Datalog as a Query Language for Data Exchange Systems

Marcelo Arenas[1], Pablo Barceló[2], and Juan L. Reutter[3]

[1] Dept. of Computer Science, Pontificia Universidad Católica de Chile
[2] Dept. of Computer Science, University of Chile
[3] School of Informatics, University of Edinburgh

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 for negated atoms, while most database query languages are equipped with negation. Unfortunately, adding an unrestricted form of negation to these languages yields to intractability of the problem of computing certain answers.

In order to face this challenge, we have recently proposed a language, called DATALOG$^{\mathbf{C}(\neq)}$ [5], that extends DATALOG with a restricted form of negation while keeping the good properties of DATALOG, and UCQ, for data exchange. In this article, we provide evidence in favor of the use of DATALOG$^{\mathbf{C}(\neq)}$ as a query language for data exchange systems. More precisely, we introduce the syntax and semantics of DATALOG$^{\mathbf{C}(\neq)}$, we present some of the fundamental results about this language shown in [5], and we extend those results to the case of data exchange settings that allow for constraints in the target schema. All of these results provide justification for the use of DATALOG$^{\mathbf{C}(\neq)}$ in practice.

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 [10]. Both the study of data exchange and schema mappings have become an active area of research during the last few years in the database community (see e.g. [10,11,4,9,17,13,18,12]).

In its simplest form, 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 $\forall\bar{x}\forall\bar{y}\,(\phi_{\mathbf{S}}(\bar{x},\bar{y}) \rightarrow \exists\bar{z}\,\psi_{\mathbf{T}}(\bar{x},\bar{z}))$, where $\phi_{\mathbf{S}}$ and $\psi_{\mathbf{T}}$ are conjunctions of relational atoms over $\mathbf{S}$ and $\mathbf{T}$, respectively. 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 satisfies each dependency in 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, each source instance has an infinite number of solutions. This immediately raises the question of which solution should be materialized. Initial work on data exchange [10] 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 [10] that for the class of data exchange settings defined above, a particular universal solution – called the *canonical* universal solution – can be computed in polynomial time.

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 [14,1,15,10]. 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 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 these classes by using some materialized solution. Fagin, Kolaitis, Miller, and Popa [10] 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. It is important to notice that in this result the complexity is measured only in terms of the size of the source instances (in particular, the data exchange setting and the query are assumed to be fixed). Thus, the previous result is stated in terms of *data* complexity [20].

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 retains several of the good properties of UCQ for data exchange. 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 DATALOG programs can be evaluated in polynomial time in the size of the data).

Unfortunately, both UCQ and DATALOG keeps us in the realm of the positive, while most database query languages are equipped with negation. However, adding an unrestricted form of negation to DATALOG (and even to the class of conjunctive queries) leads to intractability of the problem of computing certain answers. Thus, extending DATALOG with some form of negation that, on the one hand, allows to express interesting data exchange queries, and, on the other hand, retains the good properties of DATALOG for data exchange, is a nontrivial task that must be handled carefully.

In order to face this challenge, we have recently proposed a language, called DATALOG$^{\mathbf{C}(\neq)}$ [5], that extends DATALOG with a restricted form of negation while keeping the good properties of DATALOG, and UCQ, for data exchange. In this article, we provide evidence in favor of the use of $\text{DATALOG}^{\mathbf{C}(\neq)}$ as a query language for data exchange systems. More precisely, we start by introducing the syntax and semantics of $\text{DATALOG}^{\mathbf{C}(\neq)}$. Then we continue by presenting some of the fundamental results about this language shown in [5], which provide justification for the use of $\text{DATALOG}^{\mathbf{C}(\neq)}$ in practice. In particular, we show that the certain answers to a $\text{DATALOG}^{\mathbf{C}(\neq)}$ program can be computed in polynomial time, and that the language $\text{DATALOG}^{\mathbf{C}(\neq)}$ can be used to express interesting queries in the data exchange context, as every union of conjunctive queries with at most one inequality or negated relational atom per disjunct can be efficiently expressed as a $\text{DATALOG}^{\mathbf{C}(\neq)}$ program in the context of data exchange. We finish the paper by extending these results to the case of data exchange settings with constraints in the target, as explained below.

In addition to the data exchange scenario we have seen so far, it is common in the literature to assume that target schemas come with its own set of dependencies; i.e. each data exchange setting $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ is extended with a set Σ_t of dependencies over the schema $\mathbf{T}$, which are called target constraints. In that case, a target instance J is said to be a solution for the source instance I under the setting $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, if not only the pair (I, J) satisfies each dependency in Σ_{st}, but also J satisfies each dependency in Σ_t.

As it is to be expected, the addition of target dependencies makes the fundamental data exchange tasks more difficult, starting from the fact that it is no longer true that solutions exist for each source instance. Even worst, it follows from [13] that even for simple data exchange settings with target dependencies, the problem of checking for the existence of solutions may be undecidable. In order to solve this problem, the data exchange literature has identified a relevant class of target dependencies – those that consist of a set of equality-generating dependencies (that subsume keys) and a *weakly-acyclic* set of tuple-generating dependencies – that have the following good properties for data exchange [10]: Checking the existence of solutions is a tractable problem; and for every source instance that has a solution, a canonical universal solution can be computed in polynomial time. The latter implies that, for the class of data exchange settings extended with a set of target dependencies that consists of a set of equality-generating dependencies and a weakly-acyclic set of tuple-generating dependencies, the certain answers to each union of conjunctive queries Q can still be computed in polynomial time (by simply posing Q over the canonical universal solution J for a given source instance I, in case such J exists).

