Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 336 (2005) 89–124
Theoretical Computer Science
www.elsevier.com/locate/tcs

Data exchange: semantics and query answering☆

Ronald Fagin[a], Phokion G. Kolaitis[b,1], Renée J. Miller[c,1], Lucian Popa[a,*]

[a] *IBM Almaden Research Center*
[b] *University of California at Santa Cruz*
[c] *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 the query answering problem 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*. We show that a universal solution has no more and no less data than required for data exchange and that it represents the entire space of possible solutions. We then identify fairly general, yet 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 the "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 address other algorithmic issues that arise in data exchange. In particular, we study the problem of computing the certain answers of target queries by simply evaluating them on a

☆A preliminary version of this paper appeared in the 2003 International Conference on Database Theory [15].

*Corresponding author.
E-mail addresses: fagin@almaden.ibm.com (R. Fagin), kolaitis@cs.ucsc.edu (P.G. Kolaitis), miller@cs.toronto.edu (R.J. Miller), lucian@almaden.ibm.com (L. Popa).

[1] Research carried out while these authors were visiting scientists at the IBM Almaden Research Center. Kolaitis was also partially supported by NSF Grant IIS-9907419. Miller was also partially supported by a research grant from NSERC.

0304-3975/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2004.10.033

canonical universal solution, and we explore the boundary of what queries can and cannot be answered this way, in a data exchange setting.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Data exchange; Data integration; Dependencies; Universal solution; Chase; Query answering; Certain answers; Computational complexity; First-order inexpressibility

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 [30], 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 the query answering problem in the context of data exchange.

1.1. 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**. We assume that all of the variables in **x** appear free in $\phi_{\mathbf{S}}(\mathbf{x})$. We point out that schema mapping tools, such as Clio [26,27], permit the (semi-) automatic discovery of such source-to-target dependencies. Other data translation tools permit restricted forms of such dependencies to be specified in a rule language and, in certain cases, to be automatically derived from "correspondence" rules between objects [3].

Consider a fixed 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 source schema may also have dependencies that we assume are satisfied by the given source instance. Hence, the source dependencies do not play any direct role in defining the semantics of data exchange.

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 $\mathbf{T}$, and I is an instance over the source schema $\mathbf{S}$. 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, where one has to find the "right" answers to a query posed against a set of "possible" databases (see, for instance, [32]). An incomplete database can be thought of as the set of all databases that satisfy a certain specification, that is, all databases that are "possible" for the given specification. In this sense, the data exchange problem can be viewed as the problem of exchanging data between a source database I and an incomplete database representing all target instances J that are solutions for I (they satisfy the specifications of the data exchange problem), except that one is interested in actually materializing one of these solutions. Now, suppose that a query is posed against an incomplete database. There is general agreement that in this context, 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, as discussed above, the "possible" databases are the solutions J for the instance I. It also has the benefit that the query semantics is independent of the specific solution we select for data exchange. We thus adopt the certain answers as 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.

1.2. 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 [21], 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 $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ can be thought of as a data integration system in which $\mathbf{S}$ is the source schema, $\mathbf{T}$ 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 *local-as-view* (LAV) systems or *global-as-view* (GAV) systems [18,21,22]. In an 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 typically 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 generalizes both an LAV and a GAV system. In fact, it can be thought of as a *global-and-local-as-view* (GLAV) system [17,21].

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, it is often assumed to have no constraints. There has been, however, some recent work that considered the impact of target constraints in data integration. This research includes, in particular, the work of Duschka et al. [12], which showed how to compute maximally contained query plans of target queries in an LAV data integration system with target full dependencies, and the work of Calì et al. [6], which studied the impact of key and foreign key constraints on query answering in a GAV system. A more significant difference between data exchange and data integration 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 a high degree of 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?

1.3. 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 [26,27]. In Clio, source-to-target dependencies (forming a GLAV system) 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 (satisfying the target dependencies) that was the result of the

schema mapping. In designing the algorithms of Clio for creating the target instance, we were guided mainly by our 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 [3,29,30]. 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 nested (XML). 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.

1.4. 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. [6] 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 instance 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. Towards this goal, we use the concept of a *weakly acyclic* set of target dependencies; this concept is broad enough to contain as special cases both sets of full tuple-generating dependencies (full tgds) [5] and acyclic sets of inclusion dependencies [9]. In Section 3, we prove that if $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ is a data exchange setting such that Σ_{st} is a set of 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, a universal solution to the data exchange problem exists if and only if a solution exists. Moreover, for each data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ satisfying the above conditions, there is a polynomial-time algorithm that, given a source instance, determines whether a solution to the data exchange problem exists and, if so, produces a particular universal solution, which we call a *canonical* universal solution. These results make use of the classical *chase* procedure [5,23]. 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 study query answering in a data exchange setting. We adopt the notion of the certain answers as the semantics of target queries (that is, queries posed over the target schema) and we investigate two separate, but interlinked, issues. The first issue is to determine for which target queries the certain answers can be obtained using the materialized target instance alone, while the second is to analyze the

computational complexity of computing the certain answers of target queries. Note that the study of query answering in this context involves three different parameters: a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, a target query q, and a source instance I. Here, we focus on what could be called (following Vardi's [33] taxonomy) the *data complexity* of target queries in an arbitrary, but fixed, data exchange setting. This means that we have a fixed data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ and, for each target query q, we are interested in the computational complexity of the following problem: given a source instance I, find the certain answers of q with respect to I.

On the positive side, if the target query 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. To begin with, Abiteboul and Duschka [1] 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 this LAV setting is a special case of a data exchange setting in which a 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 a 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 [1], it was claimed that in a LAV setting 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 [2] 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 a coNP-complete problem. We also show that the problem of computing the certain answers for unions of conjunctive queries with inequalities remains in coNP, as long as we consider data exchange settings $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ in which Σ_{st} is a set of egds and Σ_t is a union of a set of egds with a weakly acyclic set of tgds. In proving this upper-bound result, we make use of an extension of the chase that can handle *disjunctive* egds, in addition to tgds and egds. We call this chase the *disjunctive chase*; it is a special case of the chase with disjunctive embedded dependencies defined in [10].

In contrast with the above-mentioned intractability results for the case of two inequalities or more, we then show that for the 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 [1] is false, unless P = NP). Moreover, even when the link between the source and the target has been severed, the certain answers of

unions of conjunctive queries with at most one inequality per disjunct can be computed from a given universal solution in time polynomial in the size of the universal solution. We point out, however, that this computation cannot be carried out by simply evaluating such queries on a 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 a canonical universal solution. We prove an impossibility result, which 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 a canonical universal solution. The proof of this theorem makes use of a novel combination of Ehrenfeucht-Fraïssé games and the chase. This result shows that, although there is a polynomial-time algorithm for finding the certain answers of q, there is no SQL query q^* that returns the certain answers of q when evaluated by a database engine on a canonical universal solution.

There is another way to view this impossibility result. Abiteboul and Duschka's co-NP completeness result implies that if P $\neq$ NP, then there is a conjunctive query q with inequalities whose certain answers cannot be obtained by evaluating any first-order query q^* on a canonical universal solution. We prove that the same conclusion holds even without the assumption that P $\neq$ NP. Moreover, it holds even for a query q with only one inequality, where we showed that there is a polynomial-time algorithm for obtaining the certain answers, and hence the assumption P $\neq$ NP cannot help.

2. The data exchange problem

A *schema* is a finite collection $\mathbf{R} = \{R_1, \ldots, R_k\}$ of relation symbols. Each relation symbol has an *arity*, which is a positive integer. A relation symbol of arity m is called *m-ary*, and has m distinct *attributes*, which intuitively correspond to column names. An *instance I over the schema* $\mathbf{R}$ is a function that associates to each m-ary relation symbol R_i an m-ary relation $I(R_i)$. 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*. An instance can be conveniently represented by its set of facts. 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 $i \leqslant n$ and $j \leqslant m$.

A *source-to-target dependency* is a dependency 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, 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 $x_1, \ldots, x_k$. We assume that all of the variables in $\mathbf{x}$ appear free in $\phi_{\mathbf{S}}(\mathbf{x})$. A *target* dependency is a

dependency over the target schema **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. Note that source dependencies may play an important role in deriving source-to-target dependencies [27] or in optimizing the evaluation of source queries; however, they do not play any direct role in defining the semantics of data exchange, because we take the source instance to be given. Hence, we do not include source dependencies in our formalism for data exchange.

Definition 2.1. A *data exchange setting* $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ consists of a source schema **S**, a target schema **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 $\mathrm{Sol}(I)$.

Note that the input to a data exchange problem is a source instance only; the data exchange setting itself (that is, source schema, target schema, and dependencies) is considered fixed.

For most practical purposes, and for most of the results of this paper, [2] each source-to-target dependency in Σ_{st} is a tgd [5] of the form

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

where $\phi_{\mathbf{S}}(\mathbf{x})$ is a conjunction of atomic formulas over **S** and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over **T**. We assume that all of the variables in $\mathbf{x}$ appear in $\phi_{\mathbf{S}}(\mathbf{x})$. Note that these dependencies also subsume dependencies of the form $\forall \mathbf{x}(\exists \mathbf{x}' \phi_{\mathbf{S}}(\mathbf{x}, \mathbf{x}') \rightarrow \exists \mathbf{y}\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y}))$, where the formula $\phi_{\mathbf{S}}(\mathbf{x}, \mathbf{x}')$ is a conjunction of atomic formulas over **S**, and where all of the variables in $\mathbf{x}$ appear in $\phi_{\mathbf{S}}(\mathbf{x})$, since the above formula is logically equivalent to $\forall \mathbf{x}\forall \mathbf{x}'(\phi_{\mathbf{S}}(\mathbf{x}, \mathbf{x}') \rightarrow \exists \mathbf{y}\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y}))$. Each target dependency in Σ_t is either a tgd (of the form shown below left) or an egd [5] (shown below right):

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

In the above, $\phi_{\mathbf{T}}(\mathbf{x})$ and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ are conjunctions of atomic formulas over **T**, where all of the variables in $\mathbf{x}$ appear in $\phi_{\mathbf{T}}(\mathbf{x})$, 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 [21] 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 [13], 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.

