Provenance Analysis for Missing Answers and Integrity Repairs

Jane Xu, Waley Zhang, Abdussalam Alawini, and Val Tannen
Dept. Computer and Information Science
University of Pennsylvania
{xuyuan, wzha, alawini, val}@seas.upenn.edu

Abstract

Data provenance approaches track how the answer to a database query derive from input items; however, prior approaches used "positive" provenance and were not directly usable for explaining "expected" but missing answers. A similar problem arises with the failure of integrity constraints. Our perspective is to offer explanations via possible (minimal) repairs using provenance. This is useful for debugging, repairing, and cleaning databases. In this paper, we introduce a novel approach to this problem for both missing/erroneous answers and integrity failures. The approach uses recent advances in provenance for first-order model checking.

1 Introduction

When a user submits a query to a relational database, she often expects certain answers to appear in her query result. Sometimes, some answers may be missing or erroneous. For example, a user queries her university's database to list all courses offered in the Fall semester. However, she is not able to find her advisors course in the result, even though she knows that he will be teaching a course in the fall. Our perspective, related to *causality* [21], assumes that it is useful to provide the user with explanations via repairs—a set of *delete* and *insert* commands that would modify the database instance so that the query returns the expected answers.

Another important problem is that of integrity constraints (e.g., conditional functional dependencies, conditional inclusion constraints, etc., see [10]) failing in databases obtained via data integration/warehousing or via mining/crawling large amounts of information. Here, *repairs* can be similarly useful for explaining why integrity constraints fail. Applying such repairs belongs to the field of *data cleaning* (see below).

In this paper we present, principally via examples, a methodology for addressing explanations and repairs for both missing/erroneous answers and failure of integrity constraints. The commonality is based on both being expressible in (unrestricted) first-order logic (FOL). This approach has become possible after recent work by the last author and Erich Gräedel [13] (see Section 2).

The problem of missing answers has been studied both in the context of relational databases e.g. [3, 15, 21] (also known as why-not provenance and provenance of non-answers), and Datalog-defined computer networks [51, 52] (known there as negative provenance). Chapman and Jagadish [7] provided a model and definitions for

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

describing a data item that is not in the result set, and for asking why it does not appear in the result. Huang et al. in [18] developed a framework and algorithm for calculating the provenance of a single non-answer to a query. Herschel and Hernandez in [17] further expanded their work by having an algorithm that calculated why-not provenance for a set of missing answers including those for aggregation queries. In [16], Herschel and Hernandez continued expanding their model by providing support for queries that include selection, projection, join, union, aggregation and grouping (SPJUA). The larger context includes work on reverse query processing [20], the work of Meliou and Suciu on reverse data management [22, 23] and perhaps view update [9]. While steady progress has been made for positive queries, negation has always presented a challenge. Although we do not address aggregation, our work deals with general first-order queries (equivalent to the full relational algebra) while exploiting a particular way of expressing them: in stratified non-recursive Datalog with negation.

Database research related to integrity constraint repairs include the work of Arenas, Chomicki and Bertossi on consistent query answering [1, 2, 8]. Attribute-based repairs—repairs that only update some of the values of tuples' attributes in the database—have been studied by Bohannon et al. in [4]. A comprehensive treatment of the problem of data cleaning has been initiated by Fan et al. in [11,12] with a focus on conditional dependencies. Kolahi and Lakshmanan in [19] showed that the problem of computing a minimal repair on databases that violate functional dependencies is NP-complete. Our discussion of repair costs (see Section 5) contributes to the line of work on *weighted repairs* in [5, 6].

Previous work on provenance (see [14] for a survey) used a framework based on commutative semirings for both expressing provenance (as polynomials) and applications such as cost, confidence, or access control. Following [13] (see Section 2) we use a semiring of *dual* polynomials with two kinds of indeterminates (provenance tokens), for positive and for negative facts to compute provenance of FOL sentences being checked in an FOL model. Next we compute explanations/repairs by solving the equation that makes such dual polynomials equal to the (semiring) 0. A solution/repair consists of a set of provenance tokens being made 0. We can then interpret these provenance tokens in semirings such as the Viterbi or tropical ones for cost or confidence associated with the repairs.

We regard this work as preliminary since it still relies on the a "closed world assumption" (CWA) to compute provenance in terms of negative tokens. This means that we only consider values that occur in the instance. We hope to expand our understanding of database repairs to a"open world assumption" (OWA) in the future via a more flexible approach that refer to data that we might need to insert into the database using labeled nulls.

This paper is organized as following. We start by reviewing the basic facts about computing provenance for FOL sentences and about dual polynomials (Section 2). Next, we show a couple of examples of repairs to missing answers and we describe a general algorithm for computing such repairs (Section 3). In Section 4, we give examples of repairs to a denial constraint and a tuple-generating constraint that goes beyond what has been studied in the literature so far. We discuss selecting optimal cost repairs in Section 5.

2 Dual Polynomials

Consider a finite relational vocabulary, $\mathscr{V}$.[1] From this vocabulary and a finite *non-empty* set A of *ground values* we construct the set Facts_A of all ground relational atoms (facts) $R(\mathbf{a})$, the set $\mathsf{NegFacts}_A$ of all negated facts $\neg R(\mathbf{a})$ and thus the set $\mathsf{Lit}_A = \mathsf{Facts}_A \cup \mathsf{NegFacts}_A$ of all *literals*, positive and negative facts, over $\mathscr{V}$ and A. By convention we will identify $\neg\neg R(\mathbf{a}) \equiv R(\mathbf{a})$) so the negation of a literal is again a literal.

