On Provenance Minimization [*]

Yael Amsterdamer
Tel Aviv University
and University of Pennsylvania
yaelamst@post.tau.ac.il

Daniel Deutch
Ben Gurion University
and University of Pennsylvania
deutchd@cs.bgu.ac.il

Tova Milo
Tel Aviv University
milo@post.tau.ac.il

Val Tannen
University of Pennsylvania
val@cis.upenn.edu

ABSTRACT

Provenance information has been proved to be very effective in capturing the computational process performed by queries, and has been used extensively as the input to many advanced data management tools (e.g. view maintenance, trust assessment, or query answering in probabilistic databases). We study here the *core* of provenance information, namely the part of provenance that appears in the computation of every query equivalent to the given one. This provenance core is informative as it describes the part of the computational process that is inherent to the query. It is also useful as a compact input to the above mentioned data management tools. We study algorithms that, given a query, compute an equivalent query that realizes the core provenance for all tuples in its result. We study these algorithms for queries of varying expressive power. Finally, we observe that, in general, one would not want to require database systems to evaluate a specific query that realizes the core provenance, but instead to be able to find, possibly off-line, the core provenance of a given tuple in the output (computed by an arbitrary equivalent query), without rewriting the query. We provide algorithms for such direct computation of the core provenance.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: [Data Models]

General Terms

Algorithms, Theory

1. INTRODUCTION

Recording *provenance* information for query results, that explains the computational process leading to their generation, is now a common technique. In particular, the work of [19] suggested capturing provenance information via *polynomials.* The idea is to associate every tuple in the input database with an annotation, and to extend the operations of relational algebra so that they will work on these annotated tuples. The resulting annotation of every output tuple is a polynomial over the original annotations, that reflects the operations performed on the original tuples to obtain the output tuple. Provenance polynomials were shown to be very useful, serving as input to many advanced data management tools, such as view maintenance, trust assessment, or query answering in probabilistic databases (See e.g. [18, 25, 30]). We note that provenance polynomials give a *trace* of the computational process associated with every tuple in the query result. For example, if the provenance annotation p of an output tuple is $x \cdot y \cdot y + z + z$ $(= x \cdot y^2 + 2z)$ where x, y, z are annotations that uniquely identify three input tuples (e.g., tuple ids) then p indicates that there are three different ways (derivations) to compute the output tuple, one uses x once and y twice while the other two use only z, once.

Consequently, the evaluation of two equivalent queries (in the standard sense of [9]), on a given input database instance, may lead to different provenance polynomials for the same output tuples. While this is sometimes useful, there are many cases in which what is of interest is not the computational process induced by the query chosen among equivalent queries (which is affected by optimizers and may change over time), but rather the "core" computational process, whose use of input tuples must be "included" in the query evaluation *for every equivalent query.* We are therefore led to a notion of *core provenance* capturing the core computation. We will define this notion formally, for provenance that is defined in a very general way (namely, using the $N[X]$ semiring [19]) and then see that the core provenance is not only informative in exposing the core of the execution, but is also *compact.* We propose using this compact representation to alleviate practical challenges that arise in data management tools due to the size of provenance information [10, 27].

Following the above observations, the present paper addresses two main challenges: defining the core provenance, and then realizing it. Towards the definition, we introduce an *order relation* $p \leq p'$ on provenance polynomials. Intuitively, this order relation captures the "terseness" in the use of tuples. For example, in this order using a tuple once

[*] This work has been partially funded by the NSF grant IIS-0629846, the NSF grant IIS-0803524, by the Israel Science Foundation, by the US-Israel Binational Science Foundation, by the EU grant MANCOOSI and by the ERC grant Webdam under agreement 226513.

PODS'11, June 13–15, 2011, Athens, Greece.

is terser than using it twice, within the alternative ways of computing the same answer. In a more complex example, $x{\cdot}y^2+2z \leq x{\cdot}y^2+x{\cdot}z+y{\cdot}z$ but the opposite is not true. The ordering of polynomials lifts naturally to queries. Given two equivalent queries Q, Q' (denoted $Q \equiv Q'$), we say that Q is "terser" w.r.t. provenance than Q', and we write $Q \subseteq_P Q'$ if for any input database D and any tuple t in the output, where the annotation of t in $Q(D)$ is p and that of t in $Q'(D)$ is p', we have $p \leq p'$. Now, given a class of queries $\mathcal{C}$, we can formally define core provenance for a query $Q \in \mathcal{C}$: it is the provenance yielded by a query $Q' \in \mathcal{C}$ such that $Q' \equiv Q$ and for any other $Q'' \in \mathcal{C}$ equivalent to Q, we have $Q' \subseteq_P Q''$. In that case, we say that Q' is a provenance-minimal (p-minimal) query in $\mathcal{C}$ equivalent to Q.

We then study the realization of this core provenance, first via queries and then by direct computation, as follows.

Realization via Queries. Given a query Q and a class of queries $\mathcal{C}$ we aim at finding a p-minimal query in $\mathcal{C}$ equivalent to Q. Since the identification of the p-minimal query (and thus the core provenance) depend on a "context" class of alternative query plans $\mathcal{C}$, the choices for this class become important. In this paper we study the p-minimization in query classes of increasing expressivity, for which there exist solid foundations of provenance management using polynomials: conjunctive queries (CQ), conjunctive queries with disequalities ($CQ^{\neq}$) and unions of conjunctive queries with disequalities ($UCQ^{\neq}$). When considering disequalities, we further distinguish the classes of *complete* queries, i.e. queries that disequate all of their distinct arguments. The classes of complete conjunctive queries and unions thereof are denoted $cCQ^{\neq}$ and $cUCQ^{\neq}$, respectively.

We note that query minimization in terms of the query length (or number of joins [9], referred to hereinafter as "standard minimization") has been extensively studied for these classes of queries. Interestingly, in general the queries that realize the core provenance may be very different than the minimal ones in the sense of [9]. In particular, there are conjunctive queries for which an equivalent query in $UCQ^{\neq}$ yields a "terser" provenance, as the latter query allows only a subset of the original derivations for each output tuple.

Table 1 summarizes the results on finding p-minimal queries, contrasted with results on standard minimization. Following the above discussion we distinguish between finding the p-minimal equivalent query among all queries in $UCQ^{\neq}$, or in a further restricted class $\mathcal{C}$. For instance, when given as input a query in $CQ^{\neq}$, there are cases where not only that terser provenance can be obtained by resorting to $UCQ^{\neq}$, but furthermore no equivalent query that realizes its core provenance in $CQ^{\neq}$ exists. I.e., for some sets of equivalent queries in $CQ^{\neq}$, each query leads to strictly more provenance than some other query in the set, for some output tuple. If the input query is in CQ, then we can use standard minimization to obtain an equivalent CQ query which is p-minimal among all those in CQ, but an equivalent query entailing terser provenance may be still found in $UCQ^{\neq}$.

The complexity of our p-minimization algorithms is in general exponential in the query size (with the exception of $cCQ^{\neq}$ for which we suggest a PTIME algorithm). This is unavoidable as the corresponding decision problem (described in the sequel) is DP-complete [16].

Query Class	**"Standard" Minimal in Class/Overall**	**P-minimal in Class**	**P-minimal Overall**
$CQ^{\neq}$	In $CQ^{\neq}$	No p-minimal query exists	In $UCQ^{\neq}$, EXPTIME
CQ	In CQ	same as "standard" minimization	In $UCQ^{\neq}$, EXPTIME
$cCQ^{\neq}$	In $cCQ^{\neq}$	same as "standard" minimization	In $cCQ^{\neq}$, PTIME
$UCQ^{\neq}$	In $UCQ^{\neq}$	different than "standard" minimization	In $UCQ^{\neq}$, EXPTIME

Table 1: Summary of Results

Realization by Direct Computation. Core provenance is also useful in improving the efficiency of provenance-based analysis tools, in the sense that they may be fed with smaller provenance polynomials. While so, we would not want database systems to be obligated to evaluate a specific query that realizes the core provenance, but would rather allow optimizers to evaluate the most efficient query. We would thus like to be able to evaluate any equivalent query, but then (possibly off-line) find the core provenance of given tuples in the result, *without re-evaluating the query.* We show that this is possible by manipulations on the provenance polynomials of the individual tuples in the query result. Moreover, we show that this can be done *even in absence of the original query* (e.g. if it is not available due to confidentiality or to its loss, etc.), assuming that we know the input database and the set of constants used in the query (if any are used).

Paper Organization. The rest of this paper is organized as follows. In Section 2 we provide the main definitions used throughout the paper. In Section 3 we study provenance-wise minimization of conjunctive queries, and in Section 4 we study it for union of conjunctive queries. In section 5 we study minimization applied directly on provenance polynomials. In Section 7 we provide an overview of related work, and we conclude in Section 8.

2. PRELIMINARIES

We provide in this section the formal definitions used throughout the paper. The definitions of these notions will be accompanied by simple examples, which nevertheless will be valuable in explaining the more complicated constructions in the sections that follow.

2.1 Classes of Queries

We start by briefly recalling the basic definitions for conjunctive queries and union thereof from [1]. We use the standard notions of relational databases and schema, without repeating their definitions in [1].

