *DC* is a boolean union of conjunctive queries **UCQ**.

Example 6. *The set DC of denial constraints depicted in Figure 1 leads to the following* **UCQ** *query:*

$$Q^{DC}() \leftarrow D(x, y, z) \wedge S(x, y, u) \wedge z > u$$
$$Q^{DC}() \leftarrow D(x, 'R2', y) \wedge V(x, 'R2', z) \wedge y > z$$
$$Q^{DC}() \leftarrow D(x, 'R4', y) \wedge V(x, 'R4', z) \wedge y > z$$

It is easy to verify that an instance $\mathcal{I}$ violates the set of denial constraints *DC* iff the query Q^{DC} evaluates to true over the instance $\mathcal{I}$ (i.e., $Q^{DC}(\mathcal{I}) = \{<>\}$, where the empty tuple <> denotes the *true* value of a boolean query). The lineage of the empty tuple <> enables the identification of the set of all contributing source tuples, and hence all the tuples that *"contribute"* to make the instance $\mathcal{I}$ inconsistent w.r.t. *DC*. We shall use the *provenance semirings* [15] to compute it.

Let $\mathcal{P}(\Gamma)$ be the powerset of the set of tuple identifiers Γ. Consider the following provenance semiring: $(\mathcal{P}(\Gamma) \cup \{\bot\}, +, ., \bot, \emptyset)$, where $\forall S, T \in \mathcal{P}(\Gamma) \cup \{\bot\}$, we have $\bot + S = S + \bot = S, \bot.S = S.\bot = \bot$

and $S + T = S.T = S \cup T$ if $S \neq \perp$ and $T \neq \perp$. This semiring consists of the powerset of Γ augmented with the distinguished element $\perp$ and equipped with the set union operation which is used both as addition and multiplication. The distinguished element $\perp$ is the neutral element of the addition and the annihilating element of the multiplication.

We convert the instance $\mathcal{I}$ over the schema $\mathcal{S}$ into a K-*instance*, denoted by $\mathcal{I}^{LP}$, with $K = \mathcal{P}(\Gamma) \cup \{\perp\}$. The K-*instance* $\mathcal{I}^{LP}$ is defined below.

Definition 1 (Provenance instances). *Let $\mathcal{I}$ be an instance over a database schema $\mathcal{S}$ and let DC be a set of denial contraints over $\mathcal{S}$. Let $K = \mathcal{P}(\Gamma) \cup \{\perp\}$. The K-instance $\mathcal{I}^{LP}$ is constructed as follows:*

- *$\forall R_i \in S$ a corresponding K-relation is created in $\mathcal{I}^{LP}$,*
- *A K-relation $\mathcal{I}^{LP}(R_i) \in \mathcal{I}^{LP}$ is populated as follows:*

$$\begin{cases} \mathcal{I}^{LP}(R_i)(t) = \{id(t)\} & if\ t \in \mathcal{I}(R_i) \\ \mathcal{I}^{LP}(R_i)(t) = \perp & otherwise \end{cases}$$

Example 7. *Figure 3 shows the provenance database hdb^{LP} obtained from the hospital database hdb by annotating each tuple $t \in hdb$ with a singleton set $\{id(t)\}$ containing the tuple identifier.*

Using the provenance semirings, we define below the inconsistent tuples of a given instance w.r.t. a set of denial constraints.

Definition 2 (Inconsistent tuples). *Given an instance I and a set of denial constraints DC, the set of inconsistent tuple identifiers, denoted by $IncT(\mathcal{I}, DC)$, is defined as follows*

$$IncT(\mathcal{I}, DC) \overset{def}{=} Q^{DC}(\mathcal{I}^{LP})(<>)$$

Consequently, a tuple $t \in \mathcal{I}$ is inconsistent w.r.t. DC if $id(t) \in IncT(\mathcal{I}, DC)$.

Example 8. *Consider the query Q^{DC} of Example 6 which is obtained from the set of denial constraints of the hdb database. The execution of the query Q^{DC} over the provenance database hdb^{LP} of Figure 3 leads to the answer true (i.e., the tuple <>) annotated with inconsistent tuples of the database hdb, i.e.,:*

$$incT(hdb, DC) = Q^{DC}(hdb^{LP})(<>) = \{t_1,\ t_{17},\ t_2,\ t_{18},\ t_3,\ t_{19},\ t_7,\ t_{10},\ t_6,\ t_{11},\ t_5,\ t_{13}\}$$

4.2 Annotating the initial database instance

Using the set $IncT(\mathcal{I}, DC)$ of inconsistent tuples, we convert the instance $\mathcal{I}$ into a K-*instance*, denoted by $\mathcal{I}^{\Gamma}$, with $K = \mathbb{N}[\Gamma]$ (see Definition 3). This is achieved by Algorithm 1 (lines 13 et 26) which permits to annotate each consistent tuple in $\mathcal{I}$ with the value 1 and each inconsistent tuple $id(t) \in IncT(\mathcal{I}, DC)$ with its identifier $id(t)$.

Definition 3 (Polynomial provenance instances). *Let $\mathcal{I}$ be an instance over a database schema $\mathcal{S}$ and let DC be a set of denial contraints over $\mathcal{S}$. Let $K = \mathbb{N}[\Gamma]$. The K-instance $\mathcal{I}^{\Gamma}$ is constructed as follows:*

- *$\forall R_i \in S$ a corresponding K-relation is created in $\mathcal{I}^{\Gamma}$,*
- *A K-relation $\mathcal{I}^{\Gamma}(R_i) \in \mathcal{I}^{\Gamma}$ is populated as follows:*

$$\begin{cases} \mathcal{I}^{\Gamma}(R_i)(t) = id(t) & if\ id(t) \in IncT(\mathcal{I}, DC) \\ \mathcal{I}^{\Gamma}(R_i)(t) = 1 & t \in \mathcal{I}(R_i) \wedge id(t) \notin IncT(\mathcal{I}, DC) \end{cases}$$

