# On Equality-Generating Dependencies in Ontology Querying - Preliminary Report

Andrea Calì<br/>1,3 and Andreas  ${\rm Pieris}^2$ 

<sup>1</sup>Dept. of Computer Science and Inf. Systems, Birkbeck University of London, UK <sup>2</sup>Computing Laboratory, University of Oxford, UK <sup>3</sup>Oxford-Man Institute of Quantitative Finance, University of Oxford, UK

andrea@dcs.bbk.ac.uk
andreas.pieris@comlab.ox.ac.uk

**Abstract.** In ontology-based data access, data are queried through an *ontology* that offers a representation of the domain of interest. In this context, correct answers are those entailed by the logical theory constituted by the data and the ontology. Traditional database constraints like tuple-generating dependencies (TGDs) and equality-generating dependencies (EGDs) are a useful tool for ontology specification. However, their interaction usually leads to intractability or undecidability of query answering; *separability* is the notion that captures the lack of interaction between TGDs and EGDs. In this paper we exhibit a novel and general sufficient condition for separability, in the case where the ontology is expressed with inclusion dependencies (a subclass of TGDs) and EGDs.

## 1 Introduction

Answering queries over ontologies has become an important problem in knowledge representation and databases. In ontology-enhanced database systems, an extensional relational database D is combined with an *ontological theory*  $\Sigma$  describing rules and constraints which derive new intensional data from the extensional data. A query is not just answered against the database D, but against the logical theory  $D \cup \Sigma$ . This problem has been addressed in several settings. For instance, the constraints of [1, 4, 6] are tailored to express Entity-Relationship schemata, while [18] deals with expressive constraints based on Answer Set Programming. The work [8] introduces and studies first-order constraints derived from a "light" version of F-logic [15], called *F-logic Lite*. Another relevant formalisms for knowledge bases, especially in the Semantic Web, is the *DL-lite family*; in [10, 17] tractable query answering techniques under DL-lite knowledge bases are presented.

In ontology-based query answering, a prominent family of languages, recently proposed, is the Datalog<sup>±</sup> family. In Datalog<sup>±</sup>, the ontological theory is expressed by means of rules of two kinds: *(i) tuple-generating dependencies* (TGDs), that is, (function-free) Horn rules enhanced with the possibility of having existentially quantified variables in the head; *(ii) equality-generating dependencies*, that is, (function-free) Horn rules with a single equality atom in the head. Several decidable and tractable  $Datalog^{\pm}$  languages have been studied [2, 3, 5, 7]. Even the least expressive  $Datalog^{\pm}$  languages, with some extra features which do not increase the complexity of query answering, are able to properly extend the DL-lite languages. This suggests that TGDs and EGDs, which are in fact "traditional" database constraints, are a powerful and flexible tool for ontology modeling. Consider the following example, adapted from [5].

Example 1. Consider the following relational schema.

dept(Dept\_ld, Mgr\_ld), emp(Emp\_ld, Dept\_ld, Area, Project\_ld), runs(Dept\_ld, Project\_ld), in\_area(Project\_ld, Area), project\_mgr(Emp\_ld, Project\_ld), external(Ext\_ld, Area, Project\_ld).

The fact that each department has an employee as manager can be expressed by the TGD

 $dept(V, W) \rightarrow \exists X \exists Y \exists Z emp(W, X, Y, Z).$ 

The following TGD expresses the fact that each employee works on some project that falls into his/her area of specialization ran by his/her department.

$$emp(V, W, X, Y) \rightarrow \exists Z \ dept(W, Z), \ runs(W, Y), \ in\_area(Y, X).$$

The fact that for each project run by some department there exists an external controller, specialized on the area of the project, that works on it can be expressed by the TGD

$$runs(W, X), in\_area(X, Y) \rightarrow \exists Z \ external(Z, Y, X).$$

With the EGD below, we specify that for each area, all project within it have the same manager.

$$\begin{array}{l} project\_mgr(X_1,Y_1), project\_mgr(X_2,Y_2),\\ in\_area(Y_1,Z), in\_area(Y_2,Z) \rightarrow X_1 = X_2. \end{array} \quad \blacksquare$$

In this paper we focus on the language of TGDs and EGDs, and we address the problem of the interaction between the two types of constraints. Notice that, when there is no limitation on how TGDs and EGDs interact, the conjunctive query answering problem is undecidable; in fact, it is undecidable already for *inclusion dependencies (IDs)* and key dependencies (KDs), two subclasses of TGDs and EGDs, respectively. For this reason the notion of separability was first proposed in [9]. A set  $\Sigma = \Sigma_T \cup \Sigma_E$ , where  $\Sigma_T$  and  $\Sigma_E$  are TGDs and EGDs, respectively, is said to be separable if, assuming the theory  $D \cup \Sigma$  to be consistent, for each database D, the answers to a conjunctive query Q under  $\Sigma$ and under  $\Sigma_T$  coincide. In other words, EGDs do not play any role in query answering, and queries can be answered by considering the TGDs only. Several conditions have been proposed to ensure separability (see Section 3); here we propose a sufficient condition in the case where the constraints are IDs together with general EGDs. We also discuss that this condition can be easily combined with the known condition for TGDs (and thus IDs) and FDs [5], hence identifying a more general sufficient condition. The result can be straightforwardly extended to *linear* TGDs and EGDs, where linear TGDs are a slight generalization of IDs consisting of TGDs with exactly one atom in the body; however, for clarity of exposition, we illustrate our results in the case of IDs.