In this paper, we investigate the feasibility of using $\text{DATALOG}^{\mathbf{C}(\neq)}$ as a query language for data exchange settings extended with equality-generating target dependencies and weakly-acyclic sets of tuple-generating target dependencies. In particular, we prove that for this class of data exchange settings, the certain answers to each $\text{DATALOG}^{\mathbf{C}(\neq)}$ program can be computed in polynomial time. Also, we study the expressiveness of $\text{DATALOG}^{\mathbf{C}(\neq)}$ in this context, and show that every union of conjunctive queries with at most one inequality or negated relational atom per disjunct can be efficiently expressed as a $\text{DATALOG}^{\mathbf{C}(\neq)}$ program if only equality-generating target dependencies are considered. We also show that this result fails if, in addition, target constraints are allowed to contain weakly-acyclic sets of tuple-generating target dependencies; indeed, we prove in the paper that there exist a data exchange setting $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where Σ_t

is the union of a set of equality-generating dependencies and a weakly-acyclic set of tuple-generating dependencies, and a conjunctive query Q over $\mathbf{T}$ with one negated relational atom such that the problem of computing certain answers to Q under $\mathcal{M}$ is undecidable.

Organization of the paper. In Section 2, we introduce the terminology used in the paper. Then, in Section 3, we define the syntax and semantics of DATALOG$^{\mathbf{C}(\neq)}$ programs, and show their good properties for data exchange. In Section 4 we study the expressive power of DATALOG$^{\mathbf{C}(\neq)}$ programs. Concluding remarks are given in Section 5.

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* 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 [10,11]. 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 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 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 [10] and [11] a more general class of data exchange settings is presented, that also includes *target* dependencies; we consider these settings in Section 4.1). As usual in the data exchange literature (e.g., [10,11]), 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 $\forall \bar{x} \forall \bar{y}\, (\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z}))$, where $\phi(\bar{x}, \bar{y})$ is a conjunction of relational atoms over $\mathbf{S}$ and $\psi(\bar{x}, \bar{z})$ is a conjunction of relational atoms over $\mathbf{T}$.[1] 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 [15]. 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) [15] 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 $\forall \bar{x}\, (S(\bar{x}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z}))$, 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 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. $\square$

2.2 Universal Solutions and Canonical Universal Solution

It has been argued in [10] 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 (Example 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. $\square$

It follows from [10] 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 [6] (see e.g. [10]).

[1] We usually omit universal quantification in front of st-tgds and express them simply as $\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})$.

Remark: Notice that each target instance J' that contains the canonical universal solution J of a source instance I, is also a solution for I. Thus, each source instance has infinitely many solutions.

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 are 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 [14,1,15,10].

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 $\mathrm{certain}_{\mathcal{M}}(Q, I) =$ `true` if and only if Q holds for every J that is a solution for I under $\mathcal{M}$. We write $\mathrm{certain}_{\mathcal{M}}(Q, I) =$ `false` if it is not the case that $\mathrm{certain}_{\mathcal{M}}(Q, I) =$ `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	: Does $\bar{t} \in \mathrm{certain}_{\mathcal{M}}(Q, I)$?

3 Extending Query Languages for Data Exchange: DATALOG$^{\mathrm{C}(\neq)}$ 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 [10]. 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 can be evaluated in polynomial time in the size of the data).

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 each DATALOG program can be evaluated in polynomial time in the size of the data, we have that the certain answers to each DATALOG query Q can be obtained efficiently by first computing a universal solution J, and then evaluating Q over J. Thus, DATALOG preserves 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 [3,8], as they can be used to express interesting queries. Next, we show an example of this fact.

Example 3. 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, $\text{certain}_{\mathcal{M}}(Q,I) = \texttt{true}$ if and only if there exist elements $a, b \in \text{dom}(\textsc{Can}(I))$ 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, $\text{certain}_{\mathcal{M}}(Q,I) = \texttt{true}$ if and only if there exist elements $a, b \in \text{dom}(I)$ 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. [16]) 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 $Q'(\textsc{Can}(I)) = \text{certain}_{\mathcal{M}}(Q',I) = \text{certain}_{\mathcal{M}}(Q,I)$, 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 [10] that inequalities also add expressive power to UCQ in the context of data exchange.

Unfortunately, adding an unrestricted form of negation to DATALOG (or even to CQ) not only destroys preservation under homomorphisms, but also easily leads to intractability of the problem of computing certain answers [1,10]. More precisely, there is a setting $\mathcal{M}$ and a query Q in CQ$^{\neq}$ such that the problem $\textsc{Certain-Answers}(\mathcal{M}, Q)$ 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.

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

We have recently proposed a language DATALOG$^{\mathbf{C}(\neq)}$ [5] that adds negation in a natural way to DATALOG, while keeping the good properties of this language for data exchange. We define this language below.

Definition 1 (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 $(i \in [1, m])$.

Further, 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:

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

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 $\mathit{Pred}(\Pi)$, is the set of predicate symbols mentioned in Π, while the set of intensional predicates of Π, denoted by $\mathit{IPred}(\Pi)$, is the set of predicates symbols $R \in \mathit{Pred}(\Pi)$ 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 $\mathit{Pred}(\Pi)$. Then $\mathcal{T}(I)$ is an instance of $\mathit{Pred}(\Pi)$ such that for every $R \in \mathit{Pred}(\Pi)$ 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 $\mathbf{R} = \mathit{Pred}(\Pi) \smallsetminus \mathit{IPred}(\Pi)$ and Answer $\in \mathit{IPred}(\Pi)$. Given an instance I of $\mathbf{R}$ and a tuple $\bar{t}$ in $\mathrm{dom}(I)^n$, 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 $R \in \mathit{IPred}(\Pi)$.

3.2 Certain Answers for Datalog$^{\mathbf{C}(\neq)}$ Programs

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. Thus, Datalog$^{\mathbf{C}(\neq)}$ programs preserve the good properties of Datalog, and UCQ, for data exchange.

Proposition 1 ([5]). *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,* $\bar{t} \in \mathrm{certain}_{\mathcal{M}}(\Pi, I)$ *iff* $\bar{t} \in \Pi(J)$.