Let $(K, +, \cdot, 0, 1)$ be a commutative semiring. Very roughly speaking, $0 \in K$ is intended to interpret false assertions, while an element $a \neq 0$ in K provides a "nuanced" interpretation for true assertions (call them

[1] For simplicity we omit constants from definitions, but we may make use of them in (counter)examples.

"a-true").

Next, K-interpretations will map literals to elements of K and are then extended to all formulae. Disjunction and existential quantification are interpreted by the addition operation of K. Conjunction and universal quantification are interpreted by the multiplication operation of K. For quantifiers, the finiteness of the universe A of ground values will be essential. For negation we use the well-known syntactic transformation to *negation normal form (NNF)*, denoted $\psi \mapsto \mathsf{nnf}(\psi)$. Note that $\mathsf{nnf}(\psi)$ is a formula constructed from literals (positive and negative facts) and equality/inequality atoms using just $\wedge, \vee, \exists, \forall$.

Definition 1: A **K-interpretation** is a mapping $\pi : \mathsf{Lit}_A \to K$. This is extended to FO formula given valuations $\nu : \mathsf{Vars} \to A$:

$$\pi[\![R(\mathbf{x})]\!]_\nu = \pi(R(\nu(\mathbf{x})) \qquad \pi[\![\neg R(\mathbf{x})]\!]_\nu = \pi(\neg R(\nu(\mathbf{x}))$$

$$\pi[\![x \ \mathsf{op}\ y]\!]_\nu = \text{if } \nu(x) \ \mathsf{op}\ \nu(y) \text{ then } 1 \text{ else } 0 \qquad \pi[\![\varphi \wedge \psi]\!]_\nu = \pi[\![\varphi]\!]_\nu \cdot \pi[\![\psi]\!]_\nu$$

$$\pi[\![\varphi \vee \psi]\!]_\nu = \pi[\![\varphi]\!]_\nu + \pi[\![\psi]\!]_\nu \qquad \pi[\![\exists x\, \varphi]\!]_\nu = \textstyle\sum_{a \in A} \pi[\![\varphi]\!]_{\nu[x \mapsto a]}$$

$$\pi[\![\forall x\, \varphi]\!]_\nu = \textstyle\prod_{a \in A} \pi[\![\varphi]\!]_{\nu[x \mapsto a]} \qquad \pi[\![\neg\varphi]\!]_\nu = \pi[\![\mathsf{nnf}(\neg\varphi)]\!]_\nu$$

The symbol op stands for either $=$ or $\neq$. As you can see from the definition, the equality and inequality atoms are interpreted in K as 0 or 1, i.e., their provenance is not tracked.

Let $X, \bar{X}$ be two disjoint sets together with a one-to-one correspondence $X \longleftrightarrow \bar{X}$. We denote by $p \in X$ and $\bar{p} \in \bar{X}$ two elements that are in this correspondence. We refer to the elements of $X \cup \bar{X}$ as **provenance tokens** as they will be used to label/annotate some of the "data", i.e., literals over some ground values, via the concept of K-interpretation that we defined previously. Indeed, if, as before, we fix a finite non-empty set A and consider $\mathsf{Lit}_A = \mathsf{Facts}_A \cup \mathsf{NegFacts}_A$ then we shall use X for Facts_A and $\bar{X}$ for $\mathsf{NegFacts}_A$. By convention, if we annotate $R(\mathbf{a})$ with the "positive" token p then the "negative" token $\bar{p}$ can only be used to annotate $\neg R(\mathbf{a})$, and vice versa. We refer to p and $\bar{p}$ as *complementary* tokens.

Further, we denote by $\mathbb{N}[X, \bar{X}]$ the quotient of the semiring of polynomials $\mathbb{N}[X \cup \bar{X}]$ by the congruence generated by the equalities $p \cdot \bar{p} = 0$ for all $p \in X$.[2] Observe that two polynomials $\mathfrak{p}, \mathfrak{q} \in \mathbb{N}[X \cup \bar{X}]$ are congruent iff they become identical after deleting from each of them the monomials that contain complementary tokens. Hence, the congruence classes in $\mathbb{N}[X, \bar{X}]$ are in one-to-one correspondence with the polynomials in $\mathbb{N}[X \cup \bar{X}]$ such that none of their monomials contain complementary tokens. We shall call these **dual-indeterminate polynomials** although we might often omit "-indeterminate" just use "dual polynomials". The following is the universality property of the semiring of dual polynomials:

Proposition 2: For any commutative semiring K and for any $f : X \cup \bar{X} \to K$ such that $\forall p \in X,\ f(p) \cdot f(\bar{p}) = 0$ there exists a unique semiring homomorphism $h : \mathbb{N}[X, \bar{X}] \to K$ such that $\forall x \in X \cup \bar{X}\ h(x) = f(x)$.

We note that $\mathbb{N}[X, \bar{X}]$ is "+-positive", that is, $\mathfrak{p} + \mathfrak{q} = 0$ implies $\mathfrak{p} = \mathfrak{q} = 0$, however, it has divisors of 0. Examples: $p \cdot \bar{p} = 0$; $(p + \bar{q})\bar{p}q = 0$; $(p\bar{q} + \bar{p}q)(pq + \bar{p}\bar{q}) = 0$. We also note that a dual polynomial is identically 0 iff it evaluates to 0 for every assignment $X \cup \bar{X} \to \{0, 1\}$ that sets complementary tokens to $0/1$ or $1/0$.

Definition 3: A **provenance-tracking** interpretation is a $\mathbb{N}[X, \bar{X}]$-interpretation $\pi : \mathsf{Lit}_A \to \mathbb{N}[X, \bar{X}]$ such that $\pi(\mathsf{Facts}_A) \subseteq X \cup \{0, 1\}$ and $\pi(\mathsf{NegFacts}_A) \subseteq \bar{X} \cup \{0, 1\}$.