We do believe that our preliminary results pave the way to the discovery of more general separability conditions between EGDs (or their restrictions) and decidable classes of TGDs such as guarded TGDs (guarded Datalog<sup>±</sup>) [2] or sticky sets of TGDs (sticky Datalog<sup>±</sup>) [5, 7].

# 2 Preliminaries

In this section we recall some basics on databases, queries, tuple-generating dependencies, equality-generating dependencies, and the chase procedure.

**General.** We define the following pairwise disjoint (infinite) sets of symbols: (i) a set  $\Gamma$  of constants (constitute the "normal" domain of a database), (ii) a set  $\Gamma_N$  of labeled nulls (used as placeholders for unknown values, and thus can be also seen as variables), and (iii) a set  $\Gamma_V$  of variables (used in queries and dependencies). Different constants represent different values (unique name assumption), while different nulls may represent the same value. A lexicographic order is defined on  $\Gamma \cup \Gamma_N$ , such that every value in  $\Gamma_N$  follows all those in  $\Gamma$ . Throughout the paper, we denote by **X** sequences of variables  $X_1, \ldots, X_k$ , where  $k \ge 0$ . Also, let [n] be the set  $\{1, \ldots, n\}$ , for any integer  $n \ge 1$ .

A relational schema  $\mathcal{R}$  (or simply schema) is a set of relational symbols (or predicates), each with its associated arity. A position r[i] (in a schema  $\mathcal{R}$ ) is identified by a predicate  $r \in \mathcal{R}$  and its *i*-th argument (or attribute). A term *t* is a constant, null, or variable. An atomic formula (or simply atom) has the form  $r(t_1, \ldots, t_n)$ , where *r* is an *n*-ary relation, and  $t_1, \ldots, t_n$  are terms. Conjunctions of atoms are often identified with the sets of their atoms. A database (instance) *D* for a schema  $\mathcal{R}$  is a (possibly infinite) set of atoms of the form  $r(\mathbf{t})$  (a.k.a. facts), where *r* is an *n*-ary predicate of  $\mathcal{R}$ , and  $\mathbf{t} \in (\Gamma \cup \Gamma_N)^n$ . We denote as r(D) the set  $\{\mathbf{t} \mid r(\mathbf{t}) \in D\}$ .

A substitution from one set of symbols  $S_1$  to another set of symbols  $S_2$ is a function  $h: S_1 \to S_2$  defined as follows:  $(i) \oslash$  is a substitution (empty substitution), (ii) if h is a substitution, then  $h \cup \{X \to Y\}$  is a substitution, where  $X \in S_1$  and  $Y \in S_2$ , and h does not already contain some  $X \to Z$  with  $Y \neq Z$ . If  $X \to Y \in h$  we write h(X) = Y. A homomorphism from a set of atoms  $A_1$  to a set of atoms  $A_2$ , both over the same schema  $\mathcal{R}$ , is a substitution  $h: \Gamma \cup \Gamma_N \cup \Gamma_V \to \Gamma \cup \Gamma_N \cup \Gamma_V$  such that: (i) if  $t \in \Gamma$ , then h(t) = t, and (ii) if  $r(t_1, \ldots, t_n)$  is in  $A_1$ , then  $h(r(t_1, \ldots, t_n)) = r(h(t_1), \ldots, h(t_n))$  is in  $A_2$ . If there are homomorphisms from  $A_1$  to  $A_2$  and vice-versa, then  $A_1$  and  $A_2$  are homomorphically equivalent. The notion of homomorphism naturally extends to conjunctions of atoms. Given a set of symbols S, two atoms  $\underline{a}_1$  and  $\underline{a}_2$  are S-*isomorphic* iff there exists a bijection h such that  $h(\underline{a}_1) = \underline{a}_2$ ,  $h^{-1}(\underline{a}_2) = \underline{a}_1$ , and h(X) = X, for each  $X \in S$ .

**Conjunctive Queries.** A conjunctive query (CQ) Q of arity n over a schema  $\mathcal{R}$  has the form  $q(\mathbf{X}) \leftarrow \varphi(\mathbf{X}, \mathbf{Y})$ , where  $\varphi(\mathbf{X}, \mathbf{Y})$  is a conjunction of atoms over  $\mathcal{R}$ ,  $\mathbf{X}$  and  $\mathbf{Y}$  are sequences of variables or constants in  $\Gamma$ , and q is an n-ary predicate that does not occur in  $\mathcal{R}$ .  $\varphi(\mathbf{X}, \mathbf{Y})$  is called the *body* of q, denoted as body(q). A *Boolean CQ (BCQ)* is a CQ of zero arity.

The answer to an n-ary CQ Q of the form  $q(\mathbf{X}) \leftarrow \varphi(\mathbf{X}, \mathbf{Y})$  over a database D, denoted as Q(D), is the set of all n-tuples  $\mathbf{t} \in \Gamma^n$  for which there exists a homomorphism  $h : \mathbf{X} \cup \mathbf{Y} \to \Gamma \cup \Gamma_N$  such that  $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$  and  $h(\mathbf{X}) = \mathbf{t}$ . A BCQ has only the empty tuple  $\langle \rangle$  as possible answer, in which case it is said that has positive answer. Formally, a BCQ Q has positive answer over D, denoted as  $D \models Q$ , iff  $\langle \rangle \in Q(D)$ , or, equivalently,  $Q(D) \neq \emptyset$ .