We assume in the sequel the existence of a domain $\mathcal{V}$ of variables and a domain $\mathcal{C}$ of constants. Conjunctive queries are then defined as follows:

Definition 2.1. *A rule based conjunctive query Q with disequalities, over a database schema S, is an expression of the form:*

$$ans(u_0) := R_1(u_1), ..., R_n(u_n), E_1, ..., E_m \text{ where:}$$

- *$R_1, ..., R_n$ are relation names in S, ans is a relation name not in S.*

- *Each u_i is a vector $(l_1, ..., l_k)$, where $\forall j \in \{1, .., k\}$ $l_j \in \mathcal{V} \cup \mathcal{C}$.*
- *Each E_i is an expression of the form $l_j \neq l_k$ where $l_j \in \mathcal{V}$ and $l_k \in \mathcal{V} \cup \mathcal{C}$.*

$R_1(u_1), ..., R_n(u_n)$ are called the relational atoms *of Q, and $E_1, ..., E_m$ are the* disequality atoms. *We require that every variable that appears in a disequality E_i appears also in a relational atom u_j of the query. $ans(u_0)$ is called the rule head, and is denoted $head(Q)$, while $R_1(u_1), ..., R_n(u_n)$, $E_1, ..., E_m$ is the rule body, denoted $body(Q)$. The variables and constants in the body of Q are called the arguments of Q, and are denoted $Var(Q)$ and $Const(Q)$ respectively. The variables appearing in $head(Q)$ are called the* distinguished *variables of Q, and each of them must also appear in $body(Q)$. Finally, if $head(Q)$ is of arity 0, we say that Q is* boolean.

We use $CQ^{\neq}$ to denote the set of all conjunctive queries with disequalities; the subset of queries in which no disequality expression appears is denoted by CQ.

We say that a conjunctive query is *complete* (following [21]) if it explicitly specifies all disequalities between pairs of distinct variables (or a variable and a constant) occurring in it. More formally:

Definition 2.2. *A query $Q \in CQ^{\neq}$ is* complete *if (1) for every pair of distinct variables $x, y \in Var(Q)$, the query contains the disequality $x \neq y$ (or $y \neq x$), (2) for every $x \in Var(Q)$ and $c \in Const(Q)$, it contains $x \neq c$.*

We use $cCQ^{\neq}$ to denote the class of all *complete* conjunctive queries with disequalities.

Example 2.3. *In the following example, x, y are variables and c is a constant. Q, Q' are both in $CQ^{\neq}$ but only Q' is complete (i.e. $Q' \in cCQ^{\neq}$):*

Q: ans(x,y):=R(x,y),S(y,c), $x \neq y$, $y \neq c$
Q': ans(x,y):=R(x,y),S(y,c), $x \neq y$, $y \neq c$, $x \neq c$

We next also recall the definition of union of conjunctive queries, as follows:

Definition 2.4. *A union of conjunctive queries with disequalities is an expression of the form $Q = Q_1 \cup Q_2 \cup ... \cup Q_m$ where for each $i \in \{1, ..., m\}$, $Q_i \in CQ^{\neq}$, and for each $i, j \in \{1, ..., m\}$ $head(Q_i)$ and $head(Q_j)$ are of the same relation. We say that each Q_i is an* adjunct *of the query Q. The set of adjuncts of Q is denoted by $Adj(Q)$.*

We use UCQ to denote the class of union of conjunctive queries with no disequalities, use $UCQ^{\neq}$ where the adjuncts may include disequalities, and $cUCQ^{\neq}$ where each adjunct is complete.

We further intuitively extend the definitions of Var and $Const$ for unions of conjunctive queries, such that $Var(Q) = \bigcup_{Q_i \in Adj(Q)} Var(Q_i)$ and $Const(Q) = \bigcup_{Q_i \in Adj(Q)} Const(Q_i)$

Example 2.5. *Figure 1 depicts the query Q_{union}, which is in $cUCQ^{\neq}$. Intuitively, its first adjunct Q_1 looks for pairs of different tuples (since it requires $x \neq y$) where the value of the first (second) attribute in the first tuple equals the value of the second (first) attribute in the second tuple, while the second adjunct Q_2 seeks for a single tuples, where the values in the two attributes are equal. For both adjuncts, the head relation is a tuple which contains a single variable x.*

$$\begin{array}{lll}
\mathbf{Q_1}: & ans(x) := & R(x,y), R(y,x), x \neq y \\
\mathbf{Q_2}: & ans(x) := & R(x,x) \\
\mathbf{Q_{union}}: & Q_1 \cup Q_2 & \\
\mathbf{Q_{conj}}: & ans(x) := & R(x,y), R(y,x)
\end{array}$$

Figure 1: Example Queries

Assignments and Query Results. We formally define the notion of *assignments* of database tuples to query relational atoms, and use it to define query results, as follows:

Definition 2.6. *An* assignment *α of a query $Q \in CQ^{\neq}$ to a database instance D is a mapping of the relational atoms of Q to tuples in D that respects relation names and induces a mapping over arguments, i.e. if a relational atom $R(l_0, ..., l_n)$ is mapped to a tuple $R(a_0, ..., a_n)$ then we say that l_i is mapped to a_i (denoted $\alpha(l_i) = a_i$) and we require that a variable l_i will not be mapped to multiple distinct values, and a constant l_i will be mapped to itself. We also require the induced mapping over arguments to respect disequalities appearing in Q.*

Given such an assignment α, we define $\alpha(head(Q))$ as the tuple obtained from $head(Q)$ by replacing each occurrence of a variable l_i by $\alpha(l_i)$, and a constant l_i by its value.

The set of all such assignments for a database instance D is denoted $A(Q, D)$, and the result of evaluating a query $Q \in CQ^{\neq}$, denoted $Q(D)$, is then defined as $\bigcup_{\alpha \in A(Q,D)}$ $\alpha(head(Q))$. For $Q = Q_1 \cup Q_2 \cup ... \cup Q_n$, we further denote $Q(D) = \bigcup_{i=1,...,n} Q_i(D)$, and $A(Q, D) = \bigcup_{i=1,...,n} A(Q_i, D)$ Finally, we define $A(t, Q, D) = \{\alpha \in A(Q, D) \mid t = \alpha(head(Q))\}$ as the set of all assignments yielding t.

A	B	Provenance
a	a	s_1
a	b	s_2
b	a	s_3
b	b	s_4

Table 2: Relation R

A	Provenance
a	$s_2 \cdot s_3 + s_1$
b	$s_3 \cdot s_2 + s_4$

Table 3: Relation ans

Example 2.7. *Consider the query Q_{union} depicted in Figure 1, and the database D whose single relation R is depicted in Table 2 (ignore for now the Provenance column). There are two possible assignments for the first adjunct Q_1, the first(second) maps the atom R(x,y) to the tuple (a,b) (the tuple (b,a)), the atom R(y,x) to the tuple (b,a) (the tuple (a,b)), and the head to the tuple (a) (the tuple (b)); there are two possible assignments for Q_2, mapping its single atom either to (a,a) or (b,b), and its head to (a) or (b) respectively. $A((a), Q_{union}, D)$, for instance, contains exactly the first assignment of Q_1 and the first of Q_2, since these are the only assignments that map the head to (a).*

2.2 Query Containment and Homomorphisms

We next recall the definitions of query containment and equivalence, as well as the definition of homomorphism between queries.

Definition 2.8. *Given queries Q and Q' over a database schema R, we say that Q is contained in a query Q' (denoted*

$Q \subseteq Q'$) if for every database instance D of the schema R, $Q(D) \subseteq Q'(D)$. We say that Q, Q' are equivalent (denoted $Q \equiv Q'$) if $Q \subseteq Q'$ and $Q' \subseteq Q$.

EXAMPLE 2.9. *Consider the queries Q_2 and Q_{conj} in Fig. 1. It is easy to verify that $Q_2 \subseteq Q_{conj}$.*

Homomorphisms between Queries. We next define homomorphism between conjunctive queries, as follows:

DEFINITION 2.10. *Let $Q, Q' \in CQ^{\neq}$; a homomorphism $h : Q \to Q'$ is a mapping from the atoms of Q to those of Q', inducing a mapping on the instances of arguments occurring in these atoms, such that:*

1. *If an atom a uses a relation R (resp. is a disequality) so does (is) $h(\mathsf{a})$.*
2. *The head of Q is mapped to the head of Q'.*
3. *If one instance of a variable $x \in Var(Q)$ is mapped to an instance of $y \in Var(Q')$ then all instances of x are mapped to instances of y.*
4. *Each occurrence of a constant $c \in Const(Q)$ must be mapped to an occurrence of c.*

EXAMPLE 2.11. *Reconsider Q_2, Q_{conj} from figure 1. There exists a homomorphism from Q_{conj} to Q_2 mapping the two atoms of Q_{conj} to the single atom of Q_2. The induced homomorphism over variables maps both x, y to x. Note that there is no homomorphism from Q_2 to Q_{conj} because x will necessarily be mapped to both x and y.*

2.3 Provenance of Query Results

We use here provenance annotations that are elements from the *provenance semiring* [19], as these annotations allow to capture the computational process inflicted by a given query evaluation. A semiring is an algebraic structure with two operations: addition and multiplication [23]; the provenance semiring is defined as $(\mathbb{N}[X], +, \cdot, 0, 1)$, where $\mathbb{N}[X]$ is the set of all polynomials with natural numbers as coefficients, over some pre-defined set of variables X. Each multiplicative term in a polynomial is called a monomial. For instance, the monomials of $x + x \cdot y \cdot z$ are x and $x \cdot y \cdot z$.

An $\mathbb{N}[X]$-relation P maps each tuple t to a provenance annotation $P(t)$. We consider here input $\mathbb{N}[X]$-relations that are *abstractly-tagged* [19], meaning that $\forall t \ P(t) \in X$, and $\forall t \neq t' \ P(t) \neq P(t')$ (the case of non-abstractly-tagged input relations is discussed in Section 6). For instance, reconsider the relation depicted in Table 2: the provenance annotations are depicted in the last column; all annotations are distinct and the relation is abstractly-tagged.

