A General Datalog-Based Framework for Tractable Query Answering over Ontologies (Extended Abstract)*

Andrea Calì[2,1], Georg Gottlob[1,2], and Thomas Lukasiewicz[1,‡]

[1] Computing Laboratory, University of Oxford, UK
firstname.lastname@comlab.ox.ac.uk
[2] Oxford-Man Institute of Quantitative Finance, University of Oxford, UK

1 Introduction

Ontologies play a key role in the Semantic Web [4], data modeling, and information integration [16]. Recent trends in ontological reasoning have shifted from decidability issues to tractability ones, as e.g. reflected by the work on the *DL-Lite* family of tractable description logics (DLs) [11, 19]. An important result of these works is that the main variants of *DL-Lite* are not only decidable, but that answering (unions of) conjunctive queries for these logics is in LOGSPACE, and actually in AC_0, in data complexity (i.e., the complexity where both the query and the constraints are fixed), and query answering in *DL-Lite* is FO-rewritable [11]. As observed in [18], the lack of value creation makes plain Datalog not very well suited for ontological reasoning with inclusion axioms either. It is thus natural to extend Datalog in order to nicely accommodate ontological axioms and constraints such as those expressible in the *DL-Lite* family.

This paper proposes and studies variants of Datalog that are suited for efficient ontological reasoning, and, in particular, for tractable ontology-based query answering. We introduce the Datalog$^{\pm}$ family of Datalog variants, which extend plain Datalog by the possibility of existential quantification in rule heads, and by a number of other features, and, at the same time, restrict the rule syntax in order to achieve tractability. In the Datalog$^{\pm}$ family, rules are a restricted form of *tuple-generating dependencies (TGDs)* and *equality-generating dependencies (EGDs)*. More specifically, in *guarded Datalog*$^{\pm}$ rules are *guarded TGDs (GTGDs)*, i.e., TGDs where a single atom in the body that contains all the universally quantified variables; in *linear Datalog*$^{\pm}$ rules are *linear TGDs (LTGDs)*, i.e., TGDs that have a single atom in the body. We characterize the data complexity of query answering for both the above sublanguages of Datalog$^{\pm}$. We show that query answering is in PTIME-complete and in AC_0 (which is the complexity of evaluating fixed first-order formulas over a database or finite structure), when the query is fixed, for fixed sets of GTGDs and LTGDs, respectively.

Further Results. In [7], we enrich Datalog$^{\pm}$ by adding negative constraints, showing that their introduction does not increase the complexity of query answering in Datalog$^{\pm}$.

* This work is a short version of [7].

‡ Alternative address: Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, 1040 Wien, Austria.

As a second extension, we then add *non-conflicting keys*, which are special EGDs that do not interact with TGDs, and thus also do not increase the complexity of query answering in Datalog$^{\pm}$. We deal only with *keys*, since this suffices to capture the most common tractable ontology languages in the literature. The class of *non-conflicting keys* is a generalization of the one in [9]. We also show that the Datalog$^{\pm}$ family is able to express the most common tractable ontology languages, in particular, the *DL-Lite* family [11] (see Section 5 for an example) and F-Logic Lite [8]. We finally present an extension of Datalog$^{\pm}$ with stratified negation. We provide a canonical model and a perfect model semantics, and we show that they coincide. We thus provide a natural stratified negation for query answering over ontologies, which has been an open problem to date, since it is in general based on several strata of infinite models.

Related Work. The results of the present paper are related to but very different from the results in [6], where complexity issues of GTGDs as well as of another class, called *weakly guarded TGDs*, were first studied. There, it was shown that the combined complexity of query answering and fact inference with GTGDs is 2-EXPTIME complete, and it was noted that the data complexity is polynomial, which is refined by the present paper. Extensions of Datalog that allow the introduction of values not appearing in the active domain of the input database have been proposed in the literature [1, 5, 10, 9]; the introduction of such values is usually called *value invention*. Other works integrate Datalog with DL knowledge bases, which is a different perspective.

