Data Exchange: Semantics and Query Answering

Ronald Fagin[1], Phokion G. Kolaitis[2*], Renée J. Miller[3*], and Lucian Popa[1]

[1] IBM Almaden Research Center
[2] UC Santa Cruz
[3] University of Toronto

Abstract. Data exchange is the problem of taking data structured under a source schema and creating an instance of a target schema that reflects the source data as accurately as possible. In this paper, we address foundational and algorithmic issues related to the semantics of data exchange and to query answering in the context of data exchange. These issues arise because, given a source instance, there may be many target instances that satisfy the constraints of the data exchange problem. We give an algebraic specification that selects, among all solutions to the data exchange problem, a special class of solutions that we call *universal*. A universal solution has no more and no less data than required for data exchange and it represents the entire space of possible solutions. We then identify fairly general, and practical, conditions that guarantee the existence of a universal solution and yield algorithms to compute a canonical universal solution efficiently. We adopt the notion of "certain answers" in indefinite databases for the semantics for query answering in data exchange. We investigate the computational complexity of computing the certain answers in this context and also study the problem of computing the certain answers of target queries by simply evaluating them on a canonical universal solution.

1 Introduction

In *data exchange*, data structured under one schema (which we call a *source schema*) must be restructured and translated into an instance of a different schema (a *target schema*). Data exchange is used in many tasks that require data to be transferred between existing, independently created applications. The first systems supporting the restructuring and translation of data were built several decades ago. An early such system was EXPRESS [21], which performed data exchange between hierarchical schemas. The need for systems supporting data exchange has persisted over the years. Recently this need has become more pronounced, as the terrain for data exchange has expanded with the proliferation of web data that are stored in different formats, such as traditional relational database schemas, semi-structured schemas (for example, DTDs or XML schemas), and various scientific formats. In this paper, we address several foundational and algorithmic issues related to the semantics of data exchange and to query answering in the context of data exchange.

The data exchange problem. In a data exchange setting, we have a source schema **S** and a target schema **T**, where we assume that **S** and **T** are disjoint. Since **T** can be an independently created schema, it may have its own constraints that are given as a set Σ_t of sentences in some logical formalism over **T**. In addition, we must have a way of modeling the relationship between the source and target schemas. This essential element of data exchange is captured by *source-to-target dependencies* that specify how and what source data should appear in the target. These dependencies are assertions between a source query and a target query. Formally, we have a set Σ_{st} of *source-to-target dependencies* of the form $\forall\mathbf{x}(\phi_\mathbf{S}(\mathbf{x}) \rightarrow \chi_\mathbf{T}(\mathbf{x}))$, where $\phi_\mathbf{S}(\mathbf{x})$ is a formula in some logical formalism over **S** and $\chi_\mathbf{T}(\mathbf{x})$ is a formula in some (perhaps different) logical formalism over **T**.

Consider a data exchange setting determined by **S**, **T**, Σ_{st}, and Σ_t as above. This setting gives rise to the following *data exchange problem:* given an instance I over the source schema **S**, materialize an instance J over the target schema **T** such that the target dependencies Σ_t are satisfied by J, and the source-to-target dependencies Σ_{st} are satisfied by I and J together. The first crucial observation is that there may be many solutions (or none) for a given instance of the data exchange problem. Hence, several conceptual and technical questions arise concerning the semantics of data exchange. First, when does a solution exist? If many solutions exist, which solution should we materialize and what properties should it have, so that it reflects the source data as accurately as possible? Finally, can such a "good" solution be efficiently computed?

We consider the semantics of the data exchange problem to be one of the two main issues in data exchange. We believe that the other main issue is query answering. Specifically, suppose that q is a query over the target schema **T** and I is an instance over the source schema **S**. What does answering q with respect to I mean? Clearly, there is an ambiguity arising from the fact that, as mentioned earlier, there may be many solutions J for I and, as a result, different such solutions J may produce different answers $q(J)$. This conceptual difficulty was first encountered in the context of *incomplete* or *indefinite* databases (see, for instance, [23]), where one has to find the "right" answers to a query posed against a set of "possible" databases. There is general agreement that in the context of incomplete databases, the "right" answers are the *certain* answers, that is, the answers that occur in the intersection of all $q(J)$'s, as J varies over all "possible" databases. This notion makes good sense for data exchange as well, where the "possible" databases are the solutions J for the instance I. We thus adopt the certain answers for the semantics of query answering in the data exchange setting and investigate the complexity of computing the certain answers in the data exchange setting. A related important question is whether the certain answers of a query can be computed by query evaluation on the "good" target instance that we chose to materialize.

Data exchange vs. data integration. Before describing our results on data exchange, we briefly compare and contrast data exchange with *data integration*. Following the terminology and notation in the recent overview [13], a *data integration system* is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G}$ is the *global schema*, $\mathcal{S}$ is the *source schema*, and $\mathcal{M}$ is a set of *assertions* relating elements of the global schema with elements of the source schema. Both $\mathcal{G}$ and $\mathcal{S}$ are specified in suitable languages that may allow for the expression of various constraints. In this generality, a data exchange setting $\langle \mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t \rangle$ can be thought of as a data integration system in which **S** is the source schema, **T** and Σ_t form the global schema, and the source-to-target dependencies in Σ_{st} are the assertions of the

data integration system. In practice, however, most data integration systems studied to date are either LAV (*local-as-view*) systems or GAV (*global-as-view*) systems [11,13, 14]. In a LAV system, each assertion in $\mathcal{M}$ relates one element of the source schema $\mathcal{S}$ to a query (a view) over the global schema $\mathcal{G}$; moreover, it is usually assumed that there are no target constraints ($\Sigma_t = \emptyset$). In a GAV system the reverse holds, that is, each assertion in $\mathcal{M}$ relates one element of the global schema $\mathcal{G}$ to a query (a view) over the source schema $\mathcal{S}$. Since the source-to-target dependencies Σ_{st} relate a query over the source schema **S** to a query over the target schema **T**, a data exchange setting is neither a LAV nor a GAV system. Instead, it can be thought of as a GLAV (*global-and-local-as-view*) system [10,13].

The above similarities notwithstanding, there are important differences between data exchange and data integration. As mentioned earlier, in data exchange scenarios, the target schema is often independently created and comes with its own constraints. In data integration, however, the global schema $\mathcal{G}$ is commonly assumed to be a reconciled, virtual view of a heterogeneous collection of sources and, as such, has no constraints. In fact, with the notable exception of [5], which studied the impact of key and foreign key constraints on query answering in a GAV system, most research on data integration has not taken target constraints into account. Still, a more significant difference is that in a data exchange setting we have to actually materialize a finite target instance that best reflects the given source instance. In data integration no such exchange of data is required. For query answering, both data exchange and data integration use the certain answers as the standard semantics of queries over the target (global) schema. In data integration, the source instances are used to compute the certain answers of queries over the global schema. In contrast, in a data exchange setting, it may not be feasible to couple applications together in a manner that data may be retrieved and shared on-demand at query time. This may occur, for instance, in peer-to-peer applications that must share data, yet maintain autonomy. Hence, queries over the target schema may have to be answered using the materialized target instance alone, without reference to the original source instance. This leads to the following problem in data exchange: under what conditions and for which queries can the certain answers be computed using just the materialized target instance?