Thus, the certain answers of a $\text{DATALOG}^{\mathbf{C}(\neq)}$ program Π over I can be computed by directly posing Π over $\text{CAN}(I)$ and discarding tuples that contain nulls. This implies that for each data exchange setting $\mathcal{M}$, the problem $\text{CERTAIN-ANSWERS}(\mathcal{M}, \Pi)$ can be solved in polynomial time if Π is a $\text{DATALOG}^{\mathbf{C}(\neq)}$ program (since $\text{CAN}(I)$ can be computed in polynomial time and Π can be evaluated in polynomial time in the size of the data).

Corollary 1. *The problem* $\text{CERTAIN-ANSWERS}(\mathcal{M}, \Pi)$ *can be solved in polynomial time, for every data exchange setting* $\mathcal{M}$ *and* $\text{DATALOG}^{\mathbf{C}(\neq)}$ *program* Π*.*

4 On the Expressive Power of $\text{DATALOG}^{\mathbf{C}(\neq)}$ Programs

We have shown in [5] that $\text{DATALOG}^{\mathbf{C}(\neq)}$ programs are capable of expressing relevant data exchange properties. In particular, 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 (that is, the class of queries in $\text{UCQ}^{\neq}$ with at most one inequality per disjunct) 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 [10].

Theorem 1 ([5]). *Let Q be a* $\text{UCQ}^{\neq,\neg}$ *query over a schema* $\mathbf{T}$*, with at most one inequality or negated relational atom per disjunct. Then there exists a* $\text{DATALOG}^{\mathbf{C}(\neq)}$ *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.*

We sketch the proof of this theorem by means of an example, since we prove a stronger result later (Theorem 4).

Example 4. 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:

$$\text{DOM}(x) \leftarrow G(x,z) \tag{2}$$

$$\text{DOM}(x) \leftarrow G(z,x) \tag{3}$$

$$\text{DOM}(x) \leftarrow P(x) \tag{4}$$

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

$$\textsc{Equal}(x, x, z) \leftarrow \textsc{Dom}(x), \textsc{Dom}(z) \tag{5}$$

$$\textsc{Equal}(x, y, z) \leftarrow \textsc{Equal}(y, x, z) \tag{6}$$

$$\textsc{Equal}(x, y, z) \leftarrow \textsc{Equal}(x, w, z), \textsc{Equal}(w, y, z) \tag{7}$$

Predicate $\textsc{Equal}$ 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 $\textsc{Equal}(x, x, z) \leftarrow$ to say that $\textsc{Equal}$ is reflexive, as $\textsc{Datalog}^{\mathbf{C}(\neq)}$ 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:

$$U_1(x, y, z) \leftarrow G(x, y), \textsc{Dom}(z) \tag{8}$$

$$U_2(x, z) \leftarrow P(x), \textsc{Dom}(z) \tag{9}$$

$$U_1(x, y, z) \leftarrow U_1(u, v, z), \textsc{Equal}(u, x, z), \textsc{Equal}(v, y, z) \tag{10}$$

$$U_2(x, z) \leftarrow U_2(u, z), \textsc{Equal}(u, x, z) \tag{11}$$

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 $Q(x)$:

$$U_1(x, y, x) \leftarrow U_1(x, z, x), U_1(z, y, x) \tag{12}$$

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 $Q(x)$:

$$\textsc{Equal}(x, y, x) \leftarrow U_1(x, y, x) \tag{13}$$

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 $Q(x)$:

$$\textsc{Answer}(x) \leftarrow U_2(x, x), U_1(x, x, x), \mathbf{C}(x) \tag{14}$$

$$\textsc{Answer}(x) \leftarrow \textsc{Equal}(y, z, x), \mathbf{C}(x), \mathbf{C}(y), \mathbf{C}(z), y \neq z \tag{15}$$

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 ($\textsc{Equal}(b, c, a)$ 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 $h : \text{CAN}(I) \to J$. 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. □

As an immediate corollary to Theorem 1 and Corollary 1 we obtain the following:

Corollary 2. *The problem* CERTAIN-ANSWERS$(\mathcal{M}, Q)$ *can be solved in polynomial time, for every data exchange setting* $\mathcal{M}$ *and every union of conjunctive queries Q with at most one inequality or negated relational atom per disjunct.*

We note that this slightly generalizes one of the polynomial time results in [10], which is stated for the class of unions of conjunctive queries with at most one inequality per disjunct. The proof of the result in [10] uses different techniques, based on the chase procedure.

It is important to notice that Corollary 2 is, in a sense, optimal, as there is a LAV data exchange setting $\mathcal{M}$ and a conjunctive query with two inequalities, such that CERTAIN-ANSWERS$(\mathcal{M}, Q)$ is CONP-complete [19]. This shows that Theorem 1 cannot be further extended to deal with arbitrary conjunctive queries with negated atoms.

A natural question at this point is whether the problem CERTAIN-ANSWERS$(\mathcal{M}, Q)$ is PTIME-complete for some data exchange setting $\mathcal{M}$ and union of conjunctive queries Q with at most one negated atom per disjunct. The following proposition shows that this is indeed the case.

Proposition 2 ([5]). *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, under* LOGSPACE *reductions.*

The previous result establishes a difference with the class of unions of conjunctive queries (UCQ), for which the problem of computing certain answers under a setting $\mathcal{M}$ can be solved in LOGSPACE.

