Data Exchange: Query Answering for Incomplete Data Sources

Foto Afrati
National Technical University of Athens
afrati@softlab.ntua.gr

Chen Li
University of California, Irvine
chenli@ics.uci.edu

Vassia Pavlaki
National Technical University of Athens
vpavlaki@softlab.ntua.gr

ABSTRACT

Data exchange is the problem of transforming data structured under a schema, called the source schema, into data structured under another schema, called the target schema. Existing work on data exchange considers settings where the source instance does not contain incomplete information. In this paper we study semantics and address algorithmic issues for data exchange settings where the source instance may contain incomplete data. We investigate the query answering problem in such data exchange settings. First we give two different meaningful semantics to *certain answers*: One via the certain answers in the corresponding complete data exchange problems and the other via the set of all solutions of the corresponding complete data exchange problems. We use the chase to compute a *universal instance* which is materialized over the target schema and is used to compute the certain answers to unions of conjunctive queries. We prove that computing certain answers (under both semantics) for unions of conjunctive queries can be done in polynomial time when the schema mapping contains constraints that consist of a weakly acyclic set of tuple-generating dependencies and equality-generating dependencies.

1. INTRODUCTION

Data exchange, also known as data translation, is the problem of transforming data structured under a schema, called the source schema, into data structured under another schema, called the target schema. Schema mappings are typically used to define formally such a transformation. One of the most important goals of data exchange is to actually materialize a target instance that satisfies the relationship between the schemas and then use this instance to answer queries posed on the target schema. Data exchange has received considerable attention recently, for relational databases [13, 14, 9, 4, 16, 23, 22, 25, 2, 1] and XML databases [5]. A schema mapping tool, *Clio*, was implemented at the IBM Almaden Research Center [28, 29] which grew over the years [18, 3]. On the other hand, due to new applications (e.g., sensor and scientific data, information extraction) a considerable interest has been shown in issues concerning *incomplete data* and uncertain data. Significant work on querying incomplete data [21] and recently on querying uncertain data [8, 11, 30] has been done in the literature.

To the best of our knowledge, all existing work on answering queries in data exchange settings assumes that the source instance does not contain incomplete or uncertain data; we refer to this setting as *complete data exchange setting*. A complete data exchange setting assumes a set of constraints, Σ, that consist of source-to-target constraints Σ_{st} and target constraints Σ_t. In this paper we consider relational data exchange settings in which the source instance I may be incomplete. To characterize the worlds represented by I, incomplete databases may be accompanied by data dependencies. Therefore, the *incomplete data exchange setting* that we introduce in this paper assumes also source constraints denoted Σ_s. Figure 1 illustrates an incomplete data exchange setting $\mathbf{M}$ where I is the source instance on schema $\mathbf{S}$ and J is the target instance (to be materialized) on schema $\mathbf{T}$.

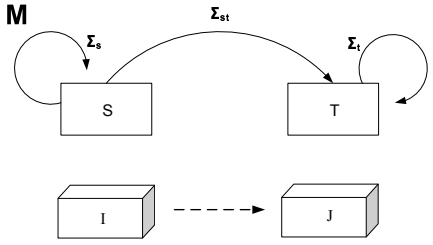

Figure 1: An incomplete data exchange setting.

In this paper, we investigate the following problem: Given a source instance and a set of constraints (that encompass schema mappings and integrity constraints), what instance to materialize on the target so that we can use it to answer unions of conjunctive queries. Work on data exchange (and data integration in general) has often adopted the notion of *certain answers* for the semantics of query answering (see e.g., [13]). The semantics of certain answers are typically based on the concept of possible worlds. For complete data exchange settings, we know how to define meaningful semantics for certain answers for unions of conjunctive

queries based on the possible target instances that satisfy the constraints (called solutions).

In a data exchange setting, the source database instance and the target database instance are on different schemas. Thus, even in the cases where the source instance does not contain incomplete data, the materialized target instance most probably does. Hence, this target instance actually is seen as representing a set of possible worlds. Thus, we define a fact to be in the certain answers of a query q posed on the target if this fact is in the answer to query q posed on every target possible world. In this paper, we adopt the notion of certain answers as the query answering semantics, i.e., we are interested in obtaining answers to the query that do not contain nulls and are guaranteed to be answers to the query on every possible world. However, we need to revisit the definition of the set of possible worlds because we have two sources of incompleteness now, a) the incomplete source instance and b) the incomplete target instance due to difference of the schemas of the source and the target. A natural way to attack this problem is to first think how to exchange data for every possible world represented by the incomplete source instance in separate, and then integrate.

The set of possible worlds represented by the incomplete source instance and the source constraints are the complete databases that are derived from it by replacing the nulls with constants and adding missing facts and for which the source constraints are satisfied. This gives rise to two natural semantics of certain answers: The first semantics is based on deriving first the certain answers for every possible world of the source instance (viewed as a data exchange problem over complete data) and then take the intersection. The second semantics is based on considering again every possible world of the source instance but, instead of computing the certain answers, we consider all solutions, compute the query on every solution and take the intersection of the answers. The difference lies in that even if some possible worlds of the source do not have a solution, the set of certain answers may be non-empty because we just ignore these worlds. I.e., given an incomplete instance I over the source schema and a set of source constrains, in the first semantics, an element in the set of possible worlds is a complete data exchange problem for one of the complete source instances represented by I. In the second semantics, an element in the set of possible worlds is a solution of one of these complete data exchange problems. Figure 2 illustrates the two distinct semantics introduced in this paper. We will revisit this figure later after we introduce formally the two semantics. In this paper, we show that the two semantics are closely related to each other, actually one is strictly more liberal than the other. Interestingly, however, the computation of each set of certain answers poses a different algorithmic challenge. We show that for both our semantics, the certain answers to unions of conjunctive queries can be computed in polynomial time under the same assumptions for the constraints as for complete data exchange settings (i.e., for sets of weakly acyclic tuple-generating dependencies). The following example shows that the two semantics are natural and that both may be of interest in applications.

EXAMPLE 1.1. *Consider an online store. The customer service department contains a database* **A** *with a single table* `Members`. *Visitors of the site of the store who wish to become members are requested to fill a Web form where they are asked to provide information in many fields of which some are mandatory and some optional. For simplicity, in this example, we assume that the provided information involves only three attributes and is stored in the table*

$$\texttt{Members}\ (\texttt{Email}, \texttt{Name}, \texttt{CCName}),$$

which contains information on members email (`Email`)*, name* (`Name`)*, and credit card holder's name* (`CCName`)*. The latter attribute is optional. That is, visitors have the option to provide only their email in the first attribute, their name in the second attribute of the relation* `Members` *and leave the third attribute blank.*

The online store has also a sales department with a database **B** *consisting of a single table, namely*

```
Customers (Email,Name,CCName).
```

For business reasons, the store needs to transfer the data from the customer service department to the sales department (i.e., from the database **A** *to the database* **B**)*. The following source-to-target tgd is used for this transfer (in this example, it is a simple copying tgd)*

$$\mathtt{d_{st}} : \texttt{Members}(\texttt{Email}, \texttt{Name}, \texttt{CCName}) \rightarrow \texttt{Customers}(\texttt{Email}, \texttt{Name}, \texttt{CCName}).$$

The sales department however maintains a database with the following integrity constraint holding (A reason for that could be, e.g., to avoid fraud in the transactions):

$$\mathtt{d_t} : \texttt{Customers}(\texttt{Email}, \texttt{Name}, \texttt{CCName}) \rightarrow \texttt{Name} = \texttt{CCName}.$$

Thus, we have egd d_t as a target dependency. Suppose $I = \{\texttt{Members}(john@gmail.com, John, null)\}$ is an instance of database **A** *where null denotes the null value.*

The target instance J should be such that $I \cup J$ satisfies the constraints d_t and d_{st}. There is an infinite number of such target instances. However, for the sake of argument, we consider here the following three candidate instances as the most natural to materialize in the target: $J_1 = \emptyset$, $J_2 = \{Customers(john@gmail.com, null, null)\}$ and $J_3 = \{Customers(john@gmail.com, John, John)\}$.