**Dependencies.** Given a schema  $\mathcal{R}$ , a tuple-generating dependency (TGD)  $\sigma$  over  $\mathcal{R}$  is a first-order formula  $\forall \mathbf{X} \forall \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}) \to \exists \mathbf{Z} \psi(\mathbf{X}, \mathbf{Z})$ , where  $\varphi(\mathbf{X}, \mathbf{Y})$  and  $\psi(\mathbf{X}, \mathbf{Z})$  are conjunctions of atoms over  $\mathcal{R}$ , called the *body* and the *head* of  $\sigma$ , denoted as  $body(\sigma)$  and  $head(\sigma)$ , respectively. Henceforth, to avoid notational clutter, we will omit the universal quantifiers in TGDs. Such  $\sigma$  is satisfied by a database D for  $\mathcal{R}$  iff, whenever there exists a homomorphism h such that  $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$ , there exists an extension h' of h (i.e.,  $h' \supseteq h$ ) where  $h'(\psi(\mathbf{X}, \mathbf{Z})) \subseteq D$ .

A restricted class of TGDs which is of special interest in this paper is the class of *inclusion dependencies (IDs)*. In fact, IDs are the simplest class of TGDs; they have just one body-atom and one head-atom, without repetition of variables neither in the body nor in the head.

An equality-generating dependency (EGD)  $\eta$  over  $\mathcal{R}$  is a first-order formula  $\forall \mathbf{X} \varphi(\mathbf{X}) \to X_i = X_j$ , where  $\varphi(\mathbf{X})$  is a conjunction of atoms over  $\mathcal{R}$ , called the body and denoted as  $body(\eta)$ , and  $X_i = X_j$  is an equality among variables of  $\mathbf{X}$ . Henceforth, for brevity, we will omit the universal quantifiers in EGDs. Such  $\eta$  is satisfied by a database D for  $\mathcal{R}$  iff, whenever there exists a homomorphism h such that  $h(\varphi(\mathbf{X})) \subseteq D$ , then  $h(X_i) = h(X_j)$ .

**CQ** Answering under Dependencies. We now define the notion of query answering under TGDs and EGDs. Given a database D for  $\mathcal{R}$ , and a set  $\Sigma$  of TGDs and EGDs over  $\mathcal{R}$ , the models of D w.r.t.  $\Sigma$ , denoted as  $mods(D, \Sigma)$ , is the set of all databases B such that  $B \models D \cup \Sigma$ , i.e.,  $B \supseteq D$  and B satisfies  $\Sigma$ . The answer to a CQ Q w.r.t. D and  $\Sigma$ , denoted as  $ans(Q, D, \Sigma)$ , is the set  $\{\mathbf{t} \mid \mathbf{t} \in Q(B), \text{ for each } B \in mods(D, \Sigma)\}$ . The answer to a BCQ Q w.r.t. Dand  $\Sigma$  is positive, denoted as  $D \cup \Sigma \models Q$ , iff  $ans(Q, D, \Sigma) \neq \emptyset$ . Note that query answering under general TGDs and EGDs is undecidable. In fact, this is true even in extremely simple cases such as that of IDs and keys [11].

We recall that the two problems of CQ and BCQ answering under TGDs and EGDs are LOGSPACE-equivalent [2]. Moreover, it is easy to see that the query output tuple problem (as a decision version of CQ answering) and BCQ evaluation are  $AC_0$ -reducible to each other. Thus, we henceforth focus only on the BCQ answering problem.

The Chase Procedure. The chase procedure (or simply chase) is a fundamental algorithmic tool introduced for checking implication of dependencies [16], and later for checking query containment [14]. Informally, the chase is a process of repairing a database w.r.t. a set of dependencies so that the resulted database satisfies the dependencies. We shall use the term chase interchangeably for both the procedure and its result. The chase works on an instance through the socalled TGD and EGD chase rules. The TGD chase rule comes in two different equivalent fashions: oblivious and restricted [2], where the restricted one repairs TGDs only when they are not satisfied. In the sequel, we focus on the oblivious one for better technical clarity. The chase rules follow.

<u>TGD CHASE RULE.</u> Consider a database D for a schema  $\mathcal{R}$ , and a TGD  $\sigma$ of the form  $\varphi(\mathbf{X}, \mathbf{Y}) \to \exists \mathbf{Z} \psi(\mathbf{X}, \mathbf{Z})$  over  $\mathcal{R}$ . If  $\sigma$  is *applicable* to D, i.e., there exists a homomorphism h such that  $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$ , then: (i) define  $h' \supseteq h$ such that  $h'(Z_i) = z_i$ , for each  $Z_i \in \mathbf{Z}$ , where  $z_i \in \Gamma_N$  is a "fresh" labeled null not introduced before, and following lexicographically all those introduced so far, and (ii) add to D the set of atoms  $h'(\psi(\mathbf{X}, \mathbf{Z}))$ , if not already in D.

EGD CHASE RULE. Consider a database D for a schema  $\mathcal{R}$ , and an EGD  $\eta$ of the form  $\varphi(\mathbf{X}) \to X_i = X_j$  over  $\mathcal{R}$ . If  $\eta$  is applicable to D, i.e., there exists a homomorphism h such that  $h(\varphi(\mathbf{X})) \subseteq D$  and  $h(X_i) \neq h(X_j)$ , then: (i) if  $h(X_i)$ and  $h(X_j)$  are both constants of  $\Gamma$ , then there is a hard violation of  $\eta$ , and the chase fails, otherwise (ii) replace each occurrence of  $h(X_j)$  with  $h(X_i)$ , if  $h(X_i)$ precedes  $h(X_j)$  in the lexicographic order, or vice-versa otherwise.