Only provenance-tracking interpretations (annotations) are considered in the sections ahead.

[2] This is the same as quotienting by the ideal generated by the polynomials $p\bar{p}$ for all $p \in X$.

Email	*Sender*	*Day*	*Receiver*	
	Bob	Mon	Danny	
	Carla	Tue	Bob	
	Danny	Tue	Bob	
	Danny	Tue	Carla	y
	Bob	Mon	Carla	$\bar{p}$
	Bob	Tue	Carla	$\bar{q}$
	Danny	Mon	Carla	$\bar{z}$

Class	*Student*	*Course*	
	Bob	Bio	r
	Bob	Calc II	s
	Carla	Chem	t
	Carla	Calc II	u
	Danny	Bio	x
	Danny	Chem	p_2
	Bob	Chem	$\bar{p}_1$
	Carla	Bio	$\bar{w}$
	Danny	Calc II	$\bar{v}$

Figure 1: Database for missing answers examples

3 Missing Answers

We present a small example database containing a table of student email exchanges and another one with the classes they are taking. *Email* and *Class* are shown below. The tuples that are present in the instance under consideration are above the dashed lines and some of them are annotated (labeled) with positive provenance tokens. We apply the framework described in Section 2 with A as the active domain of this database. The tuples below the dashed lines are some of the closed-world-absent tuples from the instance and they are annotated with negative provenance tokens. This constitutes a provenance tracking interpretation. We hope that the reader will keep the use of the variables x, y as provenance tokens distinct from the use of the same as Datalog variables. To keep the picture less cluttered we only annotated the tuples that we actually use in the examples.

We consider the following **query:** return the pairs (sender, receiver) for the non-Monday emails where the receiver is either not taking Chem or is taking every class that the sender is taking. The query, denoted Q, is expressed in non-recursive Datalog with negation as follows (c is a variable!):

$$S(x,y) \leftarrow \textit{Class}(x,c), \neg\textit{Class}(y,c), \textit{Class}(y, \text{“Chem”})$$
$$Q(x,y) \leftarrow \textit{Email}(x,d,y), \neg S(x,y), d \neq \text{“Mon”}$$

In first-order logic notation we have:

$$S = \{(x,y) |\, \exists c\, \textit{Class}(x,c) \wedge \neg\textit{Class}(y,c) \wedge \textit{Class}(y, \text{“Chem”})\}$$
$$Q = \{(x,y) \mid \exists d\, \textit{Email}(x,d,y) \wedge \neg S(x,y) \wedge d \neq \text{“Mon”}\}$$

Notice that the output of Q consists of (“Carla”, “Bob”) and (“Danny”, “Bob”). We are interested in why the tuple (“Bob”, “Carla”) is not in the set of outputs. Our approach is to find explanations for such questions by finding all the minimal repairs that would make Q(“Bob”, “Carla”) true. And we find these repairs by computing the dual polynomial $\mathfrak{p}$ corresponding to $\neg Q$(“Bob”, “Carla”) and then “solving” the equation $\mathfrak{p} = 0$, that is, finding which tokens can be set to 0 to insure that $\mathfrak{p}$ becomes 0. We have:

$$\neg Q(\text{“Bob”}, \text{“Carla”}) \;=\; \forall d\, \neg\textit{Email}(\text{“Bob”}, d, \text{“Carla”}) \;\vee\; S(\text{“Bob”}, \text{“Carla”}) \;\vee\; [d = \text{“Mon”}]$$

The computations of dual polynomials is done according to Definition 1. When it comes to quantifiers we need, in principle, to range over all the elements of the active domain for every variable. However, it is clear that it suffices to range over the elements that actually occur in the columns that correspond to the variables. For example, it suffices to take $d \in \{\text{“Mon”}, \text{“Tue”}\}$ and $c \in \{\text{“Bio”}, \text{“Calc II”}, \text{“Chem”}\}$. We call this the *typed active domain* approach and note that it is widely applicable in practice.

Now we calculate the polynomial corresponding to $\neg Q$(“Bob”, “Carla”):

$$\mathfrak{p} \;=\; (\bar{p} + \mathfrak{q} + [\text{“Mon”} = \text{“Mon”}])(\bar{q} + \mathfrak{q} + [\text{“Tue”} = \text{“Mon”}]) \;=\; (\bar{p} + \mathfrak{q} + 1)(\bar{q} + \mathfrak{q})$$

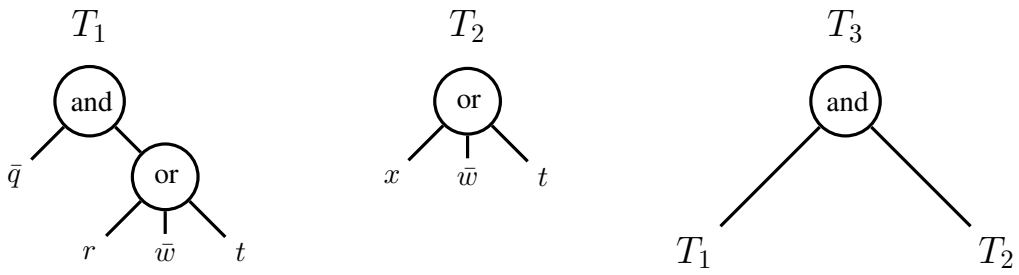

Figure 2: Repair and-or trees for missing answers Q("Bob", "Carla") and Q("Danny", "Carla")

where $\mathfrak{q}$ is the polynomial corresponding to S("Bob", "Carla"). We have:

$$\begin{aligned} S(\text{"Bob"}, \text{"Carla"}) &= \exists c\, \textit{Class}(x, c) \wedge \neg \textit{Class}(y, c) \wedge \textit{Class}(y, \text{"Chem"}) \\ \mathfrak{q} &= r\bar{w}t + s\bar{u}t + p_1\bar{t}t \;=\; r\bar{w}t + s\bar{u}t \end{aligned}$$

Notice that the polynomial $\mathfrak{q}$ can be further simplified because our database already includes the tuple *Class*("Carla", "Calc II") annotated with u, hence we can already assume that $\bar{u} = 0$. This avoids computing solutions that unnecessarily include $\bar{u} = 0$ (they would ask to insert a tuple that is already in the database). It follows that we can take $\mathfrak{q} = r\bar{w}t$.