2 Preliminaries

We briefly recall some basics on databases, queries, TGDs, and the chase.

Databases and Queries. We assume (i) an infinite universe of *data constants* Δ (which constitute the "normal" domain of a database), (ii) an infinite set of *(labeled) nulls* Δ_N (used as "fresh" Skolem terms, which are placeholders for unknown values, and can thus be seen as variables), and (iii) an infinite set of variables $\mathcal{X}$ (used in dependencies and queries). Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. We assume a lexicographic order on $\Delta \cup \Delta_N$, with every symbol in Δ_N following all symbols in Δ. We denote by $\mathbf{X}$ sequences of variables $X_1, \ldots, X_k$ with $k \geqslant 0$. We assume a *relational schema* $\mathcal{R}$, which is a finite set of *relation names* (or *predicates*). A *term* t is a constant, null, or variable. An *atomic formula* (or *atom*) $\mathbf{a}$ has the form $P(t_1, ..., t_n)$, where P is an n-ary predicate, and $t_1, ..., t_n$ are terms. We denote by $pred(\mathbf{a})$ (resp., $dom(\mathbf{a})$) the set of all its arguments (resp., its predicate). These notations naturally extend to sets and conjunctions of atoms. Conjunctions of atoms are often identified with the sets of their atoms. A *database (instance)* D for $\mathcal{R}$ is a (possibly infinite) set of atoms with predicates from $\mathcal{R}$ and arguments from $\Delta \cup \Delta_N$. Such D is *ground* iff it contains only atoms with arguments from Δ. A *conjunctive query (CQ)* over $\mathcal{R}$ has the form $Q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$, where $\Phi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms with the variables $\mathbf{X}$ and $\mathbf{Y}$. A *Boolean CQ (BCQ)* over $\mathcal{R}$ is a CQ of the form $Q()$. Answers to CQs and BCQs are defined via *homomorphisms*, which are mappings $\mu\colon \Delta \cup \Delta_N \cup \mathcal{X} \rightarrow \Delta \cup \Delta_N \cup \mathcal{X}$ such that (i) $c \in \Delta$ implies $\mu(c) = c$, (ii) $c \in \Delta_N$ implies $\mu(c) \in \Delta \cup \Delta_N$, and (iii) μ is naturally extended to atoms, sets of atoms, and conjunctions of atoms. The set of all

answers to a CQ $Q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$ over a database D, denoted $Q(D)$, is the set of all tuples $\mathbf{t}$ over Δ for which there exists a homomorphism $\mu \colon \mathbf{X} \cup \mathbf{Y} \to \Delta \cup \Delta_N$ such that $\mu(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$ and $\mu(\mathbf{X}) = \mathbf{t}$. The *answer* to a BCQ $Q()$ over D is *Yes*, denoted $D \models Q$, iff $Q(D) \neq \varnothing$.