Given a database D and a set of dependencies  $\Sigma = \Sigma_T \cup \Sigma_E$ , where  $\Sigma_T$  are TGDs and  $\Sigma_E$  are EGDs, the chase algorithm for D and  $\Sigma$  consists of an exhaustive application of the chase rules in a breadth-first fashion, which leads to a (possibly infinite) database. Roughly, the chase of D w.r.t.  $\Sigma$ , denoted as  $chase(D, \Sigma)$ , is the (possibly infinite) instance constructed by iteratively applying (i) the TGD chase rule once, and (ii) the EGD chase rule as long as it is applicable (i.e., until a fixed point is reached). A formal definition of the chase algorithm is given, e.g., in [5].

*Example 2.* Let  $\mathcal{R} = \{r, s\}$ . Consider the set  $\Sigma$  of TGDs and EGDs over  $\mathcal{R}$  constituted by the TGDs  $\sigma_1 = r(X, Y) \to \exists Z r(Z, X), s(Z)$  and  $\sigma_2 = r(X, Y) \to r(Y, X)$ , and the EGD  $\eta = r(X, Y), r(X', Y) \to X = X'$ . Let D be the database for  $\mathcal{R}$  consisting of the single atom r(a, b). During the construction of  $chase(D, \Sigma)$  we first apply  $\sigma_1$ , and we add the atoms  $r(z_1, a), s(z_1)$ , where  $z_1$  is a "fresh" null of  $\Gamma_N$ . Moreover,  $\sigma_2$  is applicable and we add the atom r(b, a). Now, the EGD  $\eta$  is applicable and we replace each occurrence of  $z_1$  with the constant b; thus, we get the atom s(b). We continue by applying exhaustively the chase rules as explained above.

The (possibly infinite) chase of D w.r.t.  $\Sigma$  is a *universal model* of D w.r.t.  $\Sigma$ , i.e., for each database  $B \in mods(D, \Sigma)$ , there exists a homomorphism from  $chase(D, \Sigma)$  to B [13, 12]. Using this fact it can be shown that the chase is a formal tool for query answering under TGDs and EGDs. In particular, given a BCQ  $Q, D \cup \Sigma \models Q$  iff  $chase(D, \Sigma) \models Q$ , providing that the chase does not fail. If the chase fails, then the set of models of D w.r.t.  $\Sigma$  is empty, and  $D \cup \Sigma \models Q$  trivially.

# 3 Overview of Decidable Classes

As already mentioned, the interaction of general TGDs and EGDs has been proved to lead to undecidability of query answering. A semantic notion that ensures decidability of query answering under sets of TGDs and EGDs, providing that the set of TGDs falls in a decidable class, is *separability* [9,3]. Roughly speaking, separability guarantees that queries can be answered by considering only the set of TGDs (apart from an initial check whether the chase fails); the formal definition is given in Section 4. Several sufficient syntactic conditions for separability have been proposed in the literature. In this section we give a brief overview of these conditions.

An early separable class of IDs and KDs, called *key-based*, was proposed in the seminal work of Johnson and Klug [14]. In short, given an ID  $\sigma$  of the form  $r(\mathbf{X}, \mathbf{Y}) \to \exists \mathbf{Z} s(\mathbf{X}, \mathbf{Z})$ , (i) the set of **X**-attributes of  $head(\sigma)$  must be strictly contained in the set of key attributes of the relation s, and also (ii) the **X**-attributes of  $body(\sigma)$  must be disjoint from the set of key attributes of the relation r.

As observed by Calì et al. [9], the first condition of key-based sets of IDs and keys, as explained above, can be relaxed so that the set of **X**-attributes of *head*( $\sigma$ ) can be also equal to the set of key attributes of *s*. Also, the second condition, which imposes a restriction on the bodies of the IDs, is not needed. In particular, the class of *non-key-conflicting* (*NKC*) *IDs* was defined: given an ID  $\sigma = r(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} s(\mathbf{X}, \mathbf{Z})$ , the set of **X**-attributes of *head*( $\sigma$ ) is not a proper superset of the set of key attributes of *s*. Note that NKC IDs capture the wellknown class of *foreign key dependencies*, which corresponds to the case where the set of **X**-attributes of *head*( $\sigma$ ) is equal to the set of key attributes of *s*.

The class of NKC IDs was generalized in [3] to the context of arbitrary (single-head) TGDs by defining the class of non-key-conflicting TGDs. Actually, the underlying idea is the same: given a TGD  $\sigma = \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} r(\mathbf{X}, \mathbf{Z})$ , the set of **X**-attributes of head( $\sigma$ ) is not a proper superset of the set of key attributes of r; moreover, each existentially quantified variable in head( $\sigma$ ) must occur only once. In [5], it was observed that the class of non-key-conflicting TGDs can be effortless extended to treat, not just keys, but functional dependencies.

The main reason due to which the above classes are separable is because, if the given database satisfies the set of EGDs, we know that it is not possible to apply an EGD during the chase procedure. The separable classes of IDs and KDs introduced in [4,6] in the context of Entity-Relationship schemata, instead, are such that KDs can be applied during the chase. The separable class of IDs and EGDs that we propose in this paper is actually a generalization of the classes of IDs and KDs introduced in [4,6].

#### 4 Separable IDs and EGDs

In this section we exhibit a sufficient syntactic condition for separability between a set of IDs and a set of EGDs. Before we proceed further, let us give the formal definition of separability [9, 3].