For $\mathfrak{p} = 0$ one possible solution, by inspection, is $\{\bar{q} = t = 0\}$. Setting a negative token to 0 corresponds to invalidating a negative fact, i.e., inserting a tuple. Similarly, setting a positive token to 0 corresponds to deleting a tuple. Therefore this solution (that we found in an ad-hoc manner) corresponds to the repair that inserts $[+\textit{Email}(\text{"Bob"}, \text{"Tue"}, \text{"Carla"})]$ and that deletes $[-\textit{Class}(\text{"Carla"}, \text{"Chem"})]$ and thus brings ("Bob", "Carla") into the output of the query Q.

However, we need to systematically find *all* minimal repairs and to do this, anticipating dealing with more general queries, we proceed *without* substituting $\mathfrak{q}$ in $\mathfrak{p}$ and thus taking advantage of the structure of the query as a stratified, non-recursive Datalog$^\neg$ program. The first observation is that $\mathfrak{p}$ is a product of factors and in general it is possible for $\mathfrak{p}$ to be identically 0 (hence all minimal solutions would be empty) since $\mathbb{N}[X, \bar{X}]$ has divisors of 0. But this is not the case here since, for example, $\bar{p}\bar{q}$ occurs in $\mathfrak{p}$. Therefore, any solution makes at least one of the two factors 0. Note that each term in each factor of $\mathfrak{p}$ is either a single token or $\mathfrak{q}$ or a 1. The factor that contain a 1 cannot be made 0. Thus we must have $\bar{q} + \mathfrak{q} = 0$, hence, $\bar{q} = 0$ and $\mathfrak{q} = 0$. Notice that the stratified structure can lead to explanations for missing answers that may entail wrong answers for some of the subqueries (non-answer intensional predicates; in this example S("Bob", "Carla")). Thus, the missing answers and the wrong answers problems are intricately related for this query language.

Since S is an intensional predicate we must continue until we find the repairs to the actual database. $\mathfrak{q} = 0$ has three (minimal) solutions; $\{t = 0\}$ is one, the others are $\{r = 0\}$ and $\{\bar{w} = 0\}$. This yields three minimal solutions for $\mathfrak{p} = 0$: $\{\bar{q} = t = 0\}$, $\{\bar{q} = r = 0\}$ and $\{\bar{q} = \bar{w} = 0\}$. The reader can check that the repairs corresponding to each of these solutions, for example, insert $[+\textit{Email}(\text{"Bob"}, \text{"Tue"}, \text{"Carla"})]$ and insert $[+\textit{Class}(\text{"Carla"}, \text{"Bio"})]$ for the third solution, also bring ("Bob", "Carla") into the output of Q. All the minimal solutions are nicely captured by the and-or tree T_1 in Figure 2 (on the leaves of these trees we have partial solutions $token = 0$ which are represented, for simplicity, just by $token$.

Let us now find the repairs for another answer missing from the output of Q, namely ("Danny", "Carla"). The dual polynomial corresponding to $\neg Q$("Danny", "Carla") is $\mathfrak{r} = (\bar{z}+\mathfrak{s}+1)(\bar{y}+\mathfrak{s})$ where $\mathfrak{s} = x\bar{w}t+v\bar{u}t+p_2\bar{t}t = x\bar{w}t + v\bar{u}t$. As before, we can simplify the polynomials by setting $\bar{y} = \bar{u} = 0$ since the tuples annotated with y and u are already in the database, obtaining $\mathfrak{r} = (\bar{z} + \mathfrak{s} + 1)x\bar{w}t$. Although it leads to the same result, we could

also have set $v = 0$ since the tuple annotated with $\bar{v}$ is *not* in the database (and a repair that contained $v = 0$ would have asked that we delete a tuple already absent. All the solutions to $\mathfrak{r} = 0$ are captured by the and-or tree T_2 in Figure 2.

What if we wish to repair *both* Q("Bob", "Carla") and Q("Danny", "Carla")? This would need the dual polynomial corresponding to $\neg[Q(\text{"Bob"}, \text{"Carla"}) \wedge Q(\text{"Danny"}, \text{"Carla"})]$, i.e., $\mathfrak{p} + \mathfrak{r}$. The solutions to $\mathfrak{p} + \mathfrak{r} = 0$ are captured by the and-or tree T_3 in the same figure. Observe that T_1 describes 3 minimal repairs, and so does T_2. Now T_3 describes $3 \times 3 = 9$ repairs, but *only* 3 of them are minimal: $\{\bar{q} = r = x = 0\}, \{\bar{q} = \bar{w} = 0\}$ and $\{\bar{q} = t = 0\}$.

In general, our approach is not guaranteed to compute only minimal repairs, as can be seen by considering $Q'(y) \leftarrow Q(x, y)$ and the missing answer Q'("Carla"). However, we can prove that all the minimal repairs are included among those computed by the algorithm shown below. It is also clear that redundant, non-minimal repairs, can be pruned away.