Consider the following conjunctive queries posed on database **B***:*

$$\mathtt{Q_1}(\texttt{Email}) \text{ :- } \texttt{Customers}(\texttt{Email}, \texttt{Name}, \texttt{CCName}).$$
$$\mathtt{Q_2}(\texttt{Email}, \texttt{CCName}) \text{ :- } \texttt{Customers}(\texttt{Email}, \texttt{Name}, \texttt{CCName}).$$

$\mathtt{Q_1}$ *asks only for a list of customer emails whereas* $\mathtt{Q_2}$ *asks for a list of customer emails together with the corresponding credit card holder's name. Intuitively, the* certain answers *expected in the first query is $\{(john@gmail.com)\}$, whereas the certain answers expected in the second query is $\emptyset$. The reason is that query $\mathtt{Q_1}$ only asks for the emails of customers (and john@gmail.com is an email address of a customer), whereas for query $\mathtt{Q_2}$, if we get in the answers a fact that contains john@gmail.com in the first attribute, we face the risk to get a non-valid CCName in the second attribute of the query. On candidate target instances J_1 and J_2, both queries compute the empty answer (because a tuple with nulls is definitely not in the certain answers, since there is a possible world which does not derive this answer). On candidate instance J_3, query Q_1 computes $\{(john@gmail.com)\}$ as desired and query Q_2 computes also $\{(john@gmail.com, John)\}$ which however contains false information. This shows the necessity for adopting two semantics for certain answers. The more liberal semantics (which*

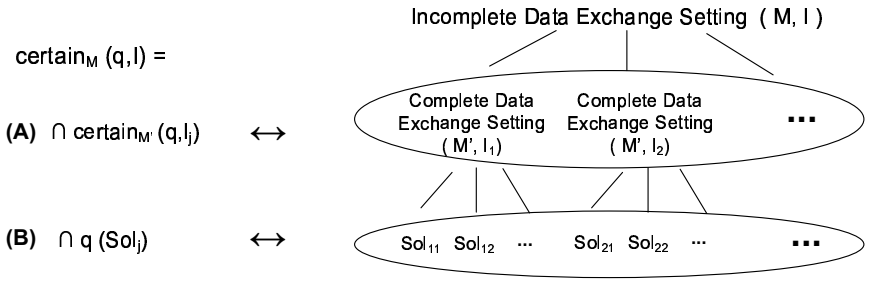

Figure 2: The two definitions of certain answers in IDE settings.

actually materializes J_3) suffices to compute the answers of $\mathtt{Q}_1$*. However, we need to adopt the stricter semantics to compute the answers of* $\mathtt{Q}_2$ *if we want to avoid getting false information. We will revisit this example after we define formally the two semantics of certain answers.*

In our setting, an *incomplete database* is a database instance which may have unknown values (nulls) or even some facts may be missing. We allow for integrity constraints to come with the incomplete database. These constraints may be functional dependencies or in general any equality-generating dependencies and tuple-generating dependencies (e.g., inclusion dependencies). In the above example, for simplicity, we did not have any source constraints. E.g., we could have the functional dependency $Email \rightarrow Name$, in which case the instance $I_1 = \{\mathtt{Members}(john@gmail.com, John, null), \mathtt{Members}(john@gmail.com, null, null)\}$ would have been the same as the instance $I = \{\mathtt{Members}(john@gmail.com, John, null)\}$.

1.1 Our Contributions

We consider sets of constraints that are equality-generating dependencies and tuple-generating dependencies. The contributions of this work are summarized as follows:

(a) We define two meaningful semantics of certain answers for data exchange settings in which the source instance may contain incomplete data.

(b) We show that the two semantics are closely related and we identify the cases in which they coincide.

(c) We use the chase to compute a *universal instance* which is materialized in the target and is used to compute the certain answers of unions of conjunctive queries.

(d) We investigate the complexity of query answering under both semantics. In particular, we show that if the source constraints consist of egds and weakly acyclic tgds, the source-to-target constrains of tgds and the target constraints of egds and weakly acyclic tgds, then we can compute certain answers in PTIME.

1.2 Related Work

In the literature several frameworks for sharing data between independent applications have been proposed. Many of them use the notion of certain answers for query answering. It is well recognized that for different query languages some semantics are not meaningful (see e.g., [2, 1]) and, therefore, the definition of certain answers problem has to be revisited. Thus, for complete data exchange settings, work has been done on defining certain answers for aggregate queries [1] and for data exchange settings which may incorporate arithmetic comparisons in their constraints and for queries with arithmetic comparisons [2]. Work in [25] focuses on giving various semantics for query answering that are meaningful in fragments of first order logic.

On the other end, for the query language of unions of conjunctive queries, and after the work in [13], broader frameworks have been considered and complexity results are obtained. In [15] the *peer data exchange setting* (PDE in short) was proposed where target-to-source constraints are also assumed and the target may have its own data. In this framework, under certain assumptions about the rules for exchanging the data, computing certain answers becomes intractable (coNP-complete). In arbitrary *peer data management systems* (PDMS in short) the relationship between peers is specified using constraints that can be in either direction (from one peer to another, and vice versa). In PDMS, query answering, is undecidable [19, 31]. In [10], computing certain answers for a special case of PDMS is shown to be in Π_2^P. Incomplete data exchange settings, introduced in this paper, are more expressive settings than complete data exchange and the problem of query answering is shown in this paper to be tractable whereas in peer data exchange settings and full peer data management systems it is proven intractable (see Figure 3 for an illustration of the discussion on the complexity of query answering for unions of conjunctive queries).

2. PRELIMINARIES

In this section we review the classes of tuple-generating dependencies and equality-generating dependencies, and the concepts of instance and ground instance. We review the results on complete data exchange settings, and in particular the semantics of certain answers in complete data exchange settings.

2.1 Data Dependencies and Instances.

We review the classes of tuple-generating dependencies (tgds) and equality-generating dependencies (egds).

Definition 2.1. (*tgd - egd*)
Let $\mathbf{D}$ *be a database schema.*
- A tuple generating dependency (*in short* tgd) *is a first-order formula of the form*

$$\forall \mathbf{x}\big(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})\big).$$

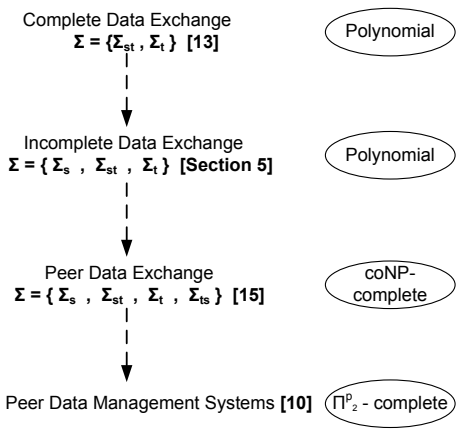

Figure 3: Hierarchy of data sharing settings w.r.t. complexity of computing certain answers.

