Query Languages for Data Exchange: Beyond Unions of Conjunctive Queries

Marcelo Arenas
Dept. of Computer Science
PUC Chile
marenas@ing.puc.cl

Pablo Barceló
Dept. of Computer Science
Univ. of Chile
pbarcelo@dcc.uchile.cl

Juan Reutter
Dept. of Computer Science
PUC Chile
jlreutte@puc.cl

ABSTRACT

The class of unions of conjunctive queries (UCQ) has been shown to be particularly well-behaved for data exchange; its certain answers can be computed in polynomial time (in terms of data complexity). However, this is not the only class with this property; the certain answers to any DATALOG program can also can be computed in polynomial time. The problem is that both UCQ and DATALOG do not allow negated atoms, as adding an unrestricted form of negation to these languages yields to intractability.

In this paper, we propose a language called DATALOG$^{\mathbf{C}(\neq)}$ that extends DATALOG with a restricted form of negation, and study some of its fundamental properties. In particular, we show that the certain answers to a DATALOG$^{\mathbf{C}(\neq)}$ program can be computed in polynomial time (in terms of data complexity), and that every union of conjunctive queries with at most one inequality or negated relational atom per disjunct, can be efficiently rewritten as a DATALOG$^{\mathbf{C}(\neq)}$ program in the context of data exchange. Furthermore, we show that this is also the case for a syntactic restriction of the class of unions of conjunctive queries with at most two inequalities per disjunct. This syntactic restriction is given by two conditions that are optimal, in the sense that computing certain answers becomes intractable if one removes any of them. Finally, we provide a thorough analysis of the combined complexity of computing certain answers to DATALOG$^{\mathbf{C}(\neq)}$ programs and other related query languages. In particular, we show that this problem is EXPTIME-complete for DATALOG$^{\mathbf{C}(\neq)}$, even if one restricts to conjunctive queries with single inequalities, which is a fragment of DATALOG$^{\mathbf{C}(\neq)}$ by the result mentioned above. Furthermore, we show that the combined complexity is CONEXPTIME-complete for the case of conjunctive queries with k inequalities, for every $k \geq 2$.

1. INTRODUCTION

Data exchange is the problem of computing an instance of a *target* schema, given an instance of a *source* schema and a specification of the relationship between source and target data. Although data exchange is considered to be an old database problem, its theoretical foundations have only been laid out very recently by the seminal work of Fagin, Kolaitis, Miller and Popa [8]. Both the study of data exchange and schema mappings have become an active area of research during the last years in the database community (see e.g. [8, 9, 4, 7, 16, 12, 17, 11]).

In formal terms, a data exchange setting is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, where $\mathbf{S}$ is a *source* schema, $\mathbf{T}$ is a *target* schema, and Σ_{st} is a mapping defined as a set of *source-to-target* dependencies of the form $\forall \bar{x}(\phi_{\mathbf{S}}(\bar{x}) \rightarrow \exists \bar{y} \psi_{\mathbf{T}}(\bar{x}, \bar{y}))$, where $\phi_{\mathbf{S}}$ and $\psi_{\mathbf{T}}$ are conjunctions of relational atoms over $\mathbf{S}$ and $\mathbf{T}$, respectively (some studies have also included target constraints, but here we focus on data exchange settings without dependencies over $\mathbf{T}$). Given a source instance I, the goal in data exchange is to materialize a target instance J that is a *solution* for I, that is, J together with I must conform to the mapping Σ_{st}.

An important issue in data exchange is that the existing specification languages usually do not completely determine the relationship between source and target data and, thus, there may be many solutions for a given source instance. This immediately raises the question of which solution should be materialized. Initial work on data exchange [8] has identified a class of "good" solutions, called *universal* solutions. In formal terms, a solution is universal if it can be homomorphically embedded into every other solution. It was proved in [8] that for the class of data exchange settings studied in this paper, a particular universal solution called the *canonical* universal solution can be computed in polynomial time. It is important to notice that in this result the complexity is measured in terms of the size of the source instance, and the data exchange specification Σ_{st} is assumed to be fixed. Thus, this result is stated in terms of *data* complexity [19].

A second important issue in data exchange is query answering. Queries in the data exchange context are posed over the target schema, and –given that there may be many solutions for a source instance– there is a general agreement in the literature that their semantics should be defined in terms of *certain* answers [13, 1, 14, 8]. More formally, given a data exchange setting $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ and a query Q over $\mathbf{T}$, a tuple $\bar{t}$ is said to be a certain answer to Q over I under $\mathcal{M}$, if $\bar{t}$ belongs to the evaluation of Q over every possible solution J for I under $\mathcal{M}$.

The definition of certain answers is highly non-effective, as it involves computing the intersection of (potentially) infinitely many sets. Thus, it becomes particularly important to understand for which classes of relevant queries, the certain answers can be computed efficiently. In particular, it becomes relevant to understand whether it is possible to compute the certain answers to any of

ICDT 2009, March 23–25, 2009, Saint Petersburg, Russia.

these classes by using some materialized solution. Fagin, Kolaitis, Miller, and Popa [8] have shown that this is the case for the class of union of conjunctive queries (UCQ); the certain answers to each union of conjunctive queries Q over a source instance I can be computed in polynomial time by directly posing Q over the canonical universal solution for I. Again, it is important to notice that this result is stated in terms of data complexity, that is, the complexity is measured in terms of the size of the source instance, and both the data exchange specification Σ_{st} and the query Q are assumed to be fixed.

The good properties of UCQ for data exchange can be completely explained by the fact that unions of conjunctive queries are preserved under homomorphisms. But this is not the only language that satisfies this condition, as queries definable in DATALOG, the recursive extension of UCQ, are also preserved under homomorphisms. Thus, DATALOG is as good as UCQ for data exchange purposes. In particular, the certain answers to a DATALOG program Π over a source instance I, can be computed efficiently by first materializing the canonical universal solution J for I, and then evaluating Π over J (since the data complexity of a DATALOG program is polynomial).

Unfortunately, both UCQ and DATALOG keeps us in the realm of the positive, while most database query languages are equipped with negation. Thus, the first goal of this paper is to investigate what forms of negation can be added to DATALOG while keeping all the good properties of DATALOG, and UCQ, for data exchange. It should be noticed that this is not a trivial problem, as there is a trade-off between expressibility and complexity in this context. On the one hand, one would like to have a query language expressive enough to be able to pose interesting queries in the data exchange context. But, on the other hand, it has been shown that adding an unrestricted form of negation to DATALOG (or even to conjunctive queries) yields to intractability of the problem of computing certain answers [1, 8]. In this respect, the following are our main contributions.

- We introduce a query language called DATALOG$^{\mathbf{C}(\neq)}$ that extends DATALOG with a restricted form of negation, and that has the same good properties for data exchange as DATALOG. In particular, we prove that the certain answers to a DATALOG$^{\mathbf{C}(\neq)}$ program Π over a source instance I can be computed by evaluating Π over the canonical universal solution for I. As a corollary, we obtain that computing certain answers to a DATALOG$^{\mathbf{C}(\neq)}$ program can be done in polynomial time (in terms of data complexity).

- To show that DATALOG$^{\mathbf{C}(\neq)}$ can be used to express interesting queries in the data exchange context, we prove that every union of conjunctive queries with at most one inequality or negated relational atom per disjunct, can be efficiently expressed as a DATALOG$^{\mathbf{C}(\neq)}$ program in the context of data exchange.

- It follows from the previous result that the certain answers to every union of conjunctive queries with at most one inequality or negated relational atom per disjunct, can be computed in polynomial time (in terms of data complexity). Although this corollary is not new (it is a simple extension of a result in [8]), the use of DATALOG$^{\mathbf{C}(\neq)}$ in the context of data exchange opens the possibility of finding new tractable classes of query languages with negation. In fact, we also use DATALOG$^{\mathbf{C}(\neq)}$ to find a tractable fragment of the class of conjunctive queries with two inequalities.

  It is known that for the class of conjunctive queries with inequalities, the problem of computing certain answers is CONP-complete [1, 8] (in terms of data complexity). In fact, it has been shown that the intractability holds even for the case of two inequalities [18]. However, very little is known about tractable fragments of these classes. In this paper, we provide a syntactic restriction for the class of unions of conjunctive queries with at most two inequalities per disjunct, and prove that every query conforming to it can be expressed as a DATALOG$^{\mathbf{C}(\neq)}$ program in the context of data exchange. It immediately follows that the data complexity of computing certain answers to a query conforming to this restriction is polynomial.

  The syntactic restriction mentioned above is given by two conditions. We conclude this part of the investigation by showing that these conditions are optimal for tractability, in the sense that computing certain answers becomes intractable if one removes any of them. It should be noticed that this gives a new proof of the fact that the problem of computing certain answer to a conjunctive query with two inequalities is CONP-complete.