TGDs. Given a relational schema $\mathcal{R}$, a *tuple-generating dependency (TGD)* σ is a first-order formula of the form $\forall \mathbf{X} \forall \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y}) \to \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$, where $\Phi(\mathbf{X}, \mathbf{Y})$ and $\Psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$, called the *body* and the *head* of σ, denoted $body(\sigma)$ and $head(\sigma)$, respectively. We usually omit the universal quantifiers in TGDs. Such σ is satisfied in a database D for $\mathcal{R}$ iff, whenever there is a homomorphism h that maps the atoms of $\Phi(\mathbf{X}, \mathbf{Y})$ to atoms of D, there exists an extension h' of h that maps the atoms of $\Psi(\mathbf{X}, \mathbf{Z})$ to atoms of D. All sets of TGDs are finite here. The notion of *query answering* under TGDs is defined as follows. For a set of TGDs Σ on $\mathcal{R}$, and a database D for $\mathcal{R}$, the set of *models* of D given Σ, denoted $mods(\Sigma, D)$, is the set of all databases B such that $B \models D \cup \Sigma$. The set of *answers* to a CQ Q on D given Σ, denoted $ans(Q, \Sigma, D)$, is the set of all tuples $\mathbf{a}$ such that $\mathbf{a} \in Q(B)$ for all $B \in mods(\Sigma, D)$. The *answer* to a BCQ Q over D given Σ is *Yes*, denoted $D \cup \Sigma \models Q$, iff $ans(Q, \Sigma, D) \neq \varnothing$. Note that query answering under general TGDs is undecidable [3], even when the schema and TGDs are fixed [6]. Query answering under a certain class $\mathcal{C}$ of TGDs is said to be *FO-rewritable* if, for every given CQ Q and for every given set Σ of TGDs in $\mathcal{C}$, there exists a first order query Q_{FO} such that, for every instance D, it holds $Q_{FO}(D) = ans(Q, \Sigma, D)$. We recall that the two problems of CQ and BCQ evaluation under TGDs are LOGSPACE-equivalent [12, 15, 14, 13]. Moreover, it is easy to see that the query output tuple (QOT) problem (as a decision version of CQ evaluation) and BCQ evaluation are AC_0-reducible to each other. Henceforth, we thus focus only on the BCQ evaluation problem. All complexity results carry over to the other problems. We also recall that query answering under TGDs is equivalent to query answering under TGDs with only singleton atoms in their heads [6]. In the sequel, w.l.o.g., every TGD has a singleton atom in its head.

The TGD Chase. The *chase* was introduced for checking implication of dependencies [17], and later also for checking query containment [15]. It is a procedure for repairing a database relative to a set of dependencies, so that the result satisfies the dependencies. By "chase", we refer both to the chase procedure and to its output. The TGD chase works on a database through so-called TGD *chase rules* (for an extended chase with also equality-generating dependencies (EGDs), see [7]). The TGD chase rule comes in two flavors: *oblivious* and *restricted*, where the restricted one repairs only unsatisfied TGDs. We focus on the oblivious one, since it makes proofs technically simpler. The *(oblivious) TGD chase rule* defined below is the building block of the chase.

TGD CHASE RULE. Consider a database D for a relational schema $\mathcal{R}$, and a TGD σ on $\mathcal{R}$ of the form $\Phi(\mathbf{X}, \mathbf{Y}) \to \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$. Then, σ is *applicable* to D if there exists a homomorphism h that maps the atoms of $\Phi(\mathbf{X}, \mathbf{Y})$ to atoms of D. Let σ be applicable, and h_1 be a homomorphism that extends h as follows: for each $X_i \in \mathbf{X}$, $h_1(X_i) = h(X_i)$; for each $Z_j \in \mathbf{Z}$, $h_1(Z_j) = z_j$, where z_j is a "fresh" null, i.e., $z_j \in \Delta_N$, z_j does not occur in D, and z_j lexicographically follows all other nulls already introduced. The application of σ adds to D the atom $h_1(\Psi(\mathbf{X}, \mathbf{Z}))$ if not already in D. ■

The notion of the *(derivation) level* of an atom in a TGD chase is defined as follows. Let D be the *initial* database from which the chase is constructed. Then: (1) The atoms in D have level 0. (2) Let a TGD $\Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$ be applied at some point in the construction of the chase, and let h and h_1 be as in the TGD chase rule. If the atom with highest level among those in $h_1(\Phi(\mathbf{X}, \mathbf{Y}))$ has level k, then the added atom $h_1(\Psi(\mathbf{X}, \mathbf{Z}))$ has level $k + 1$. The chase algorithm for Σ and D consists of an exhaustive application of the TGD chase rule in a breadth-first (level-saturating) fashion, which leads as result to a (possibly infinite) *chase* for Σ and D, denoted $chase(\Sigma, D)$. The *chase of level up to* $k \geqslant 0$ for Σ and D, denoted $chase^k(\Sigma, D)$, is the set of all atoms in $chase(\Sigma, D)$ of level at most k. The (possibly infinite) chase relative to TGDs is a *universal model*, i.e., for every $B \in mods(\Sigma, D)$, there exists a homomorphism that maps $chase(\Sigma, D)$ onto B [13, 6]. This implies that $Q(chase(\Sigma, D)) = ans(Q, \Sigma, D)$.