- *An* equality generating dependency (*in short* egd) *is a first-order formula of the form*

$$\forall \mathbf{x}\big(\phi(\mathbf{x}) \rightarrow (x_1 = x_2)\big).$$

In the above formulas, $\phi(\mathbf{x})$ *is a conjunction of atomic relational formulas over* $\mathbf{D}$ *with variables in* $\mathbf{x}$*. Each variable in* $\mathbf{x}$ *occurs in at least one formula in* $\phi(\mathbf{x})$*. In addition,* $\psi(\mathbf{x}, \mathbf{y})$ *is a conjunction of atomic relational formulas with variables in* $\mathbf{x}$ *and* $\mathbf{y}$ *and each variable in* $\mathbf{y}$ *occurs in at least one formula in* $\psi(\mathbf{x}, \mathbf{y})$*.* x_1 *and* x_2 *are variables from* $\mathbf{x}$*.*

In the rest of the paper, in some cases we will drop the universal and existential quantifiers in tgds and egds implicitly assuming such quantification. We use *rhs* (*lhs*) to refer to the right hand side (left hand side) of a tgd or an egd. A tgd with no existentially quantified variables is called *full* tgd.

We assume an infinite domain of constants $\underline{\texttt{Const}}$ and an infinite set $\underline{\texttt{Var}}$ of variables, called *labeled nulls*, such that $\underline{\texttt{Const}} \cap \underline{\texttt{Var}} = \emptyset$. A fact is a relational atom over constants from $\underline{\texttt{Const}}$ and labelled nulls from $\underline{\texttt{Var}}$. We define an *instance* to be a set of facts. For an instance K, we use $\underline{\texttt{Const}}(K)$ and $\underline{\texttt{Var}}(K)$ to denote the set of constants and the set of labeled nulls in K, respectively. An instance K where $\underline{\texttt{Var}}(K)$ is empty, is called *ground instance*.

Let $d : \forall \mathbf{x}\big(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})\big)$ be a tgd and D a database instance. We say that D *satisfies* d if whenever there is a homomorphism h from $\phi(\mathbf{x})$ to D, there exists an extension h' of h, where h' is a homomorphism from the conjunction $\phi(\mathbf{x}) \wedge \psi(\mathbf{x}, \mathbf{y})$ to D.

2.2 Query Answering in Complete Data Exchange Settings.

Fagin et al. in [13] introduced a data exchange setting where the source instance is assumed to contain only constants. In the rest of the paper, we refer to this setting as *complete data exchange setting* (in short CDE).

Let $\mathbf{S} = \{S_1, \ldots, S_n\}$ and $\mathbf{T} = \{T_1, \ldots, T_m\}$ be two disjoint schemas, where each element in the sets is a relation. We refer to $\mathbf{S}$ as the source schema and to $\mathbf{T}$ as the target schema. Instances over $\mathbf{S}$ are called source instances whereas instances over $\mathbf{T}$ are called target instances. Target instances may contain values, called *labeled nulls*, that appear in the target instance but not in the source instance ("fresh" values).

A *source-to-target dependency* is a tgd of the form $\forall \mathbf{x}\big(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})\big)$. A *target dependency* is either a tgd of the form $\forall \mathbf{x}\big(\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})\big)$, or an egd of the form $\forall \mathbf{x}\big(\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow (x_i = x_j)\big)$. In all dependencies, $\phi_{\mathbf{S}}(\mathbf{x})$, $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ and $\phi_{\mathbf{T}}(\mathbf{x})$ are each a conjunction of atomic relational formulas over $\mathbf{S}$ or $\mathbf{T}$ respectively and x_i, x_j are among variables in $\mathbf{x}$. The following definition formalizes a complete data exchange setting.

DEFINITION 2.2. (*complete data exchange setting*) *A* complete data exchange setting (CDE) *is a quadruple* $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$*, such that:*

- $\boldsymbol{S}$ *is the source schema and* $\mathbf{T}$ *is the target schema;*
- Σ_{st} *is a finite set of source-to-target tgds;*
- Σ_t *is a finite set of target tgds and target egds.*

We consider query answering for conjunctive queries posed over the target schema. A *conjunctive query* $q(\mathbf{x})$ over a schema $\mathbf{T}$ is a formula of the form $\exists y \varphi(\mathbf{x}, \mathbf{y})$, where $\varphi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over $\mathbf{T}$. The semantics adopted for query answering is that of *certain answers*. Certain answers were introduced as the standard semantics used in incomplete databases [17, 27]. It then became the standard semantics of query answering in data integration [24] and data exchange [13]. Definition 2.3 defines certain answers in CDE settings. Given a CDE setting $\mathbf{M}$ and a source instance I, a target instance J is called a *solution for* I *under* $\mathbf{M}$, if $(I, J) = I \cup J$ satisfies Σ_{st}, and J satisfies Σ_t.

DEFINITION 2.3. (*certain answers in CDE settings*) *Let* $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ *be a CDE setting,* q *be a union of conjunctive queries posed over the target schema* $\mathbf{T}$*, and* I *be a source instance. The certain answers of* q *w.r.t* I *and* $\mathbf{M}$*, denoted by* $certain_M(q, I)$*, is:*

$$certain_M(q, I) = \cap\{q(K) \mid K \in SOL(M, I)\},$$

where $SOL(M, I)$ *is the set of solutions for* I *and* $q(K)$ *denotes the result of applying* q *on instance* K*.*

For a CDE setting, an infinite set of solutions may exist. Consequently, from Definition 2.3 it is clear that computing the certain answers involves checking an infinite space of solutions. In [13] it was shown that for a CDE setting $\mathbf{M}$ and a source instance I, we can use *chase* to produce an instance J called *universal solution* for I under $\mathbf{M}$. Then a universal solution is used to compute the certain answers to a union of conjunctive queries. Specifically, the chase begins with I and in each step produces an instance K_i. In general, each *chase step* considers a tgd (egd respectively) $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})$ ($\phi(\mathbf{x}) \rightarrow x_1 = x_2$, respectively) in the set of constraints Σ and the current instance K_i. For every pair of tuples $\mathbf{a}$, $\mathbf{b}$ from K_i such that $\phi(\mathbf{a}, \mathbf{b})$ is true in K_i, we introduce a fresh tuple of distinct nulls $\mathbf{u}$ and create new facts in the K_i so that $\psi(\mathbf{a}, \mathbf{u})$ holds (we either equate a_1 to a_2 or report "fail" if a_1 and a_2 are distinct constants, respectively) and thus we obtain K_{i+1}. A detailed description of the chase is given in Section 5.

More specifically, for complete data exchange settings and for queries that are unions of conjunctive queries, suppose J

is a universal solution, then we can compute certain answers to unions of conjunctive queries using J:

$$certain_M(q, I) = q(J)_{\downarrow}, \quad (1)$$

where $q(J)_{\downarrow}$ is the set of all "null-free" tuples in $q(J)$, i.e., all tuples t in $q(J)$ such that every value in t is a constant.

