Data Exchange in the Presence of Arithmetic Comparisons[*]

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

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

Vassia Pavlaki [‡]
National Technical University of Athens (NTUA)
vpavlaki@softlab.ntua.gr

ABSTRACT

Data exchange is the problem of transforming data structured under a schema (called source) into data structured under a different schema (called target). The emphasis of data exchange is to materialize a target instance (called solution) that satisfies the relationship between the schemas. Universal solutions were shown to be the most suitable solutions, mainly because they can be used to answer conjunctive queries posed over the target schema. Trying to extend this result to more expressive query languages fails, even if we only add inequalities ($\neq$) to conjunctive queries.

In this work we study data exchange in the presence of general arithmetic comparisons ($<, \leq, >, \geq, =, \neq$): (a) We consider queries posed over the target schema that belong to the class of unions of conjunctive queries with arithmetic comparisons (in short CQACs). (b) We exploit arithmetic comparisons to define more expressive data exchange settings, called DEAC settings. In particular, DEAC settings consist of constraints that involve arithmetic comparisons. For that, two new classes of dependencies (tgd-ACs and acgds) are introduced, to capture the need of arithmetic comparisons in source-to-target and target constraints.

We show that in DEAC settings the existence of solution problem is in NP. We define a novel chase procedure called AC-chase which is a tree and we prove that it produces a universal solution (appropriately defined to deal with arithmetic comparisons). We show that the new concept of universal solution is the right tool for query answering in the case of unions of CQACs. The complexity of computing certain answers for unions of CQACs is shown to be coNP-complete. Moreover, we identify polynomial cases for a) computing a universal solution and b) computing certain answers. For that, we introduce the succinct AC-chase which is a sequence instead of a tree, but its result is not necessarily a solution. We identify cases where succinct AC-chase returns indeed a universal solution and we investigate the syntactic conditions of the query under which query answering takes polynomial time. We show that the latter is feasible even in cases where the result of chase is not a universal solution.

[*]Work partially done when all authors were visiting Stanford University and when the third author was visiting the University of California, Irvine.

[†]Chen Li was partially supported by the NSF CAREER Award No. IIS-0238586

[‡]The project is co - funded by the European Social Fund (75%) and National Resources (25%) - Operational Program for Educational and Vocational Training II (EPEAEK II) and particularly the Program HERAKLEITOS.

1. INTRODUCTION

Data exchange is the problem of transforming data structured under a schema, called the *source schema* into data structured under another schema, called the *target schema*. The goal in data exchange is to actually *materialize* a target instance (called *solution*) that satisfies the relationship between the schemas and *answer queries* posed over the target schema using the materialized instance. The theoretical foundations of data exchange were set in the two seminal papers [14, 15] where a number of research issues were presented. Given a source instance, multiple solutions may exist. So which solution should we choose to materialize? How can we compute such a solution? For queries posed over the target schema, what semantics should we adopt for query answering and how can we evaluate the queries?

In [14] *universal solutions* (i.e., solutions that homomorphically map to every solution), were shown to be the most suitable solutions to be materialized. In particular, it was shown that, if we adopt the notion of *certain answers* as the semantics for query answering, universal solutions can be used to answer unions of conjunctive queries. However, this result cannot be extended to more expressive query languages. Indeed, certain answers of conjunctive queries with inequalities ($\neq$) cannot be obtained by evaluating them on a universal solution [14, 4, 23].

In this work we study data exchange in the presence of general arithmetic comparisons ($<, \leq, >, \geq, =, \neq$):

- We consider queries, posed over the target schema, that belong to the class of unions of *conjunctive queries with arithmetic comparisons* (in short CQACs), which is a richer class compared to the class of unions of conjunctive queries with inequalities ($\neq$).

- We exploit arithmetic comparisons to define more expressive data exchange settings, called *data exchange with arithmetic comparisons* (in short *DEAC*). In particular, we define the class of *tuple generating dependencies with arithmetic comparisons* (in short *tgd-ACs*), which extends the class of tuple generating dependencies (tgds) by including arithmetic comparisons in both the left and the right hand sides of the tgd. Similarly, we extend the class of equality generating dependencies (egds) and we call the new class *arithmetic comparison generating dependencies* (in short *acgds*).

The need for enriching data dependencies with arithmetic comparisons has been noticed in the litarature [6, 21, 30, 29]. The following example shows the necessity for introducing more expressive data exchange settings by incorporating arithmetic comparisons in the dependencies.

EXAMPLE 1.1. *A real estate company* A *has a database* S *that stores information on apartments in Orange County, California. Figure 1 shows the relation* `Home` *in database* S.

`Home:`

aID	apartAddr	price
13	82 Main St	450
54	12 Alton Ave	300
55	14 Alton Ave	800

Figure 1: Source instance at company A.

Due to business reasons, the company A *needs to send some data from database* S (*called source database*) *to a company* B*, which uses a different database schema, i.e., it has a database* T (*called target database*) *with two relations:*

`Apartment(aID, apartAddr, price, zip, name)`*;*
`Person(name, Addr)`.

The table `Apartment` *contains data on apartments such as ID* (`aID`)*, address* (`apartAddr`)*, price* (`price`)*, zip code* (`zip`)*, owner's name* (`name`)*. The table* `Person` *stores information concerning the name* (`name`) *and address* (`Addr`) *of owners.*

The following tgd d_{st_1} *expresses the fact that for every tuple* $\texttt{Home(I, A, P)}$ *in database* S*, we add to database* T *a tuple* $\texttt{Apartment(I, A, P, Z, N)}$.

$$d_{st_1} : \texttt{Home(I, A, P)} \rightarrow \texttt{Apartment(I, A, P, Z, N)}.$$

Now, suppose that we only want to transfer data about apartments with price less or equal to 500 (*thousand dollars*)*. Tgds cannot express this information because they cannot express the fact that only a subset of the tuples in the source database satisfying a specific condition, should be transferred. We need more expressive dependencies that would allow for arithmetic comparisons, such as* $\texttt{P} \leq \texttt{500}$*, in the antecedent.*

Moreover, there is a second type of requirement that tgds cannot capture either. Suppose we know that all apartments in database S *are in Orange County* (*translated as the zip code of the apartments being greater than* 92000)*. Notice that the attribute* `zip` *appears only in the database* T *of company* B*. So how can we store this information in the database* T*? Again, tgds cannot express this prior knowledge. Adding arithmetic comparisons, such as* $\texttt{Z} > \texttt{92000}$*, in the consequent of* d_{st_1} *would allow to record this additional information. The following tuple generating dependency with arithmetic comparisons* (*tgd-AC*) d_{st_2} *fulfills both requirements:*

$$\texttt{Home(I, A, P)}, \texttt{P} \leq \texttt{500} \rightarrow \texttt{Apartment(I, A, P, Z, N)}, \texttt{Z} > \texttt{92000}.$$

The information provided by the arithmetic comparisons in the consequent of a tgd-AC can be useful also in query answering. The following example explains this point.

`Apartment:`

aID	apartAddr	price	zip	name
13	82 Main St	450	x_1	x_3
54	12 Alton Ave	300	x_2	x_4

Figure 2: Target instance at company B.

EXAMPLE 1.2. (*Continued from Example 1.1*)*. Figure 2 shows an instance* J_1 *of database* T *after the transfer of data from database* S *according to the dependency* d_{st_2}.
Recall that the data we transfer concern apartments costing ≤ 500 (*thousand dollars*)*. Note that there are no values for the attribute* `zip` *and from the information stored in database* S *we can not associate owners with the apartments they own. Therefore, the attributes* `zip` *and* `name` *take* null *values in database* T *and appear as* labeled nulls x_1, x_2, x_3, x_4 *in* J_1.

Consider now the query Q (*posed over the target schema*)

$$\texttt{Q} : \ \texttt{q(A)} : \texttt{Apartment(I, A, P, Z, N)}, \texttt{Z} > \texttt{92000},$$

asking for addresses of apartments in the Orange County area (*again translated as the zip code of the apartments being greater than* 92000)*. If we evaluate* Q *on* J_1 *then the result would be empty. However, we know that the apartments in the database* S *of company* A *are all in Orange County. Obviously, this holds also for the apartments that were transferred to the database* T *of company* B*. Thus, if we accompany* J_1 *with* $C = \{x_1 > 92000, x_2 > 92000\}$ (*which is the information implied by the tgd-AC* d_{st_2}) *then, applying* Q *to* $\{J_1, C\}$ *returns the non-empty answer* {*82 Main St, 12 Alton Ave*}*, which is the desired answer.*

Example 1.2 showed that the presence of arithmetic comparisons in target instances as a result of data transfer from a source schema to a different target schema, enriches the amount of the information stored in the target database, in that the set of answers to a query can be more informative.

Besides tgd-ACs being useful in transferring data from a source database to a target database, they can also be used as integrity constraints, to allow database designers to express richer constraints. Example 1.3 shows how arithmetic comparisons may enrich target constraints (i.e., constraints imposed on the target instance for it to be consistent with the requirements of the system in company B).

EXAMPLE 1.3. (*Continued from Example 1.1*)*. Consider the following tgd-AC on the database* T:

$$d_t : \texttt{Apartment(I, A}_1\texttt{, P, Z, N)}, \texttt{P} > \texttt{300} \rightarrow \texttt{Person(N, A}_2\texttt{)}.$$

d_t *expresses the fact that for each apartment with price higher than* 300 (*thousand dollars*)*, there should be a record for its owner stored in the table* `Person`.

Formally, a DEAC setting is a quadruple $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ consisting of a source schema $\mathbf{S}$, a target schema $\mathbf{T}$, a set Σ_{st} of *source-to-target* (i.e., constraints specifying the relation between source and target schemas) tgd-ACs, and a set Σ_t of *target constraints* (i.e., constrains used in the design of the target schema) consisting of tgd-ACs and acgds.

A tgd-AC is a first-order formula of the form $\forall \mathbf{x}\big(\phi(\mathbf{x}) \wedge \beta_\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y}) \wedge \beta_\psi(\mathbf{x}, \mathbf{y})\big)$, whereas an acgd is a first-order formula of the form $\forall \mathbf{x}\big(\phi(\mathbf{x}) \wedge \beta_\phi(\mathbf{x}) \rightarrow (x_1\ \theta\ x_2)\big)$, with x_1 being a variable from $\mathbf{x}$ and x_2 being either a variable from $\mathbf{x}$ or a constant. In the above formulas, $\beta_\phi(\mathbf{x})$ and

$\beta_\psi(\mathbf{x},\mathbf{y})$ are each a conjunction of comparison predicates (with variables from $\phi(\mathbf{x})$ and $\psi(\mathbf{x},\mathbf{y})$ respectively) of the form "$A\theta B$", where A is a variable, B is a variable or a constant, and θ is in $\{\leq,\geq,<,>,\neq,=\}$.