Motivation from Clio. The results presented here were motivated by our experience with Clio, a prototype schema mapping and data exchange tool to whose development some of us have contributed [18,19]. In Clio, source-to-target dependencies are (semi)-automatically generated from a set of correspondences between the source schema and the target schema; these dependencies can then be used in a data integration system to compute the certain answers to target queries. Most of the applications we considered, however, were decoupled applications that would have had to be rewritten to operate cooperatively, as required in data integration. For this reason, early on in the development of Clio, we recognized the need to go farther and, given a source instance, generate a single "universal" target instance that was the result of the schema mapping. In designing the algorithms used in Clio for creating the target instance, we were guided mainly by our own intuition rather than by formal considerations. It should be noted that there is a long history of work on data translation that focuses on taking high-level, data-independent translation rules and generating efficient, executable translation programs [1,20,21]. Yet, we could not find a formal justification for the intuitive choices we made in creating

the target instance. In seeking to formalize this intuition and justify the choices made in Clio, we were led to explore foundational and algorithmic issues related to the semantics of data exchange and query answering in this setting. Clio supports schemas that are relational or semi-structured. However, challenging issues already arise in the relational case. For this reason, here we focus exclusively on data exchange between relational schemas; extending this work to other types of schemas is the subject of on-going investigation.

Summary of Results. In Section 2, we formally introduce the data exchange problem. We then give an algebraic specification that selects, among all possible solutions for a given source instance, a special class of solutions that we call *universal*. More precisely, a solution for an instance of the data exchange problem is universal if it has homomorphisms to all solutions for that instance. We show that a universal solution has "good" properties that justify its choice for the semantics of the data exchange problem. We note that Calì et al. [5] studied GAV systems with key and foreign key constraints at the target. By means of a logic program that simulates the foreign key constraints, they constructed a *canonical database*, which turns out to be a particular example of our notion of universal solution.

Given the declarative specification of universal solutions, we go on in Section 3 to identify fairly general, yet practical, sufficient conditions that guarantee the existence of a universal solution and yield algorithms to compute such a solution efficiently. We introduce the concept of a *weakly acyclic* set of target dependencies, which is broad enough to contain as special cases both sets of full tuple-generating dependencies (full tgds) [4] and acyclic sets of inclusion dependencies [7]. We then show that if Σ_{st} is a set of tuple-generating dependencies (tgds) and Σ_t is the union of a weakly acyclic set of tgds with a set of equality generating dependencies (egds), then, given a source instance of the data exchange problem, (1) a universal solution exists if and only if a solution exists, and (2) there is a polynomial-time algorithm that determines whether a solution exists and, if so, it produces a particular universal solution, which we call the *canonical* universal solution. These results make use of the classical *chase* procedure [4,15]. We note that, even though the chase has been widely used in reasoning about dependencies, we have not been able to find any explicit references to the fact that the chase can produce instances that have homomorphisms to all instances satisfying the dependencies under consideration.

After this, in Sections 4 and 5, we address algorithmic issues related to query answering in the data exchange setting. We study the computational complexity of computing the certain answers, and explore the boundary of what queries can and cannot be answered in a data exchange setting using the exchanged target instance alone. On the positive side, if q is a union of conjunctive queries, then it is easy to show that the certain answers of q can indeed be obtained by evaluating q on an arbitrary universal solution. Moreover, universal solutions are the only solutions possessing this property; this can be seen as further justification for our choice to use universal solutions for data exchange. It also follows that, whenever a universal solution can be computed in polynomial time, the certain answers of unions of conjunctive queries can be computed in polynomial time (in particular, this is true when the dependencies in Σ_{st} and Σ_t satisfy the conditions identified in Section 3).

On the negative side, a dramatic change occurs when queries have inequalities. The hardness of this problem arises from the complexity of reasoning over uncertain databases, not from the data exchange *per se*. Indeed, Abiteboul and Duschka [2] showed that in a LAV data integration system and with conjunctive queries as views, computing the certain answers of conjunctive queries with inequalities is a coNP-complete problem. Since LAV is a special case of a data exchange setting in which the canonical universal solution can be computed in polynomial time, it follows that, unless P = NP, we cannot compute the certain answers of conjunctive queries with inequalities by evaluating them on the canonical universal solution (or on any other polynomial-time computable universal solution).

We take a closer look at conjunctive queries with inequalities by focusing on the number of inequalities. In [2], it was claimed that in the LAV setting and with conjunctive queries as views, computing the certain answers of conjunctive queries with a single inequality is a coNP-hard problem. The reduction given in that paper, however, is not correct; a different reduction in the unpublished full version [3] shows that computing the certain answers of conjunctive queries with six (or more) inequalities is a coNP-complete problem. We conjecture that the minimum number of inequalities that give rise to such coNP-hardness results is two. Towards this, we show that in the same LAV setting, computing the certain answers of *unions* of conjunctive queries with at most two inequalities per disjunct is coNP-complete. This result is tight, because we show that, even for the more general data exchange setting, there is a polynomial-time algorithm for computing the certain answers of unions of conjunctive queries with at most one inequality per disjunct (thus, the claim in [2] is false, unless P = NP). Moreover, the certain answers of unions of conjunctive queries with at most one inequality per disjunct can be computed in time polynomial in the size of a given universal solution. We point out, however, that this computation cannot be carried out by simply evaluating such queries on the canonical universal solution. Thus, the question arises as to whether the certain answers of unions of conjunctive queries with at most one inequality per disjunct can be computed by evaluating some other (perhaps more complex) first-order query on the canonical universal solution. Our final theorem provides a strong negative answer to this question. It shows that there is a simple conjunctive query q with one inequality for which there is no first-order query q^* such that the certain answers of q can be computed by evaluating q^* on the canonical universal solution. The proof of this theorem makes use of a novel combination of Ehrenfeucht-Fraïssé games and the chase.

The proofs of the results of this paper can be found in the full version [9].

2   The Data Exchange Problem

A *schema* is a finite collection $\mathbf{R} = \{R_1, \ldots, R_k\}$ of relation symbols. An *instance I over the schema* $\mathbf{R}$ is a function that associates to each relation symbol R_i a relation $I(R_i)$. set of relations $I(R_1), \ldots, I(R_k)$ interpreting the corresponding relation symbols in $\mathbf{S}$. In the sequel, we will on occasion abuse the notation and use R_i to denote both the relation symbol and the relation that interprets it. Given a tuple t occurring in a relation R, we denote by $R(t)$ the association between t and R and call it a *fact*. If $\mathbf{R}$ is a schema, then a *dependency over* $\mathbf{R}$ is a sentence in some logical formalism over $\mathbf{R}$.