In the sequel we will use the notations s (or s_i for some index i) for provenance annotations, p (or p_i) for polynomials, and m (or m_i) for monomials.

Given a database instance D and a query $Q \in UCQ^{\neq}$, we adopt the definition of [20] for the provenance of each tuple in the query result[1]:

DEFINITION 2.12. *Given a query $Q \in CQ^{\neq}$, a database instance D of $\mathbb{N}[X]$-relations, a tuple $t \in Q(D)$, we define the provenance of t w.r.t. Q and D, denoted by $P(t, Q, D)$, as follows: $P(t, Q, D) = \sum_{\alpha \in A(t,Q,D)} \prod_{R_i \in body(Q)} P(\alpha(R_i))$, where $A(t, Q, D)$ is the set of all assignments yielding t as a result (See Definition 2.6). If $Q = Q_1 \cup Q_2 \cup ... \cup Q_n$ then $P(t, Q, D) = \sum_{i=1,...,n} P(t, Q_i, D)$.*

In the case of boolean queries, where the only possible tuple in the result is the empty tuple, we may use the notation $P(Q, D)$ for the provenance of this single tuple, and call it the provenance of Q for D.

EXAMPLE 2.13. *Reconsider the relation R depicted in Table 2, this time along with the provenance annotation of its tuples, and reconsider the query Q_{union} from Figure 1.*

The output relation ans along with the provenance annotations assigned to its tuples is depicted in Table 3. Consider for example the output tuple (a); it is computed from Q_1 as the result of an assignment that maps the first (second) atom to the tuples annotated by s_2 (s_3 resp.), and from Q_2 as the result of an assignment that maps its single atom to s_1. Consequently its provenance is computed as $s_2 \cdot s_3 + s_1$. Similarly, (b) is obtained due to the assignment for Q_1 that assigns s_3 to the first atom and s_2 to the second one, and the assignment that assigns the single atom of Q_2 to s_4.

We note that different queries which are equivalent with respect to containment may yield different provenance polynomials for the same database and result tuple.

EXAMPLE 2.14. *Reconsider Q_{conj} as well as Q_{union} from Figure 1. These two queries are equivalent. However, consider the provenance of the output tuple (a) for Q_{conj} and the relation R depicted in Table 2. This output tuple is yielded by an assignment that maps the first atom of Q_{conj} to s_2 and the second atom to s_3, and by an assignment that maps both atoms to s_1. Thus, the provenance of (a) for Q_{conj} is $s_2 \cdot s_3 + s_1 \cdot s_1$. The provenance of tuple (b) would be $s_3 \cdot s_2 + s_4 \cdot s_4$. In both cases, the provenance polynomial is different than the polynomial yielded by query Q_{union}.*

Note. The above definition of provenance polynomials indicates the existence of an *isomorphism* between the assignments and the monomials of the polynomial, when the monomials are written in a form where all coefficients and exponents equal 1 (This isomorphism simply maps each assignment to the monomial it yields). For ease of presentation and to have the isomorphism between assignments and monomials clearly visible, we will assume that the polynomial is indeed written in this form. When this complicates the reading, we will also give (in brackets) the "compact" expression with coefficients and exponents. For instance, in Example 2.14 above we wrote $s_2 \cdot s_3$ (rather than $s_3 \cdot s_2$) as one of the monomials in the provenance of the tuple (a) to reflect the fact that the corresponding assignment mapped the first atom of Q_{conj} to a tuple annotated with s_2, and the second to a tuple annotated with s_3. Similarly, we wrote the other monomial as $s_1 \cdot s_1$ to reflect that the corresponding assignment mapped both the first and second atoms to s_1.

So far we have repeated existing definitions for general concepts. The following subsection presents definitions which are specific to the context of core provenance.

2.4 An Order Relation

We next define an order relation over the provenance polynomials, that will allow comparing different provenance polynomials yielded by equivalent queries. The order relation we define below reflects relative "terseness" of a provenance of a given tuple in the query result, and will be used in the

[1] The definition in [20] for UCQ is an adaptation of the original definition in [19] for SPJU queries.

sequel for defining minimal provenance. Intuitively, we will say that $p \leq p'$ if there is an injective mapping of the monomials in p to the monomials in p', such that each monomial is mapped to a monomial in which it is contained. Recall the correlation between the polynomials and assignments of atoms to tuples, where each monomial corresponded to such an assignment; consequently, if a query Q (Q') generates a tuple t with provenance p (p'), then $p \leq p'$ means that each assignment of Q is contained within an assignment of Q', thus the provenance of Q is more "terse". Formally,

Definition 2.15. *Given two monomials $m = s_1 \cdot ... \cdot s_k$ and $m' = s'_1 \cdot ... \cdot s'_{k'}$ we say that $m \leq m'$ if there exists an injective mapping $I_m : \{1, ..., k\} \to \{1, ..., k'\}$ such that $s_i = s'_{I_m(i)}$ for every $1 \leq i \leq k$.*

Given two polynomials *$p = \sum_{i=1,...,n} m_i$ and $p' = \sum_{i=1,...,n'} m'_i$, we say that $p \leq p'$ if there exists an injective mapping I_p from monomials in p to monomials in p' such that $m_i \leq I_p(m_i)$ for every $1 \leq i \leq n$.*

We say that $p = p'$ if it holds that $p \leq p'$ and $p' \leq p$.

Finally, we say that $p < p'$ if $p \leq p'$ but not $p = p'$.

Example 2.16. *Let $p_1 = s_1 \cdot s_2 + s_3 + s_3$ and $p_2 = s_1 \cdot s_2 \cdot s_2 + s_2 \cdot s_3 + s_3 \cdot s_4 + s_5$, then $p_1 < p_2$. To see that this holds, observe that we can map the monomial $s_1 \cdot s_2$ to $s_1 \cdot s_2 \cdot s_2$, map the first occurrence of s_3 to $s_2 \cdot s_3$, and the second occurrence of s_3 to $s_3 \cdot s_4$; but in the other direction we cannot e.g. map the monomial $s_3 \cdot s_4$ of p_2 to any monomial of p_1.*

We then utilize the order relation over provenance polynomials to define an order relation over *queries* which are equivalent in the "standard" sense.

Definition 2.17. *For two equivalent queries Q, Q' we say that $Q \subseteq_P Q'$ if for every abstractly-tagged database instance D, and for every tuple t in the result of evaluating Q (Q') over D, $P(t, Q, D) \leq P(t, Q', D)$. We say that $Q \equiv_P Q'$ if $Q \subseteq_P Q'$ and $Q' \subseteq_P Q$; we say that $Q \subset_P Q'$ if $Q \subseteq_P Q'$ but not $Q \equiv_P Q'$.*

Example 2.18. *Reconsider the queries Q_{conj}, Q_{union} from Figure 1. As observed above, Q_{union} and Q_{conj} are equivalent. We will show in the sequel (Theorem 3.11) that $Q_{union} \subset_P Q_{conj}$. But already now we can observe that in some cases Q_{conj} entails more provenance (and thus that the queries are not equivalent in terms of provenance): in examples 2.13 and 2.14 we have shown that for the output tuple (a) the provenance of Q_{union} was $s_2 \cdot s_3 + s_1$ while the provenance of Q_{conj} was $s_2 \cdot s_3 + s_1 \cdot s_1$, which is strictly larger.*

We now define the notion of minimal query in terms of provenance, as follows:

Definition 2.19 (Minimal Provenance). *We say that a query Q is provenance-minimal (p-minimal) in a class of queries $\mathcal{C}$ [2] if $\forall Q' \in \mathcal{C}\ Q' \equiv Q \Rightarrow Q \subseteq_P Q'$.*

The provenance yielded by a p-minimal query is called *core provenance*.

We then define the PROVENANCE-MINIMIZATION problem, with respect to a given class $\mathcal{C}$ of queries, as follows: given a query $Q \in \mathcal{C}$, find, if exists, a query $Q' \in \mathcal{C}$ such that $Q' \equiv Q$ and Q' is p-minimal. We study in the sequel PROVENANCE-MINIMIZATION for the query classes $CQ, CQ^{\neq}, cCQ^{\neq}, UCQ^{\neq}$.

Note. We note that "standard" query minimization ([9, 22, 26]) aims at minimizing the query *length*, i.e. the number of relational atoms in the query (equivalently, minimizing the number of joins [22]). We show in the sequel that such minimization does not necessarily minimize the provenance. Another important note is that (unlike for comparison based on length) it can be the case that two equivalent queries will be incomparable with respect to our order relation, and we will show such an example in the sequel (See Theorem 3.5).

3. MINIMIZING CONJUNCTIVE QUERIES

In this section we focus on solving PROVENANCE-MINIMIZATION for conjunctive queries. We start by presenting a homomorphism theorem that will be of use in all of our p-minimization algorithms. We then consider first p-minimization within the most general class, $CQ^{\neq}$. Then we turn to analyze subclasses of $CQ^{\neq}$ which are of interest, namely CQ and $cCQ^{\neq}$.