Algorithm Next we present a procedure for finding repairs (including all the minimal repairs) for missing and/or wrong answers for queries expressed in non-recursive Datalog with negation (and recall that any first-order query is logically equivalent with one such). The algorithm takes advantage of the stratified structure of the language to produce a compact and-or repair tree.

The algorithm's input is a triple $(R, \mathbf{a}, \mu)$ where R is a predicate, $\mathbf{a}$ is tuple of constants from the active domain of the instance (R and $\mathbf{a}$ have the same arity), and μ is a "mode"parameter that can have two values: missing and wrong. Let Q be the answer predicate of the query. There may be multiple Datalog rules whose head can be instantiated to $Q(\mathbf{a})$: let's denote these instatiations by $Q(\mathbf{a}) \leftarrow B_i(\mathbf{a}, \mathbf{y}_i)$ for $i = 1, \ldots, k$. Then, in FOL,

$$Q(\mathbf{a}) \;=\; [\exists \mathbf{y}_1 \; B_1(\mathbf{a}, \mathbf{y}_1)] \;\vee\; \cdots \;\vee [\exists \mathbf{y}_k \; B_k(\mathbf{a}, \mathbf{y}_k)]$$

where each B_i is a *conjunction* of positive or negative atoms(extensional or intensional), equalities, or inequalities. Moreover,in NNF,

$$\neg Q(\mathbf{a}) \;=\; [\forall \mathbf{y}_1 \; \overline{B}_1(\mathbf{a}, \mathbf{y}_1)] \;\wedge\; \cdots \;\wedge [\forall \mathbf{y}_k \; \overline{B}_k(\mathbf{a}, \mathbf{y}_k)]$$

where each $\overline{B}_i$ (in NNF) is a *disjunction* of negative or positive atoms(extensional or intensional), inequalities, or equalities.

On input $(Q, \mathbf{a}, \mathsf{missing})$, the algorithm returns the repair and-or tree corresponding to the missing answer $Q(\mathbf{a})$. It does so by first computing the dual polynomial $\mathfrak{p}$ corresponding to the first-order sentence $\mathsf{nnf}(\neg Q(\mathbf{a}))$. This computation includes simplifications like those mentioned in our examples:

- If $\bar{u}$ occurs in the polynomial and u annotates a tuple already in the instance, then replace $\bar{u}$ with 0.
- If v occurs in the polynomial and v annotates a tuple that is not in the database then replace u with 0.

Next the algorithm finds solutions to $\mathfrak{p} = 0$. A solution is a set of provenance tokens, negative or positive, all set to 0. Since intensional predicates may occur in $\mathsf{nnf}(\neg Q(\mathbf{a}))$ we generate fresh provenance tokens to annotate temporarily these intentional literals. Note that $\mathfrak{p}$ is a product of factors, each factor being a sum of (positive or negative, extensional or intensional) provenance tokens or 1's. Moreover, we can prove that $\mathfrak{p}$ cannot be identically 0. Indeed, by the range-restriction condition, each rule must contain at least one positive literal. This means that each factor in $\mathsf{nnf}(\neg Q(\mathbf{a}))$ must contain at least one negative token and the product of all these negative tokens occurs as a monomial in $\mathfrak{p}$. When constructing the and-or tree of solutions to $\mathfrak{p} = 0$, we

collect trees that are either leaves or, by setting an intensional token to 0, are obtained from recursive call to the algorithm.

On input $(Q, \mathbf{a}, \mathsf{wrong})$, the algorithm returns the repair and-or tree corresponding to the wrong answer $Q(\mathbf{a})$. It does so by computing the dual polynomial $\mathfrak{q}$ corresponding to the first-order sentence $\mathsf{nnf}(Q(\mathbf{a}))$, simplified as above and with fresh intensional tokens as above. Solutions are found, again, by setting $\mathfrak{q} = 0$. Note that $\mathfrak{p}$ is a sum of terms, each term being a product of tokens (i.e., a monomial).

In stating the algorithm we use the notations $\bar{p} \sim L_1$, respectively $p \sim L_2$, to mean that the negative token $\bar{p}$ annotates the negative literal L_1, respectively the positive token p annotates the positive literal L_2.

All this results in Algorithm **??**.

Size of repair trees We can show that the number of minimal repairs is in general exponential in the size of the active domain. However, the and-or trees that we compute offer a compact representation of all these repairs. Indeed, let n be an upper bound on the size of the active domain and let q be an upper bound on the size of the query (expressed in non-recursive Datalog with negation). The algorithm calculates two kinds of polynomials:

- For missing tuples it calculates polynomials as products of factors. Each factor's size is bounded by the size of rules is $O(q)$. The number of factors is bounded by the number of rules times the number of iterations of universally quantified variables in each rule, therefore is $O(q\, n^q)$. Thus, the resulting polynomial has size $O(q^2\, n^q)$.
- For wrong tuples it calculates polynomials as sums of monomials. Since there are at most q existentially quantified variables in each rule, and there are at most q rules there are at most $q\, n^q$ monomials in this polynomial. Each monomial's size is bounded by q so the resulting polynomial also has size $O(q^2\, n^q)$.

It follows that in the repair tree each node has $O(q^2\, n^q)$ children. The height of the tree is bounded by the number of rules, hence by q. Therefore the repair tree has size $O(q^{2q}\, n^{q^2})$, thus of polynomial data complexity.

Essentially the same argument shows that Algorithm 1 runs in polynomial time (data complexity).

4 Repairing Integrity Constraints

Since most integrity constraints used in databases are expressible in first order logic, our approach will also work to "repair" them. Here is an example. Consider the table *Class1* in Figure **??**. Notice that Danny has decided to take another course–Calc II. Uh oh... adding the course wasn't a good idea. Danny gets overloaded with work and finds himself failing Chemistry. The university notices this and puts Danny on probation (of a ruthless kind!), so Danny can now take only at most 2 courses at a time. Consequently, our database now breaks an integrity constraint. How do we fix it?