EXAMPLE 2.1. *Consider our running Example 1.1 and suppose the input source instance is the following complete instance:* $I' = \{\texttt{Members}(john@gmail.com, John, John)\}$. *Then by chasing* I' *with the dependencies* d_{st} *and* d_t *in Example 1.1, we obtain the instance*

$$\{\ \texttt{Members}(john@gmail.com, John, John),\ \texttt{Customers}(john@gmail.com, John, John)\ \}$$

and, by projecting it on the target schema, we obtain the universal solution of I' *for the set of constraints* $\{d_{st}, d_t\}$ *which is* $J' = \{\texttt{Customers}(john@gmail.com, John, John)\}$. *By computing queries* $\texttt{Q}_1$ *and* $\texttt{Q}_2$ *on* J', *we get the answer* $\{(john@gmail.com)\}$ *and* $\{(john@gmail.com, John)\}$ *respectively.*

3. INCOMPLETE DATA EXCHANGE SETTINGS (IDE)

In this section we introduce *incomplete data exchange setting (in short IDE)* which generalizes CDE settings in that the source instance could be incomplete, i.e., it may either contain null values or have missing facts, and thus it may be accompanied by dependencies defined over the source schema. The meaning of these dependencies, called *source dependencies* is that the possible worlds represented by the incomplete instance should also satisfy certain constraints. Thus, in addition to source-to-target dependencies (Σ_{st}) and target dependencies (Σ_t), an IDE setting contains also *source dependencies* denoted Σ_s. A source dependency is a dependency over the source schema $\mathbf{S}$. Every source dependency in Σ_s could be a tgd of the form $\forall \mathbf{x}\big(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi_{\mathbf{S}}(\mathbf{x}, \mathbf{y})\big)$, or an egd of the form $\forall \mathbf{x}\big(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow (x_i = x_j)\big)$, where $\phi_{\mathbf{S}}(\mathbf{x})$ and $\psi_{\mathbf{S}}(\mathbf{x}, \mathbf{y})$ are conjunctions of atomic relational formulas over $\mathbf{S}$ and x_i, x_j are among variables in $\mathbf{x}$. The following definition formalizes an incomplete data exchange setting.

DEFINITION 3.1. (*incomplete data exchange setting*) *An* incomplete data exchange setting (in short IDE)*, is a quintuple* $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_s, \Sigma_{st}, \Sigma_t)$*, such that:*

- $\mathbf{S}$ *is a source schema and* $\mathbf{T}$ *is a target schema;*
- Σ_s *is a finite set of source tgds and source egds;*
- Σ_{st} *is a finite set of source-to-target tgds;*
- Σ_t *is a finite set of target tgds and target egds.*

An *IDE problem* is a pair $(I, \mathbf{M})$ where $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_s, \Sigma_{st}, \Sigma_t)$ is an IDE setting and I is an incomplete instance of schema $\mathbf{S}$.

Notice that in CDE settings the set of source-to-target constraints does not contain egds because this would amount to assuming also source constraints in the data exchange setting. In IDE settings we do not also consider source-to-target egds because these are in essence source egds.

Similarly to complete data exchange settings, we assume here an infinite domain of constants $\underline{\texttt{Const}}$ and an infinite set $\underline{\texttt{Var}}$ of variables, called *labeled nulls*, such that $\underline{\texttt{Const}}$ and $\underline{\texttt{Var}}$ are disjoint. Here however, we also assume an infinite domain $\underline{\texttt{Null}}$ of nulls which are pairwise disjoint from $\underline{\texttt{Const}}$ and $\underline{\texttt{Var}}$. An *incomplete instance* over a schema consists of relational facts with constants from $\underline{\texttt{Const}}$, nulls from $\underline{\texttt{Null}}$, and labeled nulls from $\underline{\texttt{Var}}$. A source incomplete instance in an IDE setting contains only constants and nulls.

Recall that in the CDE setting, labeled nulls are introduced because the target instance may contain incomplete data and the labeled nulls are used to model unknown values in the target instance. In IDE settings, both the source instance and the target instance may be incomplete. Thus, we use nulls to represent unknown values in the source and labeled nulls to represent unknown values in the target for reasons of clarity of presentation. However, most of our technical tools do not need this distinction.

Now towards defining meaningful semantics for certain answers in the next section, we explain, in the following paragraph, how an IDE problem can be viewed as a set of complete data exchange problems.

An incomplete instance I *represents* a set of ground instances, denoted $Rep(I)$, where $Rep(I)$ is defined to contain exactly all ground instances I_i such that there is a homomorphism from I to I_i. Further, we denote by $Sat(\Sigma)$ the set of all ground instances I_i such that Σ is satisfied on I_i. Thus, given a (possibly incomplete) source instance I and a set of source constraints Σ_s, we assume that the possible worlds represented by I and Σ_s is $Rep(I) \cap Sat(\Sigma_s)$.

Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_s, \Sigma_{st}, \Sigma_t)$ be an IDE setting. We call the CDE setting $\mathbf{M}' = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ *the corresponding to* $\mathbf{M}$ *CDE setting.* Let $(I, \mathbf{M})$ be an IDE problem. Let $I_i \in Rep(I) \cap Sat(\Sigma_s)$. We call $(I_i, \mathbf{M}')$ a *corresponding complete data exchange problem* (*CCDE problem*). We call *corresponding set of CDE problems* the set $\{(I_1, \mathbf{M}'), (I_2, \mathbf{M}') \ldots\}$ (may be infinite), where each $(I_i, \mathbf{M}')$ is a CCDE problem. Figure 4 illustrates an IDE problem $(I, \mathbf{M})$ and the CCDE problems $(I_i, \mathbf{M}')$.

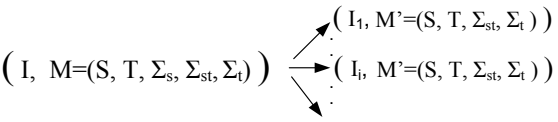

Figure 4: Corresponding complete data exchange problems.

EXAMPLE 3.1. *We revisit Example 1.1. For convenience the attribute names are replaced by single letters and the constants by integers, hence john@gmail.com is* 11 *and john is 1. Thus, the incomplete source instance is* $I : a(11, 1, p)$ *and the dependencies are as follows: There is no source dependency.*
There is one source-to-target dependency:

$$d_{st}:\ a(X, Y, Z) \rightarrow b(X, Y, Z).$$

There is one target dependency:

$$d_t:\ b(Z, X, Y) \rightarrow X = Y.$$

The source instances (infinitely many) represented by I *are succinctly described as:*

$$I_1 : a(11, 1, 1);$$
$$I_2 : a(11, 1, p'),\ p' \neq 1.$$

Actually I_2 *represents an infinite set of instances which we will denote* I_{2i} *in order to be more precise. Consider the corresponding CDE setting* $\mathbf{M}'$ *which contains* d_{st} *and* d_t. *Thus*

there is one CCDE problem $(I_1, \mathbf{M}')$ *and infinitely many CCDE problems* $(I_{2i}, \mathbf{M}')$.

4. SEMANTICS OF QUERY ANSWERING IN IDE SETTINGS.

In this section we investigate query answering in IDE settings. For the semantics of query answering we adopt the concept of *certain answers*. Certain answers are defined as the intersection of the answer of the query over all possible worlds. In general, for a setting $\mathbf{M}$, and source instance I, if we determine the set of possible worlds in the target, $PW(I, \mathbf{M})$, then the definition of certain answers is $certain_{PW(I,\mathbf{M})}(q, I) = \cap\{q(J)|J \in PW(I, \mathbf{M})\}$. For complete data exchange settings ([13]), the set of possible worlds, $PW^{CDE}(I, \mathbf{M})$, is defined as $PW^{CDE}(I, \mathbf{M}) = \{J|J \in SOL(\mathbf{M}, I)\}$.

In IDE settings, we introduce two semantics for certain answers. Suppose we are given an incomplete source instance I. We define certain answers of a query Q by resorting to the *corresponding complete data exchange problems* for all complete source instances represented by I. However there are two different perspectives to define sets of possible worlds. Either by taking all CCDE problems, for each of which we have defined certain answers (see previous sections that describe known work for complete data exchange settings), i.e., $PW^A(I, \mathbf{M}) = \{(I_i, \mathbf{M}')|(I_i, \mathbf{M}')$ *is a CCDE of* $(I, \mathbf{M})\}$; in this case we take the intersection of certain answers as computed over each $(I_i, \mathbf{M}')$ and this results to the definition of certain^A. Or, by taking all solutions for each CCDE and computing the answers to the query in each solution and then taking the intersection, i.e., in this case, $PW^B(I, \mathbf{M}) = \cup SOL(I_i, \mathbf{M}')$, for all CCDE problems $(I_i, \mathbf{M}')$. This results to the definition of certain^B.