Algorithm 1: $\mathbb{N}[\Gamma]$-*instance* from $\mathcal{I}$ denoted $\mathcal{I}^{\Gamma}$

```
Input  : $\mathcal{I}$ : database instance,
         $DC$ : set of denials constraints
Output: $\mathcal{I}^{\Gamma}$
1  $\mathcal{I}^{LP} := \emptyset$ ;
2  for $t \in \mathcal{I}$ do
3      $\mathcal{I}^{LP} := \mathcal{I}^{LP} \cup \{(t,\ \{id(t)\}\ )\}$ ;
4  end
5  /* Convert $DC$ into $Q^{DC}$ */ ;
6  for $C \in DC$ do
7      Let C= $\leftarrow R_1(u_1) \wedge ... \wedge R_n(u_n) \wedge \phi(u_1, \ldots, u_n)$ ;
8      Generate the rule:
9          $Q^{DC}() \leftarrow R_1(u_1) \wedge ... \wedge R_n(u_n) \wedge \phi(u_1, \ldots, u_n)$
10 end
11 /* Compute the lineage of the tuple <> (answer true) */
12 $IncT := Q^{DC}(\mathcal{I}^{LP})(<>)$;
13 /* Convert $\mathcal{I}$ into the $K$-relation $\mathcal{I}^{\Gamma}$ */
14 $\mathcal{I}^{\Gamma} := \emptyset$ ;
15 for $\mathcal{I}(R) \in \mathcal{I}$ do
16     $\mathcal{I}^{\Gamma}(R) := \emptyset$ ;
17     for $t \in \mathcal{I}(R)$ do
18         if $id(t) \in IncT(\mathcal{I}, DC)$ then
19             $\mathcal{I}^{\Gamma}(R) := \mathcal{I}^{\Gamma}(R) \cup \{(t,\ id(t)\ )\}$ ;
20         else
21             $\mathcal{I}^{\Gamma}(R) := \mathcal{I}^{\Gamma}(R) \cup \{(t, 1)\}$ ;
22         end
23     end
24     $\mathcal{I}^{\Gamma} := \mathcal{I}^{\Gamma} \cup \{\mathcal{I}^{\Gamma}(R)\}$
25 end
26 return $\mathcal{I}^{\Gamma}$;
```

Example 9. *Continuing with the example, Figure 4 shows the $\mathbb{N}[\Gamma]$-instance obtained from the initial database hdb by annotating consistent tuples with 1 and inconsistent tuples in $incT(hdb, DC)$ by their tuple identifiers.*

4.3 Defining inconsistency measures for query answers

We define in this section several meseaures to determine the inconsistency degrees of query answers. Given an **UCQ** query Q, we evaluate Q over the instance $\mathcal{I}^{\Gamma}$ in order to compute the answers of Q as well as the provenance polynomials semiring annotations associated with each answer. The annotations, which come in the form of polynomial expressions, are then exploited to define several inconsistency measures for query answers.

Let $\mathcal{I}$, Q and DC be, respectively, an instance, an **UCQ** query and a set of denial constraints over a database schema $\mathcal{S}$. Let $t \in Q(\mathcal{I})$ be an answer of the query Q over the instance $\mathcal{I}$. Let $K = \mathbb{N}[\Gamma]$. Applying the query Q to the K-*instance* $\mathcal{I}^{\Gamma}$, enables to compute the provenance annotation $Q(\mathcal{I}^{\Gamma})(t) = P$ associated with each answer $t \in Q(\mathcal{I})$. This annotation consists in a polynomial expression P over the set of variables Γ. Recall that the variables that appear in

$Diagnosis(D)$

PID	RefD	Date	Prov
P01	R1	2	$\{t_1\}$
P02	R7	4	$\{t_2\}$
P03	R7	10	$\{t_3\}$
P04	R1	3	$\{t_4\}$
P04	R4	8	$\{t_5\}$
P02	R4	7	$\{t_6\}$
P01	R2	5	$\{t_7\}$
P08	R7	4	$\{t_8\}$
P10	D1	5	$\{t_9\}$

$Vaccination(V)$

PID	RefD	Date	Prov
P01	R2	3	$\{t_{10}\}$
P02	R4	5	$\{t_{11}\}$
P03	R4	6	$\{t_{12}\}$
P04	R4	7	$\{t_{13}\}$
P02	R5	10	$\{t_{14}\}$
P08	R5	7	$\{t_{15}\}$
P10	R3	8	$\{t_{16}\}$

$Surgery(S)$

PID	RefD	Date	Prov
P01	R1	1	$\{t_{17}\}$
P02	R7	3	$\{t_{18}\}$
P03	R7	9	$\{t_{19}\}$
P04	R1	4	$\{t_{20}\}$
P08	R7	5	$\{t_{21}\}$

Figure 3: The lineage provenance database hdb^{LP} obtained from the hospital database hdb.

$Diagnosis(D)$

PID	RefD	Date	Prov
P01	R1	2	t_1
P02	R7	4	t_2
P03	R7	10	t_3
P04	R1	3	1
P04	R4	8	t_5
P02	R4	7	t_6
P01	R2	5	t_7
P08	R7	4	1
P10	R1	5	1

$Vaccination(V)$

PID	RefD	Date	Prov
P01	R2	3	t_{10}
P02	R4	5	t_{11}
P03	R4	6	1
P04	R4	7	t_{13}
P02	R5	10	1
P08	R5	7	1
P10	R3	8	1

$Surgery(S)$

PID	RefD	Date	Prov
P01	R1	1	t_{17}
P02	R7	3	t_{18}
P03	R7	9	t_{19}
P04	R1	4	1
P08	R7	5	1

Figure 4: *K-instance* obtained from $\mathcal{I}$ with $K = \mathbb{N}[\Gamma]$. Each inconsistent tuple is annoted by a monome that is its identifier and an inconsistent tuple is annoted by monome 1.

P correspond to identifiers of inconsistent tuples (i.e., elements of $IncT(\mathcal{I}, DC)$). Hence, the polynomial P fully documents how inconsistent source tuples contribute in the computation of the output t. In particular, each monomial $M \in \mathcal{M}(P)$ gives an alternative way to compute the output t. Based on the polynomial annotation P of a tuple t, different measures can be defined in order to quantify the inconsistency degree of t depending in particular on how one deals with the following two issues:

- How to treat the alternative ways to compute the same query answer t? in other words, the question is to determine which monomial of P one has to consider in order to compute the inconsistency degree of t. We define two classes of measures focusing either on the lower bound, hereafter called MIN alternative, or on the upper bound, hereafter called MAX alternative, of the inconsistency degree of t.
- How to deal with a source tuple that contribute more than one time in the computation of the same query answer t? We define two classes of semantics: *set semantics*, in which a contribution of a source tuple in the computation of an answer is counted at most once, and *bag semantics*, where the exact number of contributions of a source tuple is taken into account when quantifying the inconsistency degree of a given answer.