**Definition 1.** Consider a  $\Sigma_T$  of TGDs over a schema  $\mathcal{R}$ , and a set  $\Sigma_E$  of EGDs over  $\mathcal{R}$ . We say that the set  $\Sigma = \Sigma_T \cup \Sigma_E$  is separable if, for every database D for  $\mathcal{R}$ , either chase $(D, \Sigma)$  fails, or, chase $(D, \Sigma) \models Q$  iff chase $(D, \Sigma_T) \models Q$ , for every BCQ Q over  $\mathcal{R}$ .

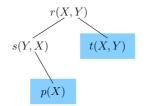
#### 4.1 Non-Conflicting Sets of IDs and EGDs

In this subsection we define when a set of IDs and EGDs is *non-conflicting*, and then establish that this condition is indeed sufficient for separability. Intuitively, the condition ensures that, if the chase does not fail, whenever "new" atoms (from the logical point of view) are created in the chase by the application of the EGD chase rule, atoms that are logically equivalent to the new ones are guaranteed to be generated also in the absence of the EGDs. This is sufficient to guarantee that EGDs do not have any impact on the chase with respect to query answering.

Before we proceed further, we need to give some preliminary definitions. First, we define the notion of *affected positions* of a relational schema w.r.t. a set of TGDs. Given a schema  $\mathcal{R}$ , and a set  $\Sigma$  of TGDs over  $\mathcal{R}$ , an *affected position* of  $\mathcal{R}$  w.r.t.  $\Sigma$  is defined inductively as follows. Let  $\pi_h$  be a position in the head of a TGD  $\sigma \in \Sigma$ . If an existentially quantified variable occurs at  $\pi_h$ , then  $\pi_h$  is affected w.r.t.  $\Sigma$ . If the same universally quantified variable X appears both in position  $\pi_h$ , and in the body of  $\sigma$  at affected positions only, then  $\pi_h$  is affected w.r.t.  $\Sigma$ . Intuitively speaking, the affected positions of a schema w.r.t. a set  $\Sigma$  of TGDs, are those positions at which a labeled null may occur during the construction of the chase under  $\Sigma$ .

A useful notion is the well-known query containment under TGDs. Given a set  $\Sigma$  of TGDs over a schema  $\mathcal{R}$ , and two CQs  $Q_1$  and  $Q_2$  over  $\mathcal{R}$ , we say that  $Q_1$  is contained in  $Q_2$  w.r.t.  $\Sigma$ , written  $Q_1 \subseteq_{\Sigma} Q_2$ , if  $Q_1(D) \subseteq Q_2(D)$ , for every database D for  $\mathcal{R}$  that satisfies  $\Sigma$ .

Consider now a set  $\Sigma_T$  of IDs over a schema  $\mathcal{R}$ , and an EGD  $\eta$  over  $\mathcal{R}$  of the form  $\varphi(\mathbf{X}) \to X_i = X_j$ , where  $\{X_i, X_j\} \subseteq \mathbf{X}$ ; we assume w.l.o.g. that  $\Sigma_T$ and  $\eta$  have no variables in common. Let  $\lambda$  be the substitution  $\{X_j \to X_i\}$ . The *derivation forest* for  $\eta$  under  $\Sigma_T$ , denoted as  $F_{\eta, \Sigma_T}$ , is constructed as follows. If at least one occurrence of the so-called *watched variable*  $X_i$  in  $\lambda(\varphi(\mathbf{X}))$  occurs at a non-affected position, then  $F_{\eta, \Sigma_T}$  is empty; otherwise, the roots of the forest are the atoms of  $\lambda(\varphi(\mathbf{X}))$ . Now, we iteratively apply the following step to every atom  $\underline{a}$  in the part of  $F_{\eta, \Sigma_T}$  constructed so far; let  $\mathcal{V}_{\eta}$  be the set of all variables appearing in the atoms of  $\varphi(\mathbf{X})$ . For each  $\sigma \in \Sigma_T$  (for which we assume w.l.o.g. that has no variables in common with any of the atoms in the part of  $F_{\eta, \Sigma_T}$  constructed so far), if there exists a homomorphism h such that  $h(head(\sigma)) = \underline{a}$ , and also



**Fig. 1.** The derivation forest  $F_{\eta, \Sigma_T}$  for Example 3.

- 1.  $h(body(\sigma))$  contains the watched variable  $X_i$ ,
- 2. all the occurrences of  $X_i$  in  $h(body(\sigma))$  occur at affected positions of  $\mathcal{R}$  w.r.t.  $\Sigma_T$ , and
- 3.  $h(body(\sigma))$  is not  $\mathcal{V}_{\eta}$ -isomorphic to some ancestor of  $\underline{a}$  in the part of  $F_{\eta, \Sigma_T}$  so far constructed,

then add  $h'(body(\sigma))$ , where  $h' \supseteq h$  maps the variables that occur in the body but not in the head of  $\sigma$  to their self, as a child of  $\underline{a}$  in  $F_{\eta,\Sigma_T}$ .

Intuitively, the atoms at the nodes of the derivation forest are representatives of "new" atoms (from the logical point of view, as explained above) that could be generated by applying the EGD chase rule in the chase. This because the forest is generated by a procedure similar to backward resolution which, starting from the root atoms, generates representatives of all possible atoms which could have generated such root atoms, and which are also affected by the EGD application. To guarantee that the new atoms are generated in the chase under the IDs alone, we use a condition based on containment of suitable conjunctive queries, as we shall explain below.

We are now ready to define formally when a set of IDs and EGDs is nonconflicting. Henceforth, for notational convenience, given a set  $\Sigma$  of dependencies, we will denote by  $\Sigma_T$  and  $\Sigma_E$  the set of IDs and EGDs, respectively.