Let $\mathbf{S} = \{S_1, \ldots, S_n\}$ and $\mathbf{T} = \{T_1, \ldots, T_m\}$ be two disjoint schemas. We refer to $\mathbf{S}$ as the *source* schema and to the S_i's as the *source* relation symbols. We refer to $\mathbf{T}$ as the *target* schema and to the T_j's as the *target* relation symbols. Similarly, instances over $\mathbf{S}$ will be called *source* instances, while instances over $\mathbf{T}$ will be called *target* instances. If I is a source instance and J is a target instance, then we write $\langle I, J \rangle$ for the instance K over the schema $\mathbf{S} \cup \mathbf{T}$ such that $K(S_i) = I(S_i)$ and $K(T_j) = J(T_j)$, for $i \leq n$ and $j \leq m$.

A *source-to-target dependency* is a dependency of the form $\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \to \chi_{\mathbf{T}}(\mathbf{x}))$, where $\phi_{\mathbf{S}}(\mathbf{x})$ is a formula, with free variables $\mathbf{x}$, of some logical formalism over $\mathbf{S}$ and $\chi_{\mathbf{T}}(\mathbf{x})$ is a formula, with free variables $\mathbf{x}$, over some logical formalism over $\mathbf{T}$ (these two logical formalisms may be different). We use the notation $\mathbf{x}$ for a vector of variables $x_1, \ldots, x_k$. A *target* dependency is a dependency over the target schema $\mathbf{T}$ (the formalism used to express a target dependency may be different from those used for the source-to-target dependencies). The source schema may also have dependencies that we assume are satisfied by every source instance. While the source dependencies may play an important role in deriving source-to-target dependencies [19], they do not play any direct role in data exchange, because we take the source instance to be given.

Definition 1. A *data exchange setting* $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ consists of a source schema $\mathbf{S}$, a target schema $\mathbf{T}$, a set Σ_{st} of source-to-target dependencies, and a set Σ_t of target dependencies. The *data exchange problem* associated with this setting is the following: given a finite source instance I, find a finite target instance J such that $\langle I, J \rangle$ satisfies Σ_{st} and J satisfies Σ_t. Such a J is called a *solution for I* or, simply a *solution* if the source instance I is understood from the context. The set of all solutions for I is denoted by $\mathrm{Sol}(I)$.

For most practical purposes, and for most of the results of this paper, each source-to-target dependency in Σ_{st} is a *tuple generating dependency (tgd)* [4] of the form

$$\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \to \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})),$$

where $\phi_{\mathbf{S}}(\mathbf{x})$ is a conjunction of atomic formulas over $\mathbf{S}$ and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over $\mathbf{T}$. Moreover, each target dependency in Σ_t is either a *tuple-generating dependency (tgd)* (of the form shown below left) or an *equality-generating dependency (egd)* [4] (shown below right):

$$\forall \mathbf{x}(\phi_{\mathbf{T}}(\mathbf{x}) \to \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})) \qquad \forall \mathbf{x}(\phi_{\mathbf{T}}(\mathbf{x}) \to (x_1 = x_2))$$

In the above, $\phi_{\mathbf{T}}(\mathbf{x})$ and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ are conjunctions of atomic formulas over $\mathbf{T}$, and x_1, x_2 are among the variables in $\mathbf{x}$. Note that data exchange settings with tgds as source-to-target dependencies include as special cases both LAV and GAV data integration systems in which the views are sound [13] and are defined by conjunctive queries. It is natural to take the target dependencies to be tgds and egds: these two classes together comprise the (embedded) implicational dependencies [8], which seem to include essentially all of the naturally-occurring constraints on relational databases. However, it is somewhat surprising that tgds, which were originally "designed" for other purposes (as constraints), turn out to be ideally suited for describing desired data transfer. For simplicity, in the rest of the paper we will drop the universal quantifiers in front of a dependency, and implicitly assume such quantification. However, we will write down all existential quantifiers.

The next example shows that there may be more than one possible solution for a given data exchange problem. The natural question is then which solution to choose.

Example 1. Consider a data exchange problem in which the source schema has three relation symbols P, Q, R, each of them with attributes A, B, C, while the target schema has one relation symbol T also with attributes A, B, C. We assume that $\Sigma_t = \emptyset$. The source-to-target dependencies and the source instance are:

$$\Sigma_{st}: \quad P(a, b, c) \rightarrow \exists Y \exists Z \; T(a, Y, Z) \qquad\qquad I = \{ \quad P(a_0, b_0', c_0'),$$
$$Q(a, b, c) \rightarrow \exists X \exists U \; T(X, b, U) \qquad\qquad\qquad Q(a_0'', b_0, c_0''),$$
$$R(a, b, c) \rightarrow \exists V \exists W \; T(V, W, c) \qquad\qquad\qquad R(a_0''', b_0''', c_0) \; \}$$

We observe first that the dependencies in Σ_{st} do not completely specify the target instance. It should be noted that such incomplete specification arises naturally in many practical scenarios of data exchange (or data integration for that matter; see [11,13]). For our example, one possible solution is:

$$J = \{T(a_0, Y_0, Z_0), T(X_0, b_0, U_0), T(V_0, W_0, c_0)\},$$

where $X_0, Y_0, \ldots$ represent "unknown" values. We will call such values *labeled nulls* and we will introduce them formally in the next section. The second observation is that there may be more than one solution. For example, the following are solutions as well:

$$J_1 = \{T(a_0, b_0, c_0)\} \qquad\qquad J_2 = \{T(a_0, b_0, Z_1), T(V_1, W_1, c_0)\}$$

In the above, Z_1, V_1 and W_1 are labeled nulls. Note that J_1 does not use labeled nulls; instead, source values are used to witness the existentially quantified variables in the dependencies. Solution J_1 seems to be less general than J, since it "assumes" that all three tuples required by the dependencies are equal to the tuple (a_0, b_0, c_0). This assumption, however, is not part of the specification. Similarly, solution J_2 has extra information that is not a consequence of the dependencies in Σ_{st} for the given source data. We argue that neither J_1 nor J_2 should be used for data exchange. In contrast, J is the "best" solution: it contains no more and no less than what the specification requires. We formalize this intuition next.

2.1 Universal Solutions

We next give an algebraic specification that selects, among all possible solutions, a special class of solutions that we call *universal*. As we will see, a universal solution has several "good" properties that justify its choice for the semantics of data exchange. Before presenting the key definition, we introduce some terminology and notation.