3.1 Homomorphisms and Provenance

In the work on "standard" minimization, where one aims at minimizing the number of joins in a given query, homomorphism mappings between queries play an important role. Evidently, this is also the case for provenance minimization, albeit in a different manner. We first recall the homomorphism theorems from [9, 21]:

Theorem 3.1 ([9, 21]). *Given two queries $Q \in CQ$ ($Q \in cCQ^{\neq}$) and $Q' \in CQ$ ($Q' \in CQ^{\neq}$), there exists a homomorphism from Q' to Q if and only if $Q \subseteq Q'$.*

We note that the requirement on Q with disequalities to be complete is essential; otherwise, homomorphism implies inclusion but the converse fails in general, as indicated by the following example:

Example 3.2 (Adapted from [22]). *Consider the following queries:*

$Q : ans() := R(x, y), R(y, z), x \neq z$
$Q' : ans() := R(x, y), x \neq y$

Q is included in Q': if there exists x, y, z such that $R(x, y)$, $R(y, z)$ and $x \neq z$, then either $x \neq y$ or $y \neq z$, in both cases there exists a tuple $R(w, t)$ with $w \neq t$ which thus matches Q'. But there is no homomorphism from Q' to Q as if we can either map $R(x, y)$ occurring in Q' to $R(x, y)$ or $R(y, z)$ occurring in Q; but in both cases the disequality $x \neq y$ could not be mapped to $x \neq z$.

We shall see below that the property exemplified in Example 3.2 will have interesting implications on the (in)existence of p-minimal queries equivalent to a given query in $CQ^{\neq}$.

We next present a counterpart theorem that will form the basis for our provenance minimization algorithms.

Theorem 3.3. *Given two equivalent queries $Q, Q' \in CQ^{\neq}$, if there exists a homomorphism h from Q' to Q that is* surjective *on relational atoms, then $Q \subseteq_P Q'$.*

Proof sketch. Let D be an input database, and let $t \in Q(D)$ (thus $t \in Q'(D)$). Let $p = \sum_i m_i$ ($p' = \sum_i m'_i$) be the provenance of t with respect to Q (Q') and D. We claim that $p \leq p'$. For that we need to show an injective mapping I mapping each monomial m_i in p and some monomial $m'_{i'}$ in p', such that $m_i \leq m'_{i'}$.

[2] $\mathcal{C}$ may e.g. be CQ, $CQ^{\neq}$, etc.

We define I as follows: for m_i, there exists an assignment α to Q that yielded it. Given the surjective mapping $h : Q' \to Q$, define β as the assignment which assigns each relational atom a' of Q', $\alpha(h(a'))$. It is straightforward to show that β is indeed a satisfying assignment for Q' (proof omitted). Since h is surjective on relational atoms, every tuple t' assigned to k different atoms by α is assigned to at least k atoms by β. Finally, let $I(m_i)$ be $m'_{i'}$, the monomial yielded by the assignment of β to Q'. Then each variable $P(t')$ which appears k times in m_i appears at least k times in $m'_{i'}$, and by definition of the order relation, $m_i \leq m'_{i'}$.

We still have to show that the mapping I between the monomials is injective. For that, consider m_i, m_j s.t. $i \neq j$. Let α_i (resp. β_i), α_j (resp. β_j) be the assignments that yielded these monomials, respectively, in the evaluation of Q (resp. Q'). Different monomials in our presentation (See note at the end of Section 2.4) are always yielded by different assignments; thus α_i and α_j must be distinct assignments; that is only possible if they assign some relational atom a in Q different tuples, let us call them t_i and t_j respectively. Then there exists some relational atom a' in Q' such that $h(a') = a$ (by the surjectiveness of h). By the construction of β_i and β_j, β_i (β_j) assigns a' the tuple $\alpha_i(h(a')) = t_i$ ($\alpha_j(h(a')) = t_j$), and thus they are distinct assignments, which yield different monomials. That means m_i and m_j are always mapped by I to different monomials in p', i.e. I is injective. □

We note that unlike the case of queries inclusion (Theorem 3.1), Theorem 3.3 requires the homomorphism to be surjective. The following example shows that a non-surjective mapping does not guarantee an order on the provenance.

Example 3.4. *Consider the following two simple boolean queries Q, Q':*

Q: ans() := R(x),R(y)
Q': ans() := R(x)

There exists a (trivial) homomorphism from Q' to Q, but no surjective one (since Q' has less atoms). For the simple unary relation R bearing a single tuple $R(a)$ with provenance s, the provenance of the (boolean) result of evaluating Q over R is $s{\cdot}s$, while for Q' the provenance is $s < s{\cdot}s$.

In contrast, mapping both atoms of Q to the one of Q' is a surjective homomorphism from Q to Q'. It is easy to see that the provenance of Q' is smaller.

In the sequel we consider different subclasses of $CQ^{\neq}$, and, where possible, utilize the above results for minimizing the provenance of a given query within the sub-class.

3.2 General Conjunctive Queries

We start with general queries in $CQ^{\neq}$. Note that [22] has shown that for standard minimization, for each query $Q \in CQ^{\neq}$ there exists a minimal equivalent query in $CQ^{\neq}$. Interestingly, this is not the case for p-minimal queries, as the following theorem holds:

Theorem 3.5. *There exists a query $Q \in CQ^{\neq}$ such that Q has no p-minimal equivalent query in $CQ^{\neq}$.*

Proof sketch. Consider the queries Q_{noPmin} and Q_{alt} from Figure 2. We can show that Q_{noPmin} is equivalent to Q_{alt}. But none of these queries is p-minimal, as the following Lemma holds:

A	B	Provenance
a	b	s_1
b	a	s_2
a	a	s_3

Table 4: Relation R in Database D

A	B	Provenance
a	b	s'_1
b	c	s'_2
c	a	s'_3
a	a	s'_4

Table 5: Relation R in Database D'

$$\begin{array}{lll}
\mathbf{Q_{noPmin}}: & ans() := & R(x_1,x_2), R(x_2,x_3), R(x_3,x_4), \\
 & & R(x_4,x_5), R(x_5,x_1), S(x_1), x_1 \neq x_2 \\
\mathbf{Q_{alt}}: & ans() := & R(x_1,x_2), R(x_2,x_3), R(x_3,x_4), \\
 & & R(x_4,x_5), R(x_5,x_1), S(x_1), x_1 \neq x_3 \\
\mathbf{Q_{alt2}}: & ans() := & R(x_1,x_2), R(x_2,x_3), R(x_3,x_4), \\
 & & R(x_4,x_5), R(x_5,x_1), S(x_1), x_1 \neq x_4 \\
\mathbf{Q_{alt3}}: & ans() := & R(x_1,x_2), R(x_2,x_3), R(x_3,x_4), \\
 & & R(x_4,x_5), R(x_5,x_1), S(x_1), x_1 \neq x_5
\end{array}$$

Figure 2: Queries in $CQ^{\neq}$

Lemma 3.6. *It neither holds that $Q_{noPmin} \subseteq_P Q_{alt}$, nor that $Q_{alt} \subseteq_P Q_{noPmin}$.*

Proof. Consider a database D with the Relation R depicted in Table 4, and the relation S consisting of a single tuple (a) annotated with s_0. The provenance expression for Q_{noPmin} is
$s_1{\cdot}s_2{\cdot}s_1{\cdot}s_2{\cdot}s_3{\cdot}s_0 + s_1{\cdot}s_2{\cdot}s_3{\cdot}s_1{\cdot}s_2{\cdot}s_0 + s_1{\cdot}s_2{\cdot}s_3{\cdot}s_3{\cdot}s_3{\cdot}s_0$
$(= 2{\cdot}(s_1)^2{\cdot}(s_2)^2{\cdot}s_3{\cdot}s_0 + s_1{\cdot}s_2{\cdot}(s_3)^3{\cdot}s_0.)$

For Q_{alt}, the provenance expression is

$s_3{\cdot}s_1{\cdot}s_2{\cdot}s_1{\cdot}s_2{\cdot}s_0 + s_3{\cdot}s_1{\cdot}s_2{\cdot}s_3{\cdot}s_3{\cdot}s_0$
$(= (s_1)^2{\cdot}(s_2)^2{\cdot}s_3{\cdot}s_0 + s_1{\cdot}s_2{\cdot}(s_3)^3{\cdot}s_0.)$

which is strictly smaller. In contrast, for a database D' with Relation R as depicted in Table 5, and relation S as before, the provenance obtained for Q_{noPmin} is

$s'_1{\cdot}s'_2{\cdot}s'_3{\cdot}s'_4{\cdot}s'_4{\cdot}s_0$

and for Q_{alt} it is

$s'_1{\cdot}s'_2{\cdot}s'_3{\cdot}s'_4{\cdot}s'_4{\cdot}s_0 + s'_4{\cdot}s'_1{\cdot}s'_2{\cdot}s'_3{\cdot}s'_4{\cdot}s_0$

which is strictly greater. □

The following lemma concludes the proof by showing that no other equivalent query in $CQ^{\neq}$ has a minimal provenance:

Lemma 3.7. *There exist database instances D, D' for which there is no query $Q \in CQ^{\neq}$ equivalent to Q_{noPmin} (and to Q_{alt}) such that both $P(Q, D) \leq P(Q_{noPmin}, D)$ and $P(Q, D') \leq P(Q_{alt}, D')$ hold.*

Proof sketch. The databases D, D' are as in the proof of Lemma 3.6 above. Assume by contradiction the existence of such Q. It must contain exactly 5 atoms in which R occur and exactly one atom in which S occur (otherwise if it contains less it will not be equivalent and if it contains more it will have greater provenance for one of the Databases). Since all the variables of Q_{noPmin} can be different, there must be at least 5 different variables in Q; it follows that Q must be of the form

$$Q : ans() := R(z_1, z_2), R(z_2, z_3), R(z_3, z_4), \\ R(z_4, z_5), R(z_5, z_1), S(z1), E$$

Where E is some conjunction of disequalities between $z_1, ..., z_5$. However, for most choices of E there exists a database instance for which the query result is not equivalent to Q_{noPmin}. There are only two other queries equivalent to Q_{noPmin} of the form of Q, depicted in figure 2: Q_{alt2}, whose provenance is equivalent to that of Q_{alt} on D, D'; and Q_{alt3}, whose provenance is equivalent to that of Q_{noPmin} on D, D'. Thus, neither of them could be Q, implying that such Q does not exist. □

This concludes the proof of Theorem 3.5. □

The construction that we have used in the proof is inspired by [17]. Interestingly, note that in the above proof we constructed two equivalent queries, both minimal in the standard sense (but not isomorphic). This settles an open problem posed in [22], indicating the correctness of the following Lemma:

Lemma 3.8. *There exists a query $Q \in CQ^{\neq}$ such that Q has no minimal equivalent query which is* unique up to isomorphism *in $CQ^{\neq}$.*

While there are queries for which no p-minimal equivalent query in $CQ^{\neq}$ exists, we will see in Section 4 that there always exists an equivalent p-minimal query in $UCQ^{\neq}$. But before that, let us consider other restricted classes in which the p-minimal query can be found. The conclusions for these restricted classes will gradually lead us towards the general solution for the overall p-minimal query in $UCQ^{\neq}$.