[2] Except for Proposition 4.2.

For simplicity of presentation, we do not allow for constants to occur anywhere inside the tgds and egds. However, all results of this paper can be suitably extended for such dependencies. Also, in the rest of the paper we will usually 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 2.2. 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:

$$\begin{array}{lll} \Sigma_{st}: & P(a,b,c) \to \exists Y \exists Z\ T(a,Y,Z), & I = \{P(a_0, b_0', c_0'), \\ & Q(a,b,c) \to \exists X \exists U\ T(X,b,U), & \quad Q(a_0'', b_0, c_0''), \\ & R(a,b,c) \to \exists V \exists W\ T(V,W,c), & \quad R(a_0''', b_0''', c_0)\}. \end{array}$$

We observe first that the dependencies in Σ_{st} do not completely specify the target instance. Indeed, the first dependency requires an A-value of a tuple in P to appear in the A column of T, but it does not specify any particular values for the B and C attributes. It should be noted that such incomplete specification arises naturally in many practical scenarios of data exchange (or data integration for that matter; see [18,21]). 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, that is values that do not occur in the source instance. 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:

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

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

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

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

Definition 2.3. Let K_1 and K_2 be two instances over $\mathbf{R}$ with values in $\underline{\text{Const}} \cup \underline{\text{Var}}$.

1. A *homomorphism* $h : K_1 \to K_2$ is a mapping from $\underline{\text{Const}} \cup \underline{\text{Var}}(K_1)$ to $\underline{\text{Const}} \cup \underline{\text{Var}}(K_2)$ such that: (1) $h(c) = c$, for every $c \in \underline{\text{Const}}$; (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 $h(t) = (h(a_1), \ldots, h(a_s))$).
2. 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 2.4 (*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 2.5. The instances J_1 and J_2 in Example 2.2 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 solutions. Thus, J is universal.

From an algebraic standpoint, being a universal solution is a property akin to being an *initial structure* [25] 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 [25]). The next result asserts that universal solutions have good properties as well.

Proposition 2.6. *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* $\text{Sol}(I) \subseteq \text{Sol}(I')$ *if and only if there is a homomorphism $h : J' \to J$. Consequently,* $\text{Sol}(I) = \text{Sol}(I')$ *if and only if J and J' are homomorphically equivalent.*

Proof. The first part follows immediately from the definitions. For the second part, assume first that $\text{Sol}(I) \subseteq \text{Sol}(I')$. Since $J \in \text{Sol}(I)$, it follows that $J \in \text{Sol}(I')$ and, hence, there is a homomorphism $h : J' \to J$ because J' is a universal solution for I'. Conversely, assume that there is a homomorphism $h : J' \to J$. Let J^* be a solution for I. We must show that

J^* is a solution for I', which amounts to showing that $\langle I', J^*\rangle \models \Sigma_{st}$ and $J^* \models \Sigma_t$. Since J^* is a solution for I, we already have that $J^* \models \Sigma_t$, so it suffices to show that $\langle I', J^*\rangle \models \Sigma_{st}$. Consider a tgd $\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y}))$ in Σ_{st}. We must show that $\langle I', J^*\rangle \models \forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y}))$. Since $\langle I', J'\rangle$ satisfies this tgd, it follows that for every vector $\mathbf{a}$ of constants from I' such that $I' \models \phi_{\mathbf{S}}(\mathbf{a})$, there is a vector $\mathbf{b}$ of elements of J' such that $J' \models \psi_{\mathbf{T}}(\mathbf{a}, \mathbf{b})$. Since J is a universal solution for I, there is a homomorphism $h^* : J \rightarrow J^*$. Hence, the composition $h^* \circ h$ is a homomorphism from J' to J^*. Since atomic formulas are preserved under homomorphisms and $h^* \circ h(\mathbf{a}) = \mathbf{a}$, it follows that $J^* \models \psi_{\mathbf{T}}(\mathbf{a}, h^* \circ h(\mathbf{b}))$. Thus, $\langle I', J^*\rangle \models \forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y}))$, as desired. □

The first part of Proposition 2.6 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 2.4 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 sets of 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 sets of 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, and of the empty instance for the target; then chase $\langle I, \emptyset\rangle$ by applying the dependencies in Σ_{st} and Σ_t in some arbitrary order and for as long as they are applicable. 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 3.3).

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 $\underline{\mathrm{Const}} \cup \underline{\mathrm{Var}}(J)$ 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 [5], except that here we chase with instances rather than symbolic tableaux.

Definition 3.1 (*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 Const then we say that *the result of applying d to K with h* is "failure", and write $K \xrightarrow{d,h} \perp$.
- 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 $K \xrightarrow{d,h} K'$.

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

Definition 3.2 (*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 $K_i \xrightarrow{d,h} K_{i+1}$, $0 \leqslant i < m$, 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 3.3. *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.*

The proof of the theorem makes use of the following basic property of a chase step. (This property was implicitly proved and used in [5,24], in slightly more restricted settings than ours and in different contexts.)

Lemma 3.4. *Let $K_1 \xrightarrow{d,h} K_2$ be a chase step where $K_2 \neq \perp$. Let K be an instance such that*: (i) *K satisfies d and* (ii) *there exists a homomorphism $h_1 : K_1 \to K$. Then there exists a homomorphism $h_2 : K_2 \to K$.*

Proof. *Case* 1: d is a tgd $\phi(\mathbf{x}) \to \exists\mathbf{y}\psi(\mathbf{x}, \mathbf{y})$. By the definition of the chase step, $h : \phi(\mathbf{x}) \to K_1$ is a homomorphism. Composing homomorphisms yields homomorphisms; thus

$$h_1 \circ h : \phi(\mathbf{x}) \to K$$

is a homomorphism. Since K satisfies d, there exists a homomorphism

$$h' : \phi(\mathbf{x}) \wedge \psi(\mathbf{x}, \mathbf{y}) \to K$$

such that h' is an extension of $h_1 \circ h$, that is $h'(\mathbf{x}) = h_1(h(\mathbf{x}))$. For each variable y in $\mathbf{y}$, denote by Λ_y the labeled null replacing y in the chase step. Define h_2 on $\underline{\mathrm{Var}}(K_2)$ as follows: $h_2(\Lambda) = h_1(\Lambda)$, if $\Lambda \in \underline{\mathrm{Var}}(K_1)$, and $h_2(\Lambda_y) = h'(y)$ for y in $\mathbf{y}$.

We need to show that h_2 is a homomorphism from K_2 to K, which means that h_2 maps facts of K_2 to corresponding facts of K. For facts of K_2 that are also in K_1 this is true because h_1 is a homomorphism. Let $\mathsf{T}(\mathbf{x_0}, \mathbf{y_0})$ be an arbitrary atom in the conjunction ψ. (Here $\mathbf{x_0}$ and $\mathbf{y_0}$ contain variables in $\mathbf{x}$ and $\mathbf{y}$, respectively.) Then K_2 contains, in addition to any facts of K_1, a fact $\mathsf{T}(h(\mathbf{x_0}), \mathbf{\Lambda_{y_0}})$. The image under h_2 of this fact is, by definition of h_2, the fact $\mathsf{T}(h_1(h(\mathbf{x_0})), h'(\mathbf{y_0}))$. Since $h'(\mathbf{x_0}) = h_1(h(\mathbf{x_0}))$, this is the same as $\mathsf{T}(h'(\mathbf{x_0}), h'(\mathbf{y_0}))$. But h' homomorphically maps all atoms of $\phi \wedge \psi$, in particular $\mathsf{T}(\mathbf{x_0}, \mathbf{y_0})$, into facts of K. Thus, h_2 is a homomorphism.

Case 2: d is an egd $\phi(\mathbf{x}) \to (x_1 = x_2)$. As in Case 1, $h_1 \circ h : \phi(\mathbf{x}) \to K$ is a homomorphism. We take h_2 to be h_1. We need to ensure that h_1 is still a homomorphism when considered from K_2 to K. The only way that h_1 can fail to be a homomorphism on K_2 is if h_1 maps $h(x_1)$ and $h(x_2)$ into two different constants or labeled nulls of K. But this is not the case, since K satisfies d and so $h_1(h(x_1)) = h_1(h(x_2))$. □

The proof of Theorem 3.3 is based on Lemma 3.4 and on the observation that the identity mapping is a homomorphism from $\langle I, \emptyset\rangle$ to $\langle I, J'\rangle$, for every solution J'. We give the full details next.

Proof of Theorem 3.3. *Part* 1: It follows from Definition 3.2 that $\langle I, J\rangle$ satisfies $\Sigma_{st} \cup \Sigma_t$. Since Σ_t uses only target relation symbols, it follows that J satisfies Σ_t. Let J' be an arbitrary solution. Thus, $\langle I, J'\rangle$ satisfies $\Sigma_{st} \cup \Sigma_t$. Moreover, the identity mapping $\underline{\mathrm{id}} : \langle I, \emptyset\rangle \to \langle I, J'\rangle$ is a homomorphism. By applying Lemma 3.4 at each chase step, we

obtain a homomorphism $h : \langle I, J\rangle \to \langle I, J'\rangle$. In particular, h is also a homomorphism from J to J'. Thus, J is universal.