We denote by $\underline{\mathsf{Const}}$ the set of all values that occur in source instances and we also call them *constants*. In addition, we assume an infinite set $\underline{\mathsf{Var}}$ of values, which we call *labeled nulls*, such that $\underline{\mathsf{Var}} \cap \underline{\mathsf{Const}} = \emptyset$. We reserve the symbols $I, I', I_1, I_2, \ldots$ for instances over the source schema I and with values in $\underline{\mathsf{Const}}$. We also reserve the symbols $J, J', J_1, J_2, \ldots$ for instances over the target schema $\mathbf{T}$ and with values in $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}$.

If $\mathbf{R} = \{R_1, \ldots, R_k)$ is a schema and K is an instance over $\mathbf{R}$ with values in $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}$, then $\underline{\mathsf{Var}}(K)$ denotes the set of labelled nulls occurring in relations in K.

Definition 2. Let K_1 and K_2 be two instances over $\mathbf{R}$ with values in $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}$.

A *homomorphism* $h : K_1 \to K_2$ is a mapping from $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}(K_1)$ to $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}(K_2)$ such that: (1) $h(c) = c$, for every $c \in \underline{\mathsf{Const}}$; (2) for every fact $R_i(t)$ of K_1, we have that $R_i(h(t))$ is a fact of K_2 (where, if $t = (a_1, \ldots, a_s)$, then $h(t) = (h(a_1), \ldots, h(a_s)))$.

K_1 is *homomorphically equivalent* to K_2 if there is a homomorphism $h : K_1 \to K_2$ and a homomorphism $h' : K_2 \to K_1$.

Definition 3 (Universal solution). Consider a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$. If I is a source instance, then a *universal solution for I* is a solution J for I such that for every solution J' for I, there exists a homomorphism $h : J \to J'$.

Example 2. The instances J_1 and J_2 in Example 1 are not universal. In particular, there is no homomorphism from J_1 to J and also there is no homomorphism from J_2 to J. This fact makes precise our earlier intuition that the instances J_1 and J_2 contain "extra" information. In contrast, there exist homomorphisms from J to both J_1 and J_2. Actually, it can be easily shown that J has homomorphisms to all other solutions. Thus, J is universal.

From an algebraic standpoint, being a universal solution is a property akin to being an *initial structure* [17] for the set of all solutions (although an initial structure for a set $\mathcal{K}$ of structures is required to have *unique* homomorphisms to all other structures in $\mathcal{K}$). Initial structures are ubiquitous in several areas of computer science, including semantics of programming languages and term rewriting, and are known to have good properties (see [17]). The next result asserts that universal solutions have good properties as well.

Proposition 1. *Let $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting.*

1. *If I is a source instance and J, J' are universal solutions for I, then J and J' are homomorphically equivalent.*

2. *Assume that Σ_{st} is a set of tgds. Let I, I' be two source instances, J a universal solution for I, and J' a universal solution for I'. Then $\mathrm{Sol}(I) \subseteq \mathrm{Sol}(I')$ if and only if there is a homomorphism $h : J' \to J$. Consequently, $\mathrm{Sol}(I) = \mathrm{Sol}(I')$ if and only if J and J' are homomorphically equivalent.*

The first part of Proposition 1 asserts that universal solutions are unique up to homomorphic equivalence. The second part implies that if J is a universal solution for two source instances I and I', then $\mathrm{Sol}(I) = \mathrm{Sol}(I')$. Thus, in a certain sense, each universal solution precisely embodies the space of solutions.

3   Computing Universal Solutions

Checking the conditions in Definition 3 requires implicitly the ability to check the (infinite) space of all solutions. Thus, it is not clear, at first hand, to what extent the notion of universal solution is a computable one. This section addresses the question of how

to check the existence of a universal solution and how to compute one (if one exists). In particular, we show that the classical chase can be used for data exchange and that every finite chase, if it does not fail, constructs a universal solution. If the chase fails, then no solution exists. However, in general, for arbitrary dependencies, there may not exist a finite chase. Hence, in Section 3.2 we introduce the class of weakly acyclic sets of tgds, for which the chase is guaranteed to terminate in polynomial time. For such dependencies, we show that: (1) the existence of a universal solution can be checked in polynomial time, (2) a universal solution exists if and only if a solution exists, and (3) a universal solution (if solutions exist) can be produced in polynomial time.

3.1   Chase: Canonical Generation of Universal Solutions

Intuitively, we apply the following procedure to produce a universal solution: start with an instance $\langle I, \emptyset \rangle$ that consists of I, for the source schema, and of the empty instance, for the target schema; then chase $\langle I, \emptyset \rangle$ by applying the dependencies in Σ_{st} and Σ_t for as long as they are applicable[1]. This process may fail (as we shall see shortly, if an attempt to identify two constants is made) or it may never terminate. But if it does terminate and if it does not fail, then the resulting instance is guaranteed to satisfy the dependencies and, moreover, to be universal (Theorem 1).

We next define chase steps. Similar to homomorphisms between instances, a homomorphism from a conjunctive formula $\phi(\mathbf{x})$ to an instance J is a mapping from the variables $\mathbf{x}$ to $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}(J)$ such that for every atom $R(x_1, \ldots, x_n)$ of ϕ, the fact $R(h(x_1), \ldots, h(x_n))$ is in J. The chase that we use is a slight variation of the classical notion of chase with tgds and egds of [4] in the sense that we chase instances rather than symbolic tableaux. Consequently, the chase may fail.

Definition 4 (Chase step). Let K be an instance.

(tgd) Let d be a tgd $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$. Let h be a homomorphism from $\phi(\mathbf{x})$ to K such that there is no extension of h to a homomorphism h' from $\phi(\mathbf{x}) \wedge \psi(\mathbf{x}, \mathbf{y})$ to K. We say that d *can be applied to K with homomorphism h.*

Let K' be the union of K with the set of facts obtained by: (a) extending h to h' such that each variable in $\mathbf{y}$ is assigned a fresh labeled null, followed by (b) taking the image of the atoms of ψ under h'. We say that *the result of applying d to K with h is K'*, and write $K \xrightarrow{d,h} K'$.

(egd) Let d be an egd $\phi(\mathbf{x}) \rightarrow (x_1 = x_2)$. Let h be a homomorphism from $\phi(\mathbf{x})$ to K such that $h(x_1) \neq h(x_2)$. We say that *d can be applied to K with homomorphism h.* We distinguish two cases.

- If both $h(x_1)$ and $h(x_2)$ are in $\underline{\mathsf{Const}}$ then we say that *the result of applying d to K with h is "failure"*, and write $K \xrightarrow{d,h} \bot$.
- Otherwise, let K' be K where we identify $h(x_1)$ and $h(x_2)$ as follows: if one is a constant, then the labeled null is replaced everywhere by the constant; if both are labeled nulls, then one is replaced everywhere by the other. We say that *the result of applying d to K with h is K'*, and write $K \xrightarrow{d,h} K'$.

---

[1] It is possible to apply first Σ_{st} as long as applicable and then apply Σ_t as long as applicable.

In the definition, $K \xrightarrow{d,h} K'$ (including the case where K' is $\perp$) defines one single chase step. We next define chase sequences and finite chases.