3.3 Conjunctive Queries without Disequalities

We consider CQ, i.e. the class of conjunctive queries with no disequalities. The following theorem shows that in CQ, the "standard" minimal query is also p-minimal. As a result, a standard minimization algorithm (such as in [9]) can be used here to obtain the p-minimal query.

Theorem 3.9. *Let $Q \in CQ$. Then Q is a minimal query (in the standard sense) iff Q is p-minimal in CQ.*

Proof sketch. Assume that Q is minimal, and consider some query $Q' \in CQ$ equivalent to Q. By the equivalence between Q and Q', and from theorem 3.1 there is a homomorphism $h : Q' \to Q$. We can show that this homomorphism is surjective on relational atoms, and thus from theorem 3.3, $Q \subseteq_P Q'$. □

Following the lines of [16] (for standard query minimization), we define the decision problem corresponding to p-minimization in CQ as follows: *given two queries $Q, Q' \in CQ$, where Q' is a sub-query of Q, decide if Q' is the p-minimal equivalent of Q.* Note that *verifying* p-minimality of a query Q is a restricted case. The following is a corollary of Theorem 3.9, resulting from the known hardness of conjunctive query minimization [16]:

Corollary 3.10. *The decision problem corresponding to* `PROVENANCE-MINIMIZATION` *in CQ is DP-Complete* [3].

In the context of "standard" minimization, the obtained minimal query is minimal also among all equivalent queries in $UCQ^{\neq}$ [9]. Interestingly, this is not the case for provenance minimization, as the following theorem holds.

Theorem 3.11. *There exists a query $Q \in CQ$ such that Q is p-minimal in CQ but there exists an equivalent query $Q' \in UCQ^{\neq}$, such that $Q' \subset_P Q$.*

Proof. Consider again the queries Q_{conj} and Q_{union} from Figure 1. It is easy to see that there is no surjective homomorphism from Q_{conj} to any of its sub-queries, and thus (following theorem 3.9) Q_{conj} is p-minimal within CQ. To show that $Q_{union} \subseteq_P Q_{conj}$, observe that every assignment to Q_{union} that yields every output tuple t is either an assignment to its first adjunct, Q_1, or its second argument Q_2. In the former case the same assignment to Q_{conj} will also yield the output tuple t, and in the latter case this assignment will have a counterpart assignment to Q_{conj} that maps two atoms to the same tuple. In both cases the provenance monomials yielded by Q_{conj} are greater or equal, and thus the provenance polynomial of each tuple for Q_{conj} is greater or equal, i.e. $Q_{union} \subseteq_P Q_{conj}$.

Last, $Q_{union} \subset_P Q_{conj}$ because there exists a database instance and output tuple for which Q_{union} yields strictly less provenance (See Example 2.18). □

The above theorem shows that even for queries which are p-minimal in CQ, there may be equivalent queries in $UCQ^{\neq}$ with smaller provenance. In the next section we show that equivalent p-minimal queries in $UCQ^{\neq}$ always exist, and explain how to compute them.

3.4 Complete Conjunctive Queries with Disequalities

The last sub-class of $CQ^{\neq}$ that we study here, and will be very useful when we consider $UCQ^{\neq}$ in the next section, is $cCQ^{\neq}$, the class of *complete* conjunctive queries with disequalities. Recall that completeness means here that for every pair of distinct variable x and argument l occurring in the query, $x \neq l$ appears in the query. In this case, we will show that like in CQ, p-minimization within the $cCQ^{\neq}$ class is equivalent to standard minimization. However, we will further show that in contrast to Theorem 3.11 the p-minimal in $cCQ^{\neq}$ is also the *overall p-minimal in $UCQ^{\neq}$*. Another difference from CQ is the complexity of (p-)minimization in $cCQ^{\neq}$, which we prove to be polynomial.

The following theorem holds:

Theorem 3.12. *Given a query $Q \in cCQ^{\neq}$, Q is minimal in the standard sense iff Q is p-minimal in $cCQ^{\neq}$, and that is iff Q is p-minimal in $UCQ^{\neq}$. The (p-)minimal equivalent of any $Q \in cCQ^{\neq}$ can be computed in time polynomial in the size of Q.*

Proof sketch. Recall the proof sketch of theorem 3.9, showing that in CQ minimality and p-minimality are the same. Note that this proof was based only on theorems 3.1 and 3.3. Since those theorems also hold for $cCQ^{\neq}$, it can be proved in a similar manner that p-minimality and minimality in $cCQ^{\neq}$ are the same. Thus, if we minimize Q we will get Q', the equivalent of Q which is p-minimal in $cCQ^{\neq}$. The p-minimality of Q' in $UCQ^{\neq}$ (thus making it the general p-minimal equivalent of Q) is a corollary of Theorem 4.6 shown in the sequel.

[3] DP is the class of decision problems that can be expressed as the intersection of an NP and a co-NP problem [16].

To show that Q' can be computed efficiently, we use the following Lemma:

LEMMA 3.13. *A query $Q \in cCQ^{\neq}$ is (p-)minimal if and only if Q does not contain duplicated relational atoms (i.e. two atoms of the same relation with the same arguments).*

Thus, a simple minimization algorithm for $cCQ^{\neq}$ will compare every two relational atoms and remove the duplicates. This can be done in PTIME. $\square$

4. MINIMIZING UNIONS

We next consider minimizing the provenance for $UCQ^{\neq}$, union of conjunctive queries with disequalities. As observed above, resorting to $UCQ^{\neq}$ in search of a query with minimal provenance may be necessary even if the input query is guaranteed to be a conjunctive query. Our study of minimization of queries in this class is done in two steps, as follows. We start by introducing the *canonical rewriting* of a query, which is essentially its rewriting as a union of *complete* queries; we then introduce a minimization algorithm, based on canonical rewritings, and we show that its result is a p-minimal equivalent of the original query.

4.1 Canonical Rewritings

Recall the queries in Example 2.18: the queries Q_{conj}, Q_{union} are equivalent, but Q_{union} has less provenance. Intuitively, this is because Q_{union} employs a by-case reasoning: each combination of equalities and disequalities among each pair of variables is dealt with separately, using distinct conjunctive queries. Intuitively, such a by-case reasoning is implemented by the canonical rewriting of a query. We next first define a *possible completion* of a query, which corresponds to a particular "case", and then the canonical rewriting, which encapsulates all the possible "cases".

DEFINITION 4.1. *Let $Q \in CQ^{\neq}$. A* possible completion *of Q is a query in $cCQ^{\neq}$ obtained by splitting the arguments $Var(Q) \cup Const(Q)$ into m disjoint subsets, $V_1, ..., V_m$, such that (1) each set contains at most one constant, and (2) if Q contains the disequality $l_i \neq l_j$, then l_i and l_j are not in the same subset. Then, for each subset V_i that contains a constant c, every occurrence of an argument from this subset is replaced in the relational atoms of Q by c; for each subset V_i without a constant, all the occurrences of arguments within it are replaced in Q by some new variable v_i; and finally all the disequalities in Q are removed, and instead a disequality $v_i \neq v_j$ for each two new variables v_i, v_j and a disequality $v_i \neq c$ for each new variable v_i and $c \in Const(Q)$ are added.*

A canonical rewriting *of Q, denoted by $Can(Q)$, is then a query in $cUCQ^{\neq}$ where each possible completion of Q is isomorphic to exactly one adjunct in Q, and vice versa.*

Finally, an extended canonical rewriting *with respect to a set of constants C which is a superset of $Const(Q)$, is defined in a similar manner, except that every possible completion is obtained by splitting $Var(Q) \cup C$ into disjoint sets as before. We denote it $Can(Q, C)$. In particular, $Can(Q) = Can(Q, Const(Q))$.*