It is well known that reasoning on logical formulas with arithmetic comparisons is more complicated than logical formulas without them. Indeed, many of the technical tools used to solve data exchange problems are closely related to query containment checking. Conjunctive query containment is recognized as a much simpler problem (NP-complete [11]) than containment of conjunctive queries with arithmetic comparisons (Π_2^P-complete [22, 32]). Thus, in DEAC settings we have to deal with additional complications that are introduced by the presence of arithmetic comparisons.

Contributions and Outline: In Section 2, we define the classes of tgd-ACs and acgds, as well as the DEAC setting.

In Section 3, we study the *existence of solution* problem (i.e., the following decision problem: consider a fixed DEAC setting $\mathbf{M}$; given a source instance I, does a solution J for I under $\mathbf{M}$ exist?). We start with defining the new concepts of *solution* and *universal solution* in a DEAC setting. As opposed to data exchange settings without arithmetic comparisons where the universal solution is a singleton, in DEAC settings a universal solution consists of a set of solutions. Interestingly, also in DEAC settings every two universal solutions are equivalent as Boolean queries (Proposition 3.1) which is an extension of a property noticed in [14] and demonstrates the robustness of the concept. We continue with introducing a novel chase procedure called *AC-chase* suitable to deal with tgd-ACs and acgds. AC-chase is a tree rather than a sequence as the majority of existing chase procedures. We identify the conditions under which AC-chase terminates and produces a universal solution (Theorem 3.1). We call the subclass of tgd-ACs that guarantee the termination of AC-chase *weakly acyclic tgd-ACs*. The class of weakly acyclic tgd-ACs is a generalization of the class of *weakly acyclic tgds* [14] (also known as constraints with *stratified witness* [12]). Interestingly, the extension, although not trivial, is simple. We show that for a set of weakly acyclic tgd-ACs and acgds an AC-chase always produces a tree of polynomial depth (on the size of the input source instance). From that, we prove that the existence of solution problem is in NP (Theorem 3.2). In all complexity results presented in this paper we consider data complexity (i.e., the data exchange setting is fixed and the complexity is measured on the size of the source instance).

In Section 4, we study the *query answering* problem for unions of *conjunctive queries with arithmetic comparisons* (in short CQACs) posed over the target schema. For the semantics of query answering we adopt the notion of *certain answers*. We show that the concept of universal solution as defined in Section 3 is the right tool for computing the certain answers of unions of CQAC queries (Theorem 4.1). Notice that universal solutions, besides constants, may have also labeled nulls. Computing conjunctive queries with arithmetic comparisons on database instances with labeled nulls is more complicated than computing conjunctive queries (without arithmetic comparisons) on such instances. Thus, we need subtler definitions and techniques. We prove that given a source instance I and a tuple t, the problem "is t in $certain_M(q,I)$?" is co-NP complete (Theorem 4.2). Although the problem remains intractable, the complexity is not greater than the significantly simpler problem of query answering for data exchange settings without arithmetic comparisons and for conjunctive queries containing solely inequalities ($\neq$) in their body.

In Section 5, we identify cases for which we can develop polynomial results. To prove the majority of the polynomial results we introduce a second chase procedure called *succinct AC-chase* (in short SAC-chase). On the positive side, the SAC-chase is a sequence instead of a tree as it is the case with AC-chase. On the negative side, unlike the AC-chase, the result of SAC-chase is not necessarily a universal solution for weakly acyclic tgd-ACs. We identify cases where succinct AC-chase returns a universal solution and we investigate the syntactic conditions of the query under which query answering takes polynomial time. We show that the latter is feasible even in cases where the result of chase is not a universal solution. In particular, we show that when there are no target constraints then, a universal solution in a DEAC setting can be computed in polynomial time (Theorem 5.1). However, computing certain answers of a CQAC query in this case is polynomial only if the query has a property called *homomorphism property* [22][1] (Theorem 5.2(a)). Moreover, we show that for full tgd-ACs (i.e., tgd-ACs without existentially quantified variables) the complexity of computing a universal solution and of computing certain answers of CQAC queries is polynomial (Theorem 5.1 and Theorem 5.2(b) respectively). Finally, for data exchange settings where dependencies do *not* have arithmetic comparisons, we identify the second class of conjunctive queries with arithmetic comparisons (the first such class was identified in [14]) for which certain answers can be computed in polynomial time (Theorem 5.3).

Table 1 summarizes complexity results on query answering for conjunctive queries with general arithmetic comparisons ($<,\leq,>,\geq,=,\neq$) or just inequalities ($\neq$). "CQ" is used to denote conjunctive queries, "UCQ" union of conjunctive queries and "(U)CQAC" (union of) conjunctive query (queries) with arithmetic comparisons. By "LAV" we mean settings where tgds have a single atom in the antecedent and "w.a." is used to denote weakly acyclic tgds or tgd-ACs.

2. DATA EXCHANGE WITH ARITHMETIC COMPARISONS

In this section we define the two new classes of data dependencies; tgd-ACs and acgds, we introduce the data exchange setting with arithmetic comparisons (in short DEAC) and we review the data exchange problem.

An *arithmetic comparison* predicate is of the form "$A\theta B$", where A is a variable, B is a variable or a constant, and θ is in $\{\leq,\geq,<,>,\neq,=\}$. The arithmetic comparisons considered in this work are interpreted over densely totally ordered domains[2]. We call an arithmetic comparison *open* if its operator is $<$ or $>$, and *closed* if its operator is $\leq$ or $\geq$. We call

[1]The usefulness of the homomorphism property is in reducing the complexity of the query containment problem in the presence of arithmetic comparisons [22, 2]. It contains large classes of queries such as queries which use only comparisons of the form $X < c$ where X is a variable and c is a constant.
[2]See example in Conclusions which illustrates the difference if arithmetic comparisons are interpreted over *discrete* to-

Constraints (Σ_{st})	**Constraints (Σ_t)**	**Query Language**	**Complexity**	**Reference**
tgds (LAV)	$\emptyset$	CQ with seven $\neq$	coNP-complete	[1]
tgds (LAV)	$\emptyset$	CQ with two $\neq$	coNP-complete	[28]
tgds	w.a. tgds and egds	UCQ with inequalities ($\neq$)	coNP-complete	[14]
tgds	$\emptyset$	Union of two CQs each with at most two $\neq$ per CQ	coNP-complete	[14]
tgd-ACs	w.a. tgd-ACs and acgds	UCQAC	coNP-complete	Section 4
tgds	w.a. tgds and egds	UCQ, at most one $\neq$ per CQ	PTIME	[14]
full tgd-ACs	full tgd-ACs and acgds	UCQAC	PTIME	Section 5
tgd-ACs	$\emptyset$	CQAC that has the homomorphism property	PTIME	Section 5
tgds	w.a. tgds and egds	CQAC that has only open (closed) LSI (RSI) comparisons	PTIME	Section 5

Table 1: Complexity results on query answering in data exchange settings.

an arithmetic comparison *left semi-interval* (in short LSI) if it is of the form $A < c$ or $A \leq c$, where A is a variable, and c is a constant. In a similar way, we call an arithmetic comparison *right semi-interval* (in short RSI) if it is of the form $A > c$ or $A \geq c$. An arithmetic comparison is called *semi-interval* if it is either an LSI or an RSI comparison.

A *conjunctive formula with arithmetic comparisons* is denoted $\mathtt{F}(\mathbf{x}) = \mathtt{F}_0(\mathbf{x}) \wedge \beta_{\mathtt{F}}(\mathbf{x})$, where $\mathtt{F}_0(\mathbf{x})$ is a conjunction of relational atoms and $\beta_{\mathtt{F}}(\mathbf{x})$ is a conjunction of arithmetic comparisons. A conjunctive formula with arithmetic comparisons is in *compact form* if the conjunction of its arithmetic comparisons $\beta_{\mathtt{F}}$ do not imply equalities. A conjunctive formula with arithmetic comparisons can be transformed into an equivalent formula in compact form in polynomial time [2]. Let $\mathtt{F} = \{\mathtt{H}(2, \mathtt{z}_1), \mathtt{H}(4, \mathtt{z}_2), \mathtt{z}_1 = \mathtt{z}_2 < 2\}$ be a conjunctive formula with arithmetic comparisons. Obviously, $\mathtt{F}$ is not compact because of the equality $\mathtt{z}_1 = \mathtt{z}_2$. A compact formula obtained from $\mathtt{F}$ is $\mathtt{F}' = \{\mathtt{H}(2, \mathtt{z}_1), \mathtt{H}(4, \mathtt{z}_1), \mathtt{z}_1 < 2\}$.

We continue with introducing the class of *tuple generating dependencies with arithmetic comparisons* (in short tgd-ACs), which extends the class of tuple-generating dependencies (tgds) and the class of *arithmetic comparison generating dependencies* (in short acgds), which extends the class of equality generating dependencies (egds). Since tgds and egds together have the same expressive power as the class of embedded implicational dependencies (eids) [13], clearly tgd-ACs and acgds together form a richer class than eids.

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

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

• *An* arithmetic comparison generating dependency (*in short* acgd) *is a first-order formula of the form*

$$\forall \mathbf{x}\big(\phi(\mathbf{x}) \wedge \beta_\phi(\mathbf{x}) \rightarrow (x_1\ \theta\ x_2)\big).$$

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

tally ordered domains.

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

Let $d : \forall \mathbf{x}\big(\phi(\mathbf{x}) \wedge \beta_\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y}) \wedge \beta_\psi(\mathbf{x}, \mathbf{y})\big)$ be a tgd-AC and D a database instance. We say that D *satisfies* d if whenever there is a homomorphism h from $\phi(\mathbf{x})$ to D such that $\beta_\phi(h(\mathbf{x}))$ is true, there exists an extension h' of h, where h' is a homomorphism from the conjunction $\phi(\mathbf{x}) \wedge \psi(\mathbf{x}, \mathbf{y})$ to D such that $\beta_\psi(h'(\mathbf{x}, \mathbf{y}))$ is true. For tgds and egds, there is always a non-empty database on which the dependency is satisfied. This property does not hold for tgd-ACs and acgds. We call a tgd-AC (or an acgd) d *consistent* if there exists a non-empty database instance on which d is satisfied. Checking consistency can be done in polynomial time [22].