4.1 Adding Target Dependencies

In addition to the simple data exchange scenario we have seen so far, it is common in the literature to assume that target schemas come with its own set of dependencies Σ_t. Formally, data exchange settings with target dependencies (as presented, for instance, in [10,11]) are tuples of the form $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where $\mathbf{S}$, $\mathbf{T}$ and Σ_{st} are as before, and Σ_t is the union of (1) a set of *tuple-generating* dependencies (tgds), i.e. dependencies of the form $\forall\bar{x}\forall\bar{y}\,(\phi(\bar{x},\bar{y}) \rightarrow \exists\bar{z}\,\psi(\bar{x},\bar{z}))$, where $\phi(\bar{x},\bar{y})$ and $\psi(\bar{x},\bar{z})$ are conjunctions of atomic formulas in $\mathbf{T}$, and (2) a set of *equality-generating* dependencies (egds), i.e. dependencies of the form $\forall\bar{x}\,(\phi(\bar{x}) \rightarrow x_i = x_j)$, where $\phi(\bar{x})$ is a conjunction of atomic formulas in $\mathbf{T}$, and x_i, x_j are variables among those in $\bar{x}$.[2]

For settings with target dependencies, the solutions also have to satisfy the dependencies in Σ_t. That is, if $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ is a data exchange setting and I is an instance of $\mathbf{S}$, then an instance J of $\mathbf{T}$ is a solution for I if not only the pair (I, J) satisfies each dependency in Σ_{st}, but also J satisfies each dependency in Σ_t.

As it is to be expected, the addition of target dependencies makes the fundamental data exchange tasks more difficult, starting from the fact that it is no longer true that solutions exist for each source instance. Even worst, it follows from [13] that there exists a data exchange setting $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ with target dependencies such that the problem of checking for the existence of solutions under $\mathcal{M}$ is undecidable.

In order to overcome the aforementioned limitations, the data exchange community has identified a relevant class of target dependencies that has good properties for data exchange. To define this class, we need to introduce some terminology. Assume that Σ is a set of tgds over a schema $\mathbf{T}$. Then the *dependency* graph G_Σ of Σ is defined as follows:

(1) add a node (R, i) to G_Σ for every relation $R \in \mathbf{T}$ and $i \in \{1, \ldots, n\}$, where n is the arity of R;
(2) add an edge $(R, i) \rightarrow (T, j)$ to G_Σ if there exist a tgd $\phi(\bar{x},\bar{y}) \rightarrow \exists\bar{z}\,\psi(\bar{x},\bar{z})$ in Σ and a variable x such that x is mentioned in $\bar{x}$, x occurs in the i-th attribute of R in ϕ and x occurs in the j-th attribute of T in ψ;
(3) add a special edge $(R, i) \rightarrow^* (T, j)$ to G_Σ if there exists a tgd $\phi(\bar{x},\bar{y}) \rightarrow \exists\bar{z}\,\psi(\bar{x},\bar{z})$ in Σ such that a variable x occurs in the i-th attribute of R in ϕ and an existentially quantified variable z occurs in the j-th attribute of T in ψ.

Moreover, Σ is said to be *weakly acyclic* if the dependency graph G_Σ of Σ does not have a cycle going through an edge labeled $*$ [10]. Next theorem shows that the class of settings with weakly acyclic sets of tgds has good properties for data exchange.

Theorem 2 ([10]). *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting, where Σ_t is the union of a set of egds and a weakly-acyclic set of tgds. Then there is a polynomial time algorithm such that for every source instance I, it first decides whether a solution for I exists, and if that is the case, it computes a* canonical *universal solution for I in polynomial time.*

[2] As usual, we omit universal quantifiers in front of tgds and egds.

The latter implies that for the class of data exchange settings whose target dependencies consist of a set of equality-generating dependencies and a weakly-acyclic set of tuple-generating dependencies, the certain answers to each union of conjunctive queries Q for a source instance I, can be computed in polynomial time by simply posing Q over the canonical universal solution J for I and then discarding the tuples that contain nulls (in case such a solution J exists). Notice, however, that using exactly the same argument one can prove the stronger result that certain answers to $\text{DATALOG}^{\mathbf{C}(\neq)}$ programs can be computed in polynomial time under the class of settings specified in Theorem 2. This is because $\text{DATALOG}^{\mathbf{C}(\neq)}$ programs are preserved under data exchange homomorphisms and can be evaluated in polynomial time in the size of the data. Indeed,

Corollary 3. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting, where Σ_t is the union of a set of egds and a weakly acyclic set of tgds, and let Π be a* $\text{DATALOG}^{\mathbf{C}(\neq)}$ *program over* $\mathbf{T}$. *Then the problem* CERTAIN-ANSWERS$(\mathcal{M}, \Pi)$ *can be solved in polynomial time.*

Let us recall Corollary 2. It says that the certain answers to a union of conjunctive queries with at most one negated atom per disjunct, can be computed in polynomial time for settings without target dependencies. A natural question at this point is whether this positive result continues to hold if target schemas are allowed to contain dependencies of the form specified in Theorem 2. The following result shows that not only this is not the case, but also that the problem of computing certain answers to unions of conjunctive queries with at most one negated atom per disjunct is undecidable for this class of settings

Theorem 3. *There exists a data exchange setting $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where Σ_t is the union of a set of egds and a weakly-acyclic set of tgds, and a Boolean* $\text{CQ}^{\neg}$ *query Q over* $\mathbf{T}$ *with a single negated relational atom such that* CERTAIN-ANSWERS$(\mathcal{M}, Q)$ *is undecidable.*