The combination of the previous dimensions leads to the following four measures of inconsistency:

- *Set semantics.* In this semantics, the quantification of an inconsistency degree from a monomial $M \in \mathcal{M}(P)$ is achieved by counting the number of variables in M (since duplicate contributions are not counted, the exponents of the variables are dropped). The set semantics leads to two measures depending on how alternatives are dealt with:
  - MIN alternative: we consider the alternative that minimizes the number of inconsistent tuples involved in the computation of each query answer. Given an answer t to Q and its associated polynomial annotation P, this option boils down to consider the monomial M_{min} of P with the smallest number of variables. In such a case, the number of variables in M_{min} gives the inconsistency degree of t, which can be interpreted as the minimal number of inconsistent tuples that contribute in the computation of t. Hereafter, this measure is denoted by IL^s_{min}.
  - MAX alternative: we consider the alternative that maximizes the number of inconsistent tuples involved in the computation of each query answer (e.g., such a measure gives an upper bound regarding the inconsistency degree of a given answer). Given that we assume a set semantics, this option turns out to consider the monomial M_{max} of P with the largest number of variables. In such a case, the number of variables in M_{max} gives an inconsistency degree of t which can be interprested as the maximal number of consistent tuples that contribute in the computation of t. Hereafter, this measure is denoted by IL^s_{max}.
- *Bag semantics.* In this semantics, the quantification of an inconsistency degree from a monomial $M \in \mathcal{M}(P)$ is achieved by computing the weight of M. Recall that the weight of a monomial M is given by the sum of the exponents of its variables. Hence, if an inconsistent source tuple is used n times in the monomial M, it will then be counted n times in the quantification of the inconsistency degree from M. The bag semantics leads to two measures depending on how alternatives are dealt with:

y	u	
D_1	D_2	$t_1 * t_{10} * t_{17}$
D_7	D_4	$t_2 * t_{18} * t_{11} + t_3 * t_{19}$
D_1	D_4	t_{13}
D_7	D_5	$t_{18} + 1$

Answers of query with their inconsisty degrees

y	u	IL^s_{min}	IL^s_{max}	IL^b_{min}	IL^b_{max}
D_1	D_2	3	3	3	3
D_7	D_4	2	3	2	3
D_1	D_4	1	1	1	1
D_7	D_5	0	1	0	1

Figure 5: Answers of query Q_{ex} over hdb^Γ and their inconsistency degrees for each respective inconsistency measure.

– The MIN alternative enables us to consider the monomial M_{min} of P with the smallest weight. Hereafter, this measure is denoted by IL^b_{min}.
– The MAX alternative corresponds to consider the monomial M_{max} of P with the largest weight. Hereafter, this measure is denoted by IL^b_{max}.

In the following, we provide the formal definition of the four proposed inconsistency measures.

Definition 4 (Inconsistency measures). *Let $\mathcal{I}$, Q and DC be respectively, a database instance, an* UCQ *query and a set of denial constraints over a database schema $\mathcal{S}$. Let $Q(\mathcal{I}^\Gamma)(t)$ be the polynomial semiring provenance of a tuple t. We define the following four measures to quantify the degree of inconsistency of a tuple $t \in Q(I)$ w.r.t DC:*

- *Set semantics, MIN alternative*
$$\mathrm{IL}^s_{min}(t, \mathcal{I}, Q, DC) \overset{def}{=} \mathcal{V}^{mi}(Q(\mathcal{I}^\Gamma)(t))$$
- *Set semantics, MAX alternative*
$$\mathrm{IL}^s_{max}(t, \mathcal{I}, Q, DC) \overset{def}{=} \mathcal{V}^{ma}(Q(\mathcal{I}^\Gamma)(t))$$
- *Bag semantics, MIN alternative*
$$\mathrm{IL}^b_{min}(t, \mathcal{I}, Q, DC) \overset{def}{=} \mathcal{W}^{mi}(Q(\mathcal{I}^\Gamma)(t))$$
- *Bag semantics, MIN alternative*
$$\mathrm{IL}^b_{max}(t, \mathcal{I}, Q, DC) \overset{def}{=} \mathcal{W}^{ma}(Q(\mathcal{I}^\Gamma)(t))$$

Given a query Q and an instance $\mathcal{I}$ and a set of $\mathcal{DC}$ DC, $Q(\mathcal{I}_{DC})$ can be computed in polynomail time in data complexity [12, 16]. Hence, all these measures of inconsistency degree of answers tuples of query Q over $\mathcal{I}$ ($Q(\mathcal{I}_{DC})$) can be computed in polynomial in the size of the database instance $\mathcal{I}$.

Example 10. *Continuing our running example, the valuation of the query Q_{ex} of Figure 1 is processed as illustrated in Table 5.*

Given a conjunctive query Q, each measure of inconsistency degree as defined above is bounded by the size of Q (i.e, $|Body(Q)|$). Also, in the case where Q is a self-join free query the set semantics and the bag semantics coincide. The lemma 4.1 shows these two properties.

Lemma 4.1. *Let $\mathcal{I}$, Q and DC be respectivelly an instance, a* CQ *query and a set of denial constraints, for each tuples $t \in Q(\mathcal{I})$*

(1) *$\alpha(t, I, Q, IC) \leq |Body(Q)|$ with $\alpha \in \{\mathrm{IL}^s_{min}, \mathrm{IL}^b_{min}, \mathrm{IL}^s_{max}, \mathrm{IL}^b_{max}\}$*
(2) *If Q is also self-join free then $\mathrm{IL}^s_{min}(t, I, Q, IC) = \mathrm{IL}^b_{min}(t, I, Q, IC)$ and $\mathrm{IL}^s_{max}(t, I, Q, IC) = \mathrm{IL}^b_{max}(t, I, Q, IC)$*

Proof. (1) Easy. $Q : Q(X) \leftarrow P_1(X_1), ..., P_n(X_n), \phi$. Each monome M in $Q(\mathcal{I}_{IC})(t)$, $\forall t \in Q(\mathcal{I})$, is in a form $a * \Pi^n_{i=1} P_i(v(X_i))$ with v a valuation for Q and $a \in \mathbb{N}$. So, $|Var(M)| \leq |W(M)| \leq n = |Body(Q)|$.
(2) $Q : Q(X) \leftarrow P_1(X_1), ..., P_n(X_n), \phi$ a CQ free self-join $\Rightarrow$ there is no valuation v of Q where $id(v(X_i)) = id(id(X_j))$ with $P_i \neq P_j$, so each monome in $Q(\mathcal{I}_{IC})(t)$ is in form $a \times x_1 \times \ldots \times x_m$ with $a \in \mathbb{N}$ and $\{x_1, ..., x_n\}$ is the set of variables (identifiers of inconsistent tuples). There is no repetition, thus the bag semantics and set semantics match one with another.

□

In the next section, we show how to use inconsistency degrees in order to filter and rank query answers.

5 INCONSISTENCY-AWARE QUERY ANSWERING

Querying inconsistent databases might lead to cope with the inconsistencies in the query answers. This occurs in many applications where inconsistencies cannot be repaired due to data protection and data ownership requirements. In this section, we rely on the consistency measures introduced above, in order to define problems related to query answering for inconsistent databases. The main idea is to compute query answers while taking into account their inconsistency degrees to filter the answers, for instance for bounding query answers with a given threshold or for selecting the top-k query answers.