This amounts to two *different* semantics where one is strictly more liberal than the other. Definitions 4.1 and 4.2 introduce the two semantics.

DEFINITION 4.1. ($certain^A$ *answers*)
Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_s, \Sigma_{st}, \Sigma_t)$ *be an IDE setting and* I *be an incomplete source instance. Let* $\mathbf{M}' = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ *be the corresponding CDE setting and* $I_1, I_2, \ldots$ *be all instances in* $Rep(I) \cap Sat(\Sigma_s)$. *Then,*

$$certain^A_M(q, I) = \cap certain_{M'}(q, I_i).$$

DEFINITION 4.2. ($certain^B$ *answers*)
Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_s, \Sigma_{st}, \Sigma_t)$ *be an IDE setting and* I *be an incomplete source instance. Let* $\mathbf{M}' = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ *be the corresponding CDE s etting,* $I_1, I_2, \ldots$ *be all instances in* $Rep(I) \cap Sat(\Sigma_s)$. *Let* A *be the set of instances such that an instance* J *is in* A *iff* J *is a solution of* I_i *under* $\mathbf{M}'$. *Then,*

$$certain^B_M(q, I) = \cap_{J_i \in A} q(J_i).$$

Figure 2 illustrates the two concepts of certain answers. The following proposition states the relationship between the two semantics.

PROPOSITION 4.1. *Let* $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_s, \Sigma_{st}, \Sigma_t)$ *be an IDE setting, and* I *be an (possibly incomplete) source instance. Then the following hold for any conjunctive query* q*:*
1. If $certain^B_M(q, I) = \emptyset$ *then* $certain^A_M(q, I) = \emptyset$.
2. $certain^A_M(q, I) = \emptyset$ *and* $certain^B_M(q, I) \neq \emptyset$ *iff there exists a CCDE setting* $\mathbf{M}'$ *and a source instance* I_i *such that* I_i *has no solution (and hence* $certain_{M'}(q, I_i) = \emptyset$*).*
3.If $certain^A_M(q, I) \neq \emptyset$ *then* $certain^A_M(q, I) = certain^B_M(q, I)$.
4.$certain^A_M(q, I) \subseteq certain^B_M(q, I)$.

PROOF. The first clause is a straightforward consequence of the definitions. The one direction of the second clause follows from Definition 4.1. For the other direction of the second clause we argue as follows: Since $certain^B_M(q, I) \neq \emptyset$, we know that there is at least one instance I_i in $Rep(I) \cap Sat(\Sigma_s)$ that has a solution under the corresponding CDE setting. This, combined with the fact that $certain^B_M(q, I) \neq \emptyset$, yields that there is at least one instance which does not have a solution. The reason is that, if all instances in $Rep(I) \cap Sat(\Sigma_s)$ have solutions, then, $certain^A_M(q, I) = certain^B_M(q, I)$. Clause 3 of the proposition is a straightforward consequence of the first two clauses. Clause 4 of the proposition is a straightforward consequence of Clause 3. □

EXAMPLE 4.1. *We continue from Example 3.1. Consider the query* $Q_1 : q(Y)$ *:-* $b(Y, Z, X)$. *Then,* $certain^B_M(q, I) = q(\{b(11, 1, 1)\}) = \{11\}$, *and* $certain^A_M(q, I) = certain_{M'}(q, I_1) \cap certain_{M'}(q, I_{2i}) = \{11\} \cap \emptyset = \emptyset$. *Clearly the answer* $\{11\}$ *is a desirable answer because 11 is indeed a customer. Hence for query* Q_1*, the desirable set of answers is* $certain^B_M(q, I)$.

Consider now the query $Q_2 : q(Y, X)$ *:-* $b(Y, Z, X)$. *Then,* $certain^B(q, I) = \{11, 1\}$ *and* $certain^A(q, I) = \emptyset$. *Here, however,* $\{11, 1\}$ *is not a desirable answer because it says that the credit card name of email 11 is 1 which is false, since we did not have such information about 11 in the source instance. Hence for query* Q_2*, the desirable set of answers is* $certain^A_M(q, I)$.

Figure 5 illustrates first all the complete data exchange problems that are derived in this example and, in a second level, the solutions for each of them. Thus, in this figure, it is shown that all the infinite instances represented by the $I_2 : a(11, 1, p')$, $p' \neq 1$ *have no solution. The instance* $I_1 : a(11, 1, 1)$ *has infinitely many solutions. In the figure, we show two of the solutions. Thus* $certain^A(q, I)$ *is empty (actually observe that it is empty for any query), whereas* $certain^B(q, I)$ *has one tuple for each of the two given queries.*

Observe that if the two semantics of certain answers compute the same set or not depends only on the instance I and the set of constraints, i.e., it does not depend on the query. There is the following semantic difference between the two definitions $certain^A$ and $certain^B$. Observe that the dependencies pose more constraints on the possible source ground instances in an indirect way. In particular, the target dependencies may cause some source ground instances to be "inconsistent", in the sense that they do not have a solution, e.g., $a(11, 1, 2)$ in Example 4.1. In such a case, $certain^B$ will ignore such source grounds instances, while $certain^A$ will say the answer is empty since the input of the problem is not "consistent". As we discussed in the example, both definitions are meaningful depending on which are the desirable answers.

5. COMPLEXITY OF QUERY ANSWERING

In this section we investigate the complexity of the query answering problem in IDE settings. We prove that we can compute certain answers under both semantics by using the result of a finite chase [26, 7, 6, 17]) on the source instance I. We define a *universal instance* to be the result of a finite chase (if chase does not fail) on I. We know that for an arbitrary set of dependencies, a finite chase may not exist. The conditions identified in [13] were characterized under

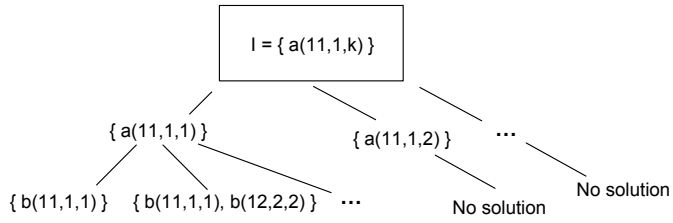

Figure 5: Explaining how to compute certain answers in Example 4.1

the term *weak acyclicity* (or under the term *stratified witness* in the independent work [12]). In Section 5.1 we show that the same conditions are still sufficient to compute certain answers to union of conjunctive queries in IDE settings.

5.1 Computing a Universal Instance

In this subsection we describe the chase procedure and describe conditions under which chase is guaranteed to terminate. We use chase to compute a universal instance for an IDE problem which will be materialized in the target and will be used to compute certain answers as we show in the next subsection. Chase is a useful tool for reasoning about dependencies (e.g. [26, 7, 6, 17]). A chase procedure may a) either never terminate; b) or fail; c) or terminate successfully. For the last case the result of chase is guaranteed to satisfy the dependencies.