Part 2: Let $\langle I, J\rangle \xrightarrow{d,h} \perp$ be the last chase step of a failing chase. Then d must be an egd of Σ_t, say $\phi(\mathbf{x}) \to (x_1 = x_2)$, and $h : \phi(\mathbf{x}) \to J$ is a homomorphism such that $h(x_1)$ and $h(x_2)$ are two *distinct* constants c_1 and, respectively, c_2. Suppose that there exists a solution J'. Following the same argument as in Part 1, we see that the identity homomorphism $\underline{\mathrm{id}} : \langle I, \emptyset\rangle \to \langle I, J'\rangle$ implies, by Lemma 3.4, the existence of a homomorphism $g : \langle I, J\rangle \to \langle I, J'\rangle$. Then $g \circ h : \phi(\mathbf{x}) \to J'$ is a homomorphism. Since J' is assumed to satisfy d, it must be the case that $g(h(x_1)) = g(h(x_2))$ and thus $g(c_1) = g(c_2)$. Homomorphisms are identities on $\underline{\mathrm{Const}}$, and so $c_1 = c_2$, which is a contradiction. □

For Part 1 of Theorem 3.3, we refer to such a solution J as *a canonical universal solution*. In further examples and proofs, when such J is unique (up to isomorphism), we will also use the term *the canonical universal solution*. We now give a simple example that shows that there need not be a unique canonical universal solution, even when there are no target dependencies.

Example 3.5. Consider a data exchange problem where the source schema has two unary relation symbols P and Q, and the target schema has one unary relation symbol R. Let Σ_{st} consist of the two source-to-target dependencies $P(x) \to R(x)$ and $Q(x) \to \exists Y\, R(Y)$, and let $\Sigma_t = \emptyset$. Let $I = \{P(a), Q(a)\}$. If we chase first with the first dependency, we obtain the canonical universal solution $\{Q(a)\}$, with only one tuple. If we chase first with the second dependency, we obtain the canonical universal solution $\{Q(Y), Q(a)\}$ with two tuples, one of which has a null. So there is not a unique 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 [24]). It is also similar to the canonical database of Calì et al. [6] defined in a more restricted setting, that of GAV with key and foreign key constraints.

The following is an example of a 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 on the class of dependencies that are allowed in the target.

Example 3.6. Consider the data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ as follows (this scenario is also graphically but informally shown in Fig. 1). The source schema $\mathbf{S}$ has one relation $\mathtt{DeptEmp}(\mathtt{dpt\_id}, \mathtt{mgr\_name}, \mathtt{eid})$ listing departments with their managers and their employees. The target schema $\mathbf{T}$ has a relation $\mathtt{Dept}$ $(\mathtt{dpt\_id}, \mathtt{mgr\_id}, \mathtt{mgr\_name})$ for departments and their managers, and a separate relation for employees $\mathtt{Emp}$ $(\mathtt{eid}, \mathtt{dpt\_id})$. The source-to-target and target dependencies are:

$$\begin{aligned}
\Sigma_{st} &= \{\, \mathtt{DeptEmp}(d, n, e) \to \exists M(\mathtt{Dept}(d, M, n) \wedge \mathtt{Emp}(e, d)) \,\},\\
\Sigma_t &= \{\, \mathtt{Dept}(d, m, n) \to \exists D\ \mathtt{Emp}(m, D),\\
&\qquad \mathtt{Emp}(e, d) \to \exists M \exists N\ \mathtt{Dept}(d, M, N) \,\}.
\end{aligned}$$

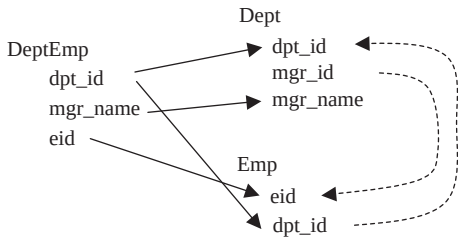

Fig. 1. Data exchange with infinite chase.

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

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

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 `Emp` as an employee id. Thus, the chase will add $\texttt{Emp}(M, D)$ 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:

$$J = \{\, \texttt{Dept}(CS, M, Mary), \texttt{Emp}(E003, CS), \texttt{Emp}(M, D), \\ \texttt{Dept}(D, M', N'), \ldots \}.$$

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

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

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 [5] 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 [7]. It is well known that chasing with inclusion dependencies may not terminate in general. *Acyclic sets of inclusion dependencies* [9] 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 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 *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 [20], 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.

This concept first arose in a conversation between the last author and Deutsch in 2001. Preliminary reports on this concept appeared independently in [15] (the conference version of this article) and in [11] (in the latter paper, under the term *constraints with stratified-witness*).

Definition 3.7 (*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 $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$ in Σ and for every x in $\mathbf{x}$ that *occurs* in ψ:

- For every occurrence of x in ϕ in position (R, A_i):
  - (a) for every occurrence of x in ψ in position (S, B_j), add an edge $(R, A_i) \rightarrow (S, B_j)$ (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* $(R, A_i) \stackrel{*}{\rightarrow} (T, C_k)$ (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

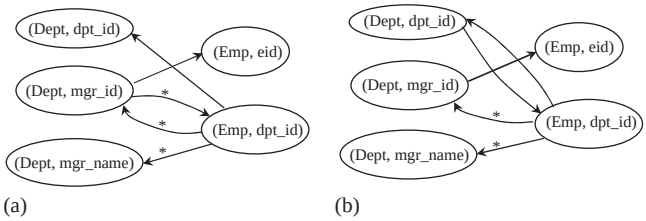

Fig. 2. Dependency graphs for: (a) a set of tgds that is not weakly acyclic, (b) a weakly acyclic set of tgds.

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 3.8. Recall Example 3.6. The dependency graph of Σ_t is shown in Fig. 2(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 3.6). 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

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

The dependency graph of Σ'_t, shown in Fig. 2(b), has no cycles going through a special edge. Thus, Σ'_t is weakly acyclic. As Theorem 3.9 will show, it is guaranteed that every chase sequence is finite. For Example 3.6, 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 Cosmadakis and Kanellakis [9].

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

Theorem 3.9. *Let Σ be the union of a weakly acyclic set of tgds with a set of egds. Then there exists a polynomial in the size of an instance K that bounds the length of every chase sequence of K with Σ.*

Proof. We give the proof for the case when Σ does not have any egds. The addition of egds does not essentially change the argument and we leave the details to the interested reader. For every node (R, A) in the dependency graph of Σ, define an *incoming path* to be any (finite or infinite) path ending in (R, A). Define the *rank* of (R, A), denoted by $\underline{\text{rank}}(R, A)$, as the maximum number of special edges on any such incoming path. Since Σ is weakly acyclic, there are no cycles going through special edges. Thus $\underline{\text{rank}}(R, A)$ is finite. Let r

be the maximum, over all positions (R, A), of $\underline{\text{rank}}(R, A)$, and let p be the total number of positions (R, A) in the schema (equal to the number of nodes in the graph). The latter number is a constant, since the schema is fixed. Moreover, r is at most p. Thus r is not only finite but bounded by a constant. The next observation is that we can partition the nodes in the dependency graph, according to their rank, into subsets $N_0, N_1, \ldots, N_r$, where N_i is the set of all nodes with rank i. Let n be the total number of distinct values (constants or labeled nulls) that occur in the instance K. Let K' be any instance obtained from K after some arbitrary chase sequence. We prove by induction on i the following claim:

For every i there exists a polynomial Q_i such that the total number of distinct values that occur in K' at positions that are restricted to be in N_i is at most $Q_i(n)$.

Base case: If (R, A) is a position in N_0, then there are no incoming paths with special edges. Thus no new values are ever created at position (R, A) during the chase. Hence, the values occurring in K' at position (R, A) are among the n values of the original instance K. Since this is true for all the positions in N_0, we can then take $Q_0(n) = n$.

Inductive case: The first kind of values that may occur in K' at a position of N_i are those values that already occur in K at the same position. The number of such values is at most n. In addition, a value may occur in K' at a position of N_i for two reasons: by being *copied* from some position in N_j with $j \neq i$, during a chase step, or by being *generated* as a new value (labeled null), also during a chase step. We count first how many values can be generated. Let (R, A) be some position of N_i. A new value can be generated in (R, A) during a chase step only due to special edges. But any special edge that may enter (R, A) must start at a node in $N_0 \cup \cdots \cup N_{i-1}$. Applying the inductive hypothesis, the number of distinct values that can exist in all the nodes in $N_0 \cup \cdots \cup N_{i-1}$ is bounded by $P(n) = Q_0(n) + \cdots + Q_{i-1}(n)$. Let d be the maximum number of special edges that enter a position, over all positions in the schema. Then for every choice of d values in $N_0 \cup \cdots \cup N_{i-1}$ (one value for each special edge that can enter a position) and for every dependency in Σ there is at most one new value that can be generated at position (R, A). (This is a consequence of the chase step definition and of how the special edges have been defined.) Thus the total number of new values that can be generated in (R, A) is at most $(P(n))^d \times D$, where D is the number of dependencies in Σ. Since the schema and Σ are fixed, this is still a polynomial in n. If we consider *all* positions (R, A) in N_i, the total number of values that can be generated is at most $p_i \times (P(n))^d \times D$ where p_i is the number of positions in N_i. Let $G(n) = p_i \times (P(n))^d \times D$. Obviously, G is a polynomial.

We count next the number of distinct values that can be copied to positions of N_i from positions of N_j with $j \neq i$. Such copying can happen only if there are non-special edges from positions in N_j with $j \neq i$ to positions in N_i. We observe first that such non-special edges can originate only at nodes in $N_0 \cup \cdots \cup N_{i-1}$, that is, they cannot originate at nodes in N_j with $j > i$. Otherwise, assume that there exists $j > i$ and there exists a non-special edge from some position of N_j to a position (R, A) of N_i. Then the rank of (R, A) would have to be larger than i, which is a contradiction. Hence, the number of distinct values that can be copied in positions of N_i is bounded by the total number of values in $N_0 \cup \cdots \cup N_{i-1}$, which is $P(n)$ from our previous consideration. Putting it all together, we can take $Q_i(n) = n + G(n) + P(n)$. Since Q_i is a polynomial, the claim is proven.