Definition 5 (Threshold-based filtering). *Let $\mathcal{I}$, Q and DC be respectively an instance, an $\mathcal{UCQ}$ query and a set of denial constraints over a database schema $\mathcal{S}$. Let δ be an integer and let $\alpha \in \{\mathrm{IL}^s_{min}, \mathrm{IL}^s_{max}, \mathrm{IL}^b_{min}, \mathrm{IL}^b_{max}\}$ then*

(1) *The less than δ answers of Q (called* **LA** *answers) over $\mathcal{I}$ w.r.t DC denoted $Q^{\delta^-,\alpha}$ is defined as follows:*
$Q^{\delta^-,\alpha}(\mathcal{I}) \overset{def}{=} \{ t \in Q(\mathcal{I}) : \alpha(t, \mathcal{I}, Q, DC) < \delta \}$
(2) *The greater than δ answers (called* **GA** *answers) of Q over I w.r.t DC denoted $Q^{\delta^+,\alpha}$ is defined as follows:*
$Q^{\delta^+,\alpha}(\mathcal{I}) \overset{def}{=} \{ t \in Q(\mathcal{I}) : \alpha(t, q, \mathcal{I}, DC) > \delta \}$
(3) *The equal to δ answers (called* **EA** *answers) of Q over $\mathcal{I}$ w.r.t DC denoted $Q^{\delta^=,\alpha}$ is defined as follows:*
$Q^{\delta^=,\alpha}(\mathcal{I}) = \{ t \in Q(\mathcal{I}) : \alpha(t, q, \mathcal{I}, DC) = \delta \}$

According to this definition, **LA** answers of a query Q w.r.t. an inconsistency measure α and a treshold δ, consists in all answers of Q that have an inconsistent degree less than δ using the inconsistency measure α. The **GA** (respectively, **EA**) answers of Q are those answers of Q that have an inconsistent degree greater than (respectively, equal to) δ using the inconsistency measure α.

We shall define below top-k queries w.r.t. inconsistency degrees.

Definition 6 (Top-k queries). *Let $\mathcal{I}$, Q and DC be respectively an instance, an $\mathcal{UCQ}$ query and a set of denial constraints over* *a database schema $\mathcal{S}$. Let k be an integer and let $\alpha \in \{\text{IL}^s_{min}, \text{IL}^s_{max}, \text{IL}^b_{min}, \text{IL}^b_{max}\}$. The top-k query answers of Q over $\mathcal{I}$ w.r.t DC and using the inconsistency measure α, denoted by $Q^{top_k,\alpha}(\mathcal{I})$, is defined as follows:*

- $Q^{top_k,\alpha}(\mathcal{I}) \subseteq Q(\mathcal{I})$, *and*
- $|Q^{top_k,\alpha}(\mathcal{I})| = Min(k, |Q(\mathcal{I})|)$, *and*
- $\forall (t_1, t_2) \in Q^{top_k,\alpha}(\mathcal{I}) \times Q(\mathcal{I})$ *then* $\alpha(t_1, Q, \mathcal{I}, DC) \leq \alpha(t_2, Q, \mathcal{I}, DC)$

The following example illustrates some threshold-base filetering and top-k queries over our example database.

Example 11. *Take $\delta = 2$ and $k = 3$. Continuing with the database example hdb and the query Q_{ex} of figure 1, we show below the results of some threshold-based queries and top-k queries.*

- $Q^{top-3,\text{IL}^s_{min}}(hdb) = Q^{top-3,\text{IL}^s_{max}}(hdb) = Q^{top-3,\text{IL}^b_{min}}(hdb) = Q^{top-3,\text{IL}^b_{max}}(hdb) = \{(D_7, D_5), (D_1, D_4), (D_7, D_4)\}$.
- $Q^{\delta^-,\text{IL}^s_{min}}(hdb) = Q^{\delta^-,\text{IL}^s_{max}}(hdb) = Q^{\delta^-,\text{IL}^b_{min}}(hdb) = Q^{\delta^-,\text{IL}^b_{max}}(hdb) = \{(D_1, D_4), (D_7, D_5)\}$.
- $Q^{\delta^+,\text{IL}^s_{min}}(hdb) = Q^{\delta^+,\text{IL}^b_{min}}(hdb) = \{(D_1, D_2)\}$.
- $Q^{\delta^+,\text{IL}^s_{max}}(hdb) = Q^{\delta^+,\text{IL}^b_{max}}(hdb) = \{(D_1, D_2), (D_7, D_4)\}$.
- $Q^{\delta^=,\text{IL}^s_{min}}(hdb) = Q^{\delta^=,\text{IL}^b_{min}}(hdb) = \{(D_7, D_4)\}$.
- $Q^{\delta^=,\text{IL}^s_{max}}(hdb) = Q^{\delta^=,\text{IL}^b_{max}}(hdb) = \emptyset$.

Below, we describe two preliminary approaches to compute the solutions to each of the problems defined above.

5.1 Full-materialized approach

Let $\mathcal{I}$, Q and DC be respectively an instance, an $\mathcal{UCQ}$ query and a set of denial constraints over a database schema $\mathcal{S}$. Let δ and k be two integers and let $\alpha \in \{\text{IL}^s_{min}, \text{IL}^s_{max}, \text{IL}^b_{min}, \text{IL}^b_{max}\}$. The simplest approach to compute the threshold-based and top-k query answers of Q is to compute beforehand all the tuples in the answers of Q together with their associated provenance semiring annotations (i.e., the materialization of the relation $Q(\mathcal{I}^\Gamma)$). Then, the answers of Q can be filtered using an α measure in order to compute **LA**, **GA** or **EA** answers. Top-k queries can also be easily handled, e.g., by sorting the answers w.r.t. inconsistency degrees. This approach is called full-materialized approach because it requires the materialization of the *K-relation* $Q(\mathcal{I}^\Gamma)$.

The shortcoming of this approach is the fact that we have to materialize all the *K-relation* $Q(\mathcal{I}^\Gamma)$ before filtering. Hence, the time and space complexity of computing threshold-based and top-k queries is a function of the size of query results. For the bounded query answering, we can do filtering in linear time in the size of query results since we have only to compare the inconsistency degree with the threshold δ. However, for top-k query answering, we have to sort the answers of a query thus obtaining a complexity in $n * log(n)$, with $n = |Q(\mathcal{I})|$ being the size of the query answers.