DEFINITION 5.1. (*Chase Step*)
Let K be an incomplete instance. We distinguish between tgds and egds:
(tgd) *Let d be a tgd of the form : $\phi(\mathbf{x}) \rightarrow \exists\mathbf{y}\psi(\mathbf{x},\mathbf{y})$. Let h be a homomorphism from $\phi(\mathbf{x})$ to K such that there is no extension h' of h. 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:*
(i) *extending h to h' such that each variable in $\mathbf{y}$ is assigned a fresh labeled null, followed by*
(ii) *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 $K \xrightarrow{d,h} K'$ to denote the chase step.
(egd) *Let d be an egd of the form $\phi(\mathbf{x}) \rightarrow (x_1 = x_2)$. 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:*
(a) *if both $h(x_1)$ and $h(x_2)$ are in* Const *then we say that the result of applying d to K with h is* failure *and write $K \xrightarrow{d,h} \perp$ to denote the* chase step.
(b) *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 globally replaced by the constant; if both are labeled nulls, then one is replaced globally by the other. We say that the result of applying d to K with h is K' and write $K \xrightarrow{d,h} K'$.*

We continue with the definition of chase.

DEFINITION 5.2. (*Chase*)
Let Σ be a set of tgds and egds, and K be an incomplete instance.
• *An* chase sequence *of K with Σ is a sequence (finite or infinite) of chase steps $K_i \xrightarrow{d,h} K'_{i+1}$, 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 $K_i \xrightarrow{d,h} K'_{i+1}$, $0 \leq i < m$, where 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 results of the finite chase. We refer to case* (a) *as* failing *finite chase and to case* (b) *as* successful *finite chase.*

In the rest of the paper, given an IDE setting $\mathbf{M}$ and a source instance I where we have renamed the nulls in I into labelled nulls, we refer to the result of the chase J projected to the target schema as *universal instance.*

Weak Acyclicity.

In CDE settings a chase may never terminate because of a target constraint. In IDE settings, a similar phenomenon may also occur because of a source constraint. So we need to identify the conditions under which there is a finite chase. The notion of *weak acyclicity* guarantees the termination of chase. A characteristic example of weakly acyclic tgds is the class of *full* tgds, i.e., tgds with no existentially quantified variables. Without the weak acyclicity assumption on the set of target tgds, it has been shown that the existence of solution problem in complete data exchange settings may be undecidable [22]. The following definition formalizes the concept of weak acyclicity for a set of tgds.

DEFINITION 5.3. (*Weakly acyclic set of tgds*)
Let Σ be a set of tgds over a database schema. We construct a directed graph (called the dependency graph*) as follows:* (1) *There is a node for every pair (R, A) with a relation symbol R of the schema and an attribute A of R. We call such a pair (R, A) a* position. (2) *Add edges as follows: for every tgd $\phi(\mathbf{x}) \rightarrow \psi(\mathbf{x},\mathbf{y})$ in Σ and for every x in $\mathbf{x}$ that occurs in $\psi(\mathbf{x},\mathbf{y})$, we call x a* propagated variable*. For such a propagated variable x, for every occurrence of x in $\phi(\mathbf{x})$ in position (R, A_i), do the following:*
(i) *For every occurrence of x in $\psi(\mathbf{x},\mathbf{y})$ in position (S, B_j), add an edge $(R, A_i) \rightarrow (S, B_j)$ (if it does not already exist);*
(ii) *In addition, for every existentially quantified variable y in $\mathbf{y}$ and for every occurrence of y in $\psi(\mathbf{x},\mathbf{y})$ in position (T, C_k), add a* special edge *$(R, A_i) \rightarrow (T, C_k)$ (if it does not already exist).*
We say Σ is weakly acyclic *if the dependency graph has no cycle going through a special edge.*

The following lemma from [13] establishes the polynomial time bound in Theorems 5.1 and 5.3.

LEMMA 5.1. [13] *Let Σ be a set of weakly acyclic tgds and a set of egds. Then, there is a polynomial in the size of an instance K that bounds the length of every chase sequence of K with Σ.*

THEOREM 5.1. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_s, \Sigma_{st}, \Sigma_t)$ be an IDE setting where Σ_{st} is a set of tgds and Σ_s, Σ_t are both unions of a weakly acyclic set of tgs with a set of egds. Let I be the incomplete source instance. Then, computing a universal instance for I under $\mathbf{M}$ can done in polynomial time.*

PROOF. We apply chase on the source instance I after renaming all nulls to labelled nulls. From Lemma 5.1 we know that the length of chase in CDE settings is bounded

by polynomial time. Therefore, a universal instance can be computed in polynomial time in the size of I since the incomplete data exchange setting is fixed. □

The following is an interesting observation: If Σ_t is a weakly acyclic set of tgds then $certain^A_M(q, I) = certain^B_M(q, I)$. This is a consequence of Proposition 4.1 and the fact that if there is an instance I_i in $Rep(I) \cap Sat(\Sigma_s)$ that has no solution then this can be caused only by the existence of an egd. The reason is that, for complete data exchange settings that have a weakly acyclic set of tgds as target dependencies (i.e., no egds) there is always a solution[13].

The following is a known fact that shows the robustness of the notion of universal instance. There can be more than one chase sequence given an input instance I. However, we can prove that the results of any chase on I are guaranteed to be homomorphically equivalent. The proof is similar to an analogous result about universal solutions in [13].

5.2 Computing Certain Answers in IDE Settings.

We prove now that, by using the universal instance, we can compute certain answers in polynomial time for unions of conjunctive queries for both semantics and for IDE settings where the constraints contain a weakly acyclic set of tgds and a set of egds. In particular, given an IDE setting $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_s, \Sigma_{st}, \Sigma_t)$, and a (possibly incomplete) source instance I, let J be a universal instance and let (I_i, M') be a CCDE problem and J_i be the universal solution of (I_i, M'). According to our definitions, $\text{Certain}^A(q, I) = \cap q(J_i)$. However, $\text{Certain}^A(q, I) \neq q(J)_\downarrow$, whereas $\text{Certain}^B(q, I) = q(J)_\downarrow$, where we use $\downarrow$ to denote the restriction of $q(J)$ to facts that only contain constants. Thus, computing $\text{Certain}^A(q, I)$ is not done in a straightforward way using J; we need to construct an algorithm to decide whether there exists I_i such that the CCDE (I_i, M') has no solution. If there is not, then $\text{Certain}^A(q, I) = \cap q(J_i) = \text{Certain}^B(q, I) = q(J)_\downarrow$; otherwise, $\text{Certain}^A(q, I) = \emptyset$.

Thus, as we prove in detail below, in order to compute $\text{Certain}^B_{\mathbf{M}}(q, I)$, we chase I with the set $\Sigma = \Sigma_s \cup \Sigma_{st} \cup \Sigma_t$, compute a universal instance J (if there is one) and finally we compute $q(J)_\downarrow$. In order to compute $\text{Certain}^A_{\mathbf{M}}(q, I)$, we run the algorithm which decides whether there is a CCDE with no solution and then we either compute $q(J)_\downarrow$ or we decide that the set of answers is empty.

5.2.1 Complexity of computing certain^B answers.

Here we show that the result of chase, i.e., the universal instance, can be used to compute certain^B answers to unions of conjunctive queries. We simply compute the query on the universal instance. The following theorem states it formally. Theorem 5.3 shows that the complexity of computing certain^B answers of conjunctive queries is PTIME.

THEOREM 5.2. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_s, \Sigma_{st}, \Sigma_t)$ be an IDE setting and q be a union of conjunctive queries posed over the target schema $\mathbf{T}$. For every (possibly incomplete) source instance I, if the result of chase on I is J then it holds: $certain^B(q, I) = q(J)_\downarrow$.*

The proof of Theorem 5.2 is a consequence of the lemmas that are proved in the rest of this subsection.