The integrity constraint "NO STUDENT CAN TAKE THREE OR MORE DISTINCT COURSES" can be written

$$I \;=\; \forall s \;\; \neg(\exists x, y, z \;\; \mathit{Class1}(s, x) \;\wedge\; \mathit{Class1}(s, y) \;\wedge\; \mathit{Class1}(s, z) \;\wedge\; x \neq y \;\wedge\; x \neq z \;\wedge\; y \neq z)$$

This is an example of a *denial constraint* [2]. Of course, it fails in the database *Class1*. To find out the reasons it fails as well as to obtain repairs to the database we compute the dual provenance polynomial of $\neg I$. We have

$$\mathsf{nnf}(\neg I) \;=\; \exists s, x, y, z \;\; \mathit{Class1}(s, x) \;\wedge\; \mathit{Class1}(s, y) \;\wedge\; \mathit{Class1}(s, z) \;\wedge\; x \neq y \;\wedge\; x \neq z \;\wedge\; y \neq z$$

Input : μ tells us whether $R(\mathbf{a})$ is missing or is wrong.
Output: An and-or tree of updates
if R *is an extensional predicate* **then**
  if $\mu = \mathsf{missing}$ **then**
    Return a single node tree labeled with token $\bar{p} \sim \neg R(\mathbf{a})$.
  end
  if $\mu = \mathsf{wrong}$ **then**
    Return a single node tree labeled with token $p \sim R(\mathbf{a})$.
  end
end
if R *is an intensional predicate* **then**
  $\mathscr{C} := \emptyset$.
  if $\mu = \mathsf{missing}$ **then**
    Compute, as a product of factors, the polynomial $\mathfrak{p}$ corresponding to $\mathsf{nnf}(\neg R(\mathbf{a}))$.
    for *each factor* f *in* $\mathfrak{p}$ **do**
      $\mathscr{T} := \emptyset$.
      for *each term* t *in the factor* f **do**
        if $t \sim$ *positive literal* $S(\mathbf{b})$ **then**
          $\mathscr{T} := \mathscr{T} \cup \{RepTree(S, \mathbf{b}, \mathsf{wrong})\}$.
        end
        if $t \sim$ *negative literal* $\neg S(\mathbf{b})$ **then**
          $\mathscr{T} := \mathscr{T} \cup \{RepTree(S, \mathbf{b}, \mathsf{missing})\}$.
        end
        if $t \sim 1$ **then**
          $\mathscr{T} := \mathscr{T} \cup \{"never - zero"\}$.
        end
      end
      if $"never - zero" \notin \mathscr{T}$ **then**
        Build an **and**-tree T with the trees collected in $\mathscr{T}$ as children of the root.
        $\mathscr{C} := \mathscr{C} \cup \{T\}$.
      end
    end
    Build an **or**-tree T with the trees collected in $\mathscr{C}$ as children of the root. Return with T.
  end
  if $\mu = \mathsf{wrong}$ **then**
    Compute, as a sum of terms, the polynomial $\mathfrak{q}$ corresponding to $\mathsf{nnf}(R(\mathbf{a}))$.
    for *each term* t *in* $\mathfrak{q}$ **do**
      $\mathscr{T} := \emptyset$.
      for *each factor* f *in the term* t **do**
        if $f \sim$ *positive literal* $S(\mathbf{b})$ **then**
          $\mathscr{T} := \mathscr{T} \cup \{RepTree(S, \mathbf{b}, \mathsf{wrong})\}$.
        end
        if $f \sim$ *negative literal* $\neg S(\mathbf{b})$ **then**
          $\mathscr{T} := \mathscr{T} \cup \{RepTree(S, \mathbf{b}, \mathsf{missing})\}$.
        end
      end
      Build an **or**-tree T with the trees collected in $\mathscr{T}$ as children of the root.
      $\mathscr{C} := \mathscr{C} \cup \{T\}$.
    end
    Build an **and**-tree T with the trees collected in $\mathscr{C}$ as children of the root. Return with T.
  end
end

Algorithm 1: $RepTree(R, \mathbf{a}, \mu)$ builds a repair tree

Class1	*Student*	*Course*	
	Bob	Bio	
	Bob	Calc II	
	Carla	Chem	
	Carla	Calc II	
	Danny	Bio	r
	Danny	Chem	s
	Danny	Calc II	t
	Bob	Chem	$\bar{u}$
	Carla	Bio	$\bar{v}$

Admin	*Course*	*Dept*	
	Calc II	Math	p
	Chem	Sci	q
	Bio	Sci	r
	Calc II	Sci	$\bar{t}$
	Chem	Math	$\bar{u}$
	Bio	Math	$\bar{v}$

Class2	*Student*	*Course*	
	Bob	Bio	w
	Bob	Calc II	x
	Danny	Bio	w_1
	Danny	Chem	y_1
	Bob	Chem	$\bar{y}$
	Danny	Calc II	$\bar{z}$

Figure 3: Tables for the integrity constraint examples

The corresponding dual polynomial $\mathfrak{p}$ is a sum. Using the *typed* active domain approach, this sum will have, in principle, $3 \times 3 \times 3 \times 3 = 81$ terms. However, most of the inequalities yield 0, namely when x, y, z are not assigned distinct values. We are left with $3 \times (3!)$ terms. Moreover, $2 \times (3!)$ of these are monomials containing $\bar{u}$ or $\bar{v}$ which we can set to 0 (because Bob and Carla are not breaking the constraint :). We are left with $\mathfrak{p} = 3rst$ which yields a repair **or**-tree with three leaves. As expected, we find that the integrity constraint is broken because all three tuples annotated r, s, or t are in the database. Also as expected, we have three possible minimal repairs, each corresponding to deleting one of the three tuples.