The study of the complexity of computing certain answers to DATALOG$^{\mathbf{C}(\neq)}$ programs will not be complete if one does not consider the notion of *combined* complexity. Although the notion of data complexity has shown to be very useful in understanding the complexity of evaluating a query language, one should also study the complexity of this problem when none of its parameters is considered to be fixed. This corresponds to the notion of combined complexity introduced in [19], and it means the following in the context of data exchange. Given a data exchange setting $\mathcal{M}$, a query Q over the target and a source instance I, one considers I as well as Q and $\mathcal{M}$ as part of the input when computing the certain answers to Q over I under $\mathcal{M}$. In this paper, we study this problem and establish the following results.

- We show that the combined complexity of the problem of computing certain answers to DATALOG$^{\mathbf{C}(\neq)}$ programs is EXPTIME-complete, even if one restricts to the class of conjunctive queries with single inequalities (which is a fragment of DATALOG$^{\mathbf{C}(\neq)}$ by the result mentioned above). This refines a result in [12] that shows that the combined complexity of the problem of computing certain answers to *unions* of conjunctive queries with at most one inequality per disjunct is EXPTIME-complete.

- We also consider the class of conjunctive queries with an arbitrary number of inequalities per disjunct. More specifically, we show that the combined complexity of the problem of computing certain answers is CONEXPTIME-complete for the case of conjunctive queries with k inequalities, for every $k \geq 2$.

- One of the reasons for the high combined complexity of the previous problems is the fact that if data exchange settings are not considered to be fixed, then one has to deal with canonical universal solutions of exponential size. A natural way to reduce the size of these solutions is to focus on the class of LAV data exchange settings [14], which are frequently used in practice.

For the case of DATALOG$^{\mathbf{C}(\neq)}$ programs, the combined complexity is inherently exponential, and thus focusing on LAV settings does not reduce the complexity of computing certain answers. However, we show in the paper that if one focus on LAV settings, then the combined complexity is considerably lower for the class of conjunctive queries with inequalities. More specifically, we show that the combined complexity goes down to NP-complete for the case of conjunctive queries with single inequalities, and to Π_2^p-complete for the case of conjunctive queries with k inequalities, for every $k \geq 2$.

Organization of the paper. In Section 2, we introduce the terminology used in the paper. In Section 3, we define the syntax and semantics of DATALOG$^{\mathbf{C}(\neq)}$ programs. In Section 4, we study some of the fundamental properties of DATALOG$^{\mathbf{C}(\neq)}$ programs concerning complexity and expressiveness. In Section 5, we study a syntactic restriction that leads to tractability of the problem of computing certain answers for unions of conjunctive queries with two inequalities. In Section 6, we provide a thorough analysis of the combined complexity of computing certain answers to DATALOG$^{\mathbf{C}(\neq)}$ programs and other related query languages. Concluding remarks are in Section 7.

2. BACKGROUND

A *schema* $\mathbf{R}$ is a finite set $\{R_1, \ldots, R_k\}$ of relation symbols, with each R_i having a fixed arity $n_i > 0$. Let $\mathbf{D}$ be a countably infinite domain. An *instance* I of $\mathbf{R}$ assigns to each relation symbol R_i of $\mathbf{R}$ a finite n_i-ary relation $R_i^I \subseteq \mathbf{D}^{n_i}$. The *domain* $\text{dom}(I)$ of instance I is the set of all elements that occur in any of the relations R_i^I. We often define instances by simply listing the tuples attached to the corresponding relation symbols.

We assume familiarity with first-order logic (FO) and DATALOG. In this paper, CQ is the class of conjunctive queries and UCQ is the class of unions of conjunctive queries. If we extend these classes by allowing inequalities or negation (of relational atoms), then we use superscripts $\neq$ and $\neg$, respectively. Thus, for example, CQ$^{\neq}$ is the class of conjunctive queries with inequalities, and UCQ$^{\neg}$ is the class of unions of conjunctive queries with negation. As usual in the database literature, we assume that every query Q in UCQ$^{\neq,\neg}$ is *safe*: (1) if Q_1 and Q_2 are disjuncts of Q, then Q_1 and Q_2 have the same free variables, (2) if Q_1 is a disjunct of Q and $x \neq y$ is a conjunct of Q_1, then x and y appear in some non-negated relational atoms of Q_1, (3) if Q_1 is a disjunct of Q and $\neg R(\bar{x})$ is a conjunct of Q_1, then every variable in $\bar{x}$ appears in a non-negated relational atom of Q_1.

2.1 Data exchange settings and solutions

As is customary in the data exchange literature, we consider instances with two types of values: constants and nulls [8, 9]. More precisely, let $\mathbf{C}$ and $\mathbf{N}$ be infinite and disjoint sets of constants and nulls, respectively, and assume that $\mathbf{D} = \mathbf{C} \cup \mathbf{N}$. If we refer to a schema $\mathbf{S}$ as a *source* schema, then we will assume that for every instance I of $\mathbf{S}$, it holds that $\text{dom}(I) \subseteq \mathbf{C}$. On the other hand, if we refer to a schema $\mathbf{T}$ as a *target* schema, then for every instance J of $\mathbf{T}$, it holds that $\text{dom}(J) \subseteq \mathbf{C} \cup \mathbf{N}$. Slightly abusing notation, we also use $\mathbf{C}$ to denote a built-in unary predicate such that $\mathbf{C}(a)$ holds if and only if a is a constant, that is $a \in \mathbf{C}$.

A *data exchange setting* is a tuple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, where $\mathbf{S}$ is a source schema, $\mathbf{T}$ is a target schema, $\mathbf{S}$ and $\mathbf{T}$ do not have predicate symbols in common and Σ_{st} is a set of FO-dependencies over $\mathbf{S} \cup \mathbf{T}$ (in [8] and [9] a more general class of data exchange settings is presented, that also includes *target* dependencies). As usual in the data exchange literature (e.g., [8, 9]), we restrict the study to data exchange settings in which Σ_{st} consists of a set of *source-to-target tuple-generating* dependencies. A source-to-target tuple-generating dependency (st-tgd) is an FO-sentence of the form $\forall \bar{x}\,(\phi(\bar{x}) \rightarrow \exists \bar{y}\,\psi(\bar{x}, \bar{y}))$, where $\phi(\bar{x})$ is a conjunction of relational atoms over $\mathbf{S}$ and $\psi(\bar{x}, \bar{y})$ is a conjunction of relational atoms over $\mathbf{T}$. A *source* (resp. *target*) instance K for $\mathcal{M}$ is an instance of $\mathbf{S}$ (resp. $\mathbf{T}$). We usually denote source instances by $I, I', I_1, \ldots,$ and target instances by $J, J', J_1, \ldots$.

The class of data exchange settings considered in this paper is usually called GLAV (global-&-local-as-view) in the database literature [14]. One of the restricted forms of this class that has been extensively studied for data integration and exchange is the class of LAV settings. Formally, a LAV setting (local-as-view) [14] is a data exchange setting $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, in which every st-tgd in Σ_{st} is of the form $\forall \bar{x}\,(S(\bar{x}) \rightarrow \psi(\bar{x}))$, for some $S \in \mathbf{S}$.

An instance J of $\mathbf{T}$ is said to be a *solution* for an instance I under $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, if the instance $K = (I, J)$ of $\mathbf{S} \cup \mathbf{T}$ satisfies Σ_{st}, where $S^K = S^I$ for every $S \in \mathbf{S}$ and $T^K = T^J$ for every $T \in \mathbf{T}$. If $\mathcal{M}$ is clear from the context, we shall say that J is a solution for I.

EXAMPLE 2.1. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ be a data exchange setting. Assume that $\mathbf{S}$ consists of one binary relation symbol P, and $\mathbf{T}$ consists of two binary relation symbols Q and R. Further, assume that Σ_{st} consists of st-tgds $P(x, y) \rightarrow Q(x, y)$ and $P(x, y) \rightarrow \exists z R(x, z)$. Then $\mathcal{M}$ is also a LAV setting.