Let $\mathbf{S} = \{S_1, \ldots, S_n\}$ and $\mathbf{T} = \{T_1, \ldots, T_m\}$ be two disjoint schemas, where each element in the sets is a relation. A *source-to-target dependency* is a tgd-AC of the form $\forall \mathbf{x}\big(\phi_{\mathbf{S}}(\mathbf{x}) \wedge \beta_\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y}) \wedge \beta_\psi(\mathbf{x}, \mathbf{y})\big)$. A *target dependency* is either a tgd-AC of the form $\forall \mathbf{x}\big(\phi_{\mathbf{T}}(\mathbf{x}) \wedge \beta_\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y}) \wedge \beta_\psi(\mathbf{x}, \mathbf{y})\big)$, or an acgd of the form $\forall \mathbf{x}\big(\phi_{\mathbf{T}}(\mathbf{x}) \wedge \beta_\phi(\mathbf{x}) \rightarrow (x_1 \theta x_2)\big)$. In all dependencies, $\phi_{\mathbf{S}}(\mathbf{x})$, $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$, and $\phi_{\mathbf{T}}(\mathbf{x})$ are each a conjunction of atomic relational formulas over $\mathbf{S}$ or $\mathbf{T}$ respectively, and β_ϕ and β_ψ are conjunctions of arithmetic comparison predicates.

Definition 2.2. (*DEAC setting*)
A data exchange setting with arithmetic comparisons (in short DEAC) is a quadruple $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ consisting of a source schema $\mathbf{S}$, a target schema $\mathbf{T}$, a set Σ_{st} of source-to-target tgd-ACs, and a set Σ_t of target tgd-ACs and acgds.

Recall that in data exchange settings without arithmetic comparisons [14], the set of source-to-target constraints does not contain egds because this would amount to assuming also source constraints in the data exchange setting. Similarly, in DEAC settings, source-to-target constraints consist solely of tgd-ACs (and not acgds). However, because of the fact that a tgd-AC d contains arithmetic comparisons in its rhs, we need to impose additional restrictions for the case where d is a source-to-target constraint. More precisely, we require that the arithmetic comparisons in the rhs of a (source-to-target only) tgd-AC must contain at least one existentially quantified variable. In the case of target tgd-ACs we may release the above restriction. However, a target tgd-AC with the restriction released is *equivalent*, (in the sense that it is satisfied on exactly the same set of database instances), to a set consisting of acgds and tgd-ACs with the latter satisfying the restriction (see Example 2.1). Hence, in DEAC settings we consider tgd-ACs (both source-to-target and target) with the restriction that each arithmetic comparison in the rhs must contain at least one existentially quantified variable.

EXAMPLE 2.1. *Consider the following target tgd-AC:*

$$\mathtt{d} : \mathtt{a}(\mathtt{X},\mathtt{Y}), \mathtt{b}(\mathtt{Y},\mathtt{W}) \rightarrow \mathtt{c}(\mathtt{X},\mathtt{Y}), \mathtt{X} < \mathtt{Y}.$$

The arithmetic comparison $\mathtt{X} < \mathtt{Y}$ *does not contain an existentially quantified variable.* $\mathtt{d}$ *is equivalent to a set of dependencies consisting of a tgd-AC* $\mathtt{d}_1$ *and an acgd* $\mathtt{d}_2$*:*

$$\mathtt{d}_1 : \mathtt{a}(\mathtt{X},\mathtt{Y}), \mathtt{b}(\mathtt{Y},\mathtt{W}) \rightarrow \mathtt{c}(\mathtt{X},\mathtt{Y}),$$
$$\mathtt{d}_2 : \mathtt{a}(\mathtt{X},\mathtt{Y}), \mathtt{b}(\mathtt{Y},\mathtt{W}) \rightarrow \mathtt{X} < \mathtt{Y}.$$

Observe that $\mathtt{d}_1$ *does not have an arithmetic comparison in its rhs and consequently it trivially satisfies the restriction.*

The data exchange problem is the following: consider a fixed DEAC setting determined by a source schema $\mathbf{S}$, a target schema $\mathbf{T}$ and a set of source-to-target constraints Σ_{st} and target constraints Σ_t. Given an instance I over $\mathbf{S}$, we want to materialize an instance J over $\mathbf{T}$ (called *solution*) such as the target dependencies Σ_t are satisfied by J and the source-to-target dependencies Σ_{st} are satisfied by I and J together. Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a DEAC setting. The *existence of solution problem* is the following decision problem: given a source instance I, does a solution J for I under $\mathbf{M}$ exist?

3. SOLUTION AND UNIVERSAL SOLUTION

In this section we introduce the concepts of *solution* and *universal solution* in a DEAC setting. We prove the robustness of the new concepts showing that every two universal solutions in a DEAC setting are equivalent if viewed as queries. We continue with defining a chase procedure, called *AC-chase* and we identify the conditions under which AC-chase terminates and guarantees the existence of a universal solution. Finally, we investigate the complexity of the existence of solution problem and prove that it is in NP.

3.1 Basic Definitions

We assume a domain of constants $\underline{\mathtt{Const}}$ which is an infinite densely totally ordered domain such as the rationals or reals. We assume an infinite set $\underline{\mathtt{Var}}$ of variables, called *labeled nulls*, such that $\underline{\mathtt{Const}} \cap \underline{\mathtt{Var}} = \emptyset$. We define an *instance*[3] to be a conjunctive formula with arithmetic comparisons using constants from $\underline{\mathtt{Const}}$ and labeled nulls from $\underline{\mathtt{Var}}$. For an instance K, we use $\underline{\mathtt{Const}}(K)$ and $\underline{\mathtt{Var}}(K)$ to denote the set of constants and the set of labeled nulls in K, respectively. An instance K where $\underline{\mathtt{Var}}(K)$ is empty, is called *ground instance.*

Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a DEAC setting, and I be a ground instance defined over $\mathbf{S}$. We use $\underline{\mathtt{Const}}(\Sigma_{st} \cup \Sigma_t)$ to denote the set of constants appearing in source-to-target and target dependencies. An instance K is called a *t-instance with respect to* $\mathbf{M}$ *and* I, or simply a *t-instance* (if $\mathbf{M}$ and I are understood from the context), if the arithmetic comparisons of K define a total order with respect to $\mathcal{C} = \underline{\mathtt{Const}}(K) \cup \underline{\mathtt{Var}}(K) \cup \underline{\mathtt{Const}}(\Sigma_{st} \cup \Sigma_t) \cup \mathtt{Const}(I)$. A t-instance K' is called *a t-instance induced by an instance* K if K' is produced by adding to K additional arithmetic comparisons so that a total order is defined among the labeled nulls and constants in $\mathcal{C}$ (possibly with some labeled nulls or constants equated). Definition 3.1 formalizes the concept of *t-homomorphism* as a homomorphism which maps the relational atoms of a conjunctive formula with arithmetic comparisons on a t-instance, preserving also the satisfaction of the arithmetic comparisons.

[3]Our concept of instance is a generalization of instance in [14] or of conditional table in [20] as observed in [1].

DEFINITION 3.1. (*t-homomorphism*)
Let $F(\mathbf{x}) = F_0(\mathbf{x}) \wedge \beta_F(\mathbf{x})$ *be a conjunctive formula with arithmetic comparisons and* $K(\mathbf{x}) = K_0(\mathbf{x}) \wedge \beta_K(\mathbf{x})$ *be a t-instance. A* t-homomorphism h *from* F *to* K *is a mapping from* $\underline{\mathtt{Const}}(F) \cup \underline{\mathtt{Var}}(F)$ *to* $\underline{\mathtt{Const}}(K) \cup \underline{\mathtt{Var}}\ (K)$ *with the following properties:*

- *Every constant in* F *is mapped to the same constant in* K*.*
- *For every atom* $R(x_1, \ldots, x_k)$ *in* F *we have* $R(h(x_1), \ldots, h(x_k))$ *in* K*.*
- β_K *logically implies* $h(\beta_F)$*, denoted by* $\beta_K(\mathbf{x}) \Rightarrow \beta_F(h(\mathbf{x}))$*.*

We say F t-homomorphically *maps to* K *if there is a t-homomorphism from* F *to* K*. A t-homomorphism* h *from* F *to* K *directly implies a homomorphism* h_0 *from* F_0 *to* K_0*. We call* h_0 *the* underlying homomorphism *of* h*.*

EXAMPLE 3.1. *Let* $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ *be a DEAC setting in which* $\mathbf{S}$ *has a single binary relation* $\mathtt{E}$*,* $\mathbf{T}$ *has a single binary relation* $\mathtt{H}$*,* $\Sigma_\mathtt{t} = \emptyset$*, and*

$$\Sigma_{\mathtt{st}} = \{\ \mathtt{d} :\ \mathtt{E}(\mathtt{X},\mathtt{Y}), \mathtt{X} < \mathtt{Y} \rightarrow \exists \mathtt{Z}\ (\mathtt{H}(\mathtt{X},\mathtt{Z}), \mathtt{H}(\mathtt{Z},\mathtt{Y}), \mathtt{Z} < \mathtt{X}\}.$$