The second example we will show in this section uses the tables *Admin* and *Class2* in Figure **??**. It also uses the constraint "IF A STUDENT IS TAKING MORE THAN ONE CLASS, ALL OF THEIR CLASSES MUST BE ADMINISTERED BY THE SAME DEPARTMENT". We can express this constraint as:

$$J \;=\; \forall x, y \;[\exists s \;\; \textit{Class2}(s, x) \;\wedge\; \textit{Class2}(s, y) \;\wedge\; x \neq y] \;\rightarrow\; [\exists t \;\; \textit{Admin}(x, t) \;\wedge\; \textit{Admin}(y, t)]$$

This is an example of a tuple-generating constraint, even more general than the conditional inclusion constraints mentioned in [2, 10]. It does fail in the database above since Bob saw fit to take both Bio and Calc II. As with the denial constraint we considered above, we find the reasons for failure and possible repairs by computing the dual polynomial corresponding to $\mathsf{nnf}(\neg J)$. We have

$$\mathsf{nnf}(\neg J) \;=\; \exists x, y \;[\exists s \;\; \textit{Class}(s, x) \;\wedge\; \textit{Class}(s, y) \;\wedge\; x \neq y] \;\wedge\; [\forall t \;\neg\textit{Admin}(x, t) \;\vee\; \neg\textit{Admin}(y, t)]$$

We compute the dual provenance polynomial of $\neg J$. Again, we use the *typed* active domain approach and we notice that the first factor will always be zero if we choose x and y to both be the same course. The result is a sum of polynomials, which we denote $\mathfrak{q}$:

$$\mathfrak{q} = (zw_1 + xw)(\bar{t} + \bar{r})(\bar{p} + \bar{v}) + (y_1 z + yx)(\bar{q} + \bar{t})(\bar{u} + \bar{p}) + (y_1 w_1 + yw)(\bar{q} + \bar{r})(\bar{u} + \bar{v})$$

However, given which tuples are already/are not in the database we can simplify this polynomial by setting $y = z = 0$ and $\bar{p} = \bar{q} = \bar{r} = 0$ and we are left with $\mathfrak{q} = xw\bar{t}\bar{v}$. To fix the constraint, we try to make $\mathfrak{q} = 0$. which yields a repair **or**-tree with four leaves. These correspond to four different minimal repairs (two of them are deletions and the others insertions): $[-\textit{Class}(\text{"Bob"}, \text{"Calc II"})]$, or $[-\textit{Class}(\text{"Bob"}, \text{"Bio"})]$, or $[+\textit{Admin}(\text{"Calc II"}, \text{"Sci"})]$, or $[+\textit{Admin}(\text{"Bio"}, \text{"Math"})]$.

5 Costing Repairs

For costs we use the *tropical semiring* $\mathbb{T} = (\mathbb{R}_+^\infty, \min, +, \infty, 0)$. Its elements and operations appear in *min-cost* interpretations (e.g., shortest paths). Here we are going to show how to use it to select repairs that are in some sense optimal.

Recall the integrity constraint J from Section 4. We obtained all possible minimal repairs by computing the dual polynomial $\mathfrak{q}$ corresponding to $\mathsf{nnf}(\neg J)$ and obtained four repairs: $\{x = 0\}, \{w = 0\}, \{\bar{t} = 0\}, \{\bar{v} = 0\}$. To choose among them, we now compute the dual polynomial $\mathfrak{r}$ corresponding to $\mathsf{nnf}(J)$:

$$\mathfrak{r} = [(\bar{z} + \bar{w}_1)(\bar{x} + \bar{w}) + tr + pv] \cdot [(\bar{y}_1 + \bar{z})(\bar{y} + \bar{x}) + qt + up] \cdot [(\bar{y}_1 + \bar{w}_1)(\bar{y} + \bar{w}) + qr + uv]$$

Next we apply all the simplifications allowed by the database *except* those involving the tokens that appear in the repairs, namely $\bar{w}_1 = \bar{y}_1 = u = 0$. We obtain: $\mathfrak{r} = [\bar{z}(\bar{x} + \bar{w}) + tr + pv] \cdot [\bar{z}(\bar{y} + \bar{x}) + qt] \cdot [qr]$.

Our definition of optimality[3] is to find the repair that minimizes the cost of $\mathfrak{r}$ when interpreted in $\mathbb{T}$, given costs for the remaining tokens in r, say $cost(\bar{z}) = \alpha$, $cost(r) = \beta$, $cost(p) = \gamma$, $cost(\bar{y}) = \delta$, $cost(q) = \iota$, and assuming, in turn, each of the four repairs have a cost also, let's say that a deletion costs λ and an insertion costs μ, where $\alpha, \beta, \gamma, \delta, \iota, \lambda, \mu \in \mathbb{R}^{\infty}_{+}$. Thus we have four different assignments into $\mathbb{T}$: $[\bar{x} \mapsto \lambda, \bar{w} \mapsto \infty, t \mapsto 0, v \mapsto 0]$, $[\bar{x} \mapsto \infty, \bar{w} \mapsto \lambda, t \mapsto 0, v \mapsto 0]$, $[\bar{x} \mapsto \infty, \bar{w} \mapsto \infty, t \mapsto \mu, v \mapsto 0]$, $[\bar{x} \mapsto \lambda, \bar{w} \mapsto \infty, t \mapsto 0, v \mapsto \mu]$ one for each of the four repairs. We obtain:

Repair	Cost
$[-$*Class2*(“Bob”, “Calc II”)$]$	$\min(\alpha + \lambda, \beta, \gamma) + \min(\alpha + \min(\delta, \lambda), \iota) + \iota + \beta$
$[-$*Class2*(“Bob”, “Bio”)$]$	same
$[+$*Admin*(“Calc II”, “Sci”)$]$	$\min(\mu + \beta, \gamma) + \min(\alpha + \delta, \iota + \mu) + \iota + \beta$
$[+$*Admin*(“Bio”, “Math”)$]$	$\gamma + \mu + \alpha + \delta + \iota + \beta$

For example, taking $\lambda = \mu = \alpha = \beta = \gamma = \delta = \iota = 10$ will result in the first (or second) repair to be cheapest.