Let $I = \{P(a, b),\ P(a, c)\}$ be a source instance. Then $J_1 = \{Q(a, b),\ Q(a, c),\ R(a, b)\}$ and $J_2 = \{Q(a, b),\ Q(a, c),\ R(a, n)\}$, where $n \in \mathbf{N}$, are solutions for I. In fact, I has infinitely many solutions. □

2.2 Universal solutions and canonical universal solution

It has been argued in [8] that the preferred solutions in data exchange are the *universal* solutions. In order to define this notion, we first have to revise the concept of *homomorphism* in data exchange. Let K_1 and K_2 be instances of the same schema $\mathbf{R}$. A *homomorphism* h from K_1 to K_2 is a function $h : \text{dom}(K_1) \rightarrow \text{dom}(K_2)$ such that: (1) $h(c) = c$ for every $c \in \mathbf{C} \cap \text{dom}(K_1)$, and (2) for every $R \in \mathbf{R}$ and every tuple $\bar{a} = (a_1, \ldots, a_k) \in R^{K_1}$, it holds that $h(\bar{a}) = (h(a_1), \ldots, h(a_k)) \in R^{K_2}$. Notice that this definition of homomorphism slightly differs from the usual one, as the additional constraint that homomorphisms are the identity on the constants is imposed.

Let $\mathcal{M}$ be a data exchange setting, I a source instance and J a solution for I under $\mathcal{M}$. Then J is a *universal solution* for I under $\mathcal{M}$, if for every solution J' for I under $\mathcal{M}$, there exists a homomorphism from J to J'.

EXAMPLE 2.2 (EXAMPLE 2.1 CONTINUED). Solution J_2 is a universal solution for I, while J_1 is not since there is no homomorphism from J_1 to J_2. □

It follows from [8] that for the class of data exchange settings studied in this paper, every source instance has universal solutions. In particular, one of these solutions - called the *canonical universal solution* - can be constructed in polynomial time from the given source instance (assuming the setting to be fixed), using the *chase* procedure [5]. We shall define canonical universal solutions directly as in [4, 16].

In the following, we show how to compute the canonical universal solution of a source instance I in a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st})$. For each st-tgd in Σ_{st} of the form:

$$\phi(\bar{x}, \bar{y}) \quad \rightarrow \quad \exists \bar{w}\,(T_1(\bar{x}_1, \bar{w}_1) \wedge \cdots \wedge T_k(\bar{x}_k, \bar{w}_k)),$$

where $\bar{x} = \bar{x}_1 \cup \cdots \cup \bar{x}_k$ and $\bar{w} = \bar{w}_1 \cup \cdots \cup \bar{w}_k$, and for each tuple $\bar{a}$ from $\mathrm{dom}(I)$ of length $|\bar{x}|$, find all tuples $\bar{b}_1, \ldots, \bar{b}_m$ such that $I \models \phi(\bar{a}, \bar{b}_i)$, $i \in [1, m]$. Then choose m tuples $\bar{n}_1, \ldots, \bar{n}_m$ of length $|\bar{w}|$ of fresh distinct null values over $\mathbf{N}$. Relation T_i ($i \in [1, k]$) in the canonical universal solution for I contains tuples $(\pi_{\bar{x}_i}(\bar{a}), \pi_{\bar{w}_i}(\bar{n}_j))$, for each $j \in [1, m]$, where $\pi_{\bar{x}_i}(\bar{a})$ refers to the components of $\bar{a}$ that occur in the positions of $\bar{x}_i$. Furthermore, relation T_i in the canonical universal solution for I only contains tuples that are obtained by applying this algorithm.

This definition differs from the one given in [8], where a canonical universal solution is obtained by using the classical chase procedure. Since the result of the chase used in [8] is not necessarily unique (it depends on the order in which the chase steps are applied), there may be multiple non-isomorphic canonical universal solutions. Clearly, under our definition, the canonical universal solution is unique up to isomorphism and can be computed in polynomial time from I. For a fixed data exchange setting $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, we denote by CAN the transformation from source instances to target instances, such that $\textsc{Can}(I)$ is the canonical universal solution for I under $\mathcal{M}$.

2.3 Certain answers

Queries in a data exchange setting $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ are posed over the target schema $\mathbf{T}$. Given that there may be (even infinitely) many solutions for a given source instance I with respect to $\mathcal{M}$, the standard approach in the data exchange literature is to define the semantics of the query based on the notion of certain answers [13, 1, 14, 8].

Let I be a source instance. For a query Q of arity $n \geq 0$, in any of our logical formalisms, we denote by $\mathrm{certain}_{\mathcal{M}}(Q, I)$ the set of *certain answers* of Q over I under $\mathcal{M}$, that is, the set of n-tuples $\bar{t}$ such that $\bar{t} \in Q(J)$, for every J that is a solution for I under $\mathcal{M}$. If $n = 0$, then we say that Q is *Boolean*, and $\mathrm{certain}_{\mathcal{M}}(Q, I) = \texttt{true}$ iff Q holds for every J that is a solution for I under $\mathcal{M}$. We write $\mathrm{certain}_{\mathcal{M}}(Q, I) = \texttt{false}$ if it is not the case that $\mathrm{certain}_{\mathcal{M}}(Q, I) = \texttt{true}$.

Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ be a data exchange setting and Q a query over $\mathbf{T}$. The main problem studied in this paper is:

PROBLEM	:	CERTAIN-ANSWERS($\mathcal{M}$, Q).
INPUT	:	A source instance I and a tuple $\bar{t}$ of constants from I.
QUESTION	:	Is $\bar{t} \in \mathrm{certain}_{\mathcal{M}}(Q, I)$?

Since in the above definition both the setting and the query are fixed, it corresponds (in terms of Vardi's taxonomy [19]) to the *data* complexity of the problem of computing certain answers. Later, in Section 6, we also study the *combined* complexity of this problem.

3. EXTENDING QUERY LANGUAGES FOR DATA EXCHANGE: DATALOG$^{\mathbf{C}(\neq)}$ PROGRAMS

The class of unions of conjunctive queries is particularly well-behaved for data exchange; the certain answers of each union of conjunctive queries Q can be computed by directly posing Q over an arbitrary universal solution [8]. More formally, given a data exchange setting $\mathcal{M}$, a source instance I, a universal solution J for I under $\mathcal{M}$, and a tuple $\bar{t}$ of constants, $\bar{t} \in \mathrm{certain}_{\mathcal{M}}(Q, I)$ if and only if $\bar{t} \in Q(J)$. This implies that for each data exchange setting $\mathcal{M}$, the problem CERTAIN-ANSWERS($\mathcal{M}, Q$) can be solved in polynomial time if Q is a union of conjunctive queries (because the canonical universal solution for I can be computed in polynomial time and Q has polynomial time data complexity).

The fact that the certain answers of a union of conjunctive queries Q can be computed by posing Q over a universal solution, can be fully explained by the fact that Q is *preserved* under homomorphisms, that is, for every pair of instances J, J', homomorphism h from J to J', and tuple $\bar{a}$ of elements in J, if $\bar{a} \in Q(J)$, then $h(\bar{a}) \in Q(J')$. But UCQ is not the only class of queries that is preserved under homomorphisms; also DATALOG, the *recursive* extension of the class UCQ, has this property. Since DATALOG has polynomial time data complexity, we have that the certain answers of each DATALOG query Q can be obtained efficiently by first computing a universal solution J, and then evaluating Q over J. Thus, DATALOG preserves all the good properties of UCQ for data exchange.

Unfortunately, both UCQ and DATALOG keep us in the realm of the positive (i.e. negated atoms are not allowed in queries), while most database query languages are equipped with negation. It seems then natural to extend UCQ (or DATALOG) in the context of data exchange with some form of negation. Indeed, query languages with different forms of negation have been considered in the data exchange context [3, 6], as they can be used to express interesting queries. Next, we show an example of this fact.

EXAMPLE 3.1. Consider a data exchange setting with $\mathbf{S} = \{E(\cdot,\cdot), A(\cdot), B(\cdot)\}$, $\mathbf{T} = \{G(\cdot,\cdot), P(\cdot), R(\cdot)\}$ and $\Sigma_{st} = \{E(x,y) \rightarrow G(x,y), A(x) \rightarrow P(x), B(x) \rightarrow R(x)\}$. Notice that if I is a source instance, then the canonical universal solution $\textsc{Can}(I)$ for I is such that $E^I = G^{\textsc{Can}(I)}$, $A^I = P^{\textsc{Can}(I)}$ and $B^I = R^{\textsc{Can}(I)}$.