Definition 5 (Chase). Let Σ be a set of tgds and egds, and let K be an instance.

- A *chase sequence of K with Σ* is a sequence (finite or infinite) of chase steps $K_i \xrightarrow{d_i, h_i} K_{i+1}$, with $i = 0, 1, \ldots$, with $K = K_0$ and d_i a dependency in Σ.

- A *finite chase of K with Σ* is a finite chase sequence $K_i \xrightarrow{d,h} K_{i+1}, 0 \leq i < m$, with the requirement that either (a) $K_m = \perp$ or (b) there is no dependency d_i of Σ and there is no homomorphism h_i such that d_i can be applied to K_m with h_i. We say that K_m is the result of the finite chase. We refer to case (a) as the case of a *failing finite chase* and we refer to case (b) as the case of a *successful finite chase*.

In general, there may not exist a finite chase of an instance (cyclic sets of dependencies could cause infinite application of chase steps). Infinite chases can be defined as well, but for this paper we do not need to do so. Also, different chase sequences may yield different results. However, each result, if not $\perp$, satisfies Σ.

For data exchange, we note first that, due to the nature of our dependencies, any chase sequence that starts with $\langle I, \emptyset \rangle$ does not change or add tuples in I. Then, if a finite chase exists, its result $\langle I, J \rangle$ is such that J is a solution. Furthermore, J is universal, a fact that does not seem to have been explicitly noted in the literature on the chase. The next theorem states this, and also states that the chase can be used to check the existence of a solution.

Theorem 1. *Assume a data exchange setting where Σ_{st} consists of tgds and Σ_t consists of tgds and egds.*

1. *Let $\langle I, J \rangle$ be the result of some successful finite chase of $\langle I, \emptyset \rangle$ with $\Sigma_{st} \cup \Sigma_t$. Then J is a universal solution.*

2. *If there exists some failing finite chase of $\langle I, \emptyset \rangle$ with $\Sigma_{st} \cup \Sigma_t$, then there is no solution.*

For case 1 of Theorem 1 we refer to such J as *a canonical universal solution*. In further examples and proofs, when such J is unique, we will also use the term *the canonical universal solution*. We note that a canonical universal solution is similar, in its construction, to the representative instance defined in the work on the universal relation (see [16]).

The following is an example of cyclic set of inclusion dependencies for which there is no finite chase; thus, we cannot produce a universal solution by the chase. Still, a finite solution does exist. This illustrates the need for introducing restrictions in the class of dependencies that are allowed in the target.

Example 3. Consider the data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ as follows. The source schema $\mathbf{S}$ has one relation $\texttt{DeptEmp}(\texttt{dpt\_id}, \texttt{mgr\_name}, \texttt{eid})$ listing departments with their managers and their employees. The target schema $\mathbf{T}$ has a relation $\texttt{Dept}(\texttt{dpt\_id}, \texttt{mgr\_id}, \texttt{mgr\_name})$ for departments and their managers, and a separate relation for employees $\texttt{Emp}(\texttt{eid}, \texttt{dpt\_id})$. The source-to-target and target dependencies are:

$$\Sigma_{st} = \{ \texttt{DeptEmp}(d, n, e) \rightarrow \exists M.\texttt{Dept}(d, M, n) \wedge \texttt{Emp}(e, d) \}$$
$$\Sigma_t = \{ \texttt{Dept}(d, m, n) \rightarrow \exists D.\texttt{Emp}(m, D), \quad \texttt{Emp}(e, d) \rightarrow \exists M \exists N.\texttt{Dept}(d, M, N) \}$$

Assume now that the source instance I has one tuple in $\mathtt{DeptEmp}$, for department *CS* with manager *Mary* and employee *E003*. Chasing $\langle I, \emptyset \rangle$ with Σ_{st} yields the target instance:

$$J_1 = \{\mathtt{Dept}(CS, M, Mary), \mathtt{Emp}(E003, CS)\}$$

where M is a labeled null that instantiates the existentially quantified variable of the tgd, and encodes the unknown manager id of *Mary*. However, J_1 does not satisfy Σ_t; therefore, the chase does not stop at J_1. The first tgd in Σ_t requires M to appear in $\mathtt{Emp}$ as an employee id. Thus, the chase will add $\mathtt{Emp}(M, D)$ where D is a labeled null representing the unknown department in which Mary is employed. Then the second tgd becomes applicable, and so on. It is easy to see that there is no finite chase. Satisfying all the dependencies would require building an infinite instance:

$$J = \{\mathtt{Dept}(CS, M, Mary), \mathtt{Emp}(E003, CS), \mathtt{Emp}(M, D), \mathtt{Dept}(D, M', N'), \dots\}$$

On the other hand, finite solutions exist. Two such examples are:

$$J' = \{\mathtt{Dept}(CS, E003, Mary), \mathtt{Emp}(E003, CS)\}$$
$$J'' = \{\mathtt{Dept}(CS, M, Mary), \mathtt{Emp}(E003, CS), \mathtt{Emp}(M, CS)\}$$

However, neither J' nor J'' are universal: there is no homomorphism from J' to J'' and there is no homomorphism from J'' to J'. We argue that neither should be used for data exchange. In particular, J' makes the assumption that the manager id of *Mary* is equal to *E003*, while J'' makes the assumption that the department in which *Mary* is employed is the same as the department (*CS*) that *Mary* manages. Neither assumption is a consequence of the given dependencies and source instance. It can be shown that no *finite* universal solution exists for this example.

We next consider sets of dependencies for which every chase sequence is guaranteed to reach its end after at most polynomially many steps (in the size of the input instance). For such sets of dependencies it follows that checking the existence of a solution, as well as generating a universal solution, can be carried out in polynomial time.

3.2 Polynomial-Length Chase

We first discuss sets of *full tgds* (tgds with no existentially quantified variables). It has been proven in [4] that every chase sequence with a set Σ of full tgds has at most finite length. Moreover every chase has the same result. It is simple to show that the length of the chase is bounded by a polynomial in the size of the input instance (the dependencies and the schema are fixed). Also, any set of egds can be added to Σ without affecting the uniqueness of the result or the polynomial bound.

Although full tgds enjoy nice properties, they are not very useful in practice. Most dependencies occurring in real schemas are non-full, for example, foreign key constraints or, more generally, inclusion dependencies [6]. It is well known that chasing with inclusion dependencies may not terminate in general. *Acyclic sets of inclusion dependencies* [7] are a special case for which every chase sequence has a length that is polynomial in the size of the input instance. Such dependencies can be described by defining a directed graph in which the nodes are the relation symbols, and such that
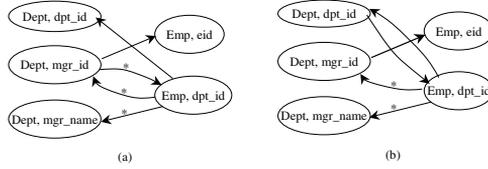
Fig. 1. Dependency graphs for: (a) a set of tgds that is not weakly acyclic, (b) a weakly acyclic set of tgds.

there exists an edge from R to S whenever there is an inclusion dependency from R to S. A set of inclusion dependencies is acyclic if there is no cycle in this graph. We define next a *weakly acyclic sets of tgds*, a notion that strictly includes both sets of full tgds and acyclic sets of inclusion dependencies. This notion is inspired by the definition of weakly recursive ILOG [12], even though the latter is not directly related to dependencies. Informally, a set of tgds is weakly acyclic if it does not allow for cascading of labeled null creation during the chase.