THEOREM 5.3. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_s, \Sigma_{st}, \Sigma_t)$ be an IDE setting and q be a union of conjunctive queries posed over the target schema $\mathbf{T}$. For every (possibly incomplete) source instance I, computing $certain^B(q, I)$ can be done in polynomial time.*

PROOF. The proof of the theorem is a straightforward consequence of Theorem 5.2. We can evaluate q on a universal instance J in polynomial time and we can decide in polynomial time whether a tuple t is in the certain answers of q on J. Therefore, computing $\text{certain}^B(q, I)$ can be done in polynomial time. □

In the rest of this subsection we prove the lemmas that are used in the proof of Theorem 5.2. Lemma 5.2 is proved in [13].

LEMMA 5.2. *Let d be a tgd, K be an instance (possibly with nulls), and $K \xrightarrow{d,h} K'$ be a chase step (where $K' \neq \perp$) caused by a homomorphism h from the lhs of d to K. Let E be an instance such that:* (i) *E satisfies d; and* (ii) *K homomorphically maps to E. Then, K' homomorphically maps to E.*

The following lemma proves that a universal instance computed by the chase has a similar property as a universal solution in the complete data exchange settings, i.e., it can be homomorphically mapped to all solutions for all CCDEs.

LEMMA 5.3. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_s, \Sigma_{st}, \Sigma_t)$ be an IDE setting, I be an incomplete source instance and J a universal instance of I under $\mathbf{M}$. Let $I_1, I_2, \ldots$ be all instances in $Rep(I) \cap Sat(\Sigma_s)$ and $J_1, J_2, \ldots$ universal solutions to the CCDE problems. Then, J homomorphically maps to every J_i, $i = 1, 2, \ldots$. Hence J homomorphically maps on every solution of every CCDE problem.*

PROOF. By definitions there is a homomorphism from I to every $J_i \cup I_i$. Moreover, $J_i \cup I_i$ satisfies the constraints. From Lemma 5.2 we get that there is a homomorphism from $J \cup I$ to every $J_i \cup I_i$. Since the J's and I's are on disjoint schemas, we get that there is a homomorphism from J to J_i. □

The following lemma is the main technical result concerning the proof of Theorem 5.2. It says that we can use a universal instance computed by the chase to compute $\text{certain}^B(q, I)$.

LEMMA 5.4. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_s, \Sigma_{st}, \Sigma_t)$ be an IDE setting and q be a union of conjunctive queries posed over the target schema $\mathbf{T}$. Let I be a (possibly incomplete) source instance, J be a universal instance and $J_1, \ldots, J_k$ universal solutions to the CCDE problems. Then, it holds: $\cap\{q(J_i)\} = q(J)_\downarrow$.*

PROOF. We will first prove that $\cap\{q(J_i)\} \subseteq q(J)_\downarrow$. Equivalently we will prove that the following holds: if a tuple t is not in $q(J)_\downarrow$ then there is a source instance $I_i \in Rep(I) \cap Sat(\Sigma_s)$ and a universal solution J_i for I_i such that t is not in $q(J_i)$.

Intuitively, we will prove that there is a ground instance I_i which has, under the CCDE problem, a universal solution J_i such that $q(J_i)$ "reduced" to the tuples that contain only constants of I is equal to $q(J)_\downarrow$. Then the proof of the lemma is a straightforward consequence of this. We give below the technical details.

Let the result of chase on I be $I' \cup J''$ where I' are the tuples in the relations in the schema of the source and J'' are the tuples in the relations in the schema of the target. Let $I'_1 \cup J_i$ be the result after replacing all labeled nulls in $I' \cup J''$ by distinct constants (distinct from any constant present in $I' \cup J''$). (Thus, we obtain ground instances I'_1 resulting from I' and J_i resulting from J''.) Observe that if $I'_1 = I_i$ is the input to the corresponding CDE setting then, J_i is a universal solution of this setting because of isomorphism. However, J_i is isomorphic to J'' hence $q(J'') \downarrow = q(J_i) \uparrow$ where the symbol $\uparrow$ denotes that we consider the tuples that use only constants from J (not the new constants introduced to obtain J_1 from J).

For the other direction, let t be in $q(J)_\downarrow$. According to Lemma 5.3 the homomorphism from q to J creates a homomorphism from q to J_i. Hence t is in $q(J_i)_\downarrow$ for every i. Hence t is in $\cap q(J_i)_\downarrow = \cap_{I_l \in Rep(I) \cap Sat(\Sigma_s)} q(J_{il}) = certain_M(q, I)$. □

Proof. (of Theorem 5.2) From Definition 4.2 we have that $certain_M(q, I) = \cap q(J_i)_\downarrow$. From Lemma 5.4 we know that $\cap q(J_i)_\downarrow = q(J)_\downarrow$. It follows that $certain_M(q, I) = q(J)_\downarrow$. From Theorem 5.1 we get that a universal instance can be computed in polynomial time. □

The following example shows how the result of chase can be used to compute certain answers.

Example 5.1. *We revisit Example 4.1. A universal instance of* $I = \{a(11, 1, N)\}$ *under* $\mathbf{M}$ *is* $J = \{b(11, 1, 1)\}$. *Thus, we compute the certain*^{B} *of* Q_1 *to be equal to* $q(J) = \{(11)\}$ *and the certain*^{B} *of* Q_2 *to be equal to* $q(J) = \{(11, 1)\}$.

Interesting Observation.

We include an interesting observation from a clear theoretical point of view. Our results extend previous results related to incomplete database theory. Imielinski and Lipski in [20] proved a property of the already known chase procedure for incomplete databases. In particular, they proved that we may ignore dependencies while computing answers to conjunctive queries on incomplete, provided the database is always maintained in a "chased" form.

Let D be an incomplete database. Suppose a state of D consists of a single table T (built up from constants and variables) and a set Σ of implicational dependencies (i.e., equality-generating dependencies and *total tuple-generating dependencies* (i.e., full tgds)). Let $Rep(T)$ denotes the set of possible worlds represented by T and $Sat(\Sigma)$ the set of all relations satisfying all dependencies in Σ as we defined them in our setting. Then, w.r.t. answering conjunctive queries, the following holds:

$$Rep(T) \cap Sat(\Sigma) \equiv_{CQ} Rep(chase_\Sigma(T)) \qquad (2)$$

Our results about computing certain^B essentially extend this result to tuple generating dependencies which may not be total. To see that, imagine an IDE setting with no target constraints and with source-to-target constraints to be only copying tgds.

5.2.2 *Complexity of computing certain*^{A} *answers.*

In this subsection we essentially give an algorithm which tells whether a particular IDE problem $(\mathbf{M}, I)$ has a CCDE problem which does not have a solution. If there is, then we know that certain^A answers is empty for every conjunctive query, if not then for any conjunctive query the notions of certain^A and certain^B coincide. We state it formally in the theorem below.

Theorem 5.4. *Let* $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_s, \Sigma_{st}, \Sigma_t)$ *be an IDE setting and* q *be a union of conjunctive queries posed over the target schema* $\boldsymbol{T}$. *Given an incomplete source instance* I, *computing* $certain^A_M(q, I)$ *is polynomial.*

Proof. From Proposition 4.1 we know that we only need to decide whether there exists a CCDE problem which has no solution. If it does not then certain$^A(q, I)$=certain$^B(q, I)$ and we know from the previous subsection how to compute certain$^B(q, I)$. Otherwise certain$^A(q, I) = \emptyset$. Thus, we give an algorithm here to decide whether there is a CCDE which has no solution.