Let $Q(x)$ be the following UCQ$^{\neg}$ query over $\mathbf{T}$: $\exists x \exists y\,(P(x) \wedge R(y) \wedge G(x,y)) \vee \exists x \exists y \exists z\,(G(x,z) \wedge G(z,y) \wedge \neg G(x,y))$. It is not hard to prove that for every source instance I, $\mathrm{certain}_{\mathcal{M}}(Q, I) = \texttt{true}$ iff there exist elements $a, b \in \mathrm{dom}(\textsc{Can}(I))$ such that a belongs to $P^{\textsc{Can}(I)}$, b belongs to $R^{\textsc{Can}(I)}$ and (a, b) belongs to the transitive closure of the relation $G^{\textsc{Can}(I)}$. That is, $\mathrm{certain}_{\mathcal{M}}(Q, I) = \texttt{true}$ iff there exist elements $a, b \in \mathrm{dom}(I)$ such that a belongs to A^I, b belongs to B^I and (a, b) belongs to the transitive closure of the relation E^I. □

It is well-known (see e.g. [15]) that there is no union of conjunctive queries (indeed, not even an FO-query) that defines the transitive

closure of a graph. Thus, if Q and $\mathcal{M}$ are as in the previous example, then there is no union of conjunctive queries Q' such that $Q'(\text{CAN}(I)) = \text{certain}_{\mathcal{M}}(Q', I) = \text{certain}_{\mathcal{M}}(Q, I)$, for every source instance I. It immediately follows that negated relational atoms add expressive power to the class UCQ in the context of data exchange (see also [4]). And not only that, it follows from [8] that inequalities also add expressive power to UCQ in the context of data exchange.

In this section, we propose a language that can be used to pose queries with negation, and that has all the good properties of UCQ for data exchange.

3.1 DATALOG$^{\mathbf{C}(\neq)}$ programs

Unfortunately, adding an unrestricted form of negation to DATALOG (or even to CQ) not only destroys preservation under homomorphisms, but also easily yields to intractability of the problem of computing certain answers [1, 8]. More precisely, there is a setting $\mathcal{M}$ and a query Q in UCQ$^{\neq}$ such that the problem CERTAIN-ANSWERS$(\mathcal{M}, Q)$ cannot be solved in polynomial time (unless PTIME = NP). In particular, the set of certain answers of Q cannot be computed by evaluating Q over a polynomial-time computable universal solution. Next we show that there is a natural way of adding negation to DATALOG while keeping all of the good properties of this language for data exchange. In Section 4, we show that such a restricted form of negation can be used to express many relevant queries (some including negation) for data exchange.

DEFINITION 3.2 (DATALOG$^{\mathbf{C}(\neq)}$ PROGRAMS). *A* constant-inequality Datalog rule *is a rule of the form:*

$$S(\bar{x}) \leftarrow S_1(\bar{x}_1), \ldots, S_\ell(\bar{x}_\ell), \mathbf{C}(y_1), \ldots, \mathbf{C}(y_m), u_1 \neq v_1, \ldots, u_n \neq v_n, \quad (1)$$

where

- *(a) S, S_1, ..., S_ℓ are (non necessarily distinct) predicate symbols,*
- *(b) every variable in $\bar{x}$ is mentioned in some tuple $\bar{x}_i$ ($i \in [1, \ell]$),*
- *(c) every variable y_j ($j \in [1, m]$) is mentioned in some tuple $\bar{x}_i$ ($i \in [1, \ell]$), and*
- *(d) every variable u_j ($j \in [1, n]$), and every variable v_j, is equal to some variable y_i ($i \in [1, m]$).*

Moreover, a constant-inequality Datalog program (DATALOG$^{\mathbf{C}(\neq)}$ *program*) Π *is a finite set of constant-inequality Datalog rules.*

For example, the following is a constant-inequality Datalog program:

$$\begin{aligned} R(x, y) &\leftarrow T(x, z), S(z, y), \mathbf{C}(x), \mathbf{C}(z), x \neq z \\ S(x) &\leftarrow U(x, u, v, w), \mathbf{C}(x), \mathbf{C}(u), \\ &\qquad \mathbf{C}(v), \mathbf{C}(w), u \neq v, u \neq w \end{aligned}$$

For a rule of the form (1), we say that $S(\bar{x})$ is its head. The set of predicates of a DATALOG$^{\mathbf{C}(\neq)}$ program Π, denoted by $Pred(\Pi)$, is the set of predicate symbols mentioned in Π, while the set of intensional predicates of Π, denoted by $IPred(\Pi)$, is the set of predicates symbols $R \in Pred(\Pi)$ such that $R(\bar{x})$ appears as the head of some rule of Π.

Assume that Π is a DATALOG$^{\mathbf{C}(\neq)}$ program and I is a database instance of the relational schema $Pred(\Pi)$. Then $\mathcal{T}(I)$ is an instance of $Pred(\Pi)$ such that for every $R \in Pred(\Pi)$ and every tuple $\bar{t}$, it holds that $\bar{t} \in R^{\mathcal{T}(I)}$ if and only if there exists a rule $R(\bar{x}) \leftarrow R_1(\bar{x}_1), \ldots, R_\ell(\bar{x}_\ell), \mathbf{C}(y_1), \ldots, \mathbf{C}(y_m), u_1 \neq v_1, \ldots, u_n \neq v_n$ in Π and a variable assignment σ such that (a) $\sigma(\bar{x}) = \bar{t}$, (b) $\sigma(\bar{x}_i) \in R_i^I$, for every $i \in [1, \ell]$, (c) $\sigma(y_i)$ is a constant, for every $i \in [1, m]$, and (d) $\sigma(u_i) \neq \sigma(v_i)$, for every $i \in [1, n]$. Operator $\mathcal{T}$ is used to define the semantics of constant-inequality Datalog programs. More precisely, define $\mathcal{T}_\Pi^0(I)$ to be I and $\mathcal{T}_\Pi^{n+1}(I)$ to be $\mathcal{T}(\mathcal{T}_\Pi^n(I)) \cup \mathcal{T}_\Pi^n(I)$, for every $n \geq 0$. Then the evaluation of Π over I is defined as $\mathcal{T}_\Pi^\infty(I) = \bigcup_{n \geq 0} \mathcal{T}_\Pi^n(I)$.

A constant-inequality Datalog program Π is said to be defined over a relational schema $\mathbf{R}$ if $\mathbf{R} = Pred(\Pi) \setminus IPred(\Pi)$ and ANSWER $\in IPred(\Pi)$. Given an instance I of $\mathbf{R}$ and a tuple $\bar{t}$ in $\text{dom}(I)^n$, where n is the arity of ANSWER, we say that $\bar{t} \in \Pi(I)$ if $\bar{t} \in \text{ANSWER}^{\mathcal{T}_\Pi^\infty(I_0)}$, where I_0 is an extension of I defined as: $R^{I_0} = R^I$ for $R \in \mathbf{R}$ and $R^{I_0} = \emptyset$ for $R \in IPred(\Pi)$.

As we mentioned before, the homomorphisms in data exchange are not arbitrary; they are the identity on the constants. Thus, given that inequalities are witnessed by constants in DATALOG$^{\mathbf{C}(\neq)}$ programs, we have that these programs are preserved under homomorphisms. From this we conclude that the certain answers to a DATALOG$^{\mathbf{C}(\neq)}$ program Π can be computed by directly evaluating Π over a universal solution.

PROPOSITION 3.3. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ be a data exchange setting, I a source instance, J a universal solution for I under $\mathcal{M}$, and Π a* DATALOG$^{\mathbf{C}(\neq)}$ *program over $\mathbf{T}$. Then for every tuple $\bar{t}$ of constants, $\bar{t} \in \text{certain}_{\mathcal{M}}(\Pi, I)$ iff $\bar{t} \in \Pi(J)$.*

This proposition will be used in Section 4 to show that DATALOG$^{\mathbf{C}(\neq)}$ programs preserve the good properties of conjunctive queries for data exchange.

4. ON THE COMPLEXITY AND EXPRESSIVENESS OF DATALOG$^{\mathbf{C}(\neq)}$ PROGRAMS

We start this section by studying the expressive power of DATALOG$^{\mathbf{C}(\neq)}$ programs. In particular, we show that these programs are expressive enough to capture the class of unions of conjunctive queries with at most one negated atom per disjunct. This class has proved to be relevant for data exchange, as its restriction with inequalities not only can express relevant queries but also is one of the few known extensions of the class UCQ for which the problem of computing certain answers is tractable [8].

THEOREM 4.1. *Let Q be a* UCQ *query over a schema $\mathbf{T}$, with at most one inequality or negated relational atom per disjunct. Then there exists a* DATALOG$^{\mathbf{C}(\neq)}$ *program Π_Q over $\mathbf{T}$ such that for every data exchange setting $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ and instance I of $\mathbf{S}$, $\text{certain}_{\mathcal{M}}(Q, I) = \text{certain}_{\mathcal{M}}(\Pi_Q, I)$. Moreover, Π_Q can be effectively constructed from Q in polynomial time.*

In the following example, we sketch the proof of Theorem 4.1.