5.2 Semi-materialized approach

In this section, we propose a second approach to deal with threshold-based filtering and top-k queries. Let $\mathcal{I}$, Q and DC be respectively an instance, an $\mathcal{UCQ}$ query and a set of denial constraints over a database schema $\mathcal{S}$. Let δ and k be two integers and let $\alpha \in \{\text{IL}^s_{min}, \text{IL}^s_{max}, \text{IL}^b_{min}, \text{IL}^b_{max}\}$. We compute the set of inconsistent tuples $IncT(\mathcal{I}, DC)$ and then use it to annotate each tuple in $\mathcal{I}$ with the value 1 if t is inconsistent (i.e., $id(t) \in IncT(\mathcal{I}, DC)$) and with the value 0 if t is consistent. The result, a $\{0, 1\}$-*instance* denoted by $\mathcal{I}^b$, is materialized and used to compute inconsistency-aware answers to user queries. This second approach is called *semi-materialized* because only the $\{0, 1\}$-*instance* $\mathcal{I}^b$ is precomputed and stored beforehand while the entire set of query answers is not materialized in advance as in the previous approach.

In this setting, we see each inconsistency measure $\alpha \in \{\text{IL}^s_{min}, \text{IL}^s_{max}, \text{IL}^b_{min}, \text{IL}^b_{max}\}$ as a scoring function over the annotations of the tuples in $\mathcal{I}^b$. Then, given a query Q, we consider the generic enumeration problem of *how to efficiently enumerate* the answers of Q according to an order defined over their inconsistency degrees. We investigate this problem in the enumeration framework defined in [25], where an algorithm can be decomposed into two phases:

- a preprocessing phase that is performed in time linear in the size of the database,
- an enumeration phase that outputs $Q(\mathcal{I})$ with no repetition and a delay depending only on Q between any two consecutive outputs. The enumeration phase has full read access to the output of the preprocessing phase and can use extra memory whose size depends only on Q.

We adapt the approach developped in [14] for enumerating the results of conjunctive queries to our setting. [14] proposes an efficient ranked enumeration algorithm for conjunctive query results that combines the use of priority queues and hash maps in conjunction with query decomposition techniques. The proposed algorithm starts with a preprocessing step and builds priority queues that maintain partial tuples at each node of the query decomposition tree. In second step, namely the enumeration phase, the algorithm computes the output of the subquery formed by subtree rooted at each node of the decomposition in sorted order according to the ranking function. In order to allow partial aggregation of tuple scores during the enumeration phase, the proposed approach requires a ranking function that satisfies a decomposability and a compatibility property w.r.t. a tree decomposition of the considered query. We recall below the classical notion of tree decompostion of a conjunctive query as well as the decomposability and compatibility properties introduced in [14].

Tree decomposition of a conjunctive query. A natural join is a full CQ, with no constants and no repeated variables in the same atom. A natural join can be represented as a hypergraph $H_Q = (V_Q, E_Q)$, where V_Q is the set of variables that appear in Q, and for each hyperedge $E \in E_Q$ there exists a relation R_F with variables F. A tree decomposition of a natural join Q is a tuple $(T, (B_n)_{n \in V(T)})$ where T is a tree, and every B_n is a subset of V, called the bag of n, such that: *(i)* each edge in E is contained in some bag; and *(ii)* for each variable $x \in V$, the set of nodes $\{n | x \in B_n\}$ is connected in T. Given a rooted tree decomposition, the notation $p(n)$ is used to denote the parent of a node $n \in V(T)$ and the notation $key(n) = B_n \cap B_{p(n)}$ denotes the common variables that are shared between the bage B_n and its parent $B_{p(n)}$. Note that, results of [14] applies to natural join queries that have bounded *fractional hypertree width* (fhw) decompositions [23].

Decomposable ranking functions. Let f be a ranking function over a set of variables Y. Let $X \subseteq Y$. We say that f is X-decomposable if there exists a valuation φ^* over $Y \setminus X$, such that for every valuation φ over $Y \setminus X$, and any two valuations θ_1, θ_2 over X we have: $f(\varphi^* \circ \theta_1) \geq f(\varphi^* \circ \theta_2) \Rightarrow f(\varphi \circ \theta_1) \geq f(\varphi \circ \theta_2)$, where $\circ$ denotes extension of two valuations to form an unique valuation.
Compatible ranking functions. Let f be a ranking function over a set of variables V, and $X, Y \subseteq V$ such that $X \cap Y = \emptyset$. We say that f is Y-decomposable conditioned on X if for every valuation θ over X, the function $f_\theta(\varphi) = f(\theta \circ \varphi)$ defined over $V \setminus X$ is Y-decomposable. A ranking function f is *compatible* with a tree T of a query Q if for every node n it is $(B_n^{\prec} \setminus key(n))$-decomposable conditioned on $key(t)$, where $B_n^{\prec}$ denotes the union of all bags in the subtree rooted at n (including B_n).
Ranked enumeration of conjunctive query results. Given a natural join CQ, an instance $\mathcal{I}$ and a ranking function f compatible with to tree associated to Q, then Shaleen et al. [14] generates, in polynomial time and polynomial space in data complexity, a specific data structure associated with each node of the tree of Q. The obtained tree is used to enumerate each tuple t in the answers of a query Q over instance $\mathcal{I}$, with complexity $p * log(|\mathcal{I}|)$ in data complexity where p is the rank of t. The valuations are organized into a hashmap structure where the keys are valuations of common variables of node with its parent in order to not loose the join condition. By key (one valuation of common variable with nœud's parent) of the hashmap, we can have many such valuations. Moreover, these valuations are organized in a priority queue (with a top value that is always available in $O(1)$), and thanks to the fact that the ranking function is compatible with a tree, we can locally choose the head (minimum) of this queue and aggregate using this ranking function along the tree. For more details about this technique, we refer the reader to [14].
The hashmap of the root node contains one element (priority queue) having as key an empty tuple (<>) because the root has no parent. At each enumeration, the top (with minimum rank) in the priority queue of root node is chosen with its children recursively in the tree. After each enumeration, the structure is updated. During updating, the top valuation is removed and recursively its children also are removed. Some children of the top valuation previously chosen can be involved in other valuations. For this reason, before removal of a child c the valuation after c in the same priority queue is associated to c (field 'next' is used).
Enumerating query answers w.r.t. their inconsistency degrees. We first observe that under the bag semantics, the inconsistency measures proposed in this paper are compatible with a tree decomposition of a full natural join, hence the results of [14] can be easily translated to our context to enable efficient enumeration of query results w.r.t. inconsistency degrees. However, under the set semantics, our inconsistency measures turn to be non decomposable and hence cannot be used in the context of the algorithm of [14]. We shall show below that this limitation can be lifted in the case of self-join free queries and hence, for this class of queries, we can exploit the result of [14] using our inconsistency measures both under bag and set semantics. To achieve this task, we first show that for full conjunctive queries, using polynomial semirings leads to an annotation of query answers by monomials.