**Definition 2.** Consider a set  $\Sigma$  of IDs and EGDs over a schema  $\mathcal{R}$ . We say that  $\Sigma$  is non-conflicting if, for each  $\eta \in \Sigma_E$  of the form  $\varphi(\mathbf{X}) \to X_i = X_j$ , the following condition holds. For each atom  $\underline{a}$  in  $F_{\eta, \Sigma_T}$ ,  $Q_1 \subseteq_{\Sigma_T} Q_2$ , where  $Q_1$  and  $Q_2$  are the conjunctive queries  $q(\mathbf{Y}) \leftarrow \varphi(\mathbf{X})$  and  $q(\mathbf{Y}) \leftarrow \underline{a}$ , respectively, where  $\mathbf{Y}$  are the variables that appear both in  $\varphi(\mathbf{X})$  and  $\underline{a}$ .

*Example 3.* Consider the set  $\Sigma$  consisting by the dependencies

$$\sigma_{1} : s(X, Y) \to r(Y, X)$$
  

$$\sigma_{2} : p(X) \to \exists Y s(Y, X)$$
  

$$\sigma_{3} : t(X, Y) \to r(X, Y)$$
  

$$\sigma_{4} : r(X, Y) \to s(Y, X)$$
  

$$\eta : r(X, Y), r(X, Z) \to Y = Z$$

The derivation forest  $F_{\eta, \Sigma_T}$  for  $\eta$  and  $\Sigma_T$  is depicted in Figure 1. Note that the shaded nodes are not part of the forest. The atom t(X, Y) is not added since

the watched variable Y occurs at a non-affected position, while the atom p(X) is not added since it does not contain the watched variable Y. It is not difficult to see that  $Q_1 \subseteq_{\Sigma_T} Q_2$  and  $Q_1 \subseteq_{\Sigma_T} Q_3$ , where

$$\begin{array}{l} Q_1:q(Y)\leftarrow r(X,Y),r(X,Z)\\ Q_2:q(Y)\leftarrow r(X,Y)\\ Q_3:q(Y)\leftarrow s(Y,X). \end{array}$$

Consequently,  $\Sigma_E$  is non-conflicting with  $\Sigma_T$ . This, as we shall state in Theorem 2, implies separability of  $\Sigma$ . Let us give an informal explanation of why this holds. If we apply  $\eta$  to two generic atoms of the form  $r(\zeta_1, \zeta_2)$ ,  $r(\zeta_1, \zeta_3)$  in the chase under  $\Sigma$  (with  $\{\zeta_1, \zeta_2, \zeta_3\} \subseteq \Gamma \cup \Gamma_N$ , and such that the chase does not fail due to such application), we get  $r(\zeta_1, \zeta_2)$ . The condition  $Q_1 \subseteq_{\Sigma_T} Q_2$  ensures that the same atom is generated from  $r(\zeta_1, \zeta_2)$ ,  $r(\zeta_1, \zeta_3)$  also in the chase under  $\Sigma_T$  only, that is, the application of  $\eta$  does not create new atoms. Notice that in this particular case the containment holds trivially as the result of the application of  $\eta$  is a mere elimination of an atom. As for  $Q_1 \subseteq_{\Sigma_T} Q_3$ , suppose now to have the atoms of the form  $s(\zeta_3, \zeta_1)$  and  $r(\zeta_1, \zeta_2)$  in the chase under  $\Sigma$ . By virtue of  $\sigma_1$ , we also have  $r(\zeta_1, \zeta_3)$ , and by applying  $\eta$  we have to replace  $\zeta_3$  with  $\zeta_2$ , and this creates the atom  $s(\zeta_2, \zeta_1)$ . The condition  $Q_1 \subseteq_{\Sigma_T} Q_3$  guarantees that  $s(\zeta_2, \zeta_1)$  is generated in the chase under  $\Sigma_T$  only.

**Identifying Non-Conflicting Sets.** Let us now establish that the derivation forest of an EGD under a set of IDs is always finite.

**Proposition 1.** Consider a set  $\Sigma_T$  of IDs over a schema  $\mathcal{R}$ , and an EGD  $\eta$  over  $\mathcal{R}$ . The derivation forest of  $\eta$  under  $\Sigma_T$  is finite.

Proof. It suffices to show that on a certain path P of the derivation forest of  $\eta$  under  $\Sigma_T$  only finitely many non- $\mathcal{V}_\eta$ -isomorphic atoms can appear. Let  $\eta$  be of the form  $\varphi(\mathbf{X}) \to X_i = X_j$ , and  $\lambda = \{X_j \to X_i\}$ . Observe that, two atoms  $\underline{a}$  and  $\underline{b}$  of P are  $\mathcal{V}_\eta$ -isomorphic iff  $\underline{a}^* = \underline{b}^*$ , where  $\underline{a}^*$  and  $\underline{b}^*$  are the atoms obtained by replacing in  $\underline{a}$  and  $\underline{b}$ , respectively, the variables that do not occur in  $\lambda(\varphi(\mathbf{X}))$  with the "don't care" character " $\star$ ". Therefore, the maximum number of non- $\mathcal{V}_\eta$ -isomorphic atoms that we can have on P is  $|\mathcal{R}| \cdot (|S| + 1)^w$ , where w is the maximum arity over all predicates of  $\mathcal{R}$ , and S is the set of symbols that can appear on P, that is, the variables and constants that appear in the root node of P, and the constants that occur in  $\Sigma_T$ . Since both  $\mathcal{R}$  and  $\Sigma_T$  are finite, the claim follows.