3 Guarded Datalog$^{\pm}$

We now introduce guarded Datalog$^{\pm}$ as a class of special TGDs that show computational data tractability, while being at the same time expressive enough to model ontologies. BCQs relative to such TGDs can be evaluated on a finite part of the chase of constant size in the data complexity.

A TGD σ is *guarded* iff it contains an atom in its body that contains all universally quantified variables of σ. The leftmost such atom is the *guard atom* (or *guard*) of σ. The non-guard atoms in the body of σ are the *side atoms* of σ.

Example 3.1. The TGD $r(X, Y), s(Y, X, Z) \rightarrow \exists W\, s(Z, X, W)$ is guarded (via the guard $s(Y, X, Z)$), while the TGD $r(X, Y), r(Y, Z) \rightarrow r(X, Z)$ is not guarded.

Note that sets of guarded TGDs (with single-atom heads) are theories in the guarded fragment of first-order logic [2]. In the sequel, let $\mathcal{R}$ be a relational schema, let D be a database for $\mathcal{R}$, and let Σ be a set of guarded TGDs on $\mathcal{R}$. We first give some preliminary definitions as follows. The *chase graph* for Σ and D is the directed graph consisting of $chase(\Sigma, D)$ as the set of nodes and having an arc from $\mathbf{a}$ to $\mathbf{b}$ iff $\mathbf{b}$ is obtained from $\mathbf{a}$ and possibly other atoms by a one-step application of a TGD $\sigma \in \Sigma$. Here, we mark $\mathbf{a}$ as *guard* iff $\mathbf{a}$ is the guard of σ. The *guarded chase forest* for Σ and D is the restriction of the chase graph for Σ and D to all atoms marked as guards and their children. The *subtree* of an atom $\mathbf{a}$ in this forest, denoted $subtree(\mathbf{a})$, is the restriction of the forest to all successors of $\mathbf{a}$. The *type* of an atom $\mathbf{a}$, denoted $type(\mathbf{a})$, is the set of all atoms $\mathbf{b}$ in $chase(\Sigma, D)$ that have only constants and nulls from $\mathbf{a}$ as arguments. Informally, the type of $\mathbf{a}$ is the set of all atoms that determine the subtree of $\mathbf{a}$ in the guarded chase forest.

Example 3.2. Consider the TGDs $\sigma_1 : r_1(X, Y), r_2(Y) \rightarrow \exists Z r_1(Z, X)$ and $\sigma_2 : r_1(X, Y) \rightarrow r_2(X)$, applied on an instance $D = \{r_1(a, b), r_2(b)\}$. The first part of the (infinite) guarded chase forest for $\{\sigma_1, \sigma_2\}$ and D is shown in Fig. 3, where every arc is labeled with the applied TGD.

Given a finite set $S \subseteq \Delta \cup \Delta_N$, two sets of atoms A_1 and A_2 are *S-isomorphic* (or *isomorphic* if $S = \varnothing$) iff there exists a bijection $\beta\colon A_1 \cup dom(A_1) \to A_2 \cup dom(A_2)$ such that (i) β and β^{-1} are homomorphisms, and (ii) $\beta(c) = c = \beta^{-1}(c)$ for all $c \in S$. Two atoms a_1 and a_2 are *S-isomorphic* (or *isomorphic* if $S = \varnothing$) iff $\{a_1\}$ and $\{a_2\}$ are S-isomorphic. The notion of S-isomorphism (or isomorphism if $S = \varnothing$) is naturally extended to more complex structures, such as pairs of two subtrees (V_1, E_1) and (V_2, E_2) of the guarded chase forest, and two pairs $(\mathbf{b}_1, S_1)$ and $(\mathbf{b}_2, S_2)$, where $\mathbf{b}_1$ and $\mathbf{b}_2$ are atoms, and S_1 and S_2 are sets of atoms. The following lemma shows that if two atoms have S-isomorphic types, then their subtrees are also S-isomorphic.