In the above claim, i is bounded by the maximum rank r, which is a constant. Hence, there exists a fixed polynomial Q such that the number of distinct values that can exist in

K', over all positions, is bounded by $Q(n)$. In particular, the number of distinct values that can exist in K' at a single position is also bounded by $Q(n)$. Then the total number of tuples that can exist in one relation in K' is bounded by $Q(n)^p$ since the maximum number of attributes in one relation is bounded by p (recall that p is the total number of positions in the schema). It follows that the total number of tuples that can exist in K', over all relations, is at most $s \times Q(n)^p$, where s is the number of relations in the schema. This is a polynomial in n since s and p are assumed to be constant. Finally, since every chase step with a tgd adds at least some tuple to K', it follows that the length of any chase sequence is at most $s \times (Q(n))^p$. □

Corollary 3.10. *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 4.1. Let $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting.

- Let q be a k-ary query, for $k \geqslant 0$, 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 $t \in q(J)$.
- In particular, let q be a Boolean (that is, 0-ary) query over the target schema $\mathbf{T}$ and I a source instance. If we let $\underline{\text{true}}$ denote the set with one 0-ary tuple and $\underline{\text{false}}$ denote the empty set, then $q(J) = \underline{\text{true}}$ and $q(J) = \underline{\text{false}}$ each have their usual meanings for Boolean queries q. Note that $certain(q, I) = \underline{\text{true}}$ means that for every solution J of this instance of the data exchange problem, we have that $q(J) = \underline{\text{true}}$; moreover, $certain(q, I) = \underline{\text{false}}$ means 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 let us define $q(J)_{\downarrow}$ to be 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 $\underline{\text{false}}$).

A *conjunctive query* $q(\mathbf{x})$ over a schema $\mathbf{R}$ is a formula of the form $\exists \mathbf{y} \phi(\mathbf{x}, \mathbf{y})$ where $\phi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over $\mathbf{R}$. If, in addition to atomic formulas, the

conjunction $\phi(\mathbf{x}, \mathbf{y})$ is allowed to contain inequalities of the form $z_i \neq z_j$, where z_i, z_j are variables among $\mathbf{x}$ and $\mathbf{y}$, we call $q(\mathbf{x})$ a *conjunctive query with inequalities.* A *union of conjunctive queries* (*with inequalities*) is a disjunction $q(\mathbf{x}) = q_1(\mathbf{x}) \vee \cdots \vee q_n(\mathbf{x})$ where $q_1(\mathbf{x}), \ldots, q_n(\mathbf{x})$ are conjunctive queries (with inequalities).

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 [1]. Another instance of this result has also been noted for the GAV setting with key/foreign key constraints in [6]. The proposition shows that 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 4.2. *Consider a data exchange setting with* $\mathbf{S}$ *as the source schema,* $\mathbf{T}$ *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* $\mathbf{T}$. *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* $\mathbf{T}$, *we have that* $certain(q, I) = q(J)_{\downarrow}$. *Then J is a universal solution.*

Proof. *Part* 1: Let q be a k-ary query that is a union of conjunctive queries and let t be a k-tuple of constants from the source instance I. If $t \in certain(q, I)$, then $t \in q(J)$, since J is a solution. Conversely, assume that $t \in q(J)_{\downarrow}$. Then t consists only of constants. Also there exists a conjunctive query $\exists \mathbf{y} \phi(\mathbf{x}, \mathbf{y})$ that is a disjunct of q and a homomorphism $g : \phi(\mathbf{x}, \mathbf{y}) \to J$ such that $g(\mathbf{x}) = t$. Let J' be an arbitrary solution. Since J is a universal solution, there is a homomorphism $h : J \to J'$. Then $h \circ g$ is a homomorphism from $\phi(\mathbf{x}, \mathbf{y})$ to J'. Homomorphisms are identities on constants, hence $h(g(\mathbf{x})) = h(t) = t$. Thus $t \in q(J')$.

Part 2: Let q^J be the *canonical* conjunctive query associated with J (i.e., q^J is the Boolean conjunctive query obtained by taking the conjunction of all the facts of J in which the labeled nulls are replaced by existentially quantified variables). Now $certain(q^J, I) = q^J(J)_{\downarrow} = q^J(J)$, where the first equality follows from our assumption about J, and where the second equality follows from the fact that q^J is a Boolean query. Since also $q^J(J) = \underline{\text{true}}$, we have $certain(q^J, I) = \underline{\text{true}}$. Therefore, if J' is an arbitrary solution, then $q^J(J') = \underline{\text{true}}$. As first shown by Chandra and Merlin [8], this implies the existence of a homomorphism $h : J \to J'$. Hence, J is universal. □

In the preceding Proposition 4.2, the query q can be a finite or an infinite union of conjunctive queries. Thus, this proposition holds for arbitrary Datalog queries.

The following result follows from Corollary 3.10 and Part 1 of Proposition 4.2.

Corollary 4.3. *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.*

Conjunctive queries with inequalities: 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 4.4. *Let S be a binary source relation symbol, T a binary target relation symbol,* $S(x, y) \rightarrow \exists z(T(x, z) \wedge T(z, y))$ *a source-to-target dependency, and q the following Boolean conjunctive query with one inequality*: $\exists x \exists y(T(x, y) \wedge (x \neq y))$.

1. *There is a source instance I such that certain*$(q, I) =$ *false, but* $q(J) =$ *true for every universal solution J.*
2. *Let* q^* *be the query* $\exists x \exists y \exists z(T(x, z) \wedge T(z, y) \wedge (x \neq y))$. *If I is a source instance and J is the canonical universal solution, then certain*$(q, I) = q^*(J)$.

Proof. *Part* 1: Let I be the source instance with $I(S) = \{(a, a)\}$, where a is some constant. Note that *certain*$(q, I) =$ false, because $J_1(T) = \{(a, a)\}$ is a solution and $q(J_1) =$ false. Let J be an arbitrary universal solution. We will prove that $q(J) =$ true by showing that $J(T)$ must contain two tuples (a, X) and (X, a) with $a \neq X$. Towards this goal, first note that J must contain two tuples of the form (a, X) and (X, a), because J is a solution. Consider now the solution J_2 with $J_2(T) = \{(a, b), (b, a)\}$, where $b \neq a$. Since J is a universal solution, there is a homomorphism h from J to J_2. It follows that $J(T)$ must contain two tuples of the form (a, X) and (X, a) with $X \neq a$, since, otherwise, $(a, a) \in J(T)$ and $(h(a), h(a)) = (a, a) \notin J_2(T)$.

Part 2: Let I be a source instance and J be the canonical universal solution (it is easy to see that in this case, the canonical universal solution is unique up to isomorphism). We have to show that *certain*$(q, I) = q^*(J)$. For this, we consider two cases.

Case 1: $I(S)$ has a tuple (a, b) with $a \neq b$. If J' is an arbitrary solution, then $J'(T)$ contains two tuples (a, X) and (X, b). If $X = a$, then $J'(T)$ contains (a, b) with $a \neq b$; if $X \neq a$, then $J'(T)$ contains (a, X) with $a \neq X$. In either case, we have that $q(J') =$ true, hence *certain*$(q, I) =$ true. Moreover, in either case we have that $q^*(J) =$ true, since J, being a solution, must contain two tuples of the form (a, X) and (X, b), and $a \neq b$. Note that the only property of J we used here was that it is a solution.

Case 2: $I(S)$ has no tuple (a, b) with $a \neq b$. Hence, $I(S)$ is a relation consisting entirely of reflexive tuples (a, a). If J' is the solution with $J'(T) = I(S)$, then $q(J') =$ false and, consequently, *certain*$(q, I) =$ false. At the same time, the canonical universal solution J consists of tuples of the form (a, X_a), (X_a, a) such that $(a, a) \in I(S)$, where a different labeled null X_a is used for each constant a. Consequently, $q^*(J) =$ false. □

In view of Proposition 4.4, 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 a 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 [1]. It follows that in the data exchange setting, it is not possible to compute the certain answers of such queries q by evaluating q (or any associated query q^* with polynomial-time evaluation) on a canonical universal solution or on any universal solution that is generated in polynomial time (unless P = NP). In Section 5.1, we take a closer look at conjunctive queries with inequalities. First, we show (Theorem 5.2) 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 5.12) 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 5.11) 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 a canonical universal solution. We begin by formally introducing the decision problem associated with the computation of the set of certain answers.

Definition 5.1. 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 simply as a *LAV setting*.

5.1.1. An upper bound

Abiteboul and Duschka [1] 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 general 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 5.2. *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". Intuitively, the small model property says that if there is a "witness" to the satisfaction or failure of some property, then there is a "witness" of bounded size (essentially this argument was used in [1] 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, that we call the *disjunctive chase* and define shortly.

To decide whether $t \in \mathit{certain}(q, I)$, we substitute t into the query q to obtain a Boolean query. We thereby reduce the problem of deciding whether $t \in \mathit{certain}(q, I)$ for arbitrary queries q to the problem of deciding whether $\mathit{certain}(q, I) = \underline{\text{true}}$ for Boolean queries q. Hence, we can assume that q is a Boolean query. We know that q is equivalent to a query of the form $q_1 \vee q_2$, where q_1 is the disjunction of a set C of conjunctive queries with no inequalities, and q_2 is the disjunction of a set C' of conjunctive queries each with at least one inequality. Each element of C' has the form:

$$\exists \mathbf{x} \left(\phi(\mathbf{x}) \wedge \left(\bigwedge_i (x_i^1 \neq x_i^2) \right) \right),$$

where $\phi(\mathbf{x})$ is a conjunction of atomic formulas. Hence, it is easy to see that the negation of q_2 is equivalent to the conjunction of a set E of formulas of the form:

$$\forall \mathbf{x} \left(\phi(\mathbf{x}) \rightarrow \left(\bigvee_i (x_i^1 = x_i^2) \right) \right).$$

We will call such formulas *disjunctive egds*. As in the case of tgds and egds, for simplicity, we will drop the universal quantifiers in front of a disjunctive egd. Note that an egd is a particular case of a disjunctive egd where the right-hand side of the logical implication sign

has only one equality. We observe next the following fact (easy to verify):

Lemma 5.3. *The following statements are equivalent*:
(1) $certain(q, I) = \underline{false}$.
(2) *There exists a solution* J^* *for* I *such that* J^* *satisfies* E *and* J^* *does not satisfy any of the conjunctive queries in* C.