6 Conclusions

In this paper, we introduced a novel approach for computing all possible repairs for missing and wrong answers to a query on a database instance. Our algorithm is based on a new framework for reasoning over the provenance of non-answers under the CWA for queries expressible in first-order logic, including negation. This framework treats relational queries as logical structures (unions and conjunctions of statements) and applies algebraic operations to these logical structures, allowing the use of dual polynomials as a way to represent provenance.

Our approach is not limited to repairing queries with missing and wrong answers; it can also repair a database instance that does not satisfy certain integrity constraints. Since integrity constraints can be expressed in first-order logic, our algorithm is able to determine where constraints are broken. In cases of missing answers, wrong answers, and broken integrity constraints, our algorithm is able to suggest repairs for the database in the form of a set of insertions and deletions of tuples.

Our use of dual polynomials rather than boolean expressions provides a way to weigh certain tuples over others by assigning tokens different weights. Questions such as finding a minimal cost solution from an and-or-tree are NP-hard (for example by reduction from vertex cover). In the future, we aim to explore approximation techniques for these questions. Additionally, we would like to expand our approach to handle open world assumption. Currently, our approach fixes databases by introducing tuples consisting only of values from our active domain. However, it is unrealistic to always expect repairs to be derived from within the database. Thus, it is important for us to move beyond CWA and develop a working framework in OWA.

[3]Other definitions are possible and we plan to compare them in future work.

Acknowledgments. The authors would like to thank Sanjeev Khanna and Tova Milo for helpful discussions. Our work was partially supported by Penn's Center for Undergraduate Research and Fellowships as well as by NSF grants 1302212 and 1547360 and by NIH grant U01EB02095401.

References

[1] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania, USA*, pages 68–79, 1999.

[2] L. Bertossi. Database repairing and consistent query answering. *Synthesis Lectures on Data Management*, 3(5):1–121, 2011.

[3] N. Bidoit, M. Herschel, and K. Tzompanaki. Immutably answering why-not questions for equivalent conjunctive queries. In *6th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2014)*, Cologne, 2014. USENIX Association.

[4] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 143–154, New York, NY, USA, 2005. ACM.

[5] D. Burdick, R. Fagin, P. G. Kolaitis, L. Popa, and W. Tan. A declarative framework for linking entities. *ACM Trans. Database Syst.*, 41(3):17:1–17:38, 2016.

[6] D. Burdick, R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Expressive power of entity-linking frameworks. In *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, pages 10:1–10:18, 2017.

[7] A. Chapman and H. V. Jagadish. Why not? In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 523–534, New York, NY, USA, 2009. ACM.

[8] J. Chomicki. *Consistent Query Answering: Five Easy Pieces*, pages 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[9] U. Dayal and P. A. Bernstein. On the updatability of relational views. In *Proceedings of the Fourth International Conference on Very Large Data Bases - Volume 4*, VLDB '78, pages 368–377. VLDB Endowment, 1978.

[10] W. Fan and F. Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[11] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.*, 33(2):6:1–6:48, June 2008.

[12] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *Proc. VLDB Endow.*, 3(1-2):173–184, Sept. 2010.

[13] E. Grädel and V. Tannen. Semiring provenance for first-order model checking. *CoRR*, abs/1712.01980, 2017.

[14] T. J. Green and V. Tannen. The semiring framework for database provenance. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 93–99, 2017.

[15] M. Herschel. A hybrid approach to answering why-not questions on relational query results. *J. Data and Information Quality*, 5(3):10:1–10:29, Mar. 2015.

[16] M. Herschel and M. A. Hernández. Explaining missing answers to spjua queries. *Proc. VLDB Endow.*, 3(1-2):185–196, Sept. 2010.

[17] M. Herschel, M. A. Hernández, and W.-C. Tan. Artemis: A system for analyzing missing answers. *Proc. VLDB Endow.*, 2(2):1550–1553, Aug. 2009.

[18] J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *PVLDB*, 1(1):736–747, 2008.

[19] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pages 53–62, New York, NY, USA, 2009. ACM.

[20] D. Kossmann, E. Lo, and C. Binnig. Reverse query processing. *2007 IEEE 23rd International Conference on Data Engineering*, 00:506–515, 2007.

[21] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. Why so? or why no? functional causality for explaining query answers. *CoRR*, abs/0912.5340, 2009.

[22] A. Meliou, W. Gatterbauer, and D. Suciu. Reverse data management. *PVLDB*, 4(12):1490–1493, 2011.

[23] A. Meliou and D. Suciu. Tiresias: the database oracle for how-to queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 337–348, 2012.

[24] Y. Wu, A. Haeberlen, W. Zhou, and B. T. Loo. Answering why-not queries in software-defined networks with negative provenance. In *Proceedings of the 12th ACM Workshop on Hot Topics in Networks (HotNets-XII)*, Nov. 2013.

[25] Y. Wu, M. Zhao, A. Haeberlen, W. Zhou, and B. T. Loo. Diagnosing missing events in distributed systems negative provenance. In *Proceedings of ACM SIGCOMM 2014*, Aug. 2014.