Let $\mathtt{I} = \{\mathtt{E}(2,5), \mathtt{E}(4,7)\}$ *be a ground instance defined over the source schema* $\mathbf{S}$*. Consider the following instances.*

$$\mathtt{J}_1 = \{\mathtt{H}(2,\mathtt{z}_1), \mathtt{H}(\mathtt{z}_1,5), \mathtt{H}(4,\mathtt{z}_2), \mathtt{H}(\mathtt{z}_2,7), \mathtt{z}_2 < \mathtt{z}_1 < 2\};$$
$$\mathtt{J}_2 = \{\mathtt{H}(2,\mathtt{z}_1), \mathtt{H}(\mathtt{z}_1,5), \mathtt{H}(4,\mathtt{z}_2), \mathtt{H}(\mathtt{z}_2,7), \mathtt{z}_1 = \mathtt{z}_2 < 2\};$$
$$\mathtt{J}_3 = \{\mathtt{H}(2,\mathtt{z}_1), \mathtt{H}(\mathtt{z}_1,5), \mathtt{H}(4,\mathtt{z}_2), \mathtt{H}(\mathtt{z}_2,7), \mathtt{z}_1 < \mathtt{z}_2 < 2\};$$
$$\mathtt{J}_4 = \{\mathtt{H}(2,\mathtt{z}_1), \mathtt{H}(\mathtt{z}_1,5), \mathtt{H}(4,\mathtt{z}_2), \mathtt{H}(\mathtt{z}_2,7), \mathtt{z}_1 < 2 = \mathtt{z}_2\};$$
$$\mathtt{J}_5 = \{\mathtt{H}(2,\mathtt{z}_1), \mathtt{H}(\mathtt{z}_1,5), \mathtt{H}(4,\mathtt{z}_2), \mathtt{H}(\mathtt{z}_2,7), \mathtt{z}_1 < 2 < \mathtt{z}_2 < 4\};$$
$$\mathtt{J}_6 = \{\mathtt{H}(2,\mathtt{z}_1), \mathtt{H}(\mathtt{z}_1,5), \mathtt{H}(4,\mathtt{z}_2), \mathtt{H}(\mathtt{z}_2,7), \mathtt{z}_1 < 2, \mathtt{z}_2 > 7\};$$
$$\mathtt{J}_7 = \{\mathtt{H}(2,\mathtt{z}_1), \mathtt{H}(\mathtt{z}_1,5), \mathtt{H}(4,\mathtt{z}_2), \mathtt{H}(\mathtt{z}_2,7), \mathtt{z}_2 < \mathtt{z}_1 < 3\};$$
$$\mathtt{J}_8 = \{\mathtt{H}(2,\mathtt{z}_1), \mathtt{H}(\mathtt{z}_1,5), \mathtt{H}(4,\mathtt{z}_2), \mathtt{H}(\mathtt{z}_2,7), \mathtt{z}_1 < 2, \mathtt{z}_2 \leq 2\};$$
$$\mathtt{J}_9 = \{\mathtt{H}(2,\mathtt{z}_1), \mathtt{H}(\mathtt{z}_1,5), \mathtt{H}(4,\mathtt{z}_2), \mathtt{H}(\mathtt{z}_2,7), \mathtt{z}_1 < 2, \mathtt{z}_2 < 4\};$$
$$\mathtt{J}_{10} = \{\mathtt{H}(2,1), \mathtt{H}(1,5), \mathtt{H}(4,0), \mathtt{H}(0,7)\};$$
$$\mathtt{J}_{11} = \{\mathtt{H}(2,2), \mathtt{H}(2,5), \mathtt{H}(4,0), \mathtt{H}(0,7), \mathtt{H}(8,7)\}.$$

Note that on all instances above we have omitted the trivial total order $2 < 4 < 5 < 7$*. For instance, the total order in* $\mathtt{J}_1$ *should be* $\mathtt{z}_2 < \mathtt{z}_1 < 2 < 4 < 5 < 7$*.* $\mathtt{J}_1$ *is a t-instance, since its arithmetic comparisons define a total order of its labeled nulls* (z_1 *and* z_2)*, its constants* (2, 4, 5 *and* 7) *and the constants in* $\mathtt{I}$ (*there is no constant in the tgd-AC*)*. Similarly,* $\mathtt{J}_2$*,* $\mathtt{J}_3$*,* $\mathtt{J}_4$*,* $\mathtt{J}_5$ *and* $\mathtt{J}_6$ *are all t-instances.* $\mathtt{J}_7$*,* $\mathtt{J}_8$ *and* $\mathtt{J}_9$ *are instances, but not t-instances. Both* $\mathtt{J}_{10}$ *and* $\mathtt{J}_{11}$ *are ground instances over* $\mathbf{T}$ *since they do not contain labeled nulls.*

There is a t-homomorphism h *from* $\mathtt{J}_7$ *to* $\mathtt{J}_1$*, where* h *is the identity mapping from* $\underline{\mathtt{Const}}(\mathtt{J}_7) \cup \underline{\mathtt{Var}}(\mathtt{J}_7)$ *to* $\underline{\mathtt{Const}}(\mathtt{J}_1) \cup \underline{\mathtt{Var}}(\mathtt{J}_1)$*. In particular, the mapping turns* $z_2 < z_1 < 3$ *to* $z_2 < z_1 < 3$*, which is implied by* $z_2 < z_1 < 2$*. Similarly,* $\mathtt{J}_1$ *t-homomorphically maps to the ground instance* $\mathtt{J}_{10}$*. However, there is no t-homomorphism from* $\mathtt{J}_6$ *to* $\mathtt{J}_1$*. Finally,* $\mathtt{J}_1$*,* $\mathtt{J}_2$*,* $\mathtt{J}_3$*, and* $\mathtt{J}_4$ *are all the t-instances induced by* $\mathtt{J}_8$*.*

Let d_1 be a tgd-AC, d_2 be an acgd and K be a t-instance. We say that K *satisfies* d_1 if whenever there is a t-homomorphism h from the lhs of d_1 to K, there exists an *extension* h' of h, where h' is a t-homomorphism from the conjunction of the lhs and the rhs of d_1 to K. We say that K *satisfies*

d_2 if whenever there is a t-homomorphism h from the lhs of d_2 to K then, the image of the rhs of d_2 under h is true.

3.2 Solution and Universal Solution

We continue with defining the concepts of *solution* and *universal solution* in a DEAC setting. Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a DEAC setting and I be a ground instance over the source schema $\mathbf{S}$. We call an instance J over $\mathbf{T}$, *t-solution for I under* $\mathbf{M}$ (or, simply *t-solution* if I and $\mathbf{M}$ are understood from the context), if J is a t-instance and satisfies Σ_t and the instance $\langle I, J\rangle = I \cup J$ satisfies $\Sigma_{st} \cup \Sigma_t$. A t-solution that is a ground instance is called a *ground solution.* We call an instance J of $\mathbf{T}$, a *solution for I under* $\mathbf{M}$ (or, simply a *solution* if I and $\mathbf{M}$ are understood from the context), if every t-instance induced by J is a t-solution. A *universal solution* for I under $\mathbf{M}$, or simply a *universal solution*, is a *set* $\mathcal{J} = \{J_1, \ldots, J_m\}$ of solutions for I, such that for every ground solution K, there is a solution $J_i \in \mathcal{J}$ that t-homomorphically maps to K. A *universal t-solution* is a universal solution which consists solely of t-solutions.

Example 3.2. (*Continued from Example 3.1*). *The set $\mathcal{J}_1 = \{\mathtt{J}_1, \ldots, \mathtt{J}_5\}$ is a universal t-solution. $\mathcal{J}_2 = \{\mathtt{J}_5, \mathtt{J}_8\}$ and $\mathcal{J}_3 = \{\mathtt{J}_9\}$ are two universal solutions. It is easy to check that the source-to-target dependency d is satisfied on the union of I and every t-instance in $\mathcal{J}_1$. d is also satisfied on $\mathtt{J}_8$, since $\mathtt{J}_1, \ldots, \mathtt{J}_4$ are all the induced instances of $\mathtt{J}_8$. Observe that since $\mathcal{J}_3$ is a universal solution, $\mathcal{J}_2$ is also a universal solution because if there is a t-homomorphism from an instance* $\mathtt{F}$ *to a t-instance* $\mathtt{K}$*, then there is some induced t-instance of* $\mathtt{F}$ *which t-homomorphically maps on* $\mathtt{K}$.

A solution in DEAC settings does not necessarily qualify for solution in [14]. For instance, consider the instance $\mathtt{K} : \mathtt{a}(5, \mathtt{N}_1), \mathtt{b}(5, \mathtt{N}_2)$ and the tgd $\mathtt{d} : \mathtt{a}(5, \mathtt{N}), \mathtt{b}(5, \mathtt{N}) \rightarrow \mathtt{c}(\mathtt{X}, \mathtt{N})$. Observe that $\mathtt{K}$ is a solution in [14] since there is no homomorphism from the lhs of $\mathtt{d}$ to $\mathtt{K}$. However, $\mathtt{K}$ is not a solution in a DEAC setting because a n induced ground instance of $\mathtt{K}$ is $\mathtt{K}' : \mathtt{a}(5, 8), \mathtt{b}(5, 8)$ and there exists a homomorphism from the lhs of $\mathtt{d}$ to $\mathtt{K}'$ that can not be extended.

We now establish a result which shows the robustness of our definition of universal solution. To every instance K we associate a boolean conjunctive query with arithmetic comparisons q^K, whose body contains exactly all the facts in K as subgoals (labeled nulls are replaced by variables) and the (possibly partial) order is added to q^K as arithmetic comparison predicates. For the predicate in the head we use a fresh predicate name. We call q^K the *corresponding boolean query* of K. To a set of instances we associate a corresponding boolean query by creating a boolean query for each instance with the same head predicate and considering the union of them. Proposition 3.1 states that any two universal solutions are equivalent if viewed as queries. In what follows, we use $SOL(\mathbf{M}, I)$ to denote the set of all solutions for the source instance I under the DEAC setting $\mathbf{M}$.

Proposition 3.1. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a DEAC setting. The following hold:*

1. *If $\mathcal{J}$ and $\mathcal{J}'$ are two universal solutions for a source instance I, then the corresponding boolean queries $q^{\mathcal{J}}$ and $q^{\mathcal{J}'}$ are equivalent.*
2. *Let I, I' be two source instances, $\mathcal{J}$ be a universal solution for I under $\mathbf{M}$, and $\mathcal{J}'$ be a universal solution for I' under $\mathbf{M}$. Then,*
   - *$SOL(\mathbf{M}, I) \subseteq SOL(\mathbf{M}, I')$ iff $q^{\mathcal{J}}$ is contained in $q^{\mathcal{J}'}$;*
   - *$SOL(\mathbf{M}, I) = SOL(\mathbf{M}, I')$ iff $q^{\mathcal{J}}$ and $q^{\mathcal{J}'}$ are equivalent.*

Recall that in [14] any two universal solutions are homomorphically equivalent which means that they are equivalent if viewed as queries. Proposition 3.1 showed that this property carries over to universal solutions in DEAC settings.

3.3 Computing Universal Solutions

In the previous subsection we defined what a universal solution is in DEAC settings. In this subsection we show how to compute a universal solution. We start with defining a chase procedure, called *AC-chase* and then prove that the result of a finite successful AC-chase is a universal solution.