Next we will show that the problem of deciding the above condition (2) is in NP, under the conditions stated in Theorem 5.2. Theorem 5.2 follows then immediately. To prove the membership in NP of the aforementioned problem, we need to define first the disjunctive chase. Deutsch and Tannen [10] introduced an extension of the classical chase in order to make use, in the process of query optimization, of a very general class of dependencies with disjunction, called disjunctive embedded dependencies (DEDs). For our purposes, we need an extension only to deal with disjunctive egds, which are a particular case of DEDs. Hence, the next definition is a particular case of the definition in [10]. We note, however, that the subsequent properties of the chase that we prove and then use in this subsection are new.

Definition 5.4 (*Disjunctive chase step*). Let K be an instance and let e be a disjunctive egd $\phi(\mathbf{x}) \rightarrow ((x_1^1 = x_1^2) \vee \cdots \vee (x_l^1 = x_l^2))$. Denote by $e_1, \ldots, e_l$ the following egds obtained from e: $\phi(\mathbf{x}) \rightarrow (x_1^1 = x_1^2), \ldots, \phi(\mathbf{x}) \rightarrow (x_l^1 = x_l^2)$, and call them the egds *associated* with e.

Let h be a homomorphism from $\phi(\mathbf{x})$ to K such that $h(x_1^1) \neq h(x_1^2), \ldots, h(x_l^1) \neq h(x_l^2)$. We say that *$e$ can be applied to K with homomorphism h*. Note that it is also the case that each of $e_1, \ldots, e_l$ can be applied to K with homomorphism h, by Definition 3.1. For each $i = 1, \ldots, l$, let K_i be the result of applying e_i to K with homomorphism h (i.e., $K \xrightarrow{e_i,h} K_i$) according to Definition 3.1. (Note that some of the K_i's can be $\perp$.) We distinguish two cases:

- If all of $K_1, \ldots, K_l$ are $\perp$ then we say that *the result of applying e to K with h* is "failure" and write $K \xrightarrow{e,h} \{\perp\}$, or simply $K \xrightarrow{e,h} \perp$.
- Otherwise, let $K_{i_1}, \ldots, K_{i_p}$ be those elements in the set $\{K_1, \ldots, K_l\}$ that are not $\perp$. We say that *the result of applying e to K with h* is the set $\{K_{i_1}, \ldots, K_{i_p}\}$, and write $K \xrightarrow{e,h} \{K_{i_1}, \ldots, K_{i_p}\}$.

Note that in the case when e has only one term in the disjunction the above definition degenerates to Definition 3.1. Thus a chase step with an egd is a particular case of a disjunctive chase step. For such chase steps, we will use, as convenience may dictate, either the notation $K \xrightarrow{e,h} K_{i_1}$ as in Definition 3.1 or the full notation $K \xrightarrow{e,h} \{K_{i_1}\}$. In addition to chase steps with (disjunctive) egds, we will continue to use chase steps with tgds as in Definition 3.1. For such chase steps, we will use either the notation $K \xrightarrow{d,h} K'$ or the notation $K \xrightarrow{d,h} \{K'\}$. We next define the finite disjunctive chase.

Definition 5.5 (*Disjunctive chase*). Let Σ be a set of tgds and egds and let E be a set of disjunctive egds, and let K be an instance.

- A *chase tree of K with $\Sigma \cup E$* is a tree (finite or infinite) such that:
  - the root is K, and
  - for every node K_j in the tree, let $\{K_{j_1}, \ldots, K_{j_r}\}$ be the set of its children. Then there must exist some dependency d in $\Sigma \cup E$ and homomorphism h such that $K_j \xrightarrow{d,h} \{K_{j_1}, \ldots, K_{j_r}\}$.[3]
- A *finite disjunctive chase of K with $\Sigma \cup E$* is a finite chase tree with the requirement that each leaf K_m satisfies either (a) $K_m = \perp$ or (b) there is no dependency d in $\Sigma \cup E$ and there is no homomorphism h such that d can be applied to K_m with h.

As with the traditional chase, there may not exist in general a finite disjunctive chase of an instance. However, if the tgds involved are required to form a weakly acyclic set then we can prove the following proposition, which is similar to Theorem 3.9.

Proposition 5.6. *Let Σ be the union of a weakly acyclic set of tgds with a set of egds. Let E be a set of disjunctive egds, and let K be a instance. Then every chase tree of K with $\Sigma \cup E$ is finite. Moreover, there exists a polynomial in the size of K that bounds the depth of every such chase tree.*

Proof. Let E' be the set of all egds that are associated with some disjunctive egd of E. Let T be an arbitrary chase tree of K with $\Sigma \cup E$. Then every path of T that starts at the root forms a chase sequence of K, in the sense of Definition 3.2, where the dependencies involved are from $\Sigma \cup E'$. Since the tgds in Σ form a weakly acyclic set, we can then use Theorem 3.9 to conclude that there exists a polynomial in the size of K that bounds the length of every such path. □

We prove next that condition (2) in Lemma 5.3 can be verified by checking first that a universal solution exists (by Corollary 3.10 this can be done in polynomial time under the given assumption that the tgds of the data exchange setting form a weakly acyclic set) and then by using the disjunctive chase on the universal solution. More precisely, we prove the following proposition.

Proposition 5.7. *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. Moreover, on the target schema, assume a set E of disjunctive egds and a set C of Boolean conjunctive queries. Let I be a source instance. Then the following are equivalent*:

(i) *There exists a solution J^* for I such that J^* satisfies E and J^* does not satisfy any of the conjunctive queries in C.*

(ii) *There exists a universal solution J for I, there exists a finite disjunctive chase T of J with $\Sigma_t \cup E$, and there exists a leaf $J^* \neq \perp$ of T such that J^* does not satisfy any of the conjunctive queries in C.*

[3] Note that such a chase step can be either a disjunctive chase step as in Definition 5.4 (if d is a disjunctive egd) or a "traditional" chase step as in Definition 3.1 (if d is an egd or tgd, and so $\{K_{j_1}, \ldots, K_{j_r}\}$ is a singleton set).

The proof of Proposition 5.7 uses the following extension of Lemma 3.4, for the case of a chase step with a (disjunctive) egd. To handle chase steps with tgds, the proof of Proposition 5.7 will use directly Lemma 3.4.

Lemma 5.8. *Let $K \xrightarrow{e,h} \{K_{i_1}, \ldots, K_{i_p}\}$ be a non-failing disjunctive chase step. Let K^* be an instance such that K^* satisfies e and there exists a homomorphism $g : K \to K^*$. Then there exists $j \in \{i_1, \ldots, i_p\}$ such that $g : K_j \to K^*$ is a homomorphism.*

Proof. Assume that the disjunctive egd e is: $\phi(\mathbf{x}) \to ((x_1^1 = x_1^2) \vee \cdots \vee (x_l^1 = x_l^2))$. Then h is a homomorphism from $\phi(\mathbf{x})$ to K, and $\{i_1, \ldots, i_p\}$ is the set of those indices j among $\{1, \ldots, l\}$ such that $K \xrightarrow{e_j,h} K_j$ and $K_j \neq \perp$. We first note that $g \circ h : \phi(\mathbf{x}) \to K^*$ is a homomorphism. Since K^* satisfies e, there exists some $j \in \{1, \ldots, l\}$ such that $g(h(x_j^1)) = g(h(x_j^2))$. We show next that $j \in \{i_1, \ldots, i_p\}$. In other words, j is such that $K_j \neq \perp$. Suppose towards a contradiction that $K_j = \perp$. Since K_j is the result of applying the egd e_j to K with homomorphism h, it must be the case that $h(x_j^1) = c_1$ and $h(x_j^2) = c_2$, where c_1 and c_2 are two distinct constants. On the other hand, we have $g(c_1) = g(c_2)$, which implies $c_1 = c_2$ (since homomorphisms preserve constants). We have thus reached a contradiction. Hence $j \in \{i_1, \ldots, i_p\}$. We need to ensure that g is still a homomorphism when considered from K_j to K^*. The only difference between K_j and K is the identification of $h(x_j^1)$ and $h(x_j^2)$ within K_j. Hence, the only way that g can fail to be a homomorphism on K_j is if g maps $h(x_j^1)$ and $h(x_j^2)$ into two different constants or labeled nulls of K^*. But this cannot happen, since $g(h(x_j^1)) = g(h(x_j^2))$. □

Proof of Proposition 5.7. We prove first that (i) implies (ii). Assume that (i) is true. Since the tgds in Σ_t form a weakly acyclic set, it is the case that any chase with $\Sigma_{st} \cup \Sigma_t$ of $\langle I, \emptyset \rangle$ terminates (by Theorem 3.9). Moreover there can be no failing chase, since otherwise there would be no solution at all, by Theorem 3.3, and hence (i) would be false. Thus, the result of the chase (any chase) with $\Sigma_{st} \cup \Sigma_t$ provides a universal solution J.