Definition 6 (Weakly acyclic set of tgds). Let Σ be a set of tgds over a fixed schema. Construct a directed graph, called the *dependency graph*, as follows: (1) there is a node for every pair (R, A) with R a relation symbol of the schema and A an attribute of R; call such pair (R, A) a *position*; (2) add edges as follows: for every tgd $\phi(\mathbf{x}) \to \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$ in Σ and for every x in $\mathbf{x}$ that **occurs** in ψ:

- For every occurence of x in ϕ in position (R, A_i):
  - (a) for every occurence of x in ψ in position (S, B_j), add an edge $(R, A_i) \to (S, B_j)$ (if it does not already exist).
  - (b) in addition, for every existentially quantified variable y and for every occurrence of y in ψ in position (T, C_k), add a *special edge* $(R, A_i) \overset{*}{\to} (T, C_k)$ (if it does not already exist).

Note that there may be two edges in the same direction between two nodes, if exactly one of the two edges is special. Then Σ is *weakly acyclic* if the dependency graph has no cycle going through a special edge. ∎

Intuitively, part (a) keeps track of the fact that a value may propagate from position (R, A_i) to position (S, B_j) during the chase. Part (b), moreover, keeps track of the fact that propagation of a value into (S, B_j) also creates a labeled null in any position that has an existentially quantified variable. If a cycle goes through a special edge, then a labeled null appearing in a certain position during the chase may determine the creation of another labeled null, in the same position, at a later chase step. This process may thus continue forever. Note that the definition allows for cycles as long as they do not include special edges. In particular, a set of full tgds is a special case of a weakly acyclic set of tgds (there are no existentially quantified variables, and hence no special edges).

Example 4. Recall Example 3. The dependency graph of Σ_t is shown in Figure 1(a). The graph contains a cycle with two special edges. Hence Σ_t is not weakly acyclic and therefore a finite chase may not exist (as seen in Example 3). On the other hand, let us assume that we know that each manager of a department is employed by the *same* department. Then, we replace the set Σ_t by the set Σ_t', where

$$\Sigma'_t = \{ \text{Dept}(d, m, n) \rightarrow \text{Emp}(m, d), \ \text{Emp}(e, d) \rightarrow \exists M \exists N. \text{Dept}(d, M, N) \}$$

The dependency graph of Σ'_t, shown in Figure 1(b), has no cycles going through a special edge. Thus, Σ'_t is weakly acyclic. As Theorem 2 will show, it is guaranteed that every chase sequence is finite. For Example 3, one can see that the chase of J_1 with Σ'_t stops with result J''. Thus J'' is universal. Note that for J'' to be universal it was essential that we explicitly encoded in the dependencies the fact that managers are employed by the department they manage. Finally, we remark that Σ'_t is an example of a set of inclusion dependencies that, although weakly acyclic, is cyclic according to the definition of [7].

We now state the main results regarding weakly acyclic sets of tgds.

Theorem 2. *Let Σ be the union of a weakly acyclic set of tgds with a set of egds, and let K be an instance. Then there exists a polynomial in the size of K that bounds the length of every chase sequence of K with Σ.*

Corollary 1. *Assume a data exchange setting where Σ_{st} is a set of tgds, and Σ_t is the union of a weakly acyclic set of tgds with a set of egds. The existence of a solution can be checked in polynomial time. If a solution exists, then a universal solution can be produced in polynomial time.*

4   Query Answering

As stated earlier, we adopt the notion of certain answers for the semantics of query answering. We first give the formal definition of this notion and then address the problem of whether and to what extent the certain answers of a query over the target schema can be computed by evaluating some query (same or different) on a universal solution.

Definition 7. Let $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting.

- Let q be a k-ary query over the target schema $\mathbf{T}$ and I a source instance. The *certain answers of q with respect to I*, denoted by $certain(q, I)$, is the set of all k-tuples t of constants from I such that for every solution J of this instance of the data exchange problem, we have that $t \in q(J)$.

- Let q be a Boolean query over the target schema $\mathbf{T}$ and I a source instance. We write $certain(q, I) = \underline{\text{true}}$ to denote that for every solution J of this instance of the data exchange problem, we have that $q(J) = \underline{\text{true}}$. We also write $certain(q, I) = \underline{\text{false}}$ to denote that there is a solution J such that $q(J) = \underline{\text{false}}$.

On the face of it, the definition of certain answers entails a computation over the entire set of solutions of a given instance of the data exchange problem. Since this set may very well be infinite, it is desirable to identify situations in which the certain answers of a query q can be computed by evaluating q on a particular fixed solution and then keeping only the tuples that consist entirely of constants. More formally, if q is a k-ary query and J is a target instance, then $q(J)_{\downarrow}$ is the set of all k-tuples t of constants such that $t \in q(J)$. We extend the notation to Boolean queries by agreeing that if q is a Boolean query, then $q(J)_{\downarrow} = q(J)$ ($= \underline{\text{true}}$ or $\underline{\text{false}}$).

The next proposition characterizes universal solutions with respect to query answering, when the queries under consideration are unions of conjunctive queries. First, it

shows that $certain(q, I) = q(J)_\downarrow$ whenever J is a universal solution and q is a union of conjunctive queries. Concrete instances of this result in the LAV setting have been established in [2]. Another instance of this result has also been noted for the GAV setting with key/foreign key constraints in [5]. Thus, evaluation of conjunctive queries on an arbitrarily chosen universal solution gives precisely the set of certain answers. Moreover, the second statement of the proposition shows that the universal solutions are the only solutions that have this property. This is further justification for using universal solutions for data exchange.

Proposition 2. *Consider a data exchange setting with* **S** *as the source schema,* **T** *as the target schema, and such that the dependencies in the sets* Σ_{st} *and* Σ_t *are arbitrary.*

1. *Let q be a union of conjunctive queries over the target schema* **T**. *If I is a source instance and J is a universal solution, then* $certain(q, I) = q(J)_\downarrow$.

2. *Let I be a source instance and J be a solution such that for every conjunctive query q over* **T**, *we have that* $certain(q, I) = q(J)_\downarrow$. *Then J is a universal solution.*

The following result follows from Corollary 1 and Part 1 of Proposition 2.