3.3.1 AC-chase

AC-chase shares the main idea of the chase procedures in the literature [31, 8, 7, 18]. However, it produces a tree rather than a sequence. The reason is that the result of an AC-chase step on a t-instance may not always be a t-instance. Indeed, it might be an instance containing a partial, instead of a total, order. Therefore, if K is the instance resulting from an AC-chase step, we might need to consider all t-instances induced by K before proceeding to the next AC-chase step. In [14], to prove the complexity of query answering for conjunctive queries with inequalities, a chase called *disjunctive* was presented, which is also a tree. However, disjunctive chase is tailored to a specific query, whereas, the result of a finite successful AC-chase can be used to compute the certain answers of any union of conjunctive queries with arithmetic comparisons (see Section 4). Definitions 3.2 and 3.3 define formally an AC-chase step and AC-chase tree respectively.

Definition 3.2. (*AC-chase Step*)
Let K be a t-instance. An AC-chase step *consists of 3 stages:*
Stage I: *We construct an instance K' by adding relational facts and arithmetic comparisons to K. There are two cases:*

a) (acgd): *Let d be an acgd $\phi(\mathbf{x}) \wedge \beta_\phi(\mathbf{x}) \rightarrow (x_1 \theta x_2)$. Let h be a t-homomorphism from $\phi(\mathbf{x}) \wedge \beta_\phi(\mathbf{x})$ to $K(\mathbf{x}) = K_0(\mathbf{x}) \wedge \beta_K(\mathbf{x})$ such that $h(x_1 \theta x_2)$ is not true. We say that d can be applied to K with t-homomorphism h. We add $h(x_1 \theta x_2)$ to K and produce K'.*

b) (tgd-AC): *Let $d : \phi(\mathbf{x}) \wedge \beta_\phi(\mathbf{x}) \rightarrow \psi(\mathbf{x}, \mathbf{y}) \wedge \beta_\psi(\mathbf{x}, \mathbf{y})$ be a tgd-AC. Let h be a t-homomorphism from $\phi(\mathbf{x}) \wedge \beta_\phi(\mathbf{x})$ to K such that there is no extension of h to a t-homomorphism from $\phi(\mathbf{x}) \wedge \beta_\phi(\mathbf{x}) \wedge \psi(\mathbf{x}, \mathbf{y}) \wedge \beta_\psi(\mathbf{x}, \mathbf{y})$ to K. We say that d can be applied to K with t-homomorphism h. Let h_0 be the underlying homomorphism of h and let K_0' be the union of K_0 with the set of facts obtained by:*
(i) *extending h_0 to a homomorphism h_0', such that each variable in $\mathbf{y}$ is assigned a fresh labeled null, followed by*
(ii) *taking the image of the atoms in the rhs of d under h_0'.*
For each AC $x_1 \theta x_2$ in $\beta_\psi(\mathbf{x})$ we add $h_0'(x_1 \theta x_2)$ to K_0' and obtain K' in compact form.

Stage II: *We check K' for consistency. Two cases:*
1. *If K' is not consistent, we say that the result of applying d to K with h is "failure" and write $K \xrightarrow{d,h} \perp$.*
2. *Otherwise, we say that the result of applying d to K with h is K', denoted $K \xrightarrow{d,h} K'$.*

Stage III: *We produce all induced t-instances $K_1', \ldots, K_n'$ of K' in compact form and write also (equivalently) $K \xrightarrow{d,h} \{K_1', \ldots, K_n'\}$.*

Definition 3.3. (*AC-chase Tree*)
Let Σ be a set of tgd-ACs and acgds, and J be a t-instance. An AC-chase tree *of J with Σ is a tree (finite or infinite) such that:*

- *The root is J.*
- *Each node in the tree is a t-instance.*
- *For every node K in the tree, let $\{K'_1, \dots, K'_l\}$ be the set of its children. Then, there must exist a tgd-AC or an acgd d in Σ, a t-homomorphism h from the lhs of d to K, and an instance K', such that $K \xrightarrow{d,h} K'$, and $K'_1, \dots, K'_l$ are all the t-instances induced by K'.*

A finite AC-chase *of J with Σ is a finite chase tree such that each leaf is either $\perp$, or satisfies Σ. The* result *of a* finite AC-chase *is the union of all the leaf nodes that are not $\perp$. If the result is not empty, we refer to it as the result of a* finite successful AC-chase.

Theorem 3.1 states that the result of a finite successful AC-chase is a universal solution in a DEAC setting.

Theorem 3.1. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a DEAC setting and I be a source instance.*

1. *Let $\mathcal{L} = \{\langle I \cup J_1\rangle, \dots, \langle I \cup J_k\rangle\}$ be the result of a successful finite AC-chase of I with $\Sigma = \Sigma_{st} \cup \Sigma_t$. Then, $J = \{J_1, \dots, J_k\}$ is a universal t-solution.*
2. *If the result of a finite chase of I with $\Sigma = \Sigma_{st} \cup \Sigma_t$ is empty, then there is no solution.*

Example 3.3 illustrates an AC-chase tree. Observe that even in this simple case there is no universal solution consisting of a single instance.

Example 3.3. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a DEAC setting with Σ_{st} and Σ_t being as follows:*

$$\Sigma_{st} = \{\ \mathtt{d_1 : A(Y,Z) \rightarrow E(X,Y,Z)}\}$$
$$\Sigma_t = \begin{cases} \mathtt{d_2 : E(X,Y,Z), X < Z \rightarrow X = Y} \\ \mathtt{d_3 : E(X,Y,Z), X = Z \rightarrow F(X,Y,X)} \end{cases}$$

Let $\mathtt{I} = \{\mathtt{A(5,8)}\}$ be the source instance. Figure 3 illustrates the AC-chase tree. We chase $\mathtt{I}$ with $\mathtt{d_1}$ and get $\mathtt{J} = \{\mathtt{E(X,5,8)}\}$. Then, we consider all t-instances induced by $\mathtt{J}$:

$$\begin{cases} \mathtt{J_1 = \{E(X,5,8), X < 5 < 8\}} \\ \mathtt{J_2 = \{E(X,5,8), X = 5 < 8\}} \\ \mathtt{J_3 = \{E(X,5,8), 5 < X < 8\}} \\ \mathtt{J_4 = \{E(X,5,8), 5 < X = 8\}} \\ \mathtt{J_5 = \{E(X,5,8), 5 < 8 < X\}} \end{cases}$$

The acgd $\mathtt{d_2}$ is applied on the t-instances $\mathtt{J_1, J_3}$. The AC-chase fails on both of them (denoted with $\perp$ in Figure 3). $\mathtt{J_2}$ already satisfies all constraints, so AC-chase stops (the left oval in Figure 3). The same holds for the t-instance $\mathtt{J_5}$ (the right oval in Figure 3). The tgd-AC $\mathtt{d_3}$ is applied on the t-instance $\mathtt{J_4}$ giving the fact $\mathtt{F(8,5,8)}$ and the AC-chase stops successfully (the oval in the middle of Figure 3). The t-solutions that comprise the universal solution are shown in the ovals in the leaf nodes. Notice that there is no singleton universal solution. More precisely, the universal solution $\mathtt{J'}$ consists of three solutions: $\mathtt{J'_1} = \{\mathtt{E(5,5,8)}\}$, $\mathtt{J'_2} = \{\mathtt{E(8,5,8), F(8,5,8)}\}$ and $\mathtt{J'_3} = \{\mathtt{E(X,5,8), 5 < 8 < X}\}$.

Example 3.4. (*Continued from Example 3.2*). *We have now the techniques to show that the instance $\mathcal{J}_1$ is indeed a universal solution. Applying an AC-chase step to I with t-homomorphism $X \rightarrow 2, Y \rightarrow 5$ produces a number of t-instances as a consequence of Stage III in Definition 3.2. If to each of these t-instances we apply an AC-chase step with t-homomorphism $X \rightarrow 4, Y \rightarrow 7$, we produce $\mathcal{J}_1$.*

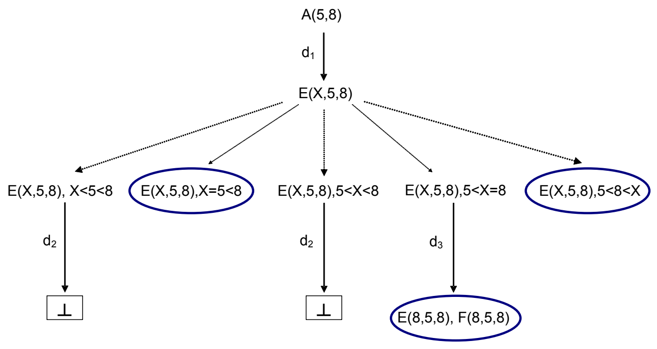

Figure 3: AC-Chase Tree for Example 3.3.

3.3.2 Complexity

We continue with identifying the conditions under which AC-chase terminates and guarantee the computation of the universal solution if there exists a solution. We also study the complexity of the *existence of solution* problem. In particular, we show that AC-chase terminates when the target dependencies consist of a set of *weakly acyclic* tgd-ACs and a set of acgds. The definition of weak acyclicity for tgd-ACs is an extension of the definition of weak acyclicity for tgds in [14].

Definition 3.4. (*Weakly acyclic set of tgd-ACs*)
Let Σ be a set of tgd-ACs in compact form. We construct a directed graph (called the dependency graph*) as follows:*
(1) *There is a node for every pair (R, A) with a relation symbol R of the schema and an attribute A of R. We call such a pair (R, A) a* position.
(2) *Add edges as follows: for every tgd-AC $\phi(\mathbf{x}) \wedge \beta_\phi(\mathbf{x}) \rightarrow \psi(\mathbf{x}, \mathbf{y}) \wedge \beta_\psi(\mathbf{x}, \mathbf{y})$ in Σ and for every x in $\mathbf{x}$ that occurs in $\psi(\mathbf{x}, \mathbf{y}) \wedge \beta_\psi(\mathbf{x}, \mathbf{y})$, we call x a* propagated variable. *For such a propagated variable x, for every occurrence of x in $\phi(\mathbf{x})$ in position (R, A_i), do the following:*
(i) *For every occurrence of x in $\psi(\mathbf{x}, \mathbf{y})$ in position (S, B_j), add an edge $(R, A_i) \rightarrow (S, B_j)$ (if it does not already exist);*
(ii) *In addition, for every existentially quantified variable y in $\mathbf{y}$ and for every occurrence of y in $\psi(\mathbf{x}, \mathbf{y})$ in position (T, C_k), add a* special edge $(R, A_i) \rightarrow (T, C_k)$ *(if it does not already exist).*
We call Σ weakly acyclic *if the dependency graph has no cycle going through a special edge.*

Observe that the presence of arithmetic comparisons may affect significantly weak acyclicity. More precisely, the above definition differs from weak acyclicity for simple tgds (without arithmetic comparisons) in that a variable from the lhs of a tgd-AC d that appears in an arithmetic comparison in the rhs of d affects acyclicity even if this variable does not appear in any relational atom in the rhs of d.