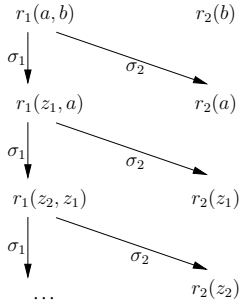

Fig. 1. Guarded chase forest in Example 3.2.

Lemma 3.1. *Let S be a set of constants and nulls, and $\mathbf{a}_1$ and $\mathbf{a}_2$ be atoms from $chase(\Sigma, D)$ with S-isomorphic types. Then, the subtree of $\mathbf{a}_1$ is S-isomorphic to the subtree of $\mathbf{a}_2$.*

The next lemma provides an upper bound for the number of all non-T-isomorphic pairs consisting of an atom and a type.

Lemma 3.2. *Let w be the maximal arity of a predicate in $\mathcal{R}$, $\delta = (2w)^w \cdot 2^{(2w)^{w \cdot |\mathcal{R}|}}$, and $\mathbf{a} \in chase(\Sigma, D)$. Let P be a set of pairs $(\mathbf{b}, S)$, each consisting of an atom $\mathbf{b}$ and a type S of atoms $\mathbf{c}$ with arguments from $\mathbf{a}$ and new nulls. If $|P| > \delta$, then P contains at least two $dom(\mathbf{a})$-isomorphic pairs.*

Using the above two lemmas, we are now ready to prove that the atoms in the type of an atom $\mathbf{a}$ in the guarded chase forest cannot be generated at a depth of the guarded chase forest that exceeds the depth of the atom $\mathbf{a}$ by a value that depends only on the TGDs. Here, the *guarded depth* of an atom $\mathbf{a}$ in the guarded chase forest for Σ and D, denoted $depth(\mathbf{a})$, is the length of the path from D to $\mathbf{a}$ in the forest. Note that this is generally different from the derivation level.

Lemma 3.3. *Let $\mathbf{a}$ be a guard in the chase graph for Σ and D, and $\mathbf{b} \in type(\mathbf{a})$. Then, $depth(\mathbf{b}) \leqslant depth(\mathbf{a}) + k$, where k depends only on Σ.*

We next show that BCQs can be evaluated using only a finite, initial portion of the guarded chase forest, whose size is determined by the TGDs only. Here, the *guarded chase* of level up to $k \geqslant 0$ for Σ and D, denoted $g\text{-}chase^k(\Sigma, D)$, is the set of all atoms in the forest of depth at most k.

Lemma 3.4. *Let Q be a BCQ over $\mathcal{R}$. If there exists a homomorphism μ that maps Q into $chase(\Sigma, D)$, then there exists a homomorphism λ that maps Q into $g\text{-}chase^k(\Sigma, D)$, where k depends only on Q and $\mathcal{R}$.*

The following definition captures a general property, where also the whole derivation of the query atoms are contained in the finite, initial portion of the guarded chase forest, whose size is determined by the TGDs only.

Definition 3.1. We say that Σ has the *bounded guard-depth property (BGDP)* iff, for every database D for $\mathcal{R}$ and for every BCQ Q, whenever there exists a homomorphism μ that maps Q into $chase(\Sigma, D)$, then there exists a homomorphism λ of this kind such that all ancestors of $\lambda(Q)$ in the chase graph for Σ and D are contained in $g\text{-}chase^{\gamma_g}(\Sigma, D)$, where γ_g depends only on Q and $\mathcal{R}$.

It is not difficult to prove, based on Lemmas 3.3 and 3.4, that guarded TGDs enjoy the BGDP. Hence, BCQ answering under guarded TGDs is in PTIME in data complexity, since by the existence of the homomorphism λ in Lemma 3.4 and in Definition 3.1, we obtain $Q(g\text{-}chase^{\gamma_g}(\Sigma, D)) = Q(chase(\Sigma, D)) = ans(Q, \Sigma, D)$. The following theorem summarizes these and other results from [7].