Proof. Let $\mathbf{S}^\star$ be a source schema consisting of a ternary relation P, $T^\star$ a target schema consisting of a ternary relation R and $\mathcal{M}^\star = (\mathbf{S}^\star, T^\star, \Sigma_{st}^\star, \Sigma_t^\star)$ a data exchange setting, where $\Sigma_{st}^\star$ consists of the following dependency:

$$P(x, y, x) \rightarrow R(x, y, z),$$

and $\Sigma_t^\star$ consists of the egd:

$$R(x, y, z) \wedge R(x, y, w) \rightarrow z = w,$$

and the following tgds:

$$\begin{aligned} &R(x, y, u) \wedge R(y, z, v) \wedge R(u, z, w) \rightarrow R(x, v, w), \\ &R(x, y, z) \wedge R(x', y', z') \rightarrow \exists w_1 \exists w_2 \exists w_3 \exists w_4 \exists w_5 \exists w_6 \exists w_7 \exists w_8 \exists w_9 \, (R(x, x', w_1) \wedge \\ &\qquad R(x, y', w_2) \wedge R(x, z', w_3) \wedge R(y, x', w_4) \wedge R(y, y', w_5) \wedge \\ &\qquad R(y, z', w_6) \wedge R(z, x', w_7) \wedge R(z, y', w_8) \wedge R(z, z', w_9)). \end{aligned}$$

In [13], it was proved that the problem of verifying, given an instance I of $\mathbf{S}^\star$, whether there exists at least one solution for I under $\mathcal{M}^\star$ is undecidable. Next we show how to reduce this problem to the complement of our problem. More precisely, we define a data exchange setting $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where $\mathbf{S} = \mathbf{S}^\star$ and Σ_t is the union of a set of egds and a weakly-acyclic set of tgds, and a Boolean CQ$^\neg$ query Q over $\mathbf{T}$ with a single negated relational atom such that for every instance I of $\mathbf{S}^\star$: There exists at least one solution for I under $\mathcal{M}^\star$ if and only if $\text{certain}_{\mathcal{M}}(Q, I) = \texttt{false}$. From this, we conclude that CERTAIN-ANSWERS$(\mathcal{M}, Q)$ is undecidable.

Let $\mathbf{S} = \mathbf{S}^\star$, $\mathbf{T} = \mathbf{T}^\star \cup \{S\}$, where S is a ternary predicate, $\Sigma_{st} = \Sigma^\star_{st}$ and Σ_t be a set of target dependencies consisting of the egd:

$$R(x, y, z) \wedge R(x, y, w) \rightarrow z = w,$$

and the following tgds:

$$\begin{aligned}
&R(x, y, u) \wedge R(y, z, v) \wedge R(u, z, w) \rightarrow R(x, v, w),\\
&R(x, y, z) \wedge R(x', y', z') \rightarrow \exists w_1 \exists w_2 \exists w_3 \exists w_4 \exists w_5 \exists w_6 \exists w_7 \exists w_8 \exists w_9\, (S(x, x', w_1) \wedge\\
&\qquad S(x, y', w_2) \wedge S(x, z', w_3) \wedge S(y, x', w_4) \wedge S(y, y', w_5) \wedge\\
&\qquad S(y, z', w_6) \wedge S(z, x', w_7) \wedge S(z, y', w_8) \wedge S(z, z', w_9)).
\end{aligned}$$

Moreover, let Q be the following Boolean query:

$$\exists x \exists y \exists z\, (S(x, y, z) \wedge \neg R(x, y, z)).$$

It is important to notice that the set of tgds in $\Sigma^\star_t$ is not weakly acyclic, while the set of tgds in Σ_t is weakly acyclic. Next we show that for every instance I of $\mathbf{S}^\star$, it holds that there exists at least one solution for I under $\mathcal{M}^\star$ if and only if $\text{certain}_{\mathcal{M}}(Q, I) = \texttt{false}$.

($\Rightarrow$) Let I be an instance of $\mathbf{S}^\star$ and $J^\star$ a solution for I under $\mathcal{M}^\star$. Define J as the following instance of $\mathbf{T}$: $R^T = R^{T^\star}$ and $S^T = R^{T^\star}$. Given that $(I, J^\star)$ satisfies $\Sigma^\star_{st}$ and $J^\star$ satisfies $\Sigma^\star_t$, we have that (I, J) satisfies Σ_{st} and J satisfies Σ_t and, therefore, J is a solution for I under $\mathcal{M}$. Thus, given that Q does not hold in J (since $R^T = S^T$), we conclude that $\text{certain}_{\mathcal{M}}(Q, I) = \texttt{false}$.

($\Leftarrow$) Assume that I is an instance of $\mathbf{S}^\star$ such that $\text{certain}_{\mathcal{M}}(Q, I) = \texttt{false}$. Then let J be a solution of I under $\mathcal{M}$ such that Q does not hold in J, and $J^\star$ an instance of $\mathbf{T}^\star$ defined as $R^{T^\star} = R^T$. Given that (I, J) satisfies Σ_{st}, we have that $(I, J^\star)$ satisfies $\Sigma^\star_{st}$. Furthermore, given that Q does not hold in J, we have that J satisfies dependency:

$$\forall x \forall y \forall z\, (S(x, y, z) \rightarrow R(x, y, z)).$$

Thus, given that J satisfies Σ_t, we conclude that $J^\star$ satisfies $\Sigma^\star_t$. Hence, we have that $J^\star$ is a solution for I under $\mathcal{M}^\star$, from which we deduce that there exists at least one solution for I under $\mathcal{M}^\star$. This concludes the proof of the theorem. □

A natural way to ensure that the problem of computing certain answers to unions of conjunctive queries with at most one negated atom per disjunct remains tractable, in the presence of target dependencies, is by restricting the class of target dependencies allowed. Indeed, we prove below that this is the case for the class of data exchange settings that only allow egds in the target. The interesting part of this is not the result itself – which is a slight extension of a result in [10] – but the fact that our proof relies again on the translation of the problem of computing certain answers for this class of queries, under the settings described above, into the problem of computing certain answers to $\text{DATALOG}^{\mathbf{C}(\neq)}$ programs.