Lemma 5.1. *Given a full CQ Q, an instance $\mathcal{I}$ and a set of denial constraints DC. Using the polynomial semiring, for any tuple $t \in Q(I)$ then $Q(\mathcal{I}^\Gamma)(t)$ is a monomial.*

Proof. As Q is full (i.e all variables are in head of Q), for each of two different valuations v_1 and v_2 of Q (i.e, there exists at least one variable $x \in Var(Q)$ where $v_1(x) \neq v_2(x)$) $v_1(head(Q)) \neq v_2(head(Q))$. Each tuple in the answers of query Q over $\mathcal{I}_{IC}$ is associated with one valuation, hence its polynomial provenance is a monomial. □

As one can note if $Q(\mathcal{I}_{IC})(t)$ is a monomial then $\mathbb{IL}^s_{min}(t, \mathcal{I}, Q, IC) = \mathbb{IL}^s_{max}(t, \mathcal{I}, Q, IC)$ and $\mathbb{IL}^b_{min}(t, \mathcal{I}, Q, IC) = \mathbb{IL}^b_{max}(t, \mathcal{I}, Q, IC)$. The reason is that since there is one possible monomial to choose, this makes the maximum and the minimum coincide.

Lemma 5.2 states that for full self-join free conjunctive query, the different inconsistency measures coincide.

Lemma 5.2. *Given a full self-join free CQ Q, an instance $\mathcal{I}$ and a set of denial constraints IC. For any tuple $t \in Q(I)$ then $\mathbb{IL}^s_{min}(t, \mathcal{I}, Q, IC) = \mathbb{IL}^s_{max}(t, \mathcal{I}, Q, IC) = \mathbb{IL}^b_{min}(t, \mathcal{I}, Q, IC) = \mathbb{IL}^b_{max}(t, \mathcal{I}, Q, IC)$.*

Proof. By lemma 5.1, we know that the polynomial provenance of a tuple t in answers of full query Q is a monomial so $\mathbb{IL}^s_{max}(t, \mathcal{I}, Q, IC) = \mathbb{IL}^s_{min}(t, \mathcal{I}, Q, IC)$ and $\mathbb{IL}^b_{max}(t, \mathcal{I}, Q, IC) = \mathbb{IL}^s_{min}(t, \mathcal{I}, Q, IC)$. Q is self-join free, there is no repeating in tuples used in computing of each tuple in answers of Q, so the bag semantic and set semantic coincide (c.f., lemma 4.1). As a consequence $\mathbb{IL}^s_{min}(t, \mathcal{I}, Q, IC) = \mathbb{IL}^s_{max}(t, \mathcal{I}, Q, IC) = \mathbb{IL}^b_{min}(t, \mathcal{I}, Q, IC) = \mathbb{IL}^b_{max}(t, \mathcal{I}, Q, IC)$ □

We show in the following theorem that the query answers in our inconsistency-aware framework can be enumerated with logarithmic delay.

Theorem 5.3. *Given a natural join CQ Q with bounded fhw decompositions, an instance $\mathcal{I}$ and a set of denial constraints IC, after a preprocessing of Q over $\mathcal{I}$ in polynomial time in $|\mathcal{I}|$:*

(1) *$Q^{top_k, \mathbb{IL}^b_{max}}(\mathcal{I})$ and $Q^{top_k, \mathbb{IL}^b_{max}}(\mathcal{I})$ can be enumerated with delay $O(\log(|\mathcal{I}|))$ between their ordered tuples.*
(2) *If Q is also self-join free, $Q^{top_k, \mathbb{IL}^s_{max}}(\mathcal{I})$ and $Q^{top_k, \mathbb{IL}^s_{min}}(\mathcal{I})$ can be enumerated with delay $O(\log(|\mathcal{I}|))$ between their ordered tuples.*

Proof. (1) The natural join CQ Q is $Q(X) \leftarrow P_(X_1), \ldots, P_n(X_n)$. Let $C : \mathcal{I} \rightarrow \mathbb{R}$ be a function that assoicates to each tuple t in $\mathcal{I}$ a cost $C(t)$ as follows: if $t \in IncT(\mathcal{I}, IC)$ then $C(t) = 1$ Otherwise $C(t) = 0$.
Q is a natural join query $\Rightarrow \forall t \in \mathcal{I}, Q(\mathcal{I}_{IC})(t) = x_1^{n_1} * \ldots * x_k^{n_k}$ (by lemma 5.1) and $n_1 + \ldots + n_k = \mathbb{IL}^b_{max}(t, \mathcal{I}, Q, IC) = \mathbb{IL}^b_{min}(t, \mathcal{I}, Q, IC) \leq |Body(Q)|$ (by lemma 4.1) with $\{x_1, \ldots, x_n\} \subseteq \{id(v(X_1)), \ldots, id(v(X_n))\}$, $v(X) = t$ and v a valuation of Q over $\mathcal{I}$. Then $\mathbb{IL}^b_{min}(t, \mathcal{I}, Q, IC) = \mathbb{IL}^b_{max}(t, \mathcal{I}, Q, IC) = n_1 * C(t_1) + \ldots + n_k * C(t_k)$ with $id(t_i) = x_i, \forall i \in \{1, \ldots, n\}$. Also we have, $n_1 * C(t_1) + \ldots +$

$n_k * C(t_k) = C(v(X_1)) + \ldots + C(v(X_n)) = \sum_{i=1}^{n} C(v(X_i))$. So, $\mathrm{IL}^b_{min}(t, \mathcal{I}, Q, IC) = \mathrm{IL}^b_{max}(t, \mathcal{I}, Q, IC) = \sum_{i=1}^{n} C(v(X_i))$ and the ranking function to use is $r = \sum_{i=1}^{n} C(v(X_i))$ that they call in [14] tuple-based ranking function. In [14], in lemma 2, they show that r is compatible with any tree decomposition of a natural join CQ query. As a consequence, using a technique in [14], $Q^{top_k, \mathrm{IL}^b_{max}}(\mathcal{I})$ and $Q^{top_k, \mathrm{IL}^b_{min}}(\mathcal{I})$ can be enumerated in delay $O(log|\mathcal{I}|)$ between their ranked tuples with a prepocessing of $\mathcal{I}$ in polynomial time in data complexity.

(2) By lemma 5.2 and (1) of Theorem 5.3.

□

It is easy to verify that the results of Theorem 5.3 extend to threshold-based filtering problems under the same restrictions on the considered queries.