EXAMPLE 4.2. *Consider for example the query*

$Q : ans(x, y) := R(x, y), x \neq a, x \neq y$

Its canonical rewriting with respect to $C = \{a, b\}$ (which contains $Const(Q)$) is

$$\begin{array}{lll}
\multicolumn{3}{l}{Can(Q, C) := Q_1 \cup Q_2 \cup Q_3 \cup Q_4 \cup Q_5} \\
Q_1 : & ans(v_1, a) := & R(v_1, a), v_1 \neq a, v_1 \neq b \\
Q_2 : & ans(v_1, b) := & R(v_1, b), v_1 \neq a, v_1 \neq b \\
Q_3 : & ans(v_1, v_2) := & R(v_1, v_2), v_1 \neq a, v_1 \neq v_2, v_2 \neq a, \\
 & & v_1 \neq b, v_2 \neq b \\
Q_4 : & ans(b, a) := & R(b, a) \\
Q_5 : & ans(b, v_2) := & R(b, v_2), v_2 \neq a, v_2 \neq b
\end{array}$$

We next study some basic properties of the (extended) canonical rewriting. First, it is easy to show that it preserves the query results:

THEOREM 4.3. *For any query $Q \in CQ^{\neq}$, and any superset C of the constants in Q, $Q \equiv Can(Q, C)$.*

Furthermore, we show that the output tuples of the canonical representation bear the same provenance as those of the original query:

THEOREM 4.4. *For every query $Q \in CQ^{\neq}$ and any superset C of the constants in Q, $Q \equiv_P Can(Q, C)$.*

PROOF SKETCH. We use the following lemma:

LEMMA 4.5. *For every query $Q \in CQ^{\neq}$, and two different adjuncts Q_1, Q_2 in $Can(Q)$, if α is an assignment for Q_1, then it is not valid for Q_2.*

Now, let D be an input database and t a tuple in the query result. It follows from the lemma that each assignment to $Can(Q)$ satisfies exactly one of its adjuncts, and thus has the same provenance as its corresponding assignment for Q, i.e. $P(t, Can(Q), D) \leq P(t, Q, D)$. Conversely an assignment to Q decides (in)equalities between variables (and constants), consequently corresponds to an assignment to exactly one adjunct of $Can(Q)$, thus $P(t, Q, D) \leq P(t, Can(Q), D)$; since this holds for every tuple t in the result set, it follows that $Can(Q) \equiv_P Q$. $\square$

4.2 Provenance Minimization Algorithm

We are now ready to present an algorithm for finding a p-minimal equivalent of a query in $UCQ^{\neq}$. By doing so, we will prove the following theorem:

THEOREM 4.6. *Given a query $Q \in UCQ^{\neq}$, there exists a p-minimal equivalent query $Q' \in UCQ^{\neq}$; Q' may be found in time exponential in the size of Q.*

We next give the algorithm, analyze it and then show that its exponential complexity is inevitable.

Algorithm. The minimization method $MinProv$ is depicted in Algorithm 1 and operates in 3 steps. First (step I), it replaces each adjunct of the input query Q by its canonical rewriting with respect to the full set of constants in Q, obtaining a query Q_I as a result. Then, in step II, each of the adjuncts is minimized separately, using any efficient algorithm for minimization of $cCQ^{\neq}$ queries (such as the simple algorithm depicted in the proof of theorem 3.12), obtaining Q_{II} as the union of minimized queries. Finally, in step III the algorithm checks every pair of adjuncts in Q_{II} for query containment: recall that since Q_{II} is a canonical rewriting,

Algorithm 1: MinProv

```
Input: Query $Q$
Output: An equivalent p-minimal query
// Step I
$Q_I \leftarrow \phi$ ;
$C \leftarrow Const(Q)$ ;
foreach $Q_i \in Adj(Q)$ do
    foreach $Q_{i,j}$ adjunct in $Can(Q_i, C)$ do
        Add $Q_{i,j}$ to $Q_I$;
    end
end
// Step II
$Q_{II} \leftarrow \phi$ ;
foreach $Q_i \in Adj(Q_I)$ do
    Minimize $Q_i$ and add to $Q_{II}$
end
// Step III
$Q_{III} \leftarrow Q_{II}$ ;
foreach $Q_i \in Adj(Q_{III})$ do
    foreach $Q_j \in Adj(Q_{III})$ other than $Q_i$ do
        if $Q_j \subseteq Q_i$ then
            Remove $Q_j$ from $Q_{III}$
        end
    end
end
return $Q_{III}$ ;
```

each adjunct in it is complete, therefore Theorem 3.1 holds and checking for containment amounts to checking for the existence of a homomorphism. Whenever a contained query is found, it is omitted by the algorithm. Q_{III} (and the algorithm output) is then Q_{II} without the contained adjuncts.

EXAMPLE 4.7. *Consider the queries depicted in figure 3. We apply the $MinProv$ algorithm on $\hat{Q}$, showing the obtained intermediate queries. Step I of the algorithm computes the canonical rewriting of (the single adjunct of) $\hat{Q}$, resulting in $\hat{Q}_I$ Then, in step II, each of the adjuncts of $\hat{Q}_I$ are (p-)minimized. Recall (Lemma 3.13) that a complete conjunctive query is p-minimal if and only if it has no duplicated relational atoms; the only adjunct which is not p-minimal is $\hat{Q}_1$. We remove the duplicated atoms to obtain its p-minimal equivalent, $\hat{Q}_{min1}$. Consequently the result of step II of the algorithm is $\hat{Q}_{II}$. Finally, in step III of the algorithm we eliminate contained adjuncts. In this case we can easily verify that $\hat{Q}_2, \hat{Q}_3$ and $\hat{Q}_4$ are contained in $\hat{Q}_{min1}$. There are no other containments between adjuncts; thus the result of step III (and the algorithm output), $\hat{Q}_{III}$, is the union of the remaining adjuncts, $\hat{Q}_{min1}$ and $\hat{Q}_5$.*

Correctness. First, it is easy to see that $Q \equiv MinProv(Q)$. We next show that the query that is computed is indeed the minimal one.

PROPOSITION 4.8. *For every two equivalent queries, $Q, Q' \in UCQ^{\neq}$, $MinProv(Q) \subseteq_P Q'$, i.e. $MinProv$ returns a p-minimal query equivalent to Q.*

PROOF. We say in the sequel that a query Q is complete w.r.t. a set of constants C if it is complete and additionally contains a disequality $v \neq c$ for every $v \in Var(Q)$, $c \in C$. The following lemma is an adaptation of a Theorem in [26], adapted to consider queries with disequalities.

$\mathbf{\hat{Q}}$:	$ans() := R(x,y), R(y,z), R(z,x)$
$\mathbf{\hat{Q}}_I$:	$\hat{Q}_1 \cup \hat{Q}_2 \cup \hat{Q}_3 \cup \hat{Q}_4 \cup \hat{Q}_5$
$\hat{Q}_1$:	$ans() := R(v_1,v_1), R(v_1,v_1), R(v_1,v_1)$
$\hat{Q}_2$:	$ans() := R(v_1,v_2), R(v_2,v_1), R(v_1,v_1), v_1 \neq v_2$
$\hat{Q}_3$:	$ans() := R(v_1,v_2), R(v_2,v_2), R(v_2,v_1), v_1 \neq v_2$
$\hat{Q}_4$:	$ans() := R(v_1,v_1), R(v_1,v_2), R(v_2,v_1), v_1 \neq v_2$
$\hat{Q}_5$:	$ans() := R(v_1,v_2), R(v_2,v_3), R(v_3,v_1), v_1 \neq v_2,$ $v_2 \neq v_3, v_1 \neq v_3$
$\mathbf{\hat{Q}}_{II}$:	$\hat{Q}_{min1} \cup \hat{Q}_2 \cup \hat{Q}_3 \cup \hat{Q}_4 \cup \hat{Q}_5$
$\hat{Q}_{min1}$:	$ans() := R(v_1,v_1)$
$\mathbf{\hat{Q}}_{III}$:	$\hat{Q}_{min1} \cup \hat{Q}_5$

Figure 3: Example for $MinProv$, Step by Step

LEMMA 4.9. *Let $Q \in cCQ^{\neq}, Q' \in UCQ^{\neq}$, such that Q is complete with respect to the constants in Q', $Q \subseteq Q'$ iff there exists an adjunct $Q'_i \in Adj(Q')$ s.t. $Q \subseteq Q'_i$.*

Now, let $C = Const(MinProv(Q)) \cup Const(Q')$. We next show that $Can(MinProv(Q), C) \subseteq_P Q'$; since $Can(MinProv(Q), C) \equiv_P MinProv(Q)$, this will prove proposition 4.8. Let Q_i be one of the adjuncts in $Can(MinProv(Q), C)$. Since we removed contained adjuncts in step III of the algorithm, and thus every two adjuncts must differ in some equality/disequality, every monomial contributed to the provenance by Q_i must be unique. Since $Can(MinProv(Q), C) \subseteq Can(Q', C)$, it holds that $Q_i \subseteq Can(Q', C)$. Thus, by lemma 4.9 (Q_i is complete on all the constants in Q') there exists some adjunct Q'_j in $Can(Q', C)$ s.t. $Q_i \subseteq Q'_j$. The containment and completeness on the other direction also holds, thus there exists some adjunct Q_k in $Can(MinProv(Q), C)$ s.t. $Q'_j \subseteq Q_k$. We get that $Q_i \subseteq Q'_j \subseteq Q_k$, and since we removed all containments from $MinProv(Q)$, $Q_i \equiv Q_k$, thus $Q_i \subseteq Q'_j \subseteq Q_i$, i.e. $Q_i \equiv Q'_j$.