Corollary 2. *Assume a data exchange setting where* Σ_{st} *is a set of tgds, and* Σ_t *is the union of a weakly acyclic set of tgds with a set of egds. Let q be a union of conjunctive queries. For every source instance I, the set* $certain(q, I)$ *can be computed in polynomial time in the size of I.*

The state of affairs changes dramatically when conjunctive queries with inequalities are considered. The next proposition shows that there is a simple Boolean conjunctive query q with inequalities such that no universal solution can be used to obtain the certain answers of q by evaluating q on that universal solution. This proposition also shows that in this particular case, there is another conjunctive query q^* with inequalities such that the certain answers of q can be obtained by evaluating q^* on the canonical universal solution.

Proposition 3. *Let S be a binary source relation symbol, T a binary target relation symbol,* $S(x, y) \to \exists z(T(x, z) \land T(z, y))$ *a source-to-target dependency, and q the following Boolean conjunctive query with one inequality:* $\exists x \exists y(T(x, y) \land (x \neq y))$.

1. *There is a source instance I such that* $certain(q, I) = \underline{\text{false}}$, *but* $q(J) = \underline{\text{true}}$ *for every universal solution J.*

2. *Let* q^* *be the query* $\exists x \exists y \exists z(T(x, z) \land T(z, y) \land (x \neq y))$. *If I is a source instance and J is the canonical universal solution, then* $certain(q, I) = q^*(J)$.

In view of Proposition 3, we address next the question of whether, given a conjunctive query with inequalities, it is always possible to find a query (not necessarily the same) that computes the certain answers when evaluated on the canonical universal solution.

5   Query Answering: Complexity and Inexpressibility

It is known that in LAV data integration systems, computing the certain answers of conjunctive queries with inequalities is a coNP-hard problem [2]. It follows that in the data exchange setting, it is not possible to compute the certain answers of such queries

by evaluating them on the canonical universal solution or on any universal solution that is generated in polynomial time (unless $P = NP$). We take in Section 5.1 a closer look at conjunctive queries with inequalities. First we show (Theorem 3) that, in the data exchange setting, the problem of computing the certain answers for unions of conjunctive queries with inequalities is in coNP. Surprisingly, we show (Theorem 6) that there is a polynomial-time algorithm that computes the certain answers of unions of conjunctive queries with at most one inequality per disjunct. This is an optimal result because we also show (Theorem 5) that it is coNP-hard to compute the certain answers of unions of conjunctive queries with at most two inequalities per disjunct.

In the case of unions of conjunctive queries with at most one inequality per disjunct, the certain answers can be computed in polynomial time from an arbitrary universal solution. However, Section 5.2 shows (with no unproven complexity-theoretic assumptions such as $P \neq NP$) that there is a conjunctive query q with one inequality whose certain answers cannot be computed by rewriting q to a first-order query q^* and then evaluating q^* on the canonical universal solution. We begin by formally introducing the decision problem associated with the computation of the set of certain answers.

Definition 8. Let $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting.

1. Let q be a k-ary query over the target schema $\mathbf{T}$. *Computing the certain answers of* q is the following decision problem: given a source instance I over $\mathbf{S}$ and a k-tuple t of constants from I, is it the case that $t \in certain(q, I)$?

2. Let q be a Boolean query over the target schema $\mathbf{T}$. *Computing the certain answers of* q is the following decision problem: given a source instance I over $\mathbf{S}$, is it the case that $certain(q, I) = \underline{\text{true}}$?

3. Let $\mathcal{C}$ be a complexity class and $\mathcal{Q}$ a class of queries over the target schema $\mathbf{T}$. We say that *computing the certain answers of queries in $\mathcal{Q}$ is in $\mathcal{C}$* if for every query $q \in \mathcal{Q}$, computing the certain answers of q is in $\mathcal{C}$. We say that *computing the certain answers of queries in $\mathcal{Q}$ is $\mathcal{C}$-complete* if it is in $\mathcal{C}$ and there is at least one query $q \in \mathcal{Q}$ such that computing the certain answers of q is a $\mathcal{C}$-complete problem.

Thus, computing the certain answers of a k-ary query q is a decision problem. One can also consider a related function problem: given a source instance I, find the set $certain(q, I)$. The latter problem has a polynomial-time reduction to the former, since there are polynomially many k-tuples from I and so we can compute the set $certain(q, I)$ by going over each such k-tuple t and deciding whether or not $t \in certain(q, I)$.

5.1   Computational Complexity

Since the complexity-theoretic lower bounds and inexpressibility results presented in the sequel hold for LAV data integration systems with sound views defined by conjunctive queries, we review the definition of this type of data integration system first. A LAV data integration system with sound views defined by conjunctive queries is a special case of a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ in which $\Sigma_t = \emptyset$ and each source-to-target dependency in Σ_{st} is a tgd of the form $S_i(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$, where S_i is some relation symbol of the source schema $\mathbf{S}$ and $\psi_{\mathbf{T}}$ is an arbitrary conjunction of atomic formulas over the target schema $\mathbf{T}$. In what follows we will refer to such a setting as a LAV *setting*.

Abiteboul and Duschka [2] showed that in the LAV setting computing the certain answers of unions of conjunctive queries with inequalities is in coNP. We extend this by showing that the same upper bound holds in the data exchange setting, provided Σ_{st} is a set of tgds and Σ_t is a union of a set of egds with a weakly acyclic set of tgds.

Theorem 3. *Consider a data exchange setting in which Σ_{st} is a set of tgds and Σ_t is a union of a set of egds with a weakly acyclic set of tgds. Let q be a union of conjunctive queries with inequalities. Then computing the certain answers of q is in* coNP.

We first note that, in the particular case when all the tgds in Σ_t are full, the theorem can be proved by using the "small model property" (essentially this argument was used in [2] for the LAV setting). However, for the more general case when the tgds in Σ_t may have existentially quantified variables, the proof is more involved. It is based on an extension of the chase, called the *disjunctive chase*; see the full version [9] for details.

Theorem 3 yields an upper bound in a fairly general data exchange setting for the complexity of computing the certain answers of unions of conjunctive queries with inequalities. It turns out, as we discuss next, that this upper bound is tight, even in fairly restricted data exchange settings. Specifically, computing certain answers for such queries is coNP-complete. Therefore no polynomial algorithm exists for computing the certain answers when the input is a universal solution, unless $P = NP$.

Abiteboul and Duschka [2] showed that in the LAV setting, computing certain answers of conjunctive queries with inequalities is coNP-complete. They also sketched a proof which, if correct, would establish that this problem is coNP-complete even for conjunctive queries with a single inequality. Unfortunately, the reduction is erroneous. A correct reduction cannot be produced without increasing the number of inequalities, since here we show that in the LAV setting, there is a polynomial-time algorithm for computing the certain answers of unions of conjunctive queries with at most one inequality per disjunct. Still, the result of Abiteboul and Duschka [2] is correct; in fact, the unpublished full version [3] of that paper contains a proof to the effect that in the LAV setting, computing certain answers of Boolean conjunctive queries with six inequalities is coNP-complete. A different proof of the same result can be extracted by slightly modifying the proof of Theorem 3.2 in van der Meyden [22]. Thus, the next result provides a matching lower bound for the complexity of computing the certain answers of conjunctive queries with inequalities.