EXAMPLE 4.2. Let $\mathcal{M}$ be a data exchange setting such that $\mathbf{S} = \{E(\cdot,\cdot), A(\cdot)\}$, $\mathbf{T} = \{G(\cdot,\cdot), P(\cdot)\}$ and

$$\Sigma_{st} \;=\; \{E(x,y) \rightarrow \exists z(G(x,z) \wedge G(z,y)),\; A(x) \rightarrow P(x)\}.$$

Also, let $Q(x)$ be the following query in $\text{UCQ}^{\neq,\neg}$:

$$(P(x) \wedge G(x,x)) \;\vee\; \exists y\,(G(x,y) \wedge x \neq y) \\ \vee\; \exists y \exists z\,(G(x,z) \wedge G(z,y) \wedge \neg G(x,y)).$$

We construct a $\text{DATALOG}^{\mathbf{C}(\neq)}$ program Π_Q such that $\text{certain}_{\mathcal{M}}(Q,I) = \text{certain}_{\mathcal{M}}(\Pi_Q, I)$. The set of intensional predicates of the $\text{DATALOG}^{\mathbf{C}(\neq)}$ program Π_Q is $\{U_1(\cdot,\cdot,\cdot), U_2(\cdot,\cdot), \text{dom}(\cdot), \text{EQUAL}(\cdot,\cdot,\cdot), \text{ANSWER}(\cdot)\}$. The program Π_Q over $\mathbf{T}$ is defined as follows.

- First, the program collects in $\text{dom}(x)$ all the elements that belong to the active domain of the instance of $\mathbf{T}$ where Π_Q is evaluated:

$$\text{dom}(x) \;\leftarrow\; G(x,z) \quad (2)$$
$$\text{dom}(x) \;\leftarrow\; G(z,x) \quad (3)$$
$$\text{dom}(x) \;\leftarrow\; P(x) \quad (4)$$

- Second, the program Π_Q includes the following rules that formalize the idea that $\text{EQUAL}(x,y,z)$ holds if x and y are the same elements:

$$\text{EQUAL}(x,x,z) \leftarrow \text{dom}(x), \text{dom}(z) \quad (5)$$
$$\text{EQUAL}(x,y,z) \leftarrow \text{EQUAL}(y,x,z) \quad (6)$$
$$\text{EQUAL}(x,y,z) \leftarrow \text{EQUAL}(x,w,z), \text{EQUAL}(w,y,z) \quad (7)$$

  Predicate EQUAL includes an extra argument that keeps track of the element z where the query is being evaluated. Notice that we cannot simply use the rule $\text{EQUAL}(x,x,z) \leftarrow$ to say that EQUAL is reflexive, as $\text{DATALOG}^{\mathbf{C}(\neq)}$ programs are *safe*, i.e. every variable that appears in the head of a rule also has to appear in its body.

- Third, Π_Q includes the rules:

$$U_1(x,y,z) \;\leftarrow\; G(x,y), \text{dom}(z) \quad (8)$$
$$U_2(x,z) \;\leftarrow\; P(x), \text{dom}(z) \quad (9)$$
$$U_1(x,y,z) \;\leftarrow\; U_1(u,v,z), \text{EQUAL}(u,x,z), \text{EQUAL}(v,y,z) \quad (10)$$
$$U_2(x,z) \;\leftarrow\; U_2(u,z), \text{EQUAL}(u,x,z) \quad (11)$$

  Intuitively, the first two rules create in U_1 and U_2 a copy of G and P, respectively, but again with an extra argument for keeping track of the element where Π_Q is being evaluated. The last two rules allow to replace equal elements in the interpretation of U_1 and U_2.

- Fourth, Π_Q includes the following rule for the third disjunct of $Q(x)$:

$$U_1(x,y,x) \;\leftarrow\; U_1(x,z,x), U_1(z,y,x) \quad (12)$$

  Intuitively, this rule expresses that if a is an element that does not belong to the set of certain answers to $Q(x)$, then for every pair of elements b and c such that (a,b) and (b,c) belong to the interpretation of G, it must be the case that (a,c) also belongs to it.

- Fifth, Π_Q includes the following rule for the second disjunct of $Q(x)$:

$$\text{EQUAL}(x,y,x) \;\leftarrow\; U_1(x,y,x) \quad (13)$$

  Intuitively, this rule expresses that if a is an element that does not belong to the set of certain answers to $Q(x)$, then for every element b such that the pair (a,b) belongs to the interpretation of G, it must be the case that $a = b$.

- Finally, Π_Q includes two rules for collecting the certain answers to $Q(x)$:

$$\text{ANSWER}(x) \leftarrow U_2(x,x), U_1(x,x,x), \mathbf{C}(x) \quad (14)$$
$$\text{ANSWER}(x) \leftarrow \text{EQUAL}(y,z,x), \mathbf{C}(y), \mathbf{C}(z), y \neq z \quad (15)$$

  Intuitively, rule (14) says that if a constant a belongs to the interpretation of P and (a,a) belongs to the interpretation of G, then a belongs to the set of certain answers to $Q(x)$. Indeed, this means that if J is an arbitrary solution where the program is being evaluated, then a belongs to the evaluation of the first disjunct of $Q(x)$ over J.

  Rule (15) says that if in the process of evaluating Π_Q with parameter a, two distinct constants b and c are declared to be equal ($\text{EQUAL}(b,c,a)$ holds), then a belongs to the set of certain answers to $Q(x)$. We show the application of this rule with an example. Let I be a source instance, and assume that (a,n) and (n,b) belong to G in the canonical universal solution for I, where n is a null value. By applying rule (2), we have that $\text{dom}(a)$ holds in $\text{CAN}(I)$. Thus, we conclude by applying rule (8) that $U_1(a,n,a)$ and $U_1(n,b,a)$ hold in $\text{CAN}(I)$ and, therefore, we obtain by using rule (13) that $\text{EQUAL}(a,n,a)$ holds in $\text{CAN}(I)$. Notice that this rule is trying to prove that a is not in the certain answers to $Q(x)$ and, hence, it forces n to be equal to a. Now by using rule (6), we obtain that $\text{EQUAL}(n,a,a)$ holds in $\text{CAN}(I)$. But we also have that $\text{EQUAL}(b,b,a)$ holds in $\text{CAN}(I)$ (by applying rules (3) and (5)). Thus, by applying rule (10), we obtain that $U_1(a,b,a)$ holds in $\text{CAN}(I)$. Therefore, by applying rule (13) again, we obtain that $\text{EQUAL}(a,b,a)$ holds in $\text{CAN}(I)$. This time, rule (13) tries to prove that a is not in the certain answers to $Q(x)$ by forcing constants a and b to be the same value. But this cannot be the case since a and b are distinct constants and, thus, rule (15) is used to conclude that a is in the certain answers to $Q(x)$. It is important to notice that this conclusion is correct. If J is an arbitrary solution for I, then we have that there exists a homomorphism $h : \text{CAN}(I) \rightarrow J$. Given that a and b are distinct constants, we have that $a \neq h(n)$ or $b \neq h(n)$. It follows that there is an element c in J such that $a \neq c$ and the pair (a,c) belongs to the interpretation of G. Thus, we conclude that a belongs to the evaluation of the second disjunct of $Q(x)$ over J.

It is now an easy exercise to show that the set of certain answers to $Q(x)$ coincide with the set of certain answers to Π_Q, for every source instance I. □

At this point, a natural question about $\text{DATALOG}^{\mathbf{C}(\neq)}$ programs is whether the different components of this language are really needed, that is, whether inequalities and recursion are essential for this language. Next, we show that this is indeed the case and, in particular, we conclude that both inequalities and recursion are essential for Theorem 4.1.

It was shown in [8] that there exist a data exchange setting $\mathcal{M}$ and a conjunctive query Q with one inequality for which there is no first-order query $Q^\star$ such that $\text{certain}_\mathcal{M}(Q, I) = Q^\star(\text{CAN}(I))$ holds, for every source instance I. Thus, given that a non-recursive DATALOG$^{\mathbf{C}(\neq)}$ program is equivalent to a first-order query, we conclude from Proposition 3.3 that recursion is necessary for capturing the class of unions of conjunctive queries with at most one negated atom per disjunct.

PROPOSITION 4.3 ([8]). *There exist a data exchange setting $\mathcal{M}$ and a Boolean conjunctive query Q with a single inequality such that for every non-recursive* DATALOG$^{\mathbf{C}(\neq)}$ *program* Π*, it holds that* $\text{certain}_\mathcal{M}(Q, I) \neq \text{certain}_\mathcal{M}(\Pi, I)$ *for some source instance* I.

In the following proposition, we show that the use of inequalities is also necessary for capturing the class of unions of conjunctive queries with at most one negated atom per disjunct. We note that this cannot be obtained from the result in [8] mentioned above, as there are DATALOG$^{\mathbf{C}(\neq)}$ programs without inequalities that are not expressible in first-order logic. The proof of this proposition follows from the fact that DATALOG$^{\mathbf{C}(\neq)}$ programs without inequalities are preserved under homomorphisms, while conjunctive queries with inequalities are only preserved under one-to-one homomorphisms.