Since at step II we minimized each adjunct, Q_i is p-minimal. In particular, $Q_i \subseteq_P Q'_j$. Thus, for every input database D and $t \in Q(D)$, Q_i contributes at most the provenance that Q'_j does. This is true for every such adjunct Q_i in $Can(MinProv(Q), C)$, thus $P(t, Can(MinProv(Q), C), D) \leq P(t, Can(Q', C), D)$. Since it is true for every t and D, $Can(MinProv(Q), C) \subseteq_P Can(Q', C)$. □

Complexity Upper Bound. The complexity of $MinProv$ is EXPTIME in the size of its input query: step I of the algorithm replaces each adjunct of the original query Q with its canonical rewriting, which is of exponential size. Then we may have an exponential number of adjuncts, but each of them is of *polynomial size* (by definition of the canonical rewriting). Then the overall complexity of steps II and III is (respectively) polynomial and exponential in the maximal size (number of relational atoms) of an adjunct in their input query.

This concludes the proof of Theorem 4.6 above.

Complexity Lower bound. The EXPTIME complexity of $MinProv$ is inevitable as the following theorem holds:

THEOREM 4.10. *For every $n \in \mathbb{N}$ there exists a query Q_n of size $\Theta(n)$ such that any p-minimal equivalent of Q_n is of size $2^{\Omega(n)}$.*

A	B	Provenance
a	a	s_1
a	b	s_2
b	a	s_3
b	c	s_4
c	a	s_5

Table 6: Relation R in $\hat{D}$

The proof is based on constructing a query Q_n which is of the form
$ans() := R_1(x_1, y_1), R_1(y_1, x_1), ..., R_n(x_n, y_n), R_n(y_n, x_n)$. We can show that a p-minimal equivalent of Q_n must consider exponentially many "cases" of (dis)equalities, hence is of exponential size.

It is interesting to compare this result to "standard" minimization, where the output is at most of the same size of the input (albeit minimization still requires EXPTIME).

5. DIRECT PROVENANCE MINIMIZATION

In the previous sections we have discussed the computation of a p-minimal query equivalent to a given query. As explained above, this p-minimal query realizes the core provenance for the query and every database instance. However, in some real-life scenarios we may wish not to change the query that is evaluated, since it is determined by optimizers and governed by evaluation time considerations. In such cases it is preferable to compute the core provenance directly from the provenance of tuples in the query result. This section shows how such a direct computation can be done. We present the next theorem.

Theorem 5.1. *Let $Q \in UCQ^{\neq}$ be a query, D be some input database, t a tuple in $Q(D)$ and let $p = P(t, Q, D)$. Let p' be the provenance of t as computed by Q', a p-minimal query equivalent to Q.*

- *Given p, (and without any other information on Q, Q', D or t), p' can be computed in time polynomial in the size of p, up to the number of occurrences of equal monomials (corresponding to its coefficient).*
- *Given p, D, t and $Const(Q)$, and without any other information on Q or Q', p' can be exactly computed (*including *the correct number of occurrences of equal monomials) in time exponential in the size of p.*

The rest of the section is dedicated to proving Theorem 5.1. Our proof technique is to (i) closely examine the three steps of Algorithm $MinProv$ (given in Section 4), (ii) study the effect that each step has on the provenance polynomials, and (iii) prove that we can simulate this effect by directly working on the polynomials.

Several notations will be used in the sequel: given a query $Q \in UCQ^{\neq}$, an input instance D and a tuple t in $Q(D)$, we use $Q_I, Q_{II}, Q_{III} \in cUCQ^{\neq}$ to denote the queries obtained after step I,II,III respectively of applying $MinProv$ on Q; and the corresponding provenance polynomials $p_I = P(t, Q_I, D)$, $p_{II} = P(t, Q_{II}, D)$, $p_{III} = P(t, Q_{III}, D)$.

5.1 Step I of MinProv

Recall that the first step of $MinProv$ replaces each adjunct with its canonical rewriting. By theorem 4.4, this does not affect the provenance. We exemplify this next.

Example 5.2. *Recall the query $\hat{Q}$ from Example 4.7 (depicted in Figure 3). Let $\hat{D}$ be a database with R relation as in table 6. The provenance of the boolean result of $\hat{Q}$ (and thus also $\hat{Q}_I$) when evaluated on $\hat{D}$ is given next (each row contains the monomials yielded by one adjunct of $\hat{Q}_I$):*

$s_1 \cdot s_1 \cdot s_1 +$
$s_2 \cdot s_3 \cdot s_1 +$
$s_3 \cdot s_1 \cdot s_2 +$
$s_1 \cdot s_2 \cdot s_3 +$
$s_2 \cdot s_4 \cdot s_5 + s_4 \cdot s_5 \cdot s_2 + s_5 \cdot s_2 \cdot s_4$

5.2 Step II of MinProv

We next consider the impact of step II on the provenance polynomial. The following Lemma holds:

Lemma 5.3. *For each monomial m_i in p_I there is a monomial m_j in p_{II} which contains every variable s of m_i exactly once.*

Example 5.4. *Consider again $\hat{Q}_I$ from Example 4.7 and recall that only the first adjunct $\hat{Q}_1$ was further minimized. Accordingly, only the monomial contributed by the first adjunct is changed (compare to the provenance in Example 5.2), and we get the following provenance (for database $\hat{D}$).*

$s_1 +$
$s_2 \cdot s_3 \cdot s_1 +$
$s_3 \cdot s_1 \cdot s_2 +$
$s_1 \cdot s_2 \cdot s_3 +$
$s_2 \cdot s_4 \cdot s_5 + s_4 \cdot s_5 \cdot s_2 + s_5 \cdot s_2 \cdot s_4$

5.3 Step III of MinProv

Step III of Algorithm $MinProv$ eliminates adjuncts which are contained in other adjuncts. It turns out that removing *contained* adjuncts from the query, causes the elimination of *containing* monomials from the provenance polynomial. We first show this, then study the effect of step III on the number of occurrences of the remaining monomials.

Lemma 5.5. *For every monomial m_i of p_{III}, it holds that $m_i \in p_{II}$ and there is no $m_j \in p_{II}$ such that $m_j < m_i$*

Proof sketch. It is easy to see that each monomial in p_{III} is also in p_{II} (since at step III the algorithm only removes adjuncts). Assume by contradiction the existence of such m_i, m_j. Let α_i and α_j be the assignments corresponding to m_i and m_j respectively, and Q_i, Q_j the adjuncts on which α_i and α_j are applied, respectively. By definition of the order relation, there exists an injective mapping $I : m_j \to m_i$ from each multiplicand of m_j to an identical multiplicand in m_i. We define $h : Q_j \to Q_i$ s.t. every atom R_k^j in Q_j is mapped by h to R_l^i in Q_i, where $I(P(\alpha_j(R_k^j))) = P(\alpha_i(R_l^i))$. That means every atom in Q_j is mapped to an atom in Q_i which is assigned the same tuple. We can show that h is a homomorphism, thus $Q_i \subseteq Q_j$. Then, we can further show that there exists no homomorphism from Q_i to Q_j, thus $Q_i \subset Q_j$ and Q_i would have been eliminated in step III of $MinProv$. □

Our analysis so far leads to the following corollary:

Corollary 5.6. *Up to number of equal monomial occurrences, p_{III} may be obtained from p by removing all the multiple occurrences of the same variable in each monomial, and omitting every monomial m_i in p that includes some monomial m_j in p.*

This transformation can easily be done in PTIME, proving part (1) of theorem 5.1. We next prove part (2).

Computing the number of monomial occurrences. We first characterize the number of occurrences of each remaining monomial. The following Lemma holds:

LEMMA 5.7. *Let m_i be a monomial of p_{II} that was yielded by an adjunct of Q, that has k automorphisms. If m_i appears in p_{III} then k monomials equal to m_i appear in p_{III}.*

PROOF SKETCH. Let α_i be the assignment corresponding to m_i. Then we define k different assignments as follows. For every automorphism τ we define the assignment $\alpha_\tau = \alpha_i \circ \tau$. We can show that: (i) $\alpha_\tau \in A(t, Q, D)$, i.e. it is an assignment for Q that yields t; (ii) for $\tau \neq \tau'$, $\alpha_\tau \neq \alpha_{\tau'}$, and (iii) an assignment contributes a monomial equal to m_i to the provenance of t iff it is in $\{\alpha_\tau | \tau$ is an automorphism of $Q\}$. □

EXAMPLE 5.8. *Recall that according to Example 4.7, the second, third and forth adjuncts of the result of step II, $\hat{Q}_{II}$, are eliminated, since they were contained in the first adjunct, $\hat{Q}_{min1}$. Consider again the provenance polynomial corresponding to $\hat{Q}_{II}$, which appears in Example 5.4. Observe that the monomials yielded by the eliminated adjuncts* strictly *contain the monomial yielded by the first adjunct. For instance, $s_1 < s_2 \cdot s_3 \cdot s_1$. The three equivalent monomials yielded by the last adjunct remain intact. Indeed, the last adjunct has exactly 3 automorphisms.*

s_1+
$s_2 \cdot s_4 \cdot s_5 + s_4 \cdot s_5 \cdot s_2 + s_5 \cdot s_2 \cdot s_4$

Computing the number of automorphisms for the adjunct corresponding to a monomial m (denoted $Aut(m)$) is straightforward to do (in EXPTIME) if we have the corresponding query adjunct in hand. But interestingly, we can do it without seeing the query, if we are given the input database and the set of *constants* used by the query (if any are used). This will conclude the proof of Theorem 5.1. Formally,

LEMMA 5.9. *Given a monomial m in polynomial $P(t, Q, D)$ on database D and the output tuple t, and given $Const(Q)$, where Q is a p-minimal query, $Aut(m)$ can be computed in time exponential in m.*

6. GENERAL ANNOTATIONS

So far we have limited our discussion to abstractly-tagged databases, i.e. databases in which every tuple has a distinct annotation. However, we note that in some cases, input relations are not abstractly-tagged, for instance if they are the result of some previous computation. We can show that since our core provenance captures the essence of the computation in terms of the participating tuples and regardless of their annotations, the p-minimal query remains the same. Intuitively, the proof is by replacing all annotations in the database with unique new annotations, and then show that the order between provenance polynomials in the result of query evaluation remains intact after these replacements.