Example 3.5. *Consider a database schema with a relation $\mathtt{R(A)}$ and a relation $\mathtt{S(B)}$. We consider first the set of* *tgds* $\Sigma = \{\mathtt{d}_1, \mathtt{d}_2\}$*:*

$$\mathtt{d}_1 : \mathtt{R}(\mathtt{X}) \rightarrow \mathtt{S}(\mathtt{Y});$$
$$\mathtt{d}_2 : \mathtt{S}(\mathtt{Y}) \rightarrow \mathtt{R}(\mathtt{Y}).$$

The dependency graph is weakly acyclic, thus the chase terminates [14]. Suppose we modify $\mathtt{d}_1$ *by adding arithmetic comparisons in its rhs. Now,* $\mathtt{d}_1$ *becomes a tgd-AC* $\mathtt{d}'_1$*:*

$$\mathtt{d}'_1 : \mathtt{R}(\mathtt{X}) \rightarrow \mathtt{S}(\mathtt{Y}), \mathtt{X} < \mathtt{Y}.$$

The dependency graph of $\Sigma' = \{\mathtt{d}'_1, \mathtt{d}_2\}$ *is no longer weakly acyclic. Indeed, AC-chase does not terminate. The reason is that every time* $\mathtt{d}'_1$ *is applied on an instance defined over the database schema, we must add an atom with a value for its argument which is greater than the value of the atom we added in the previous application of* $\mathtt{d}'_1$*.*

Theorem 3.2 shows that the *existence of solution* problem is in NP and yields a broad sufficient condition for the computation of a universal solution. Similarly to all complexity results presented in this paper, the complexity is in the size of the source instance (data complexity).

Theorem 3.2. *Let* $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ *be a DEAC setting, such that* Σ_t *consists of a set of weakly acyclic tgd-ACs and a set of acgds and let* I *be a ground source instance. Then, the following hold:*

1. *There is a polynomial that bounds the depth of every AC-chase beginning with* I*.*
2. *There is a nondeterministically polynomial-time algorithm such that it tests whether a t-solution for* I *exists.*
3. *AC-chase takes exponential time to compute a universal t-solution for* I*.*

Theorem 3.2 covers interesting cases that arise in many applications. For instance, the class of weakly acyclic tgd-ACs includes the class of (a) full tgd-ACs and (b) tgd-ACs in which the tgds alone are weakly acyclic and the arithmetic comparisons are semi-interval. In the first case, since full tgd-ACs do not have existentially quantified variables, the dependency graph does not contain special edges. In the second case, the fact that the arithmetic comparisons are semi-interval implies that every variable that propagates from the lhs of a tgd-AC d to its rhs appears necessarily also in the relational atoms in the rhs of d.

4. QUERY ANSWERING

In this section we study the *query answering* problem in DEAC settings. We consider queries posed over the target schema that belong to the class of unions of *conjunctive queries with arithmetic comparisons* (in short CQACs). A conjunctive query with arithmetic comparisons is of the form:

$$h(\overline{X}) \ :- \ e_1(\overline{X}_1), \ldots, e_k(\overline{X}_k), C_1, \ldots, C_m,$$

where each C_i is of the form "$A\theta B$" with A being a variable, B a variable or a constant and θ is in $\{\leq, \geq, <, >, \neq, =\}$. For instance, the following query is a CQAC posed over the target schema of Example 1.1 asking for the address of owners of apartments costing less than 400 (thousand dollars):

$$\mathtt{ans}(\mathtt{A}_2) : -\mathtt{Apartment}(\mathtt{I}, \mathtt{A}_1, \mathtt{P}, \mathtt{Z}, \mathtt{N}), \mathtt{Person}(\mathtt{N}, \mathtt{A}_2), \mathtt{P} < 400.$$

For the semantics of query answering, similarly to previous work [14], we adopt the concept of *certain answers*. In the rest of this section we use $certain_M(q, I)$ to denote the certain answers of a query q, given a source instance I under a DEAC setting $\mathbf{M}$. Certain answers are defined in [14] as the intersection of the answer of the query over all possible worlds. In [14] the set of possible worlds considered to define the certain answers is the set of solutions. However, in DEAC settings the domains are not flat as in [14], rather they are interpreted over densely totally ordered domains. Hence, we need to resort to *ground* solutions. I.e., in DEAC settings the set of possible worlds considered to define the certain answers is the set of all ground solutions:

Definition 4.1. (*Certain answers*) *Let* $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ *be a DEAC setting,* I *a source instance and* q *a query posed over the target schema* $\mathbf{T}$*. Then,*

$$certain_M(q, I) = \cap\{q(J) : J \in G(M, I)\},$$

where $G(M, I)$ *is the set of all ground solutions for* I *under* $\mathbf{M}$*, and* $q(J)$ *is the result of evaluating* q *on* J*.*

For a given source instance there may exist infinitely many ground solutions under a data exchange setting. So how can we compute the certain answers? In [14] it was shown that universal solutions can be used to answer conjunctive queries posed over the target schema. Trying to extend this result to more expressive query languages fails, even if we only add inequalities ($\neq$) to conjunctive queries [14, 4, 23]. In the rest of this section we show that universal solutions as defined in Section 3 can be used to answer conjunctive queries with general arithmetic comparisons ($<, \leq, >, \geq, =, \neq$).

Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a DEAC setting. We will show that if q is a union of CQAC queries posed over the target schema $\mathbf{T}$, I is a source instance and $\mathcal{J} = \{J_1, \ldots, J_k\}$ is a universal solution for I then

$$certain_M(q, I) = \cap\{q(J_i) : J_i \in \mathcal{J}\}. \qquad (1)$$

Recall that a universal solution consists of a finite set of solutions. Therefore, to compute the certain answers of a CQAC query posed over the target schema, we need first to specify how to evaluate the query on a solution. Observe that even if every solution is a t-instance, it is not straightforward how to compute the answers to the query. The reason is that the query may contain constants not appearing in the universal solution (i.e., fresh constants). Therefore, the total order of the solution may no longer form a total order with respect also to the constants of the query. The following example explains this point.

Example 4.1. *Consider the following CQAC query:*

$$\mathtt{q} : \ \mathtt{ans}(\mathtt{A}) \text{:- } \mathtt{R}(\mathtt{A}), \mathtt{A} < 3.$$

Let $\mathtt{J} = \{\mathtt{R}(\mathtt{X}), \mathtt{X} < 7\}$ *be a t-instance. If we replace the labeled null* $\mathtt{X}$ *with constant* 2*, the answer to the query would be* $\{2\}$*. However, if we replace* $\mathtt{X}$ *with another constant* 5*, the answer to the query becomes* empty*. Observe that although* $\mathtt{J}$ *defines a total order with respect to its own constants and labeled nulls, the query could introduce fresh constants* (*e.g., the constant* 3)*. Consequently, the order consisting of the labeled nulls, the constants in the t-instance and the fresh constants of the query may no longer be a total order.*

From Example 4.1 it becomes clear that we need to introduce the concept of *qt-instance* defined as a t-instance that also forms a total order w.r.t. the constants of the query. More formally, let $M = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a DEAC setting, q be a CQAC query, and J be an instance over $\mathbf{T}$. Let $\mathcal{C} = \underline{\texttt{Const}}(J) \cup \underline{\texttt{Const}}(\Sigma_{st} \cup \Sigma_t) \cup \underline{\texttt{Const}}(q)$. An instance K is called a *qt-instance* with respect to query q and instance J under $\mathbf{M}$, if it is a t-instance with respect to $\mathcal{C}$ and the labeled nulls of K. We evaluate a query q on a qt-instance K by treating the labeled nulls in K as distinct constants and remove the tuples containing labeled nulls (the latter denoted by the operator "$\downarrow$").

Definition 4.2. *Let J be an instance and q be a CQAC query. Let J_q be the set of qt-instances induced by J. The result of evaluating q on J, denoted $q(J)$, is defined as:*

$$q(J) = \cap\{q(K)_\downarrow : K \in J_q\},$$

where $q(K)$ is the set of answers of q on qt-instance K.

The following theorem states that if Definition 4.2 is used for computing answers of a query on a given instance, then universal solutions can be used to compute certain answers of unions of CQAC queries.

Theorem 4.1. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a DEAC setting. The following hold:*

- *Let q be a union of CQAC queries posed over the target schema $\mathbf{T}$, and I be a source instance. If $\mathcal{J} = \{J_1, \dots, J_k\}$ is a universal solution for I under $\mathbf{M}$, then $certain_M(q, I) = \cap\{q(J_i), J_i \in \mathcal{J}\}$.*
- *Let I be a source instance and $\mathcal{J}$ be a set of solutions, such that for every union of CQACs q posed over schema $\mathbf{T}$, it holds $certain_M(q, I) = \cap\{q(J_i), J_i \in \mathcal{J}\}$. Then, $\mathcal{J}$ is a universal solution.*

Example 4.2 illustrates how a universal solution can be used to compute the certain answers of a CQAC query.