Since the conjunctive query containment problem under the class of IDs is decidable [14], we immediately get that the non-conflicting condition as defined above is decidable. In particular, given a set  $\Sigma$  of IDs and EGDs, the problem of deciding whether  $\Sigma$  is non-conflicting is PSPACE-complete. The upper bound is obtained by exhibiting a simple non-deterministic polynomial space algorithm, while the lower bound is established by providing a reduction from the conjunctive query containment problem under IDs, which is PSPACE-hard [14].

**Theorem 1.** Consider a set  $\Sigma$  of IDs and EGDs over a schema  $\mathcal{R}$ . The problem whether  $\Sigma$  is non-conflicting is PSPACE-complete.

**Soundness and Completeness.** We now establish that non-conflicting sets of IDs and EGDs are indeed separable. Before we proceed further, let us establish an auxiliary technical lemma.

**Lemma 1.** Consider a non-conflicting set  $\Sigma$  of IDs and EGDs over a schema  $\mathcal{R}$ . If chase $(D, \Sigma)$  does not fail, then there exists a homomorphism h such that  $h(chase(D, \Sigma)) \subseteq chase(D, \Sigma_T)$ , for every database D for  $\mathcal{R}$ .

Proof (sketch). The proof is by induction on the number of applications of the (TGD or EGD) chase rule. We need to show that, for each  $k \ge 0$ , there exists a homomorphism  $h_k$  such that  $h_k(chase^{[k]}(D, \Sigma)) \subseteq chase(D, \Sigma_T)$ , where  $chase^{[k]}(D, \Sigma)$  is the initial finite part of the chase obtained by applying k times either the TGD or the EGD chase rule. The base step is trivial since  $chase^{[0]}(D, \Sigma) = D$ , and therefore  $h_0$  is the identity homomorphism. Now, suppose that during the k-th application, the TGD chase rule is applied due to an ID  $\sigma \in \Sigma_T$ , which implies that there exists a homomorphism  $\mu$  that maps  $body(\sigma)$  to  $chase^{[k-1]}(D, \Sigma)$ . The homomorphism  $h_k$  can be defined easily by exploiting the homomorphisms  $h_{k-1}$  and  $\mu$ .

The interesting case is when during the k-th application, the EGD chase rule is applied due to an EGD  $\eta$  of the form  $\varphi(\mathbf{X}) \to X_i = X_j$ . Thus, there exists a homomorphism  $\mu$  such that  $\mu(\varphi(\mathbf{X})) \subseteq chase^{[k-1]}(D, \Sigma)$ , and  $\mu(X_i) \neq \mu(X_j)$ . W.l.o.g. we assume the during the application of the EGD chase rule the substitution  $\lambda = \{\mu(X_j) \to \lambda(X_i)\}$  is applied. Consider an atom  $\underline{a} \in chase^{[k]}(D, \Sigma) \setminus chase^{[k-1]}(D, \Sigma)$ , i.e., was obtained by applying the EGD chase rule. Assume that  $\underline{a} = \lambda(\underline{a}')$ , where  $\underline{a}' \in chase^{[k-1]}(D, \Sigma)$ . We are going to show that  $h_{k-1}(\underline{a}) \in chase(D, \Sigma_T)$ . We proceed by identifying the following two cases.

Suppose first that  $\underline{a}'$  is the ancestor in the chase graph (i.e., the graph where the nodes are the atoms of  $chase(D, \Sigma)$ , and there exists an edge from  $\underline{a}$  to  $\underline{b}$  if  $\underline{b}$  is obtained from  $\underline{a}$  by a single TGD chase rule application) of some atom of  $\mu(\varphi(\mathbf{X}))$ . Clealry,  $\mu(X_i)$  occurs in  $\underline{a}'$ . Moreover, since by hypothesis  $chase(D, \Sigma)$ does not fail,  $\mu(X_j)$  is a null of  $\Gamma_N$ , and thus all the occurrences of  $\mu(X_j)$ in  $\underline{a}'$  appear at affected positions. Furthermore, by induction hypothesis,  $h_{k-1}$ maps  $\mu(\varphi(\mathbf{X}))$  into  $chase(D, \Sigma_T)$ . Since  $\Sigma$  is non-conflicting we get that there exists a homomorphism  $h_{\underline{a}} \supseteq h_{k-1} \circ \mu$  such that  $h_{\underline{a}}(\underline{a}) \in chase(D, \Sigma_T)$ . Observe that  $h_{\underline{a}} \subseteq h_{k-1}$  and  $\mu(\underline{a}) = \underline{a}$ . Therefore,  $h_{\underline{a}}(\underline{a}) = h_{k-1}(\underline{a})$  which implies that  $h_{k-1}(\underline{a}) \in chase(D, \Sigma_T)$ .

Finally, suppose that  $\underline{a}'$  is not the ancestor of any atom of  $\mu(\varphi(\mathbf{X}))$ , but there exists an atom  $\underline{a}'' \in chase^{[k-1]}(D, \Sigma)$  which is the ancestor of  $\underline{a}'$ , and also  $\underline{a}''$  is the ancestor of some atom of  $\mu(\varphi(\mathbf{X}))$ . Since, by the previous case,  $h_{k-1}(\underline{a}'') \in chase(D, \Sigma_T)$ , it is straightforward to see that  $h_{k-1}(\underline{a}') \in chase(D, \Sigma_T)$ .

The desired homomorphism is eventually  $h = \bigcup_{i=0}^{\infty} h_i$ .

Our main result follows by exploiting the above technical lemma.

**Theorem 2.** Consider a set  $\Sigma$  over a schema  $\mathcal{R}$ . If  $\Sigma$  is non-conflicting, then it is also separable.