Proposition 5.6 implies that a finite disjunctive chase T of J with $\Sigma_t \cup E$ must exist. We prove next that T contains a leaf satisfying the properties required in (ii). Let J^* be the instance guaranteed to exist by (i). Since J^* is a solution, it must be the case that there exists a homomorphism $g : J \to J^*$. Applying either Lemma 5.8 or 3.4 at each level in the chase tree, we must find in T a path $J, J_1, \ldots, J_m$, with $J_m \neq \perp$, such that there exists a homomorphism $g_m : J_m \to J^*$ and such that either (a) J_m is a leaf or (b) $J_m \xrightarrow{e,h} \perp$, for some e in $\Sigma_t \cup E$ and homomorphism h. Suppose towards a contradiction that (b) is true. We note that e must be a (disjunctive) egd for the chase step of J_m to fail. Assuming e is $\phi(\mathbf{x}) \to ((x_1^1 = x_1^2) \vee \cdots \vee (x_l^1 = x_l^2))$, we have that h is a homomorphism from $\phi(\mathbf{x})$ to J_m. Then, for every $j \in \{1, \ldots, l\}$, we have that $h(x_j^1)$ and $h(x_j^2)$ are two distinct constants of J_m (otherwise the chase step would not produce $\perp$). We also have that $g_m \circ h$ is a homomorphism from $\phi(\mathbf{x})$ to J^*. Moreover, since homomorphisms preserve constants, it follows that $g_m(h(x_j^1))$ and $g_m(h(x_j^2))$ are two distinct constants of J^*, for every $j \in \{1, \ldots, l\}$. This contradicts the fact that J^* satisfies e. Thus, we proved that T contains a leaf J_m ($J_m \neq \perp$) such that there exists a homomorphism $g_m : J_m \to J^*$.

The existence of g_m ensures that J_m cannot satisfy any of the conjunctive queries in C, or otherwise J^* would satisfy some conjunctive query of C. Hence, J_m can play the role of J^* required by (ii).

Finally, we prove that (ii) implies (i). We show that the leaf J^* guaranteed to exist, by (ii), satisfies the requirements of (i). In particular, J^* satisfies the dependencies in Σ_t and E because it is a leaf in the chase tree. It is also easy to see that the disjunctive chase with $\Sigma_t \cup E$ does not affect the satisfaction of the source-to-target dependencies (i.e., J^* continues to satisfy Σ_{st}, as the universal solution J does). □

Proof of Theorem 5.2. Based on Proposition 5.7 and Lemma 5.3, we can check that $certain(q, I) = \underline{\text{false}}$ by checking that there exists a universal solution J for I, there exists a finite disjunctive chase T of J with $\Sigma_t \cup E$ and there exists a leaf $J^* \neq \perp$ of T such that J^* does not satisfy any of the conjunctive queries in C. All this can be verified, non-deterministically, in polynomial time. More precisely, suppose that $certain(q, I) = \underline{\text{false}}$. Then we produce, in polynomial time (by Theorem 3.9), a universal solution J. Next we guess the sequence of dependencies and homomorphisms to be applied during the disjunctive chase *as well as* the branch that we pick at each step. We therefore non-deterministically find a finite disjunctive chase T *and* path within T leading to the "right" leaf J^*. The sequence of guesses is of polynomial length, by Proposition 5.6. Verifying that J^* is a leaf (i.e., that no dependency d in $\Sigma_t \cup E$ and no homomorphism h exist such that d can be applied to J^* with h) can be done in polynomial time. In addition, verifying that J^* does not satisfy any of the conjunctive queries in C can be done in polynomial time. Conversely, suppose that $certain(q, I) = \underline{\text{true}}$. Then either no universal solution exists (and no solution exists) or a universal solution exists but no sequence of guesses as above exists that could lead to acceptance. Hence, deciding whether $certain(q, I) = \underline{\text{false}}$ is in NP. Therefore, computing the certain answers, under the conditions of Theorem 5.2, is in coNP. □

5.1.2. Lower bounds

Theorem 5.2 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 [1] 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 [1] is correct; in fact, the unpublished full version [2] 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 [31]. Thus, the next result provides a matching lower bound for the complexity of computing the certain answers of conjunctive queries with inequalities.

Theorem 5.9 (*Abiteboul and Duschka [1]*). *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 5.10. *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 5.11. *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*.

Proof. As mentioned earlier in this section, membership in coNP was first established by Abiteboul and Duschka [1]. This membership also follows from Theorem 5.2 proved in Section 5.1.1 for the more general data exchange setting. The coNP-hardness is established by a reduction from the complement of POSITIVE-NOT-ALL-EQUAL-3SAT, which is the following decision problem: given a 3CNF-formula φ consisting entirely of positive clauses $(x \vee y \vee z)$, is there a truth assignment to the variables of φ such that for every clause of φ at least one variable is assigned value "true" and at least one variable is assigned value "false"? This problem is known to be NP-complete (for instance, this can be derived easily from Schaefer's [28] results on the complexity of GENERALIZED SATISFIABILITY problems).

Before embarking on the description of the reduction, we give some intuition for one of the key constructs in the reduction. Suppose that a database schema contains a binary relation symbol L' and consider an instance in which $L'(u, 0)$ and $L'(v, 1)$ hold, where u and v are two distinct elements. Suppose also that in this instance there is an element t such that $L'(u, t)$ and $L'(v, t)$ hold. Consequently, u or v is guaranteed to have two distinct L'-neighbors (it is possible that both u and v have two distinct L'-neighbors). This will make it possible to simulate disjunction and then extract a truth assignment. It should be noted that variants of this construct were first used by van der Meyden [31].

Let **S** be the source schema consisting of a ternary relation symbol P, a ternary relation symbol A, and a binary relation symbol L. Intuitively, P will consist of all triples of variables occurring in clauses of a given 3CNF-formula, while A and L will be used to assign truth values to the variables of the formula. Let **T** be the target schema consisting of a ternary relation symbol P', a ternary relation symbol A', and a binary relation symbol L'. Let Σ_{st}

be the set of the following four source-to-target dependencies:

$$P(x, y, z) \to P'(x, y, z),$$
$$A(x, u, v) \to A'(x, u, v),$$
$$L(u, v) \to L'(u, v),$$
$$A(x, u, v) \to \exists t(L'(u, t) \wedge L'(v, t)).$$

Finally, let $q = q_1 \vee q_2$ be the union of the following two queries over the target schema $\mathbf{T}$:

$$q_1 :- (\exists x, u, v, t_1, t_2, t)(A'(x, u, v) \wedge L'(u, t_1) \wedge L'(v, t_2) \wedge L'(u, t) \wedge L'(v, t) \wedge (t \neq t_1) \wedge (t \neq t_2)),$$
$$q_2 :- (\exists x_1, x_2, x_3, u_1, v_1, u_2, v_2, u_3, v_3, t)(P'(x_1, x_2, x_3) \wedge \bigwedge_{i=1}^{3} (A'(x_i, u_i, v_i) \wedge L'(u_i, t) \wedge L'(v_i, t))).$$

Given a positive 3CNF-formula φ, let I_φ be the source instance defined as follows:

- The elements of I_φ are: 0, 1, all variables of φ, and for each variable x of φ, two distinct elements u_x and v_x (different such elements are used for different variables).
- The relations of I_φ are:

$$I_\varphi(P) = \{(x, y, z) : (x \vee y \vee z) \text{ is a clause of } \varphi\},$$
$$I_\varphi(A) = \{(x, u_x, v_x) : x \text{ is a variable of } \varphi\},$$
$$I_\varphi(L) = \{(u_x, 0), (v_x, 1) : x \text{ is a variable of } \varphi\}.$$

We now claim that φ is NOT-ALL-EQUAL satisfiable if and only if $certain(q, I_\varphi) = \underline{\text{false}}$. This means that we have to show that the following two statements are equivalent:

(1) There is a truth assignment such that, for every clause of φ, at least one variable is assigned value "true" and at least one variable is assigned value "false".

(2) There is a target instance J that is a solution to the data exchange problem for I_φ and is such that $q(J) = \underline{\text{false}}$.

Of the two directions in the claimed equivalence above, (2) $\Rightarrow$ (1) is the more interesting one. Suppose that J is a solution such that $q(J) = \underline{\text{false}}$, which means that $q_1(J) = \underline{\text{false}}$ and $q_2(J) = \underline{\text{false}}$. Since J satisfies the source-to-target dependencies in Σ_{st}, but fails to satisfy q_1, it follows that for every variable x, we have that $L'(u_x, 0)$ and $L'(u_x, 1)$ hold or that $L'(v_x, 0)$ and $L'(v_x, 1)$ hold (it is conceivable that both u_x and v_x have 0 and 1 as L'-neighbors). We now assign value $\underline{\text{true}}$ to a variable x if $L'(u_x, 0)$ and $L'(u_x, 1)$ hold. Using the fact that $q_2(J) = \underline{\text{false}}$, it is not hard to verify that, for each clause of φ, at least one variable is assigned value $\underline{\text{true}}$ and at least one variable is assigned value $\underline{\text{false}}$. □

5.1.3. A polynomial-time case

For unions of conjunctive queries with inequalities, Theorem 5.11 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 5.12. *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).

Proof. As in the proof of Theorem 5.2, we can assume without loss of generality that q is a Boolean query. We know that q is equivalent to a query of the form $q_1 \vee q_2$, where q_1 is the disjunction of a set C of conjunctive queries with no inequalities, and q_2 is the disjunction of conjunctive queries with exactly one inequality. As in the proof of Theorem 5.2, we note that the negation of q_2 is equivalent to the conjunction of a set E of disjunctive egds. However, differently from that proof, we use next the fact that q_2 has exactly one inequality per disjunct. Hence, it is easy to see that for each egd in E the number of equalities that participate in the disjunction is one. Therefore, E is a set of egds in the traditional sense (i.e., no disjunction).

We now describe the algorithm, and then show that it runs in polynomial time and is correct. The algorithm is based on the chase, as in the proof of Theorem 5.2. However, since there is no disjunction in E, the chase used is the traditional one (as defined in Section 3) and not the disjunctive chase used in the proof of Theorem 5.2.

The algorithm begins by chasing the universal solution J with $\Sigma_t \cup E$.