Example 4.2. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a DEAC setting where $\boldsymbol{S}$ contains one binary predicate $\texttt{v}$, $\boldsymbol{T}$ contains three unary predicates $\texttt{a}$, $\texttt{b}$ and $\texttt{c}$ and $\Sigma_t = \emptyset$. Let the source instance be $\texttt{I} = \{\texttt{v}(7, 7)\}$. Consider the following query:*

$$\texttt{Q} : \texttt{q}() : -\texttt{a}(\texttt{X}), \texttt{c}(\texttt{Y}), \texttt{b}(\texttt{Y}), \texttt{X} < 1.$$

Suppose Σ_{st} consists of the following tgd-AC:

$$\texttt{d} : \texttt{v}(\texttt{T}, \texttt{U}) \rightarrow \texttt{a}(\texttt{S}), \texttt{c}(\texttt{T}), \texttt{b}(\texttt{U}), \texttt{S} \leq \texttt{T}.$$

Then, a universal solution $\mathcal{J}$ consists of two solutions:

$$\texttt{J}_1 = \{\texttt{a}(\texttt{L}), \texttt{c}(7), \texttt{b}(7), \texttt{L} < 7\},$$
$$\texttt{J}_2 = \{\texttt{a}(\texttt{L}), \texttt{c}(7), \texttt{b}(7), \texttt{L} = 7\}.$$

To compute the certain answers of $\texttt{Q}$ we need first consider all qt-instances of $\texttt{J}_1$ and $\texttt{J}_2$:

$$\texttt{J}_{11} = \{\texttt{a}(\texttt{L}), \texttt{c}(7), \texttt{b}(7), 1 < \texttt{L} < 7\},$$
$$\texttt{J}_{12} = \{\texttt{a}(\texttt{L}), \texttt{c}(7), \texttt{b}(7), 1 = \texttt{L} < 7\},$$
$$\texttt{J}_{13} = \{\texttt{a}(\texttt{L}), \texttt{c}(7), \texttt{b}(7), \texttt{L} < 1 < 7\},$$
$$\texttt{J}_{21} = \{\texttt{a}(\texttt{L}), \texttt{c}(7), \texttt{b}(7), 1 < \texttt{L} = 7\}.$$

The answer of $\texttt{Q}$ *on* $\texttt{J}_{11}$, $\texttt{J}_{12}$, $\texttt{J}_{21}$ *is* false, *whereas the answer of* $\texttt{Q}$ *on* $\texttt{J}_{13}$ *is* true. *Thus, the certain answers of* $\texttt{Q}$ *is* false. *Consider a second query $\texttt{Q}' : \texttt{q}()$:- $\texttt{a}(\texttt{X}), \texttt{X} \leq 7$. Now, the certain answers of $\texttt{Q}'$ is* true *because $\texttt{Q}'$ computes to* true *on both $\texttt{J}_1$ and $\texttt{J}_2$.*

Theorem 4.2. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a DEAC setting such that Σ_t is the union of a weakly acyclic set of tgd-ACs and a set of acgds. Let q be a union of CQAC queries. Given a source instance I and a tuple t, the problem "is t in $certain_M(q, I)$?" is co-NP complete.*

5. POLYNOMIAL RESULTS

In this section we identify the cases in which a) we compute a universal solution in PTIME and b) we compute the certain answers of conjunctive queries with arithmetic comparisons in PTIME. The following is a summary of the polynomial results presented in this section:

(A) We compute a universal solution in PTIME when:
  (a) there are no target constraints or
  (b) all tgd-ACs are full.

(B) We compute certain answers of CQAC queries in PTIME when:
  (a) there are no target constraints and the CQAC query has the homomorphism property or
  (b) all tgd-ACs are full or
  (c) the data exchange setting consists of tgds and egds (without arithmetic comparisons) and the CQAC query has only open (closed) LSI (RSI) arithmetic comparisons.

5.1 Compute a Universal Solution in PTIME

We investigate the conditions under which a universal solution can be computed in PTIME. Technically, we identify the cases in which a universal solution is a singleton. Example 5.1 shows such a case and illustrates the intuition behind *succinct AC-chase* (formally defined in Subsection 5.1.1). In Subsection 5.3 we present the similarities and differences of *succinct AC-chase*, *AC-chase* and the chase procedure presented in [14] for different classes of constraints.

Example 5.1. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a DEAC setting, where $\Sigma_t = \emptyset$ and $\Sigma_{st} = \{\texttt{d} : \texttt{E}(\texttt{Y}) \rightarrow \texttt{A}(\texttt{Y}, \texttt{Z}), \texttt{Y} \leq \texttt{Z}\}$. Let $I = \{E(5)\}$ be the source instance. The result of AC-chase produces a universal solution which consists of the following two solutions $\texttt{J}_1, \texttt{J}_2$:*

$$\texttt{J}_1 = \{\texttt{A}(5, \texttt{z}), 5 < \texttt{z}\},$$
$$\texttt{J}_2 = \{\texttt{A}(5, 5)\}.$$

However, there is also a singleton universal solution $\texttt{J}$:

$$\texttt{J} = \{\texttt{A}(5, \texttt{z}), 5 \leq \texttt{z}\}.$$

Therefore, it may not be always necessary to consider all t-instances induced by an instance after an AC-chase step.

5.1.1 Succinct AC-chase

To identify polynomial cases, we introduce a new chase procedure called *succinct AC-chase* (in short SAC-chase) which is a sequence rather than a tree, as the AC-chase. However, the result of a SAC-chase is not necessarily a universal solution. The reason is that some of the induced t-instances of the result of the SAC-chase may not satisfy the constraints, hence it may not be a solution.

We introduce the concept of *p-homomorphism* which naturally extends the notion of *t-homomorphism*, by allowing a mapping from a conjunctive formula with arithmetic comparisons to an instance (not necessarily t-instance). Proposition 5.1 relates a p-homomorphism to t-homomorphisms.

Proposition 5.1. *Let $F(\mathbf{x}) = F_0(\mathbf{x}) \wedge \beta_F(\mathbf{x})$ be a conjunctive formula with arithmetic comparisons and $K(\mathbf{x}) = K_0(\mathbf{x}) \wedge \beta_K(\mathbf{x})$ be an instance in compact form. If there is a p-homomorphism from F to K then, there is a t-homomorphism from F to every induced t-instance of K.*

We proceed to the definition of SAC-chase. Technically, a SAC-chase step consists of the first two stages of an AC-chase step (Definition 3.2). It omits the final decomposition of an instance to its induced t-instances in Stage III. Consequently, every node in the SAC-chase step produces a single child-instance (instead of multiple children). Hence, SAC-chase is a sequence. Notice that since the instance considered in each chase step may contain a partial order, instead of total order, we use p-homomorphism instead of t-homomorphism. The formal definition follows.

Definition 5.1. (*SAC-chase Step*)
Let K be an instance. A Succinct AC-chase *step* (*in short* SAC-chase step) *consists of 2 stages:*
Stage I: *We construct instance K' by adding relational facts and arithmetic comparisons to K. There are two cases:*
a) (acgd): *Let d be an acgd $\varphi(\mathbf{x}) \wedge \beta_\phi(\mathbf{x}) \rightarrow (x_1 \theta x_2)$. Let h be a p-homomorphism from $\varphi(\mathbf{x}) \wedge \beta_\phi(\mathbf{x})$ to $K(\mathbf{x}) = K_0(\mathbf{x}) \wedge \beta_K(\mathbf{x})$ such that $h(x_1\theta x_2)$ is not true. We say that d can be applied to K with p-homomorphism h. We add $h(x_1\theta x_2)$ to K and produce K'.*
b) (tgd-AC): *Let $d : \phi(\mathbf{x}) \wedge \beta_\phi(\mathbf{x}) \rightarrow \psi(\mathbf{x},\mathbf{y}) \wedge \beta_\psi(\mathbf{x},\mathbf{y})$ be a tgd-AC. Let h be a p-homomorphism from $\phi(\mathbf{x}) \wedge \beta_\phi(\mathbf{x})$ to K, such that there is no extension of h to a p-homomorphism from $\phi(\mathbf{x}) \wedge \beta_\phi(\mathbf{x}) \wedge \psi(\mathbf{x},\mathbf{y}) \wedge \beta_\psi(\mathbf{x},\mathbf{y})$ to K. Let h_0 be the underlying homomorphism of h and let K_0' be the union of K_0 with the set of facts obtained by:*
(i) extending h_0 to a homomorphism h_0', such that each variable in $\mathbf{y}$ is assigned a fresh labeled null, followed by
(ii) taking the image of the atoms in the rhs of d under h_0'.
For each AC $x_1\theta x_2$ in $\beta_\psi(\mathbf{x})$ we add $h_0'(x_1\theta x_2)$ to K_0' and obtain K' in compact form.
Stage II: *We check K' for consistency. Two cases:*
1. If K' is not consistent we say that the result of applying d to K with h is "failure", denoted $K \xrightarrow{d,h} \perp$.
2. Otherwise, we say that the result of applying d to K with h is K', denoted $K \xrightarrow{d,h} K'$.

Definition 5.2. (*SAC-chase Sequence*)
Let Σ be a set of tgd-ACs and acgds and K be an instance.
• *A SAC-chase sequence of K with Σ is a sequence (finite or infinite) of SAC-chase steps starting from K.*
• *A* finite SAC-chase *of K with Σ is a finite SAC-chase sequence, with the requirement that either*
(***a***) *the last instance in the sequence is $\perp$, or*
(***b***) *the last instance satisfies all the dependencies in Σ.*
The result of a finite SAC-chase *consists of the instance in the leaf node. We refer to case* (***a***) *as the case of a* failing finite SAC-chase *and to case* (***b***) *as the case of a* successful finite SAC-chase*.*

Example 5.2 illustrates a SAC-chase and shows its usefulness to obtain the universal solution $\mathcal{J}_3$ of Example 3.2.

Example 5.2. (*Continued from Example 3.2*). *Applying a SAC-chase step to the source instance $I = \{E(2,5), E(4,7)\}$ with p-homomorphism $X \rightarrow 2, Y \rightarrow 5$ produces the instance*

$$\langle \mathtt{I}, \mathtt{K}\rangle = \{\mathtt{E}(2,5), \mathtt{E}(4,7), \mathtt{H}(2,\mathtt{z}_1), \mathtt{H}(\mathtt{z}_1,5), \mathtt{z}_1 < 2\}.$$

Applying another SAC-chase step to $\langle \mathtt{I}, \mathtt{K}\rangle$ with p-homomorphism $X \rightarrow 4, Y \rightarrow 7$ produces $\mathcal{J}_3$.

Lemma 5.1. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a DEAC setting where Σ_t consists of a set of weakly acyclic tgd-ACs and a set of acgds. Let I be a source instance. Then, there is a polynomial in the size of I that bounds the length of every SAC-chase sequence of I with Σ.*

5.1.2 Computing Universal Solution in PTIME

As already mentioned, the result of a SAC-chase is not necessarily a universal solution. In this subsection we identify two cases in which the result of SAC-chase is indeed a universal solution. First, observe that in DEAC settings with no target constraints, chase is simpler. Let K be an instance obtained after a chase step and suppose there are no target constraints. When applying a dependency d on K, the variables in the lhs of d map always to constants because the labeled nulls of K are already totally ordered. Thus, there is no need to consider all the induced t-instances.

The second case is when all tgd-ACs in the DEAC setting are full. Theorem 5.1 states formally these claims.