PROPOSITION 4.4. *There exist a data exchange setting $\mathcal{M}$ and a Boolean conjunctive query Q with a single inequality such that for every* DATALOG$^{\mathbf{C}(\neq)}$ *program* Π *without inequalities,* $\text{certain}_\mathcal{M}(Q, I) \neq \text{certain}_\mathcal{M}(\Pi, I)$ *for some source instance* I.

Notice that as a corollary of Proposition 4.4 and Theorem 4.1, we obtain that DATALOG$^{\mathbf{C}(\neq)}$ programs are strictly more expressive than DATALOG$^{\mathbf{C}(\neq)}$ programs without inequalities.

We conclude this section by studying the complexity of the problem of computing certain answers to DATALOG$^{\mathbf{C}(\neq)}$ programs. It was shown in Proposition 3.3 that the certain answers of a DATALOG$^{\mathbf{C}(\neq)}$ program Π can be computed by directly posing Π over CAN(I). This implies that for each data exchange setting $\mathcal{M}$, the problem CERTAIN-ANSWERS$(\mathcal{M}, \Pi)$ can be solved in polynomial time if Π is a DATALOG$^{\mathbf{C}(\neq)}$ program (since CAN(I) can be computed in polynomial time and Π has polynomial time data complexity).

PROPOSITION 4.5. *The problem* CERTAIN-ANSWERS$(\mathcal{M}, \Pi)$ *can be solved in polynomial time, for every data exchange setting $\mathcal{M}$ and* DATALOG$^{\mathbf{C}(\neq)}$ *program* Π.

From the previous proposition and Theorem 4.1, we conclude that the certain answers to a union of conjunctive queries with at most one negated atom per disjunct can also be computed in polynomial time. We note that this slightly generalizes one of the polynomial time results in [8], which is stated for the class of unions of conjunctive queries with at most one inequality per disjunct. The proof of the result in [8] uses different techniques, based on the chase procedure. In Section 5, we show that DATALOG$^{\mathbf{C}(\neq)}$ programs can also be used to express (some) unions of conjunctive queries with two inequalities per disjunct.

A natural question at this point is whether the problem CERTAIN-ANSWERS$(\mathcal{M}, \Pi)$ is PTIME-complete for some data exchange setting $\mathcal{M}$ and DATALOG$^{\mathbf{C}(\neq)}$ program Π. It is easy to see that this is the case given that the data complexity of the evaluation problem for DATALOG programs is PTIME-complete. But more interestingly, from Theorem 4.1 we have that this result is also a corollary of a stronger result for UCQ$^\neq$ queries, namely that there exist a data exchange setting $\mathcal{M}$ and a conjunctive query Q with one inequality such that the problem CERTAIN-ANSWERS$(\mathcal{M}, Q)$ is PTIME-complete.

PROPOSITION 4.6. *There exist a* LAV *data exchange setting $\mathcal{M}$ and a Boolean conjunctive query Q with one inequality such that* CERTAIN-ANSWERS$(\mathcal{M}, Q)$ *is* PTIME*-complete.*

It is worth mentioning that it follows from Proposition 3.1 in [12] that there exist a data exchange setting $\mathcal{M}$ containing some *target* dependencies and a conjunctive query Q with one inequality such that CERTAIN-ANSWERS$(\mathcal{M}, Q)$ is PTIME-complete. Proposition 4.6 shows that this result holds even when no target dependencies are provided.

5. CONJUNCTIVE QUERIES WITH TWO INEQUALITIES

As we mentioned before, computing certain answers to conjunctive queries with more than just one inequality is an intractable problem. Indeed, there is a LAV setting $\mathcal{M}$ and a Boolean conjunctive query Q with two inequalities such that the problem CERTAIN-ANSWERS$(\mathcal{M}, Q)$ is CONP-complete [18]. Therefore, unless PTIME = NP, Theorem 4.1 is no longer valid if we remove the restriction that every disjunct of Q must contain at most one inequality.

The intractability for conjunctive queries with two inequalities is tightly related with the use of null values when joining relations and checking inequalities. In this section, we investigate this relationship, and provide a syntactic condition on the type of joins and inequalities allowed in queries. This restriction leads to tractability of the problem of computing certain answers. Indeed, this tractability is a corollary of a stronger result, namely that for every conjunctive query Q with two inequalities, if Q satisfies the syntactic condition, then one can construct a DATALOG$^{\mathbf{C}(\neq)}$ program Π_Q such that $\text{certain}_\mathcal{M}(Q, I) = \text{certain}_\mathcal{M}(\Pi_Q, I)$ for every source instance I. It should be noticed that in this case DATALOG$^{\mathbf{C}(\neq)}$ programs are used as a tool for finding a tractable class of queries for the problem of computing certain answers.

To define the syntactic restriction mentioned above, we need to introduce some terminology. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ be a data exchange setting. Then for every n-ary relation symbol T in $\mathbf{T}$, we say that the i-th attribute of T ($1 \leq i \leq n$) *can be nullified* under $\mathcal{M}$, if there is an st-tgd α in Σ_{st} such that the i-th attribute of T is existentially quantified in the right hand side of α. Notice that for each setting $\mathcal{M}$ and source instance I, if the i-th attribute of T cannot be nullified under $\mathcal{M}$, then for every tuple $(c_1, \ldots, c_n)$ that belongs to T in the canonical universal solution for I, it holds that c_i is a constant. Moreover, if Q is a UCQ$^\neq$ query over $\mathbf{T}$ and x is a variable in Q, then we say that x *can be nullified* under Q and $\mathcal{M}$, if x appears in Q as the i-th attribute of a target relation T, and the i-th attribute of T can be nullified under $\mathcal{M}$.

Let $\mathcal{M}$ be a data exchange setting and Q a conjunctive query

with two inequalities, and assume that if x appears as a variable in the inequalities of Q, then x cannot be nullified under Q and $\mathcal{M}$. In this case, it is straightforward to prove that CERTAIN-ANSWERS$(\mathcal{M}, Q)$ is tractable. Indeed, the previous condition implies that for every source instance I, if Q holds in CAN(I), then all the witnesses for Q in CAN(I) make comparisons of the form $c \neq c'$, where c and c' are constants. Thus, we have that $\text{certain}_{\mathcal{M}}(Q, I)$ can be computed by simply evaluating Q over CAN(I). Here we are interested in finding less obvious conditions that lead to tractability. In particular, we would like to find queries that do not restrict the use of null values in such a strict way.

Let Q be a conjunctive query with two inequalities over a target schema $\mathbf{T}$. Assume that the quantifier free part of Q is of the form $\phi(x_1, \ldots, x_m) \wedge u_1 \neq v_1 \wedge u_2 \neq v_2$, where ϕ is a conjunction of relational atoms over $\mathbf{T}$ and u_1, v_1, u_2 and v_2 are all mentioned in the set of variables x_1, ..., x_m (Q is a safe query [2]). We are now ready to define the two components of the syntactic restriction that leads to tractability of the problem of computing certain answers. We say that Q has *almost constant inequalities* under $\mathcal{M}$, if u_1 or v_1 cannot be nullified under Q and $\mathcal{M}$, and u_2 or v_2 cannot be nullified under Q and $\mathcal{M}$. Intuitively, this means that to satisfy Q in the canonical universal solution of a source instance, one can only make comparisons of the form $c \neq \perp$ and $c \neq c'$, where c, c' are constants and $\perp$ is a null value. Moreover, we say that Q has *constant joins* under $\mathcal{M}$, if for every variable x that appears at least twice in ϕ, x cannot be nullified under Q and $\mathcal{M}$. Intuitively, this means that to satisfy Q in the canonical universal solution of a source instance, one can only use constant values when joining relations.

EXAMPLE 5.1. Let $\mathcal{M}$ be a data exchange setting specified by st-tgds:

$$\begin{aligned} P(x, y) &\rightarrow T(x, y), \\ P(x, y) &\rightarrow \exists z\, U(x, z). \end{aligned}$$

The first and second attribute of T, as well as the first attribute of U, cannot be nullified under $\mathcal{M}$. On the other hand, the second attribute of U can be nullified under $\mathcal{M}$.