1. If the chase fails (by trying to equate two constants), then halt and say that *certain*$(q, I) =$ true.
2. If the chase does not fail, then call the result K. See if K satisfies at least one of the conjunctive queries in C.
   (a) If K satisfies at least one of the conjunctive queries in C, then halt and say that *certain*$(q, I) =$ true.
   (b) If K does not satisfy any of the conjunctive queries in C, then halt and say that *certain*$(q, I) =$ false.

Since Σ_{st} is a fixed set of tgds and Σ_t is the union of a weakly acyclic set of tgds with a set of egds, there is a polynomial-time algorithm for doing the chase (Theorem 3.9). Moreover, it is well known that for every first-order query (and in particular for every conjunctive query with inequalities), there is a polynomial-time algorithm (and even a logspace algorithm) for deciding satisfaction of the query on a given database. From these facts, it follows easily that the algorithm described above runs in polynomial time. We now show that the algorithm is correct.

Case 1: *The algorithm halts in step* 1. Since every solution is a homomorphic image of J and satisfies Σ_t, there is no solution that satisfies E. By definition of E, this tells us that *certain*$(q_2, I) =$ true, and hence *certain*$(q, I) =$ true.

Case 2: *The algorithm halts in step* 2(a). Since J is a universal solution, it is easy to see that K is a universal solution for targets that satisfy E (in addition to the requirements on Σ_{st} and Σ_t). Thus, every solution that satisfies E (that is, where q_2 fails) is a homomorphic image of K. Also, if K satisfies some conjunctive query in C, then so does every homomorphic image of K. Putting these facts together, we see that if K satisfies some conjunctive query in C, then so does every solution that satisfies E, that is, every solution where q_2 fails. So if K satisfies some conjunctive query in C, then every solution where q_2 fails satisfies some conjunctive query in C, and so satisfies q_1. Therefore, every solution satisfies either q_2 or q_1, and hence satisfies q. Hence, *certain*$(q, I) =$ true.

Case 3: *The algorithm halts in step* 2(b). As mentioned in Case 2, K is a universal solution for targets that satisfy E. In particular, K is a solution for the original data exchange problem (which does not include E). Since K does not satisfy any of the conjunctive queries in C, it does not satisfy q_1. On the other hand, K satisfies all of the egds in E, and hence does not satisfy q_2. Hence, K does not satisfy q. Since K is a solution, it follows that $certain(q, I) =$ false. □

Corollary 5.13. *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.*

Proof. We construct a two-phase algorithm. First, a canonical universal solution is constructed, by the chase, in polynomial time (see Corollary 3.10). Then we run, on this universal solution, the polynomial-time algorithm of Theorem 5.12, to compute the certain answers. □

5.2. First-order inexpressibility

We just showed that, for every conjunctive query with one inequality, the certain answers of the query can be evaluated in polynomial time. Here, we show that it is not possible to always obtain the certain answers by evaluating some first-order query on a canonical universal solution. Moreover, the certain answers may not be first-order definable over the source schema. The proof of these results combines Ehrenfeucht-Fraïssé games with the chase procedure.

Theorem 5.14. *There exists a LAV setting and a Boolean conjunctive query q with one inequality for which there is no first-order query q^* over the target schema such that, for every source instance I, there is some canonical universal solution J with $certain(q, I) = q^*(J)$.*

Proof. The source schema consists of a unary relation symbol M, and two binary relation symbols R and Q. The target schema consists of a unary relation symbol N and a binary relation symbol P. The set Σ_{st} of source-to-target dependencies consists of:

$$\begin{aligned} M(x) &\to N(x), \\ Q(x, y) &\to P(x, y), \\ R(x, y) &\to \exists z(P(x, z) \wedge P(z, y) \wedge N(z)). \end{aligned}$$

The set Σ_t of target dependencies is empty. The query q is:

$$\exists x \exists y \exists z(P(x, y) \wedge P(y, z) \wedge N(x) \wedge N(z) \wedge (x \neq z)).$$

We now define two source instances I_1 and I_2, both based on a positive integer parameter k that will be taken to be "sufficiently large" (explained later). Both I_1 and I_2 have the same domain, which consists of the $4k + 2$ distinct points (values) c, d, $e_1, \ldots, e_{2k}$, f_1,

$\ldots, f_{2k}$. In both I_1 and I_2, the unary relation corresponding to M contains the two points c and d. In both I_1 and I_2, the binary relation corresponding to R is the disjoint union of two cycles, each of size $2k$, where the first cycle contains the edges (tuples) (e_i, e_{i+1}) for $1 \leqslant i < 2k$, along with the edge (e_{2k}, e_1), and the second cycle contains the edges (f_i, f_{i+1}) for $1 \leqslant i < 2k$, along with the edge (f_{2k}, f_1). The only difference between I_1 and I_2 is that in I_1, the binary relation corresponding to Q contains the two tuples (e_1, c) and (e_k, d), whereas in I_2, the binary relation corresponding to Q contains the two tuples (e_1, c) and (f_1, d). Thus, in I_1, the points connecting to c and d are in the same cycle (but "far apart"), while in I_2, the points connecting to c and d are in different cycles. Thus, if we ignore the directions of the edges, then c and d are connected by a path in I_1, but not in I_2.

It is easy to see that up to isomorphism, there is a unique canonical universal solution J_1 for I_1 and a unique canonical universal solution J_2 for I_2. That is, the order in which we apply the chase steps does not matter. Furthermore, it is easy to see that in the canonical universal solution J_1 of I_1, in addition to the constants c, d, $e_1, \ldots, e_{2k}$, $f_1, \ldots, f_{2k}$, there are nulls $e'_1, \ldots, e'_{2k}$, $f'_1, \ldots, f'_{2k}$, such that the relation corresponding to P has the following tuples:

- (e_i, e'_i) for $1 \leqslant i \leqslant 2k$,
- (e'_i, e_{i+1}) for $1 \leqslant i < 2k$,
- (e'_{2k}, e_1),
- (f_i, f'_i) for $1 \leqslant i \leqslant 2k$,
- (f'_i, f_{i+1}) for $1 \leqslant i < 2k$,
- (f'_{2k}, f_1),
- (e_1, c),
- (e_k, d).

Intuitively, this relation consists of two cycles, each of size $4k$, along with two dangling edges that point to c and d, respectively, and that each hang off the first cycle and are far apart.

The relation corresponding to N in the canonical universal solution J_1 contains the points $c, d, e'_1, \ldots, e'_{2k}, f'_1, \ldots, f'_{2k}$. Thus, this relation contains c and d, along with the nulls.

Similarly, in the canonical universal solution J_2 of I_2, in addition to the constants $c, d, e_1, \ldots, e_{2k}, f_1, \ldots, f_{2k}$, there are nulls $e''_1, \ldots, e''_{2k}, f''_1, \ldots, f''_{2k}$, such that the relation corresponding to P has the following tuples:

- (e_i, e''_i) for $1 \leqslant i \leqslant 2k$,
- (e''_i, e_{i+1}) for $1 \leqslant i < 2k$,
- (e''_{2k}, e_1),
- (f_i, f''_i) for $1 \leqslant i \leqslant 2k$,
- (f''_i, f_{i+1}) for $1 \leqslant i < 2k$,
- (f''_{2k}, f_1),
- (e_1, c),
- (f_1, d).

Intuitively, this relation consists of two cycles, each of size $4k$, along with two dangling edges that point to c and d, respectively, where the two dangling edges hang off of different cycles.

The relation corresponding to N in the canonical universal solution J_2 contains the points $c, d, e''_1, \ldots, e''_{2k}, f''_1, \ldots, f''_{2k}$. Thus, this relation contains c and d, along with the nulls.

Let q^* be an arbitrary first-order query over the target schema. We now show that if k is sufficiently large, then $q^*(J_1) = q^*(J_2)$. We shall also show that $certain(q, I_1) = \underline{\text{true}}$ and $certain(q, I_2) = \underline{\text{false}}$. This shows that q^* does not play the role demanded of it in the statement of the theorem (namely, that $certain(q, I) = q^*(J)$). The theorem then follows.

We begin by showing that if k is sufficiently large, then $q^*(J_1) = q^*(J_2)$. This follows easily by making use of Ehrenfeucht-Fraïssé games, and in particular utilizing Hanf's technique [14].

We now show that $certain(q, I_1) = \underline{\text{true}}$. Note that $\neg q$ is equivalent to the egd

$$P(x, y) \wedge P(y, z) \wedge N(x) \wedge N(z) \rightarrow (x = z).$$

To show that $certain(q, I_1) = \underline{\text{true}}$, it is sufficient to show that if we chase J_1 with $\neg q$, the chase fails. This is because, as it is easy to see, the failure of the chase implies that no homomorphic image of J_1, and hence no solution, can satisfy $\neg q$.

In the chase, we first apply $\neg q$ to J_1 with the homomorphism h where $h(x) = e'_{2k}$, $h(y) = e_1$, and $h(z) = c$, and thereby replace e'_{2k} by c. We then apply $\neg q$ with the homomorphism h where $h(x) = c$ (which e'_{2k} has been replaced by), $h(y) = e_1$, and $h(z) = e'_1$, and therefore replace e'_1 by c. We then apply $\neg q$ with the homomorphism h where $h(x) = c$ (which e'_1 has been replaced by), $h(y) = e_2$, and $h(z) = e'_2$, and therefore replace e'_2 by c. Continuing in this manner, we replace $e'_3, e'_4, \ldots, e'_{k-1}$ by c. Finally, we apply $\neg q$ with the homomorphism h where $h(x) = c$ (which e'_{k-1} has been replaced by), $h(y) = e_k$, and $h(z) = d$, and try to replace d by c, which leads to failure, as desired.