Theorem 4. [2,22] *In the* LAV *setting, computing the certain answers of Boolean conjunctive queries with six or more inequalities is* coNP-*complete.*

It is an interesting technical problem to determine the minimum number of inequalities needed to give rise to a coNP-complete problem in this setting.

Conjecture 1. In the LAV setting, computing the certain answers of Boolean conjunctive queries with two inequalities is coNP-complete.

We have not been able to settle this conjecture, but have succeeded in pinpointing the complexity of computing the certain answers of *unions* of Boolean conjunctive queries with at most two inequalities per disjunct.

Theorem 5. *In the* LAV *setting, computing the certain answers of unions of Boolean conjunctive queries with at most two inequalities per disjunct is* coNP-*complete. In fact,*

this problem 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.*

For unions of conjunctive queries with inequalities, Theorem 5 delineates the boundary of intractability, because the next theorem asserts that computing certain answers of unions of conjunctive queries with at most one inequality per disjunct can be solved in polynomial time by an algorithm that runs on universal solutions.

Theorem 6. *Assume a data exchange setting in which Σ_{st} is a set of tgds, and Σ_t is the union of a weakly acyclic set of tgds with a set of egds. Let q be a union of conjunctive queries with at most one inequality per disjunct. Let I be a source instance and let J be an arbitrary universal solution for I. Then there is a polynomial-time algorithm with input J that computes* $certain(q, I)$.

Corollary 3. *Assume a data exchange setting in which Σ_{st} is a set of tgds, and Σ_t is the union of a weakly acyclic set of tgds with a set of egds. Let q be a union of conjunctive queries with at most one inequality per disjunct. Then there is a polynomial-time algorithm for computing the certain answers of q.*

5.2   First-Order Inexpressibility

The following theorem shows that, in general, even for a conjunctive query q with just one inequality, there is no first-order query q^* that computes the certain answers when evaluated on the canonical universal solution. This is in strong contrast with the polynomial-time algorithm that we have seen earlier (Theorem 6). It is also in contrast with the second part of Proposition 3, where we have seen a particular example for which such a q^* exists. The proof of the theorem combines Ehrenfeucht-Fraïssé games with the chase procedure.

Theorem 7. *There is a LAV setting with source I and there is a Boolean conjunctive query q with one inequality, for which there is no first-order query q^* over the canonical universal solution J such that* $certain(q, I) = q^*(J)$.

In the full version we also show that the result holds even if we allow the first-order formula q^* to contain the predicate <u>const</u> that distinguishes between constants and nulls.

The next result, of particular interest to query answering in the data integration context, is a corollary to the proof of Theorem 7. It shows that for conjunctive queries with just one inequality we cannot in general find any first-order query over the *source* schema that, when evaluated on the *source* instance, computes the certain answers.

Corollary 4. *There is a LAV setting with source I and there is a Boolean conjunctive query q with one inequality, for which there is no first-order query q^* over the source schema such that* $certain(q, I) = q^*(I)$.

6   Concluding Remarks

We plan to further investigate how universal solutions can be used for query answering in data exchange. We wish to characterize when a query q can be rewritten to a first-order query q^* such that the certain answers of q can be computed by evaluating q^* on a

universal solution. We also wish to understand how well the certain answers of a query can be approximated by evaluating the same query on a universal solution and how this differs from universal solution to universal solution. Finally, an important direction is extending the notion of universal solution to cover data exchange between nested (e.g. XML) schemas.

References

1. S. Abiteboul, S. Cluet, and T. Milo. Correspondence and Translation for Heterogeneous Data. In *ICDT*, pages 351–363, 1997.
2. S. Abiteboul and O. M. Duschka. Complexity of Answering Queries Using Materialized Views. In *PODS*, pages 254–263, 1998.
3. S. Abiteboul and O. M. Duschka. Complexity of Answering Queries Using Materialized Views. Unpublished full version of [2], 2000.
4. C. Beeri and M. Y. Vardi. A Proof Procedure for Data Dependencies. *JACM*, 31(4):718–741, 1984.
5. A. Calì, D. Calvanese, G. D. Giacomo, and M. Lenzerini. Data Integration under Integrity Constraints. In *CAiSE*, pages 262–279, 2002.
6. M. A. Casanova, R. Fagin, and C. H. Papadimitriou. Inclusion Dependencies and their Interaction with Functional Dependencies. *JCSS*, 28(1):29–59, 1984.
7. S. S. Cosmadakis and P. C. Kanellakis. Functional and Inclusion Dependencies: A Graph Theoretic Approach. In *Advances in Computing Research*, volume 3, pages 163–184. 1986.
8. R. Fagin. Horn Clauses and Database Dependencies. *JACM*, 29(4):952–985, Oct. 1982.
9. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. IBM Research Report, Nov. 2002.
10. M. Friedman, A. Y. Levy, and T. D. Millstein. Navigational Plans For Data Integration. In *AAAI*, pages 67–73, 1999.
11. A. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal*, pages 270–294, 2001.
12. R. Hull and M. Yoshikawa. ILOG: Declarative Creation and Manipulation of Object Identifiers. In *VLDB*, pages 455–468, 1990.
13. M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, 2002.
14. A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering Queries Using Views. In *PODS*, pages 95–104, May 1995.
15. D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing Implications of Data Dependencies. *ACM TODS*, 4(4):455–469, Dec. 1979.
16. D. Maier, J. D. Ullman, and M. Y. Vardi. On the Foundations of the Universal Relation Model. *ACM TODS*, 9(2):283–308, June 1984.
17. J. A. Makowsky. Why Horn Formulas Matter in Computer Science: Initial Structures and Generic Examples. *JCSS*, 34(2/3):266–292, April/June 1987.
18. R. J. Miller, L. M. Haas, and M. Hernández. Schema Mapping as Query Discovery. In *VLDB*, pages 77–88, 2000.
19. L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, 2002.
20. N. C. Shu, B. C. Housel, and V. Y. Lum. CONVERT: A High Level Translation Definition Language for Data Conversion. *Communications of the ACM*, 18(10):557–567, 1975.
21. N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. EXPRESS: A Data EXtraction, Processing, amd REStructuring System. *TODS*, 2(2):134–174, 1977.
22. R. van der Meyden. The Complexity of Querying Indefinite Data about Linearly Ordered Domains. *JCSS*, 54:113–135, 1997.
23. R. van der Meyden. Logical Approaches to Incomplete Information: A Survey. In *Logics for Databases and Information Systems*, pages 307–356. Kluwer, 1998.