Proof. Let D be a database for  $\mathcal{R}$  such that  $chase(D, \Sigma)$  does not fail. Clearly, by construction,  $chase(D, \Sigma)$  satisfies all the dependencies in  $\Sigma$ . Therefore,  $chase(D, \Sigma) \in mods(D, \Sigma) \subseteq mods(D, \Sigma_T)$ . Since  $chase(D, \Sigma_T)$  is a universal model of D w.r.t.  $\Sigma_T$  we immediately get that there exists a homomorphism hsuch that  $h(chase(D, \Sigma_T)) \subseteq chase(D, \Sigma)$ . On the other hand, Lemma 1 implies that there exists a homomorphism h' such that  $h'(chase(D, \Sigma)) \subseteq chase(D, \Sigma_T)$ Due to the existence of h and h' we get that  $chase(D, \Sigma)$  and  $chase(D, \Sigma_T)$  are homomorphically equivalent. Consequently, for every BCQ Q over  $\mathcal{R}$ , it holds that  $chase(D, \Sigma) \models Q$  iff  $chase(D, \Sigma_T) \models Q$ , and the claim follows.  $\Box$ 

Let us say that our non-conflicting condition can be combined with existing techniques in order to capture additional cases that involve functional dependencies which are not triggered during the construction of the chase. In particular, it can be combined with the non-conflicting notion proposed in [5] for general TGDs (and thus IDs) and FDs. In this case, we say that an EGD is non-conflicting with a set of IDs if the condition given in [5] is satisfied, or the non-conflicting condition defined above is satisfied.

For simplicity reasons, in this work we considered only IDs. However, the non-conflicting condition can be extended to the slightly more general class of *linear* TGDs, i.e., TGDs with just one body-atom, where repetition of variables is allowed both in the body and in the head. Roughly, this can be achieved by modifying the non-conflicting condition in such a way that, instead of a homomorphism that maps the head of a TGD  $\sigma$  to an atom <u>a</u> of the derivation forest, we need that  $head(\sigma)$  and <u>a</u> unify. Then, during the construction of the derivation forest, we exploit the most general unifier of  $head(\sigma)$  and <u>a</u>.

## 5 Conclusions

In this paper we have addressed the problem of separability between TGDs and EGDs in the context of ontological query answering. We have exhibited a sufficient, syntactically checkable condition for separability for the case of IDs and general EGDs. The result can be straightforwardly extended to linear TGDs and EGDs. We remind the reader that the condition is sufficient but not necessary. Deciding separability of sets of IDs and general EGDs is, in fact, undecidable.

Notice that, by the results of [3], answering conjunctive queries under linear TGDs is PSPACE-complete in combined complexity (complexity when both data and constraints are considered as input) and in  $AC_0^{-1}$  in data complexity (complexity where only data are considered as input). Moreover, a query Q can be evaluated against a database D under a set of linear TGDs  $\Sigma_T$  by rewriting it

<sup>&</sup>lt;sup>1</sup> The complexity class of recognizing words in languages defined by constant-depth Boolean circuits with (unlimited fan-in) AND and OR gates.

into a union of conjunctive query  $Q_{\Sigma_T}$  (expressible in SQL), and then evaluating  $Q_{\Sigma_T}$  directly over D.

By exploiting the results of this paper, it is possible to show that the complexity of conjunctive query answering under non-conflicting sets of linear TGDs and EGDs is the same as in the case of linear TGDs alone. Thus, as a corollary we get that the complexity of conjunctive query answering under non-conflicting sets of linear TGDs and EGDs is PSPACE-complete and in  $AC_0$  in combined and data complexity, respectively.

## References

- A. Calì, D. Calvanese, G. D. Giacomo, and M. Lenzerini. Accessing data integration systems through conceptual schemas. In *Proc. of ER*, pages 270–284, 2001.
- 2. A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *Proc. of KR*, pages 70–80, 2008.
- A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *Proc. of PODS*, pages 77–86, 2009.
- A. Calì, G. Gottlob, and A. Pieris. Tractable query answering over conceptual schemata. In Proc. of ER, pages 175–190, 2009.
- A. Calì, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. PVLDB, 3(1):554–565, 2010.
- 6. A. Calì, G. Gottlob, and A. Pieris. Query answering under expressive entityrelationship schemata. In *Proc. of ER*, pages 347–361, 2010.
- A. Calì, G. Gottlob, and A. Pieris. Query answering under non-guarded rules in Datalog+/-. In Proc. of RR, pages 175–190, 2010.
- A. Calì and M. Kifer. Containment of conjunctive object meta-queries. In Proc. of VLDB, pages 942–952, 2006.
- A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of PODS*, pages 260–271, 2003.
- D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-lite family. J. Autom. Reasoning, 39(3):385–429, 2007.
- 11. A. K. Chandra and M. Y. Vardi. The implication problem for functional and inclusion dependencies. *SIAM Journal of Computing*, 14:671–677, 1985.
- A. Deutsch, A. Nash, and J. B. Remmel. The chase revisisted. In *Proc. of PODS*, pages 149–158, 2008.
- R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
- D. S. Johnson and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. J. Comput. Syst. Sci., 28(1):167–189, 1984.
- M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and framebased languages. J. ACM, 42(4):741–843, 1995.
- D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. ACM Trans. Database Syst., 4(4):455–469, 1979.
- A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. J. Data Semantics, 10:133–173, 2008.
- M. Simkus and T. Eiter. DNC: Decidable non-monotonic disjunctive logic programs with function symbols. In Proc. of LPAR, pages 514–530, 2007.