A naïve approach to prove this result would be the following. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting, where Σ_t consists of a set of equality-generating dependencies, and let $\mathcal{M}'$ be the setting obtained from $\mathcal{M}$ by removing Σ_t. As we have mentioned above, for each union of conjunctive queries Q, with at most one inequality or negated relational atom per disjunct, one can construct a $\text{DATALOG}^{\mathbf{C}(\neq)}$ program Π_Q such that the certain answers to Q under $\mathcal{M}'$ coincide with the certain answers to Π_Q under $\mathcal{M}'$. Then one could implement the following algorithm for computing certain answers to Q under $\mathcal{M}$: Given a source instance I, compute the canonical universal solution J for I under $\mathcal{M}$ (in case such a solution exists); evaluate Π_Q over J; discard tuples that contain nulls.

Unfortunately, this simple algorithm is not correct for the following reason. Evaluating the program Π_Q over J may force some elements in J to be equal, which, in turn, may imply some of the dependencies in Σ_t to be triggered in the process. This suggests that if one wants to compute the certain answers to Q (under $\mathcal{M}$) with a $\text{DATALOG}^{\mathbf{C}(\neq)}$ program Π_Q, then Π_Q must take into consideration not only Q but also Σ_t. Indeed, we show next that for each union of conjunctive queries, with at most one negated atom per disjunct, it is possible to construct a $\text{DATALOG}^{\mathbf{C}(\neq)}$ program Π_{Q,Σ_t} such that the certain answers to Q and to Π_{Q,Σ_t} (under $\mathcal{M}$) coincide. Formally,

Theorem 4. *Let Q be a $\text{UCQ}^{\neq,\neg}$ k-ary query over a schema $\mathbf{T}$ ($k \geq 0$), with at most one inequality or negated relational atom per disjunct. Further, let Σ_t be a set of egds over $\mathbf{T}$. Then there exists a $\text{DATALOG}^{\mathbf{C}(\neq)}$ program Π_{Q,Σ_t} over $\mathbf{T}$ such that for every data exchange setting $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, instance I of $\mathbf{S}$ and tuple $\bar{a} \in dom(I)^k$:*

$$\bar{a} \in \text{certain}_{\mathcal{M}}(Q, I) \text{ if and only if } \bar{a} \in \text{certain}_{\mathcal{M}}(\Pi_{Q,\Sigma_t}, I).$$

Moreover, Π_{Q,Σ_t} can be effectively constructed from Q and Σ_t in polynomial time.

Proof. Assume that $\mathbf{T} = \{T_1, \ldots, T_k\}$, where each T_i has arity $n_i > 0$, that $Q(\bar{x}) = Q_1(\bar{x}) \vee \cdots \vee Q_\ell(\bar{x})$, where $\bar{x} = (x_1, \ldots, x_m)$ and each $Q_i(\bar{x})$ is a conjunctive query with at most one inequality or negated relational atom, and that $\Sigma_t = \{\alpha_1, \ldots, \alpha_q\}$ is a set of egds.

Then the set of intensional predicates of $\text{DATALOG}^{\mathbf{C}(\neq)}$ program Π_{Q,Σ_t} is

$$\{U_1, \ldots, U_k, \text{DOM}, \text{EQUAL}, \text{ANSWER}\},$$

where each U_i ($i \in [1, k]$) has arity $n_i + m$, DOM has arity 1, EQUAL has arity $2 + m$ and ANSWER has arity m. Moreover, the set of rules of Π_{Q,Σ_t} is defined as follows.

– For every predicate $T_i \in \mathbf{T}$, Π_{Q,Σ_t} includes the following k rules:

$$\begin{aligned}
\textsc{Dom}(x) &\leftarrow T_i(x, y_2, y_3, \ldots, y_{n_i-1}, y_{n_i})\\
\textsc{Dom}(x) &\leftarrow T_i(y_1, x, y_3, \ldots, y_{n_i-1}, y_{n_i})\\
&\ldots\\
\textsc{Dom}(x) &\leftarrow T_i(y_1, y_2, y_3, \ldots, y_{n_i-1}, x)
\end{aligned}$$

– Π_{Q,Σ_t} includes the following rules for predicate EQUAL:

$$\begin{aligned}
\textsc{Equal}(x, x, z_1, \ldots, z_m) &\leftarrow \textsc{Dom}(x), \textsc{Dom}(z_1), \ldots, \textsc{Dom}(z_m)\\
\textsc{Equal}(x, y, z_1, \ldots, z_m) &\leftarrow \textsc{Equal}(y, x, z_1, \ldots, z_m)\\
\textsc{Equal}(x, y, z_1, \ldots, z_m) &\leftarrow \textsc{Equal}(x, w, z_1, \ldots, z_m), \textsc{Equal}(w, y, z_1, \ldots, z_m)
\end{aligned}$$

– For every predicate U_i, Π_{Q,Σ_t} includes the following rules:

$$\begin{aligned}
U_i(y_1, \ldots, y_{n_i}, z_1, \ldots, z_m) &\leftarrow T_i(y_1, \ldots, y_{n_i}), \textsc{Dom}(z_1), \ldots, \textsc{Dom}(z_m)\\
U_i(y_1, \ldots, y_{n_i}, z_1, \ldots, z_m) &\leftarrow U_i(w_1, \ldots, w_{n_i}, z_1, \ldots, z_m),\\
&\qquad \textsc{Equal}(w_1, y_1, z_1, \ldots, z_m), \ldots,\\
&\qquad \textsc{Equal}(w_{n_i}, y_{n_i}, z_1, \ldots, z_m)
\end{aligned}$$

– Let $i \in [1, \ell]$. First, assume that $Q_i(\bar{x})$ does not contain any negated atom. Then $Q_i(\bar{x})$ is equal to $\exists \bar{u}\,(T_{p_1}(\bar{u}_1) \wedge \cdots \wedge T_{p_n}(\bar{u}_n))$, where $p_j \in [1, k]$ and every variable in $\bar{u}_j$ is mentioned in either $\bar{u}$ or $\bar{x}$, for every $j \in [1, n]$. In this case, program Π_{Q,Σ_t} includes the following rule:

$$\textsc{Answer}(\bar{x}) \leftarrow U_{p_1}(\bar{u}_1, \bar{x}), \ldots, U_{p_n}(\bar{u}_n, \bar{x}), \mathbf{C}(x_1), \ldots, \mathbf{C}(x_m) \qquad (16)$$