Let $Q(x)$ be query $\exists y \exists z (T(y, x) \wedge U(z, x) \wedge x \neq y \wedge x \neq z)$. Then we have that Q has almost constant inequalities under $\mathcal{M}$ because variables y and z cannot be nullified under Q and $\mathcal{M}$, but Q does not have constant joins because variable x appears twice in $T(y, x) \wedge U(z, x)$ and it can be nullified under Q and $\mathcal{M}$. On the other hand, query $U(x, y) \wedge U(x, z) \wedge x \neq z \wedge y \neq z$ has constant joins but does not have almost constant inequalities, and query $U(x, y) \wedge T(x, z) \wedge x \neq z \wedge y \neq z$ has both constant joins and almost constant inequalities. □

Although the notions of constant joins and almost constant inequalities were defined for CQ$^{\neq}$ queries with two inequalities, they can be easily extended to the case of conjunctive queries with an arbitrary number of inequalities. In fact, the notion of constant joins does not change in the case of an arbitrary number of inequalities, while to define the notion of almost constant inequalities in the general case, one has to say that each inequality $x \neq y$ in a query satisfies the condition that x or y cannot be nullified. With this extension, we have all the necessary ingredients for the main result of this section.

THEOREM 5.2. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ be a data exchange setting and Q a* UCQ$^{\neq}$ *query over $\mathbf{T}$ such that each disjunct of Q either (1) has at most one inequality and constant joins under $\mathcal{M}$, or (2) has two inequalities, constant joins and almost constant inequalities under $\mathcal{M}$. Then there exists a* DATALOG$^{\mathbf{C}(\neq)}$ *program Π_Q over $\mathbf{T}$ such that for every instance I of $\mathbf{S}$, $\text{certain}_{\mathcal{M}}(Q, I) = \text{certain}_{\mathcal{M}}(\Pi_Q, I)$. Moreover, Π_Q can be effectively constructed from Q and $\mathcal{M}$ in polynomial time.*

It immediately follows from Proposition 4.5 that if a data exchange setting $\mathcal{M}$ and a UCQ$^{\neq}$ query Q satisfy the conditions mentioned in Theorem 5.2, then CERTAIN-ANSWERS$(\mathcal{M}, Q)$ is in PTIME. Furthermore, it can also be shown that the properties of having constant joins and almost constant inequalities are helpful in reducing the complexity of computing certain answers to unions of conjunctive queries with at most one inequality per disjunct.

PROPOSITION 5.3. *Let Q be a* UCQ$^{\neq}$ *query with at most one inequality per disjunct. If every disjunct of Q has constant joins under a setting $\mathcal{M}$, then* CERTAIN-ANSWERS$(\mathcal{M}, Q)$ *is in* NLOGSPACE*, and if in addition every disjunct of Q has almost constant inequalities under $\mathcal{M}$, then* CERTAIN-ANSWERS$(\mathcal{M}, Q)$ *is in* LOGSPACE*.*

An obvious question at this point is how natural the conditions used in Theorem 5.2 are. Although we cannot settle this subjective question, we are at least able to show that these conditions are optimal in the sense that removing any of them leads to intractability for the class of UCQ$^{\neq}$ queries with two inequalities.

THEOREM 5.4.

(1) There exist a LAV *data exchange setting $\mathcal{M}$ and a query Q such that Q is the union of a Boolean conjunctive query and a Boolean conjunctive query with two inequalities that has both constant joins and almost constant inequalities under $\mathcal{M}$, and such that* CERTAIN-ANSWERS$(\mathcal{M}, Q)$ *is* CONP*-complete.*

(2) There exist a LAV *data exchange setting $\mathcal{M}$ and a Boolean conjunctive query Q with two inequalities, such that Q has constant joins under $\mathcal{M}$, Q does not have almost constant inequalities under $\mathcal{M}$ and* CERTAIN-ANSWERS$(\mathcal{M}, Q)$ *is* CONP*-complete.*

(3) There exist a LAV *data exchange setting $\mathcal{M}$ and a Boolean conjunctive query Q with two inequalities, such that Q has almost constant inequalities under $\mathcal{M}$, Q does not have constant joins under $\mathcal{M}$ and* CERTAIN-ANSWERS$(\mathcal{M}, Q)$ *is* CONP*-complete.*

It is important to notice that although the problem of computing certain answers to UCQ$^{\neq}$ queries has been considered in the literature, none of the results of Theorem 5.4 directly follows from any of the known results for this problem. In particular, Fagin at al. showed in [8] a similar result to (1), namely that the problem of computing certain answers is CONP-complete even for the union of two queries, the first of which is a conjunctive query and the second of which is a conjunctive query with two inequalities. The difficulty in our case is that the second query is restricted to

have constant joins and almost constant inequalities, while Fagin et al. considered a query that does not satisfy any of these conditions. Moreover, Mądry proved in [18] a similar result to (2) and (3), namely that the problem of computing certain answers is coNP-complete for conjunctive queries with two inequalities. The difficulty in our case is that we consider a query that has constant joins in (2) and a query that has almost constant inequalities in (3), while Mądry considered a query that does not satisfy any of these conditions. In fact, we provide in (2) and (3) two new proofs of the fact that the problem of computing certain answer to a conjunctive query with two inequalities is coNP-complete.

We conclude this section with a remark about the possibility of using the conditions defined in this section to obtain tractability for $\text{UCQ}^{\neq}$. As we mentioned above, the notions of constant joins and almost constant inequalities can be extended to $\text{UCQ}^{\neq}$ queries with an arbitrary number of inequalities. Thus, one may wonder whether these conditions lead to tractability in this general scenario. Unfortunately, the following proposition shows that this is not the case, even for the class of $\text{UCQ}^{\neq}$ queries with three inequalities.

PROPOSITION 5.5. *There exist a* LAV *data exchange setting $\mathcal{M}$ and a Boolean conjunctive query Q with three inequalities, such that Q has both constant joins and almost constant inequalities under $\mathcal{M}$, but the problem* CERTAIN-ANSWERS$(\mathcal{M}, Q)$ *is* coNP*-complete.*

6. THE COMBINED COMPLEXITY OF QUERY ANSWERING

Beyond the usual data complexity analysis, it is natural to ask for the combined complexity of the problem of computing certain answers: What is the complexity if data exchange settings and queries are not considered to be fixed? To state this problem, we shall extend the notation defined in Section 2. Let $\mathcal{DE}$ be a class of data exchange settings and $\mathcal{C}$ a class of queries. In this section, we study the following problem:

PROBLEM:	CERTAIN-ANSWERS$(\mathcal{DE}, \mathcal{C})$.
INPUT:	A data exchange setting $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}) \in \mathcal{DE}$, a source instance I, a query $Q \in \mathcal{C}$ and a tuple $\bar{t}$ of constants from I.
QUESTION:	Is $\bar{t} \in \text{certain}_{\mathcal{M}}(Q, I)$?

It is worth mentioning that a related study appeared in [12]. Even though the focus of that paper was the combined complexity of the existence of solutions problem, some of the results in [12] can be extended to the certain answers problem. In particular, some complexity bounds for unions of conjunctive queries with inequalities can be proved by using these results. Nevertheless, in this section we prove stronger lower bounds that consider single conjunctive queries with inequalities, and which cannot be directly proved by using the results of [12].

We start by stating the complexity for the case of $\text{DATALOG}^{\mathbf{C}(\neq)}$ queries. The study continues by considering some restrictions of $\text{DATALOG}^{\mathbf{C}(\neq)}$ that lead to lower combined complexity, and which are expressed in the form of conjunctive queries with single inequalities. We conclude this study by examining unrestricted $\text{CQ}^{\neq}$ queries, which are not rewritable in $\text{DATALOG}^{\mathbf{C}(\neq)}$ (unless PTIME = NP). The results of this section are summarized in Table 1, where we let k-$\text{CQ}^{\neq}$ be the class of $\text{CQ}^{\neq}$ queries with at most k inequalities.

6.1 Combined complexity of $\text{DATALOG}^{\mathbf{C}(\neq)}$ queries

We showed in Proposition 3.3 that the certain answers of a $\text{DATALOG}^{\mathbf{C}(\neq)}$ program can be computed by directly posing the query over the canonical universal solution. It can be shown that such an approach can compute the certain answers to a $\text{DATALOG}^{\mathbf{C}(\neq)}$ program in exponential time, although canonical universal solutions can be of exponential size if data exchange settings are not considered to be fixed. And not only that, it can be proved that this is a tight bound.

THEOREM 6.1. CERTAIN-ANSWERS$(\text{GLAV}, \text{DATALOG}^{\mathbf{C}(\neq)})$ *is* EXPTIME*-complete.*

Note that the above problem has to deal with canonical universal solutions of exponential size. Then restricting these solutions to be of polynomial size would be a natural approach to reduce the complexity of the problem. There are at least two ways to do this. The obvious one would be to fix the data exchange settings, and leave only queries and source instances as input. The less obvious but more interesting case is to restrict the class of data exchange settings to be LAV settings. However, for the case of $\text{DATALOG}^{\mathbf{C}(\neq)}$ programs, the combined complexity is inherently exponential, and thus reducing the size of canonical universal solutions does not help in improving the upper bound.