Intuition: whether there is a ground instance I_0 in $Rep(I) \cap Sat(\Sigma_s)$ which has no solution wrto M' is a "local" property, i.e., it depends on a set of tuples in I whose cardinality is bounded by a constant (which is a function of the maximum size of the tgds and egds).

Let $c = c_0^2$ where c_0 is the maximum size of tgd or egd in the constraints. The algorithm is the following:

Step 1. For each combination S of at most c nulls of I_0 we construct a sequence of I^j_S which is I after we equate the nulls in S to one of the constants in I say constant j.

Step 2. For each I^j_S, we apply chase with respect to all constraints (source, source-to-target and target constrains). If the chase does not fail we proceed to the next I^j_S. Otherwise, we report that certain$^A(q, I) = \emptyset$.

Step 3. If chase succeeds for all I^j_S then we report that certain$^A(q, I)$=certain$^B(q, I)$.

Step 4. We compute certain$^B(q, I)$.

Proof of correctness: The one direction is easy, if the chase fails, then we have a counterexample to that that certain$^A(q, I) \neq \emptyset$ a ground instance in $Rep(I) \cap Sat(\Sigma_s)$ which has no solution. For the other direction, suppose instance I_0 in $Rep(I) \cap Sat(\Sigma_s)$ has no solution. Suppose J_0 is the target instance produced after chasing with the source-to-target dependencies. Then this means that some target egd d failed on J_0 which means in turn that there is a homomorphism h from the lhs of d to J_0 (that requires, on the rhs, to equate two distinct constants). The image of h is at most c_0 tuples from J_0. These c_0 tuples were produced while applying the source-to-target tgds on I_0, hence they used at most c_0^2 tuples of I_0. Now, construct I'_0 which locally on these c_0^2 tuples is isomorphic to I_0 and otherwise is "like" I. I'_0 is one of the I^j_S that the algorithm looks at and also, by construction, it is guaranteed for the chase to fail.

In particular this is how we construct I'_0: The c tuples are isomorphic as in I_0. The rest of the tuples in I are modified in that some nulls are equated to produce the c tuples. We equate these nulls and leave the rest of the nulls as they are. And we equate each null now to a distinct constant. □

Interestingly, observe that the following idea for computing $certain^A_M(q, I)$ does not work: We use a variant of chase to compute from I an instance J such that $certain^A_M(q, I) = q(J)_\downarrow$. The following is a counterexample to this claim; it uses only one copying chase step.

Example 5.2. *Let* $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_s, \Sigma_{st}, \Sigma_t)$ *be an IDE setting where* $\Sigma_s = \emptyset$ *and*

$$\Sigma_{st} = \{d_1 : \ a(X, Y), a(Z, W) \rightarrow b(X, Y), b(Z, W)\};$$

$\Sigma_t = \{d_2 : b(X,Y), b(Y,Z) \rightarrow X = Z\}$.
Let $I = \{a(3,y), a(z,5)\}$ be an incomplete source instance. A possible world represented by I is $I_i = \{a(3,N), a(N,5)\}$. Then, chasing with d_1 results to $\{b(3,N), b(N,5)\}$ and then d_2 is applied and chase [13] fails. However, if we apply chase to I then we produce $J = \{b(3,y), b(z,5)\}$. Let $q(x) : -b(x,y)$. Then $q(J) = \{(3)\}$ whereas $certain^A_M(q,I) = \emptyset$.

6. CONCLUSION

In this work we study the query answering problem in data exchange settings where the source instance may contain null values and have missing tuples and is accompanied by dependencies. For the semantics of query answering we give two definitions of certain answers either *(a)* over all *certain answers* of the query on the corresponding complete data exchange settings or *(b)* over all *solutions* of the corresponding complete data exchange settings. Both definitions use an infinite number of possible worlds. Hence, it is a challenge to show that the problem of computing certain answers is even decidable. We show here that, for unions of conjunctive queries, we can compute certain answers in polynomial time for both semantics.

In this paper we focused on deriving those answers to the query that do not contain null values and that are guaranteed to be derived in any possible worlds. A future direction is to allow nulls in the answers and possibly derive answers that may not be guaranteed in all possible worlds but that satisfy another meaningful criterion. Another future direction is to investigate the problem of computing certain answers for incomplete data for aggregate queries and queries which have built-in predicates (e.g., arithmetic comparisons). Finally, when we have uncertain data (i.e., data with probabilities attached to it or with lineage), the problem of how to compute certain answers (in fact how to define the semantics) is open.

7. REFERENCES

[1] F. Afrati and P. Kolaitis. Answering aggregate queries in data exchange. In *PODS*, 2008.

[2] F. Afrati, C. Li, and V. Pavlaki. Data exchange in the presence of arithmetic comparisons. In *EDBT*, 2008.

[3] B. Alexe, L. Chiticariu, and W. C. Tan. Spider: a schema mapping debugger. In *VLDB*, pages 1179–1182, 2006.

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

[5] M. Arenas and L. Libkin. XML data exchange: Consistency and query answering. In *PODS*, 2005.

[6] C. Beeri and M. Y. Vardi. Formal systems for tuple and equality generating dependencies. *SIAM J. on Computing*, 13(1):76–98, 1984.

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

[8] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.

[9] P. A. Bernstein. Generic model management: A database infrastructure for schema manipulation. In *IDM 2003 Workshop*, 2003.

[10] L. Bertossi and L. Bravo. Query answering in peer to peer data exchange systems. In *EDBT Workshop on Peer to Peer Computing and Databases*, 2004.

[11] D. Burdick, P. M. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. Olap over uncertain and imprecise data. *VLDB J.*, 16(1):123–144, 2007.

[12] A. Deutsch and V. Tannen. Mars: A system for publishing xml from mixed and redundant storage. In *VLDB*, 2003.

[13] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, 2003. Full version: TCS 336(1): 89–124, 2005.

[14] R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: Getting to the core. In *PODS*, 2003. Full version: ACM TODS 30(1): 174–210, 2005.

[15] A. Fuxman, P. Kolaitis, R. Miller, and W.-C. Tan. Peer data exchange. In *PODS*, 2005.

[16] G. Gottlob. Computing cores for data exchange: New algorithms and practical solutions. In *PODS*, 2005.

[17] G. Grahne. *The problem of incomplete information in relational databases*, volume 554. Spring-Verlag, Berlin, Lecture Notes in Computer Science, 1991.

[18] L. Haas, M. Hernandez, H. Ho, L. Popa, and M. Roth. Clio grows up: from research prototype to industrial tool. In *SIGMOD*, 2005.

[19] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema mediation for large-scale semantic data sharing. *VLDB journal*, 14(1):68–83, 2005.

[20] T. Imielinski and W. Lipski. Incomplete information and dependencies in relational databases. In *SIGMOD*, pages 178–184, 1983.

[21] T. Imielinski and W. Lipski. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[22] P. Kolaitis, J. Panttaja, and W.-C. Tan. The complexity of data exchange. In *PODS*, 2006.

[23] P. G. Kolaitis. Schema Mappings, Data Exchange, and Metadata Management. In *PODS*, 2005.

[24] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.

[25] L. Libkin. Data exchange and incomplete information. In *PODS*, 2006.

[26] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM TODS*, 4(4):455–469, 1979.

[27] R. v. d. Meyden. Logical approaches to incomplete information: A survey. *In Logics for Databases and Information Systems*, pages 307–356, 1998.

[28] R. J. Miller, L. M. Haas, and M. Hernández. Schema mapping as query discovery. In *VLDB*, 2000.

[29] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating web data. In *VLDB*, 2002.

[30] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, pages 596–605, 2007.

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