We close by showing that $certain(q, I_2) = \underline{\text{false}}$. It is sufficient to show that if we chase J_2 with $\neg q$, the chase does not fail. Indeed, the result K_2 of such a chase continues to satisfy Σ_{st}, and hence it is a solution. Furthermore, K_2 satisfies $\neg q$, that is, $q(K_2) = \underline{\text{false}}$.

It is straightforward to verify that the chase of J_2 with $\neg q$ does not fail and its result, K_2, is as follows. The relation corresponding to P has the following tuples:

- (e_i, c) for $1 \leqslant i \leqslant 2k$,
- (c, e_i) for $1 \leqslant i \leqslant 2k$,
- (f_i, d) for $1 \leqslant i \leqslant 2k$,
- (d, f_i) for $1 \leqslant i \leqslant 2k$,
- (e_1, c),
- (f_1, d).

The relation corresponding to N contains only c and d. This concludes the proof. □

It follows from the above proof that the result holds even if we allow the first-order formula q^* to contain the predicate const that distinguishes between constants and nulls.

The next result, of particular interest to query answering in the data integration context, shows (by a slight modification of the proof of Theorem 5.14) 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.

Theorem 5.15. *There is a LAV setting and a Boolean conjunctive query q with one inequality, for which there is no first-order query q^* over the source schema such that $certain(q, I) = q^*(I)$ for every source instance I.*

Proof. Take the LAV setting, Boolean conjunctive query q, and source instances I_1 and I_2 exactly as in the proof of Theorem 5.14. It is shown in that proof that $certain(q, I_1) = \underline{\text{true}}$ and $certain(q, I_2) = \underline{\text{false}}$. Let q^* be an arbitrary first-order query over the source schema. If k is sufficiently large, then q^* cannot distinguish between I_1 and I_2, that is, $q^*(I_1) = q^*(I_2)$. This follows for the same reason that any given first-order query over the target schema cannot distinguish between J_1 and J_2 in the proof of Theorem 5.14 if k is sufficiently large. In both cases, this indistinguishability follows easily by making use of Ehrenfeucht-Fraïssé games, and in particular utilizing Hanf's technique [14]. This shows that q^* does not play the role demanded of it in the statement of the theorem (namely, that $certain(q, I) = q^*(I)$). The theorem then follows. □

6. Concluding remarks

Given a source instance, there may be many universal solutions. This naturally brings up the question of whether there is a "best" universal solution, and hence a best solution for data exchange. In a follow-up paper [16], we address this question and answer it by considering the well-known notion of the *core* of a structure, a notion that was first studied in graph theory (see, for instance, [19]), but has also played a role in conjunctive-query processing [8].

In Theorem 5.14, we show 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 a canonical universal solution. But this leads to the question of whether some other solution other than a canonical universal solution would have done the job. That is, is there a transformation $\mathcal{F}$ that maps each source instance I into a solution $\mathcal{F}(I)$ and a first-order rewriting q^* such that the certain answers are given by $q^*(\mathcal{F}(I))$? This question is investigated in [4], where it is shown that as long as $\mathcal{F}$ is "locally consistent" (which means intuitively that points with similar neighborhoods in the source have similar neighborhoods in the target), then there are first-order queries q with no such rewriting q^*. It is also shown in [4] that in appropriate data exchange settings, the mappings $\mathcal{F}$ that map onto the canonical universal solution or onto the core are locally consistent. Therefore, the results in [4] provide an extension of our Theorem 5.14. We feel that there is a need for further investigation of how universal solutions can be used for query answering in the data exchange setting.

Finally, we wish to go back to our original motivation from Clio, an XML-based schema mapping tool. The results we presented here are about data exchange between relational schemas. We would like to study data exchange between XML schemas and, in particular, investigate how the notion of universal solution can be extended to cover XML schemas.

Acknowledgements

We would like to thank Arnon Rosenthal, Val Tannen and Moshe Y. Vardi for helpful suggestions.

References

[1] S. Abiteboul, O.M. Duschka, Complexity of answering queries using materialized views, in: Proc. of the ACM Symp. on Principles of Database Systems (PODS), 1998, pp. 254–263.

[2] S. Abiteboul, O.M. Duschka, Complexity of answering queries using materialized views, 2000 (unpublished full version of [1]).

[3] S. Abiteboul, S. Cluet, T. Milo, Correspondence and translation for heterogeneous data, in: Proc. of the Internat. Conf. on Database Theory (ICDT), 1997, pp. 351–363.

[4] M. Arenas, P. Barceló, R. Fagin, L. Libkin, Locally consistent transformations and query answering in data exchange, in: Proc. of the ACM Symp. on Principles of Database Systems (PODS), 2004, pp. 229–240.

[5] C. Beeri, M.Y. Vardi, A proof procedure for data dependencies, J. Assoc. Comput. Mach. 31 (4) (1984) 718–741.

[6] A. Calı`, D. Calvanese, G. De Giacomo, M. Lenzerini, Data integration under integrity constraints, in: Proc. of the Internat. Conf. on Advanced Information Systems Engineering (CAiSE), 2002, pp. 262–279.

[7] M.A. Casanova, R. Fagin, C.H. Papadimitriou, Inclusion dependencies and their interaction with functional dependencies, J. Comput. System Sci. 28 (1) (1984) 29–59.

[8] A.K. Chandra, P.M. Merlin, Optimal implementation of conjunctive queries in relational data bases, in: Proc. of the ACM Symp. on Theory of Computing (STOC), 1977, pp. 77–90.

[9] S.S. Cosmadakis, P.C. Kanellakis, Functional and inclusion dependencies: a graph theoretic approach, in: P.C. Kanellakis, F.P. Preparata (Eds.), Advances in Computing Research, Vol. 3, 1986, pp. 163–184.

[10] A. Deutsch, V. Tannen, Optimization properties of conjunctive regular path queries, in: Internat. Workshop on Database Programming Languages, 2001, pp. 21–39.

[11] A. Deutsch, V. Tannen, Reformulation of XML queries and constraints, in: Proc. of the Internat. Conf. on Database Theory (ICDT), 2003, pp. 225–241.

[12] O.M. Duschka, M.R. Genesereth, A.Y. Levy, Recursive query plans for data integration, J. Logic Programming 43 (1) (2000) 49–73.

[13] R. Fagin, Horn clauses and database dependencies, J. Assoc. Comput. Mach. 29 (4) (1982) 952–985.

[14] R. Fagin, L. Stockmeyer, M.Y. Vardi, On monadic NP vs. monadic co-NP, Inform. Comput. 120 (1) (1995) 78–92.

[15] R. Fagin, Ph.G. Kolaitis, R.J. Miller, L. Popa, Data exchange: semantics and query answering, in: Proc. of the Internat. Conf. on Database Theory (ICDT), 2003, pp. 207–224.

[16] R. Fagin, Ph.G. Kolaitis, L. Popa, Data exchange: getting to the core, in: Proc. of the ACM Symp. on Principles of Database Systems (PODS), 2003, pp. 90–101.

[17] M. Friedman, A.Y. Levy, T.D. Millstein, Navigational plans for data integration, in: Proc. of the National Conf. on Artificial Intelligence (AAAI), 1999, pp. 67–73.

[18] A. Halevy, Answering queries using views: a survey, VLDB J. (2001) 270–294.

[19] P. Hell, J. Nešetřil, The core of a graph, Discrete Math. 109 (1992) 117–126.

[20] R. Hull, M. Yoshikawa, ILOG: declarative creation and manipulation of object identifiers, in: Proc. of the Internat. Conf. on Very Large Data Bases (VLDB), 1990, pp. 455–468.

[21] M. Lenzerini, Data integration: a theoretical perspective, in: Proc. of the ACM Symp. on Principles of Database Systems (PODS), 2002, pp. 233–246.

[22] A.Y. Levy, A.O. Mendelzon, Y. Sagiv, D. Srivastava, Answering queries using views, in: Proc. of the ACM Symp. on Principles of Database Systems (PODS), May 1995, pp. 95–104.

[23] D. Maier, A.O. Mendelzon, Y. Sagiv, Testing implications of data dependencies, ACM Trans. Database Systems 4 (4) (1979) 455–469.

[24] D. Maier, J.D. Ullman, M.Y. Vardi, On the foundations of the universal relation model, ACM Trans. Database Systems 9 (2) (1984) 283–308.

[25] J.A. Makowsky, Why Horn formulas matter in computer science: initial structures and generic examples, J. Comput. System Sci. 34 (2/3) (1987) 266–292.

[26] R.J. Miller, L.M. Haas, M. Hernández, Schema mapping as query discovery, in: Proc. of the Internat. Conf. on Very Large Data Bases (VLDB), 2000, pp. 77–88.

[27] L. Popa, Y. Velegrakis, R.J. Miller, M.A. Hernandez, R. Fagin, Translating web data, in: Proc. of the Internat. Conf. on Very Large Data Bases (VLDB), 2002, pp. 598–609.

[28] T.J. Schaefer, The complexity of satisfiability problems, in: Proc. of the ACM Symp. on Theory of Computing (STOC), 1978, pp. 216–226.
[29] N.C. Shu, B.C. Housel, V.Y. Lum, CONVERT: a high level translation definition language for data conversion, Comm. ACM 18 (10) (1975) 557–567.
[30] N.C. Shu, B.C. Housel, R.W. Taylor, S.P. Ghosh, V.Y. Lum, EXPRESS: a data extraction, processing, and restructuring system, ACM Trans. Database Systems 2 (2) (1977) 134–174.
[31] R. van der Meyden, The complexity of querying indefinite data about linearly ordered domains, J. Comput. System Sci. 54 (1997) 113–135.
[32] R. van der Meyden, Logical approaches to incomplete information: a survey, in: J. Chomicki, G. Saake (Eds.), Logics for Databases and Information Systems, Kluwer, Dordrecht, 1998, pp. 307–356.
[33] M.Y. Vardi, The complexity of relational query languages, in: Proc. of the ACM Symp. on Theory of Computing (STOC), 1982, pp. 137–146.