EXAMPLE 12. *Consider an instance as in Figure* 1*, where each tuple has an inconsistency degree equal to* 1 *if the tuple is inconsistent and* 0 *otherwise (the column with blue color), and consider a natural join conjunctive query:*

$Q(x, y, z, z', z_2) \leftarrow D(x, y, z),\ S(x, y, z'),\ V(x, u, v)$

The id column serves the need of identifying the tuples. A decomposition tree of Q and the data structure associated to each of its node is shown in Figure 6. In this example, the ranking function is the sum of the degrees of the corresponding valuations. Figure 7 shows the first enumerated tuple as well as the content of the tree after this first enumeration.

6 CONCLUSION

We have presented a novel framework for inconsistency-aware query answering. We have defined inconsistency degrees of query answers by leveraging four different measures of inconsistency. We have grounded the computation of inconsistency degrees in why-provenance annotations and used the polynomial semiring provenance in order to define top-k and bounded query answering in this setting. To this end, we have designed two algorithms (the Full-materialized and Semi-materialized algorithms) that actually carry out the above computation with different space requirements.

As future work, we plan to further explore the optimized computation of top-k and bounded query answering and also see its impact and performance in practice by means of an extensive experimental study. It will also be fruitful to check whether a cost-based computation is possible in this setting and to tune the latter computation in order to fit the cost computation of relational query plans, in the spirit of top-k query processing in relational databases.

REFERENCES

[1] John R. Smith Chung-Sheng Li Apostol Natsev, Yuan-Chi Chang and Jeffrey Scott Vitter. [n. d.]. Supporting Incremental Join Queries on Ranked Inputs. *VLDB '01 Proceedings of the 27th International Conference on Very Large Data Bases*, 281–290.

[2] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent query answers in inconsistent databases. In *the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 68–79.

[3] Abdallah Arioua and Angela Bonifati. 2018. User-guided Repairing of Inconsistent Knowledge Bases. In *Proceedings of the 21th International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018*. 133–144.

[4] Leopoldo Bertossi. 2011. *Database Repairing and Consistent Query Answering*. Morgan & Claypool.

[5] Leopoldo Bertossi, Anthony Hunter, and Torsten Schaub. 2005. Introduction to Inconsistency Tolerance. *Inconsistency Tolerance* 3300, 1–14.

[6] Leopoldo E. Bertossi. 2018. Repair-Based Degrees of Database Inconsistency: Computation and Complexity. *CoRR* abs/1809.10286 (2018). http://arxiv.org/abs/1809.10286

[7] L. E. Bertossi and J. Chomicki. 2013. Query answering in inconsistent databases. In *In Logics for Emerging Applications of Databases*, R. van der Meyden J. Chomicki and G. Saake (Eds.).

[8] Andrea Calì, Domenico Lembo, and Riccardo Rosati. 2003. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *Proceedings of the Twenty-second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '03)*. 260–271.

[9] Jan Chomicki, Jerzy Marcinkowski, and Slawomir Staworko. 2004. Computing Consistent Query Answers Using Conflict Hypergraphs. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management (CIKM '04)*. 417–426.

[10] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering Denial Constraints. *PVLDB* 6, 13 (2013), 1498–1509.

[11] J Lang D Dubois and H Prade. [n. d.]. Possibilistic logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming* (1994). 439–513. Oxford University Press.

[12] Sudeepa Roy Daniel Deutch, Tova Milo and Val Tannen. 2014. Circuits for Datalog Provenance. In *17th International Conference on Database Theory (ICDT), At Athens, Greece*. https://doi.org/10.5441/002/icdt.2014.22

[13] Hendrik Decker and Davide Martinenghi. 2009. Modeling, Measuring and Monitoring the Quality of Information. In *ER 2009 Workshops (CoMoL, ETheCoM, FP-UML, MOST-ONISW, QoIS, RIGiM, SeCoGIS) on Advances in Conceptual Modeling - Challenging Perspectives*. ACM.

[14] Shaleen Deep and Paraschos Koutris. 2019. Ranked Enumeration of Conjunctive Query Results. (November 2019). arXiv:1902.02698.

[15] Todd J. Green. 2009. Containment of conjunctive queries on annotated relations. In *ICDT '09 Proceedings of the 12th International Conference on Database Theory*. ACM, 296–309.

[16] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance Semirings. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 31–40.

[17] Ihab F. Ilyas Ihab F. Ilyas and Ihab F. Ilyas. [n. d.]. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)* ([n. d.]).

[18] Walid G. Aref Ihab F. Ilyas and Ahmed K. Elmagarmid. [n. d.]. Supporting top-k join queries in relational databases. *The VLDB Journal âĂŤ The International Journal on Very Large Data Bases* ([n. d.]), 207–221. https://doi.org/10.1007/s00778-004-0128-2

[19] Anthony Hunter John Grant. 2005. Measuring inconsistency in knowledgebases. *Journal of Intelligent Information Systems* (2005).

[20] Solmaz Kolahi and Laks V. S. Lakshmanan. 2009. On Approximating Optimum Repairs for Functional Dependency Violations. In *Proceedings of the 12th International Conference on Database Theory (ICDT '09)*. 53–62.

[21] Jérôme Lang and Pierre Marquis. 2010. Reasoning under inconsistency: A forgetting-based approach. *Artif. Intell.* 174, 12-13 (2010), 799–823.

[22] ELIEZER L. LOZINSKII. [n. d.]. Information and evidence in logic systems. *Journal of Experimental and Theo- retical Artificial Intelligence* ([n. d.]), 163–193.

[23] Dániel Marx. 2017. Graphs, Hypergraphs, and the Complexity of Conjunctive Database Queries (Invited Talk). In *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*. 2:1–2:1.

[24] Dany Maslowski and Jef Wijsen. 2013. A dichotomy in the complexity of counting database repairs. *J. Comput. System Sci.* (2013).

[25] Luc Segoufin. 2015. Constant Delay Enumeration for Conjunctive Queries. *SIGMOD Rec.* 44, 1 (May 2015), 10–17.

[26] Matthias Thimm. 2018. *On the evaluation of inconsistency measures*. Jhon. Grant and Maria Vania Martinez, editors. 19–60 pages. College Publications.

[27] JEF WIJSEN. 2005. Database Repairing Using Updates. *ACM Transactions on Database Systems (TODS)* (2005).