Notice that this rule is well defined since the set $\bar{x}$ is the set of free variables of $\exists \bar{u}\,(T_{p_1}(\bar{u}_1) \wedge \cdots \wedge T_{p_n}(\bar{u}_n))$. Second, assume that $Q_i(\bar{x})$ contains a negated relational atom. Then $Q_i(\bar{x})$ is equal to $\exists \bar{u}\,(T_{p_1}(\bar{u}_1) \wedge \cdots \wedge T_{p_n}(\bar{u}_n) \wedge \neg T_{p_{n+1}}(\bar{u}_{n+1}))$, where $p_j \in [1, k]$ and every variable in $\bar{u}_j$ is mentioned in either $\bar{u}$ or $\bar{x}$, for every $j \in [1, n+1]$. In this case, program Π_{Q,Σ_t} includes the following rule:

$$U_{p_{n+1}}(\bar{u}_{n+1}, \bar{x}) \leftarrow U_{p_1}(\bar{u}_1, \bar{x}), \ldots, U_{p_n}(\bar{u}_n, \bar{x}). \qquad (17)$$

This rule is well defined since $\exists \bar{u}\,(T_{p_1}(\bar{u}_1) \wedge \cdots \wedge T_{p_n}(\bar{u}_n) \wedge \neg T_{p_{n+1}}(\bar{u}_{n+1}))$ is a safe query. Finally, assume that $Q_i(\bar{x})$ contains an inequality. Then $Q_i(\bar{x})$ is equal to $\exists \bar{u}\,(T_{p_1}(\bar{u}_1) \wedge \cdots \wedge T_{p_n}(\bar{u}_n) \wedge v_1 \neq v_2)$, where $p_j \in [1, k]$ and every variable in $\bar{u}_j$ is mentioned in either $\bar{u}$ or $\bar{x}$, for every $j \in [1, n]$, and v_1, v_2 are mentioned in $\bar{u}$ or $\bar{x}$. In this case, program Π_{Q,Σ_t} includes the following rule:

$$\textsc{Equal}(v_1, v_2, \bar{x}) \leftarrow U_{p_1}(\bar{u}_1, \bar{x}), \ldots, U_{p_n}(\bar{u}_n, \bar{x}) \qquad (18)$$

We note that the rule above is well defined since $\exists \bar{u}\,(T_{p_1}(\bar{u}_1) \wedge \cdots \wedge T_{p_n}(\bar{u}_n) \wedge v_1 \neq v_2)$ is a safe query.

– For each $i \in [1, q]$, assume that dependency α_i is of form $(T_{p_1}(\bar{x}_1) \wedge \cdots \wedge T_{p_n}(\bar{x}_n) \rightarrow u = v)$, where each $p_j \in [1, k]$ and variables u and v are mentioned in $\bar{x}_1, \ldots, \bar{x}_n$. Then the program Π_{Q,Σ_t} includes the following rule:

$$\text{EQUAL}(u, v, \bar{x}) \leftarrow U_{p_1}(\bar{x}_1, \bar{x}), \ldots, U_{p_n}(\bar{x}_n, \bar{x}) \quad (19)$$

– Finally, if Q has at least one inequality, or if Σ_t is nonempty, program Π_{Q,Σ_t} includes the rule:

$$\text{ANSWER}(\bar{x}) \leftarrow \text{EQUAL}(u, v, \bar{x}), \mathbf{C}(u), \mathbf{C}(v), u \neq v, \mathbf{C}(x_1), \ldots, \mathbf{C}(x_m) \quad (20)$$

Let $\bar{a}$ be a tuple of elements from the domain of a source instance I. Each predicate U_i in Π_{Q,Σ_t} is used as a copy of T_i but with m extra arguments that store tuple $\bar{a}$. These predicates are used when testing whether $\bar{a}$ is a certain answer for Q over I. More specifically, the rules of Π_{Q,Σ_t} try to construct from the canonical universal solution $\text{CAN}(I)$ a solution J for I such that $\bar{a} \notin Q(J)$. Thus, if in a solution J for I, it holds that $\bar{a} \in Q(J)$ because $\bar{a} \in Q_i(J)$, where $Q_i(\bar{x})$ is equal to $\exists \bar{u}\, (T_{p_1}(\bar{u}_1) \wedge \cdots \wedge T_{p_n}(\bar{u}_n) \wedge \neg T_{p_{n+1}}(\bar{u}_{n+1}))$, then Π_{Q,Σ_t} uses rule (17) to create a new solution where the negative atom of Q_i does not hold. In the same way, if in a solution J for I, it holds that $\bar{a} \in Q(J)$ because $\bar{a} \in Q_i(J)$, where $Q_i(\bar{x})$ is equal to $\exists \bar{u}\, (T_{p_1}(\bar{u}_1) \wedge \cdots \wedge T_{p_n}(\bar{u}_n) \wedge v_1 \neq v_2)$, then Π_{Q,Σ_t} uses rule (18) to create a new solution where the values assigned to v_1 and v_2 are equal (predicate EQUAL is used to store this fact). If v_1 or v_2 is assigned a null value, then it is possible to create a solution where the values assigned to these variables are the same. But this is not possible if both v_1 and v_2 are assigned different constant values. In fact, it follows from [10] that this implies that it is not possible to find a solution J' for I where $\bar{a} \notin Q(J')$, and in this case rule (20) is used to indicate that $\bar{a}$ is a certain answer for Q over I. Notice, however, that every solution for I must satisfy the dependencies in Σ_t. Thus, if for some Q_i the program uses rules (17) or (18) to create an instance J such that $\bar{a} \notin Q_i(J)$, but J does not satisfies a dependency $\phi(\bar{x}) \rightarrow x_i = x_j$ in Σ_t, then rule (19) must be used to *repair* that instance, obtaining a solution J' for I in which the values assigned to x_i and x_j are the same, but still holds that $\bar{a} \notin Q_i(J)$. This will not be possible if both x_i and x_j are assigned different constant values; in this case, it can be proved using results in [10] that it is not possible to create a solution such that $\bar{a} \notin Q_i(J)$, and thus rule (20) is used to indicate that $\bar{a}$ is a certain answer for Q over I.