PROPOSITION 6.2. CERTAIN-ANSWERS$(\text{LAV}, \text{DATALOG}^{\mathbf{C}(\neq)})$ *is* EXPTIME*-complete.*

It was shown in Theorem 4.1 that every conjunctive query with one inequality can be efficiently translated into a $\text{DATALOG}^{\mathbf{C}(\neq)}$ program. Hence, the class of 1-$\text{CQ}^{\neq}$ queries form a subclass of the class of $\text{DATALOG}^{\mathbf{C}(\neq)}$ programs. Thus, it is natural to ask whether the EXPTIME lower bound carries over this class, and whether the LAV restriction could be useful in this case. These are the motivating questions for the next section.

6.2 Combined complexity of $\text{CQ}^{\neq}$

We leave the $\text{DATALOG}^{\mathbf{C}(\neq)}$ queries to concentrate on the analysis of $\text{CQ}^{\neq}$ queries in data exchange. We first study the class 1-$\text{CQ}^{\neq}$, that is, the class of conjunctive queries with only one inequality. It is worth mentioning that an EXPTIME lower bound can be obtained from [12] for the case of unions of 1-$\text{CQ}^{\neq}$ queries. We refine this result to the case of 1-$\text{CQ}^{\neq}$ queries, and therefore present a stronger lower bound:

THEOREM 6.3. CERTAIN-ANSWERS$(\text{GLAV}, \text{1-CQ}^{\neq})$ *is* EXPTIME*-complete.*

It is natural to ask what happens in the case of unrestricted queries and, more specifically, for queries with two inequalities. It was noted that the data complexity becomes higher when dealing with two inequalities, and a similar behavior should be expected for the combined complexity. Indeed, we have that:

Query	GLAV setting	LAV setting
DATALOG$^{\mathbf{C}(\neq)}$	EXPTIME-complete	EXPTIME-complete
1-CQ$^{\neq}$	EXPTIME-complete	NP-complete
k-CQ$^{\neq}$, $k \geq 2$	coNEXPTIME-complete	Π_2^p-complete
CQ$^{\neq}$	coNEXPTIME-complete	Π_2^p-complete

Table 1: Combined complexity of computing certain answers.

THEOREM 6.4. *For every $k \geq 2$,* CERTAIN-ANSWERS(GLAV, k-CQ$^{\neq}$) *is* coNEXPTIME*-complete.*

As we mentioned in the previous section, if data exchange settings are not considered to be fixed, then one has to deal with canonical universal solutions of exponential size when computing certain answers. A natural way to avoid this problem is by restricting the class of data exchange settings to be LAV settings. For the case of DATALOG$^{\mathbf{C}(\neq)}$ programs, this restriction does not help in reducing the complexity of computing certain answers. However, the evaluation of CQ$^{\neq}$ queries is not inherently exponential and, thus, we are able to considerably reduce the complexity by considering LAV settings, as we show in the following proposition.

PROPOSITION 6.5. CERTAIN-ANSWERS(LAV, 1-CQ$^{\neq}$) *is* NP*-complete, and* CERTAIN-ANSWERS(LAV, k-CQ$^{\neq}$) *is* Π_2^p*-complete for every $k \geq 2$.*

A natural question at this point is what happens with the complexity of the certain answers problem if one considers the entire class CQ$^{\neq}$. In the following theorem, we show that the same complexity bounds as in Theorem 6.4 and Proposition 6.5 hold in this case. Notice that the lower bounds in the following theorem follow from the lower bounds in these results.

THEOREM 6.6. CERTAIN-ANSWERS(GLAV, CQ$^{\neq}$) *is* coNEXPTIME*-complete and* CERTAIN-ANSWERS(LAV, CQ$^{\neq}$) *is* Π_2^p*-complete.*

We conclude this section with two remarks. First, notice that fixing data exchange settings has the same effect than restricting to LAV settings. In fact, the lower bounds in Proposition 6.5 remains the same for fixed LAV settings. Second, all the complexity bounds presented in this section remain the same if we allow unions of conjunctive queries with inequalities; if k-UCQ$^{\neq}$ is the class of unions of k-CQ$^{\neq}$ queries, then

PROPOSITION 6.7.

(1) CERTAIN-ANSWERS(GLAV, 1-UCQ$^{\neq}$) *is* EXPTIME*-complete,* CERTAIN-ANSWERS(LAV, 1-UCQ$^{\neq}$) *is* NP*-complete.*

(2) CERTAIN-ANSWERS(GLAV, k-UCQ$^{\neq}$) *is* coNEXPTIME*-complete, and* CERTAIN-ANSWERS(LAV, k-UCQ$^{\neq}$) *is* Π_2^p*-complete for every $k \geq 2$.*

(3) CERTAIN-ANSWERS(GLAV, UCQ$^{\neq}$) *is* coNEXPTIME*-complete, and* CERTAIN-ANSWERS(LAV, UCQ$^{\neq}$) *is* Π_2^p*-complete.*

7. CONCLUDING REMARKS

In this paper, we proposed the language DATALOG$^{\mathbf{C}(\neq)}$ that extends DATALOG with a restricted form of negation, and studied some of its fundamental properties. In particular, we showed that the certain answers to a DATALOG$^{\mathbf{C}(\neq)}$ program can be computed in polynomial time (in terms of data complexity), and we used this property to find tractable fragments of the class of unions of conjunctive queries with inequalities. In the paper, we also studied the combined complexity of computing certain answers to DATALOG$^{\mathbf{C}(\neq)}$ programs and other related query languages.

Many problems related to DATALOG$^{\mathbf{C}(\neq)}$ programs remain open. In particular, it would be interesting to know if it is decidable whether the certain answers to a query Q in UCQ$^{\neq}$ can be computed as the certain answers to a DATALOG$^{\mathbf{C}(\neq)}$ program Π_Q, and whether there exist a setting $\mathcal{M}$ and a query Q in UCQ$^{\neq}$ such that the problem CERTAIN-ANSWERS($\mathcal{M}, Q$) is in PTIME, but the certain answers to Q cannot be computed as the certain answers to a DATALOG$^{\mathbf{C}(\neq)}$ program Π_Q.

Acknowledgments

We are very grateful to Jorge Pérez for many helpful discussions, and to the anonymous referees for their comments. The authors were supported by: Arenas and Reutter - FONDECYT grant 1070732; Barceló - FONDECYT grant 11080011; Arenas and Barceló - grant P04-067-F from the Millennium Nucleus Centre for Web Research.

8. REFERENCES

[1] S. Abiteboul, and O. Duschka. Answering queries using materialized views. Gemo report 383.

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

[3] F. N. Afrati, C, Li, and V. Pavlaki. Data exchange in the presence of arithmetic comparisons. In *EDBT*, pages 487-498, 2008.

[4] M. Arenas, P. Barceló, R. Fagin, and L. Libkin. Locally consistent transformations and query answering in data exchange. In *PODS*, pages 229–240, 2004.

[5] C. Beeri, and M. Y. Vardi. A proof procedure for data dependencies. *Journal of the ACM*, 31(4):718–741, 1984.

[6] A. Deutsch, A. Nash, and J. B. Remmel. The chase revisited. In *PODS*, pages 149-158, 2008.

[7] R. Fagin, P. Kolaitis, L. Popa, W. C. Tan. Composing schema mappings: Second-order dependencies to the rescue. In *PODS*, pages 83–94, 2004.

[8] R. Fagin, P. G. Kolaitis, R. J. Miller, L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[9] R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: getting to the core. *ACM Transactions on Database Systems*, 30(1):174–210, 2005.

[10] G. Gottlob, C. Papadimitriou. On the complexity of single-rule datalog queries. *Information and Computation*, 183(1):104–122, 2003.

[11] P. Kolaitis. Schema mappings, data exchange, and metadata management. In *PODS*, pages 61–75, 2005.

[12] P. Kolaitis, J. Panttaja, and W.-C. Tan. The complexity of data exchange. In *PODS*, pages 30–39, 2006.

[13] T. Imielinski, W. Lipski. Incomplete information in relational databases. *Journal of the ACM* 31, 761–791, 1984.

[14] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

[15] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[16] L. Libkin. Data exchange and incomplete information. In *PODS*, pages 60–69, 2006.

[17] L. Libkin, C. Sirangelo. Data exchange and schema mappings in open and closed worlds. In *PODS*, pages 139–148, 2008.

[18] A. Mądry. Data exchange: On the complexity of answering queries with inequalities. *Information Processing Letters*, 94(6):253–257, 2005.

[19] M. Y. Vardi. The complexity of relational query languages. In *STOC*, pages 137–146, 1982.