Diagnosis(D)

PID	RefD	Date		id
P01	R1	2	1	t_1
P02	R7	4	1	t_2
P03	R7	10	1	t_3
P04	R1	3	0	t_4
P04	R4	8	1	t_5
P02	R4	7	1	t_6
P01	R2	5	1	t_7
P08	R7	4	0	t_8
P10	R1	5	0	t_9

Vaccination(V)

PID	RefD	Date		id
P01	R2	3	1	t_{10}
P02	R4	5	1	t_{11}
P03	R4	6	0	t_{12}
P04	R4	7	1	t_{13}
P02	R5	10	0	t_{14}
P08	R5	7	0	t_{15}
P10	R3	8	0	t_{16}

Surgery(S)

PID	RefD	Date		id
P01	R1	1	1	t_{17}
P02	R7	3	1	t_{18}
P03	R7	9	1	t_{19}
P04	R1	4	0	t_{20}
P08	R7	5	0	t_{21}

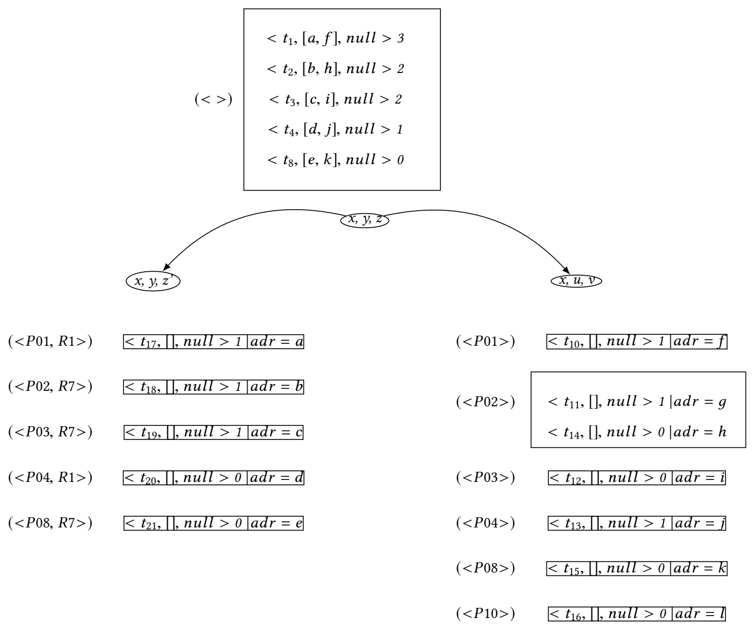
(< >)
< t1, [a, f], null > 3
< t2, [b, h], null > 2
< t3, [c, i], null > 2
< t4, [d, j], null > 1
< t8, [e, k], null > 0
x, y, z
x, y, z'
x, u, v
(<P01, R1>) < t17, [], null > 1 |adr = a
(<P02, R7>) < t18, [], null > 1 |adr = b
(<P03, R7>) < t19, [], null > 1 |adr = c
(<P04, R1>) < t20, [], null > 0 |adr = d
(<P08, R7>) < t21, [], null > 0 |adr = e
(<P01>) < t10, [], null > 1 |adr = f
(<P02>) < t11, [], null > 1 |adr = g
< t14, [], null > 0 |adr = h
(<P03>) < t12, [], null > 0 |adr = i
(<P04>) < t13, [], null > 1 |adr = j
(<P08>) < t15, [], null > 0 |adr = k
(<P10>) < t16, [], null > 0 |adr = l

Figure 6: An example of instance and a query tree for Example 12. The values in parentheses correspond to key of the hashmap and each corresponding value (values in rectangle) is a priority key. The keys of a hashmap of a node are evaluation of common variables of this node and its parent.

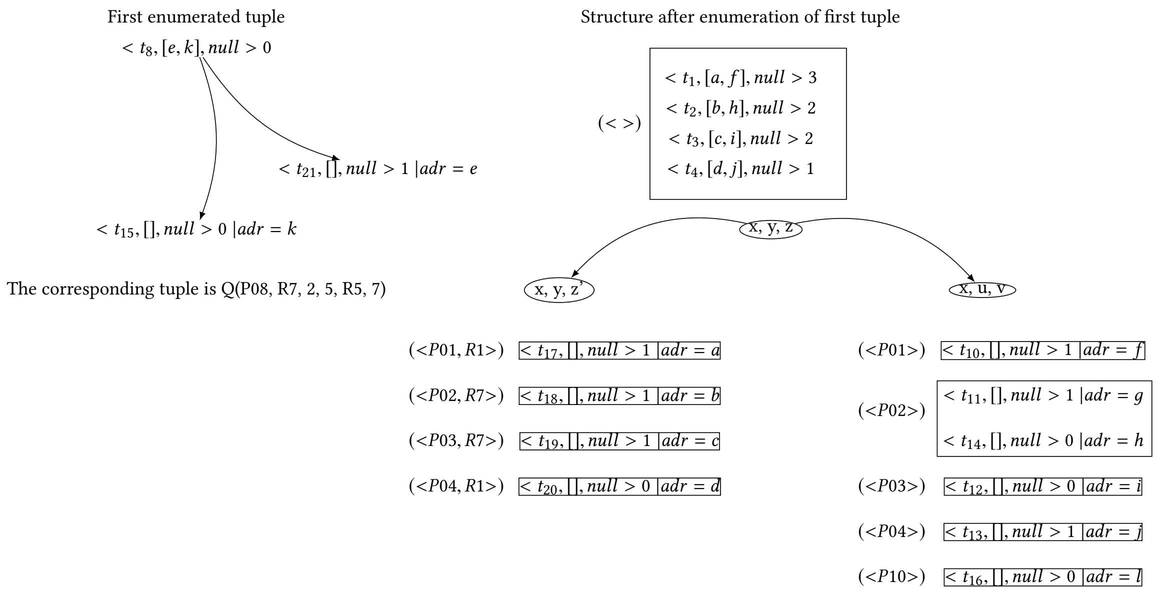
First enumerated tuple
< t8, [e, k], null > 0
< t21, [], null > 1 |adr = e
< t15, [], null > 0 |adr = k
The corresponding tuple is Q(P08, R7, 2, 5, R5, 7)
Structure after enumeration of first tuple
(< >)
< t1, [a, f], null > 3
< t2, [b, h], null > 2
< t3, [c, i], null > 2
< t4, [d, j], null > 1
x, y, z
x, y, z'
x, u, v
(<P01, R1>) < t17, [], null > 1 |adr = a
(<P02, R7>) < t18, [], null > 1 |adr = b
(<P03, R7>) < t19, [], null > 1 |adr = c
(<P04, R1>) < t20, [], null > 0 |adr = d
(<P01>) < t10, [], null > 1 |adr = f
(<P02>) < t11, [], null > 1 |adr = g
< t14, [], null > 0 |adr = h
(<P03>) < t12, [], null > 0 |adr = i
(<P04>) < t13, [], null > 1 |adr = j
(<P10>) < t16, [], null > 0 |adr = l

Figure 7: The first enumerated tuple for Example 12 (left) and the query tree after the first enumeration (right)