THEOREM 6.1. *Let Q be a p-minimal query (w.r.t abstractly tagged databases), within the class $\mathcal{C}$. Then for every equivalent query $Q' \in \mathcal{C}$, a* non-*abstractly-tagged database D and a tuple $t \in Q(D)$, $P(t, Q, D) \leq P(t, Q', D)$.*

In contrast, our results for direct provenance computation (Section 5) do not go through to non-abstractly-tagged database. We can show that such direct computation is impossible *without knowing the query*. Intuitively, this is because two occurrences of the same annotation can either come from the same tuple or different tuples.

THEOREM 6.2. *There exist a database D and a tuple t, for which there exist two non-equivalent queries Q, Q', such that $Const(Q) = Const(Q')$, $t \in Q(D) \cap Q'(D)$ and $P(t, Q, D) = P(t, Q', D)$, but $P(t, MinProv(Q), D) \neq P(t, MinProv(Q'), D)$.*

PROOF. Let D have a single relation R, with the tuples (a) and (b), both annotated with s and let t be (a). Consider the following queries:

$Q : ans(x) := R(x), R(y), x \neq y$
$Q' : ans(x) := R(x), R(x)$

The provenance of both queries is $P(t, Q, D) = P(t, Q', D) = s \cdot s$. It is easy to see that Q is p-minimal in $UCQ^{\neq}$ w.r.t. abstractly-tagged databases, thus by the above theorem, $P(t, Q, D) = P(t, MinProv(Q), D) = s \cdot s$. However, the p-minimal equivalent of Q', $MinProv(Q')$, is $ans(x) := R(x)$ (obtained by removing the duplicated atom) and thus $P(t, MinProv(Q'), D) = s \neq s \cdot s$. □

7. RELATED WORK

Management of provenance information has been extensively studied in the database literature, in multiple branches. In [6, 12, 7, 4, 5] and other works, the authors define different provenance management techniques (e.g. Why Provenance [7], Trio Provenance [4]). In [19] the authors suggest the use of a *provenance semiring*, where provenance expressions are represented as polynomials detailing the computation of the different output tuples. [20] has shown that Trio provenance can be represented as polynomials with no exponents, while Why provenance is a set of sets and can be captured as a polynomial with no exponents or coefficients. Since the current paper focuses on the core computational processes, we found the model of provenance semirings best suited to our needs. We mention in this context that [5] studies problem of identifying "maximal" provenance, namely the (possibly infinite) union of all provenance expressions obtained for equivalent queries; in contrast we study here the "minimal" provenance, for provenance polynomials. Defining and studying "maximal" provenance for provenance polynomials is an interesting future research. Our analysis of "direct" identification of the core provenance (Section 5) also sheds light on the provenance polynomials obtained, and shows that there are some "core coefficients" in the polynomials, appearing in the provenance of tuples for every equivalent query. The core provenance is more minimal than Trio provenance: containing monomials are not omitted in Trio; core provenance also gives more details on the core *computation* than both Trio and Why Provenance, due to the coefficients reflecting this core computation (in Why provenance there are no coefficients, in Trio provenance the coefficients may be different among equivalent queries). As mentioned in the Introduction, provenance information is extensively used as input to various data management tools (e.g. trust assessment, update propagation [18, 25, 30, 29]). A particular challenge here arises from the size of provenance, both in terms of storage and efficiency of the operation of the tools [10, 27].

We believe that the identification of core provenance can be the basis of optimization techniques that reduce the size of the input given to these tools, helping to alleviate these challenges.

Query minimization in a "classic" sense, that aims at minimizing the number of joins has been well investigated for the different classes discussed here. Minimization of conjunctive queries was first suggested in [9], and was extended to union of conjunctive queries in [26]. Minimization of conjunctive queries with inequalities was studied in [22]. We have compared and contrasted our results with classic minimization for all of these classes, except that we restricted the discussion to queries with disequalities ($\neq$) instead of general inequalities ($<, \leq, ...$). Query inclusion and minimization were further studied for queries of various classes, for instance, aggregation, bag semantics and arithmetic comparisons [13, 14, 2, 8]; identifying the core provenance for queries of these classes, and for queries with general inequalities ($<, \leq, ...$), is an interesting future work. Practically efficient heuristics (e.g. [28, 11]) are known for "standard" minimization of queries. For CQ queries, such algorithms will also serve as heuristics for p-minimization; for other classes their adaptation to p-minimization is an intriguing challenge.

Last, we note that our reference to the minimal provenance as "core provenance" was inspired by the notion of core of universal solutions in Data Exchange [16, 3, 15, 24], with the intuition that the core provenance be a part of the computational process, for every "solution". Studying the connection between the core in data exchange and the core provenance is an interesting future research.

8. CONCLUSIONS

We have studied in this paper the core of provenance information, namely the part of provenance that appears in the result of evaluating every query equivalent to the query in hand. We have considered query classes of varying expressive power, and studied the problem of computing an equivalent query that realizes the provenance core for the query in hand, and for every input database instance. We have analyzed the existence of such a query in the different classes, and have given algorithms for computing it where it exists. We have further presented algorithms that compute the core provenance directly from the provenance information of tuples in the query result, and showed its applicability even for cases when the input query is absent.

In the previous section we have mentioned several intriguing research directions such as exploiting the compact size of the core provenance for practical applications, and studying the nature of core provenance in the presence of additional query constructs. Due to the importance of provenance in general, and of provenance polynomials in particular, we believe them to be promising subjects for future work. Additionally, we have restricted our study to Conjunctive Queries with disequalities, and unions thereof. Considering provenance minimization for more expressive query languages, e.g. Datalog, is an additional further research challenge. Last, we note that different *physical query plans* for the same query may result in different provenance, and finding the p-minimal among those is an intriguing research challenge.

Acknowledgements. We are grateful to Georg Gottlob for providing us with a counter-example for non-unique minimal query for $CQ^{\neq}$, which helped us to prove the non-existence of a p-minimal query for this class (Theorem 3.5 and Lemma 3.8). We are also grateful to the anonymous reviewers of PODS for their comments that have helped to improve the final version of this paper.

9. REFERENCES

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

[2] F. N. Afrati, C. Li, and P. Mitra. On containment of conjunctive queries with arithmetic comparisons. In *EDBT*, 2004.

[3] M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Relational and XML Data Exchange*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[4] O. Benjelloun, A.D. Sarma, A.Y. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *VLDB J.*, 17:243–264, 2008.

[5] Deepavali Bhagwat, Laura Chiticariu, Wang-Chiew Tan, and Gaurav Vijayvargiya. An annotation management system for relational databases. In *VLDB*, 2004.

[6] P. Buneman, J. Cheney, and S. Vansummeren. On the expressiveness of implicit provenance in query and update languages. *ACM Trans. Database Syst.*, 33(4), 2008.

[7] P. Buneman, S. Khanna, and W.C. Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.

[8] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *PODS*, 1998.

[9] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, 1977.

[10] A. Chapman, H. V. Jagadish, and P. Ramanan. Efficient provenance storage. In *SIGMOD Conference*, 2008.

[11] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In *ICDT*, 1997.

[12] J. Cheney, S. Chong, N. Foster, M. I. Seltzer, and S. Vansummeren. Provenance: a future history. In *Proc. of OOPSLA*, 2009.

[13] S. Cohen. Equivalence of queries combining set and bag-set semantics. In *PODS*, pages 70–79, 2006.

[14] S. Cohen, W. Nutt, and Y. Sagiv. Rewriting queries with arbitrary aggregation functions using views. *ACM Trans. Database Syst.*, 31(2):672–715, 2006.

[15] G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. On reconciling data exchange, data integration, and peer data management. In *PODS*, 2007.

[16] R. Fagin, P.G. Kolaitis, and L. Popa. Data exchange: getting to the core. *ACM Trans. Database Syst.*, 30:174–210, 2005.

[17] G. Gottlob, 2010. private communication.

[18] T. J. Green, G. Karvounarakis, Z. Ives, and V. Tannen. Update exchange with mappings and provenance. In *VLDB*, 2007.

[19] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Proc. of PODS*, 2007.

[20] T.J. Green. Containment of conjunctive queries on annotated relations. In *ICDT*, 2009.

[21] G. Karvounarakis and V. Tannen. Conjunctive queries and mappings with unequalities. Technical report, 2008.

[22] A. Klug. On conjunctive queries containing inequalities. *J. ACM*, 35(1), 1988.

[23] W. Kuich. Semirings and formal power series. In *Handbook of Formal Languages*, 1997.

[24] L. Libkin and C. Sirangelo. Open and closed world assumptions in data exchange. In *Description Logics*, 2009.

[25] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1), 2010.

[26] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, 1980.

[27] Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34, September 2005.

[28] I. Tatarinov and A. Halevy. Efficient query reformulation in peer data management systems. In *SIGMOD*, 2004.

[29] S. Vansummeren and J. Cheney. Recording provenance for sql queries and updates. *IEEE Data Eng. Bull.*, 30(4):29–37, 2007.

[30] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao. Efficient querying and maintenance of network provenance at internet-scale. In *SIGMOD Conference*, 2010.