Theorem 5.1. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a DEAC setting where either (a) there are no target constraints ($\Sigma_t = \emptyset$); or (b) all tgd-ACs in $\Sigma_{st} \cup \Sigma_t$ are full. Then it holds:*

- *Given a source instance I, if there is a solution for I under $\mathbf{M}$, there is a universal solution for I that is a singleton.*
- *There is a polynomial-time algorithm in the size of the source instance that is based on a SAC-chase procedure such that, given a source instance I, it tests whether a solution for I exists, and if so, it produces a singleton universal solution for I.*

Example 5.3. *In Example 5.1 only a single SAC-chase step is needed to compute the universal solution* J*.*

5.2 Computing Certain Answers in PTIME

In this subsection we investigate the cases in which we can compute the certain answers of CQAC queries in polynomial time. Suppose we are given an instance K. We will show that evaluating a CQAC query in K can be computed efficiently when the query has a nice property, named *homomorphism property* [22, 2].

Definition 5.3. (*Homomorphism Property* (*HP*))
1. *Let q be a CQAC and K be an instance. We say that q* has the homomorphism property with respect to K, (*or simply q has the homomorphism property if K is understood from the context*), *if the following is true: there is a p-homomorphism from q to K such that the head of q is mapped to a tuple of constants t if and only if for every t-instance K_i induced by K, there is a t-homomorphism from q to K_i such that the head of q is mapped to t.*
2. *Let $\mathcal{Q}$ be a class of CQACs and $\mathcal{K}$ be a class of instances. We say that Q* has the homomorphism property with respect to $\mathcal{K}$ *if every q in $\mathcal{Q}$ has the property with respect to each instance in $\mathcal{K}$.*

LEMMA 5.2. *Let q be a CQAC query which has the homomorphism property. Then, for any instance K, we can compute $q(K)$ in PTIME.*

In [2] the syntactic conditions of CQAC queries that have the homomorphism property were investigated. Queries that contain either only open (closed) LSI (RSI) comparisons, were shown to have the homomorphism property.

The following theorem asserts that we can compute certain answers in polynomial time using a universal solution. It is a direct consequence of Theorem 5.1 and Lemma 5.2.

THEOREM 5.2. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a DEAC setting, I be a source instance and q be a CQAC query. Then, we can compute $certain_M(q, I)$ in PTIME for the following two cases:*

(a) *when there are no target constraints ($\Sigma_t = \emptyset$) and q has the homomorphism property.*

(b) *when all tgd-ACs in $\Sigma_{st} \cup \Sigma_t$ are full.*

An interesting observation is that, in some cases, the result of SAC-chase can be used to compute certain answers of a CQAC query, even if it is not a universal solution. One such case is stated in Theorem 5.3, for data exchange settings without arithmetic comparisons.

THEOREM 5.3. *Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting, where Σ_{st} consists of tgds and Σ_t consists of weakly acyclic tgds and egds. Let I be a source instance and q be a CQAC query with only open (closed) LSI (RSI) arithmetic comparisons. Then, the result of SAC-chase can be used to compute $certain_M(q, I)$ in PTIME.*

5.3 Discussion on the chase procedures

Chase is a useful tool for reasoning about dependencies (e.g.,[31, 8, 7, 18]). Most related to our work is the chase presented in [14]. In the sequel, we refer to this chase as *Simple chase*. We proceed to a detailed comparison of Simple chase, AC-chase and SAC-chase for classes of constraints.

When we have tgds for source-to-target constraints and tgds and egds for target constraints, then the SAC-chase reduces to the Simple chase. However the result they produce when they terminate is not a universal solution according to our DEAC setting definition. This is the reason it cannot be used to compute certain answers of CQAC queries (as noticed in [14]). The reason that it is not a DEAC universal solution is that some of its induced instances may not be solutions because one of the constraints may not be satisfied.

When the source-to-target constraints are tgds and we have no target constraints then the result of the SAC-chase (and hence the result of the Simple chase) is a universal solution in the DEAC sense and hence it can be used to compute certain answers for any CQAC query. The reason that now it is a universal solution is that any of its induced instances is guaranteed to satisfy the constraints since a) the lhs of each constraint is on a disjoint schema from the schema of the rhs of any constraint and b) the relations in instance K which are on the lhs schema contain only constants. For the same reason, the SAC-chase produces a universal solution when the source-to-target dependencies are tgd-ACs and there are no target dependencies. Notice that when there are no target dependencies (even with only tgds as source-to-target dependencies) the AC-chase still produces a tree. This is precisely the reason why the SAC-chase is a useful tool; in certain well-behaving cases it avoids the unnecessary complications of the AC-chase.

6. RELATED WORK AND CONCLUSIONS

Data exchange is an old problem [36, 9] and has recently regained the interest because of the two seminal papers [14, 15]. Based on these ideas, a schema mapping tool called *Clio* was implemented at the IBM Almaden Research Center [34, 35]. Research issues on data exchange have recently received attention for relational databases [9, 4, 16, 23, 17, 24, 27, 37] and XML databases [5].

Limited work exists on extending dependency theories for interpreted domains. Maher in [29] considers functional and tuple generating dependencies with a broad class of arithmetic comparisons (e.g., linear arithmetic). In [38] disjunctive constrained tuple generating dependencies were introduced and the query containment problem under those constraints was studied. Authors in [30] consider constrained tuple generating dependencies and a chase procedure is given, but no complexity issues are discussed. Conditional functional dependencies are introduced in [10] which extend functional dependencies with equality conditions. Most related to the dependencies introduced in this work are the constraint generating dependencies [6] defined over totally ordered domains. In the same work the implication problem was shown to be in co-NP for acgds, and in PTIME for acgds with no constraints on the antecedent. Tgd-ACs are considered implicitly in constraint checking in [21].

The notion of certain answers was introduced as the standard concept for query answering in incomplete databases [18, 33]. It then became popular for query answering in data integration [1, 25] and data exchange [14] settings. Computing certain answers of conjunctive queries with inequalities in a data exchange setting was shown to be in coNP in [14]. For unions of conjunctive queries with inequalities, where at most one inequality is allowed in every conjunctive query in the union, the problem is polynomial-time computable, whereas when two inequalities are used per conjunctive query in the union then, it is shown to be coNP-hard [14]. Abiteboul and Duschka [1] showed that there is a single conjunctive query with seven inequalities and a schema mapping with no target constraints for which computing the certain answers is a coNP-complete problem. Madry [28] proved the same result for a single conjunctive query with two inequalities. Containment tests for conjunctive queries [11] and conjunctive queries with comparisons [22, 19, 2, 3] provide some technical tools for studying data exchange in the absence or presence of comparisons.

Conclusions: In this work, we studied data exchange in the presence of arithmetic comparisons. We introduced two classes of dependencies (tgd-ACs and acgds) that allow for arithmetic comparison predicates and together they form a richer class than embedded implicational dependencies. We defined the DEAC setting as a data exchange setting where the specifications are expressed by tgd-ACs and acgds. We studied the complexity of the existence of solution problem and proved that it is in NP. Moreover, we showed that the new concept of universal solution can be used to compute the certain answers of unions of CQACs. The complexity of the problem of computing certain answers for unions of CQACs was shown to be coNP-complete. Finally, we developed polynomial results for (a) computing a universal solution and (b) query answering.

An interesting future direction is to consider discrete (instead of densely) totally ordered domains, such as the integers [26]. Interpreting the arithmetic comparisons over the integers changes the situation completely. For instance, consider the queries $q_1()$:-$r(X), 3 < X < 4$ and $q_2()$:-$r(X), 13 < X < 14$. If the arithmetic comparisons are interpreted over the integers, the answer to both queries evaluated on any database is *empty*. Hence, they are equivalent. However, if the arithmetic comparisons are interpreted over densely totally ordered domains then, the answer of q_1 on the database $D = \{r(3.5)\}$ is *true* whereas the answer of q_2 on D is *false*. We conjecture that most of the results presented in this paper will change significantly, while the initial definitions may easily carry over. Finally, AC-chase can be used to study the implication problem for tgd-ACs and acgds. Suppose $\Sigma = \{d_1, \ldots, d_k\}$ is a set of tgd-ACs and d is a tgd-AC. The implication problem tests whether Σ logically implies d. To study the implication problem we consider the lhs of d, viewed as an instance K, and chase K with Σ. Then, we check if the rhs of d holds on the result of the chase. Since K may contain a partial order, to chase K with Σ we need to consider all the induced t-instances. AC-chase provides the machinery for that.

7. REFERENCES

[1] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *PODS*, 1998.

[2] F. Afrati, C. Li, and P. Mitra. On containment of conjunctive queries with arithmetic comparisons. In *EDBT*, 2004.

[3] F. Afrati, C. Li, and P. Mitra. Rewriting queries using views in the presence of arithmetic comparisons. *TCS*, 368(1-2):88–123, 2006.

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

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

[6] M. Baudinet, J. Chomicki, and P. Wolper. Constraint-generating dependencies. *J. Comput. Syst. Sci.*, 59(1):94–115, 1999.

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

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

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

[10] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, 2007.

[11] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *STOC*, 1977.

[12] A. Deutsch and V. Tannen. Reformulation of XML queries and constraints. In *ICDT*, 2003.

[13] R. Fagin. Horn clauses and database dependencies. *J. ACM*, 29(4):952–985, 1982.

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

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

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

[17] G. Gottlob and A. Nash. Data exchange: Computing cores in polynomial time. In *PODS*, 2006.

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

[19] A. Gupta. *Partial Information Based Integrity Constraint Checking*. PhD thesis, Stanford, 1994.

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

[21] N. Ishakbeyoglu and Z. M. Ozsoyoglu. On the maintenance of implication integrity constraints. In *DEXA*, 1993.

[22] A. Klug. On conjunctive queries containing inequalities. *J. ACM*, 35(1):146–160, 1988.

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

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

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

[26] A. Levy, A. Rajaraman, and J. Ullman. Answering queries using limited external query processors. In *PODS*, 1996.

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

[28] A. Madry. Data exchange: on complexity of answering queries with inequalities. *Information Processing Letters*, 94(6):253–257, 2005.

[29] M. J. Maher. Constrained dependencies. *Theoretical Computer Science*, 173(1):113–149, 1997.

[30] M. J. Maher and D. Srivastava. Chasing constrained tuple-generating dependencies. In *PODS*, 1996.

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

[32] R. v. d. Meyden. The complexity of querying indefinite data about linearly ordered domains. In *PODS*, 1992.

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

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

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

[36] N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. Express: A data EXtraction, processing, amd REStructuring System. *ACM TODS*, 2(2):134–174, 1977.

[37] A. Vieilleribiere and M. D. Rougemont. Approximate data exchange. In *ICDT*, 2007.

[38] J. Wang, R. W. Topor, and M. J. Maher. Reasoning with disjunctive constrained tuple-generating dependencies. In *DEXA*, 2001.