By using the above observations, it is not difficult to prove that the statement of the theorem holds, which was to be shown. □

As a corollary of Theorem 4 and Corollary 1, we immediately obtain the following desired result:

Corollary 4. *Let Q be a* UCQ$^{\neq,\neg}$ *query over a schema $\mathbf{T}$, with at most one inequality or negated relational atom per disjunct, and let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting such that Σ_t consists of a set of egds. Then the problem* CERTAIN-ANSWERS$(\mathcal{M}, Q)$ *can be solved in polynomial time.*

Another possible way to retain tractability of the problem $\text{CERTAIN-ANSWERS}(\mathcal{M}, Q)$, where Q is a union of conjunctive queries with at most one negated atom per disjunct and $\mathcal{M}$ is a setting with target dependencies, is by restricting the class of queries allowed. Indeed, it has been proved in [10] that for the class of settings whose sets of target dependencies consist of egds and weakly-acyclic sets of tgds, the certain answers to a union of conjunctive queries with at most one inequality per disjunct can be computed in polynomial time by using an algorithm based on the *chase* procedure. It is an interesting open problem whether this result can also be proved with the help of $\text{DATALOG}^{\mathbf{C}(\neq)}$ programs, in the style of Theorem 4 and Corollary 4.

5 Concluding Remarks

In this paper, we presented the language $\text{DATALOG}^{\mathbf{C}(\neq)}$ 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 $\text{DATALOG}^{\mathbf{C}(\neq)}$ program can be computed in polynomial time, and we used this property to find tractable fragments of the class of unions of conjunctive queries with inequalities (even in the presence of target dependencies).

Both the problem of the existence of solutions and the computation of certain answers are defined in the paper assuming settings to be fixed. That is, in terms of Vardi's taxonomy [20], we study the *data* complexity of these problems. This makes sense in the database context, as usually specifications and queries are much smaller than source instances. However, a more refined complexity analysis of these problems should not consider any of their parameters to be fixed. This corresponds to the *combined* complexity of the problems mentioned above. The combined complexity of the problem of existence of solutions was studied in [13,7], while the combined complexity of the problem of computing certain answers was studied in [5].

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

Acknowledgments. We are very grateful to Jorge Pérez for many helpful discussions. The authors were supported by: Arenas - FONDECYT grant 1090565; Barceló - FONDECYT grant 11080011; Reutter - EPSRC grant G049165.

References

1. Abiteboul, S., Duschka, O.: Answering queries using materialized views. Gemo report 383
2. Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases. Addison-Wesley, Reading (1995)
3. Afrati, F.N., Li, C., Pavlaki, V.: Data exchange in the presence of arithmetic comparisons. In: EDBT, pp. 487–498 (2008)

4. Arenas, M., Barceló, P., Fagin, R., Libkin, L.: Locally consistent transformations and query answering in data exchange. In: PODS, pp. 229–240 (2004)
5. Arenas, M., Barceló, P., Reutter, J.: Query languages for data exchange: Beyond unions of conjunctive queries. Accepted for publication in Theory of Computing Systems, ToCS (2010); Preliminary version in Proceedings 12th International Conference on Database Theory (ICDT 2009), pp. 73–83 (2009)
6. Beeri, C., Vardi, M.Y.: A proof procedure for data dependencies. Journal of the ACM 31(4), 718–741 (1984)
7. Calì, A., Gottlob, G., Pieris, A.: Query answering under non-guarded rules in datalog+/-. In: Hitzler, P., Lukasiewicz, T. (eds.) RR 2010. LNCS, vol. 6333, pp. 1–17. Springer, Heidelberg (2010)
8. Deutsch, A., Nash, A., Remmel, J.B.: The chase revisited. In: PODS, pp. 149–158 (2008)
9. Fagin, R., Kolaitis, P., Popa, L., Tan, W.C.: Composing schema mappings: Second-order dependencies to the rescue. In: PODS, pp. 83–94 (2004)
10. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theoretical Computer Science 336(1), 89–124 (2005)
11. Fagin, R., Kolaitis, P.G., Popa, L.: Data exchange: getting to the core. ACM Transactions on Database Systems 30(1), 174–210 (2005)
12. Kolaitis, P.: Schema mappings, data exchange, and metadata management. In: PODS, pp. 61–75 (2005)
13. Kolaitis, P., Panttaja, J., Tan, W.-C.: The complexity of data exchange. In: PODS, pp. 30–39 (2006)
14. Imielinski, T., Lipski, W.: Incomplete information in relational databases. Journal of the ACM 31, 761–791 (1984)
15. Lenzerini, M.: Data integration: A theoretical perspective. In: PODS, pp. 233–246 (2002)
16. Libkin, L.: Elements of Finite Model Theory. Springer, Heidelberg (2004)
17. Libkin, L.: Data exchange and incomplete information. In: PODS, pp. 60–69 (2006)
18. Libkin, L., Sirangelo, C.: Data exchange and schema mappings in open and closed worlds. In: PODS, pp. 139–148 (2008)
19. Mądry, A.: Data exchange: On the complexity of answering queries with inequalities. Information Processing Letters 94(6), 253–257 (2005)
20. Vardi, M.Y.: The complexity of relational query languages. In: STOC, pp. 137–146 (1982)