Theorem 3.1. *Let $\mathcal{R}$ be a relational schema, D be a database for $\mathcal{R}$, Σ be a finite set of guarded TGDs on $\mathcal{R}$, and Q be a BCQ Q over $\mathcal{R}$. We have that deciding $D \cup \Sigma \models Q$ is* PTIME*-complete in data complexity. It can be decided in linear time in data complexity in case Q is atomic.*

4 Linear Datalog$^{\pm}$

We now introduce linear Datalog$^{\pm}$ as a variant of guarded Datalog$^{\pm}$, where we prove that query answering is even FO-rewritable in the data complexity. Nonetheless, linear Datalog$^{\pm}$ is still expressive enough for representing ontologies [7]. A TGD is *linear* iff it contains only a singleton body atom. Notice that linear Datalog$^{\pm}$ generalizes the well-known class of *inclusion dependencies*.

We first define the bounded derivation-depth property, which is strictly stronger than the bounded guard-depth property.

Definition 4.1. A set of TGDs Σ has the *bounded derivation-depth property (BDDP)* iff, for every database D for $\mathcal{R}$ and for every BCQ Q over $\mathcal{R}$, whenever $D \cup \Sigma \models Q$, then $chase^{\gamma_d}(\Sigma, D) \models Q$, where γ_d depends only on Q and $\mathcal{R}$.

Clearly, in the case of linear TGDs, for every $\mathbf{a} \in chase(\Sigma, D)$, the subtree of $\mathbf{a}$ is determined only by $\mathbf{a}$, instead of $type(\mathbf{a})$ (cf. Lemma 3.1). Therefore, for a single atom, its depth coincides with the number of applications of the TGD chase rule that are necessary to generate it. By this observation, as an immediate consequence of the fact that guarded TGDs enjoy the BGDP, we obtain that linear TGDs have the bounded derivation-depth property.

The next result shows that queries relative to TGDs with the bounded derivation-depth property are FO-rewritable.

Theorem 4.1. *Let $\mathcal{R}$ be a relational schema, Σ be a set of TGDs over $\mathcal{R}$, D be a database for $\mathcal{R}$, and Q be a BCQ over $\mathcal{R}$. If Σ enjoys the BDDP, then Q is FO-rewritable.*

As an immediate consequence of the fact that linear TGDs enjoy the BDDP and of Theorem 4.1, we have that BCQs are FO-rewritable in the linear case.

Corollary 4.1. *Let $\mathcal{R}$ be a relational schema, Σ be a set of linear TGDs over $\mathcal{R}$, D be a database for $\mathcal{R}$, and Q be a BCQ over $\mathcal{R}$. Then, Q is FO-rewritable.*

5 Ontology Querying

In [7], we have provided a translation of the DLs *DL-Lite*$_{\mathcal{F}}$, *DL-Lite*$_{\mathcal{R}}$, and *DL-Lite*$_{\mathcal{A}}$ of the *DL-Lite* family [11] into linear Datalog$^{\pm}$ with negative constraints and non-conflicting keys, called Datalog$^{\pm}_0$, and we have shown that the former are strictly less expressive than the latter. Note also that the other DLs of the *DL-Lite* family can be similarly translated to Datalog$^{\pm}_0$. The following example illustrates the translation.

Example 5.1. Consider the following sets of atomic concepts, abstract roles, and individuals, which represent (i) the classes of scientists, articles, conference papers, and journal papers, (ii) the binary relations "has author", "has first author", and "is author of", and (iii) two individuals, respectively:

$$\begin{aligned}\mathbf{A} &= \{\mathit{Scientist}, \mathit{Article}, \mathit{ConPaper}, \mathit{JouPaper}\},\\ \mathbf{R}_A &= \{\mathit{hasAuthor}, \mathit{hasFirstAuthor}, \mathit{isAuthorOf}\},\\ \mathbf{I} &= \{i_1, i_2\}.\end{aligned}$$

The following are concept inclusion axioms, which informally express that (i) conference and journal papers are articles, (ii) conference papers are not journal papers, (iii) every scientist has a publication, (iv) *isAuthorOf* relates scientists and articles:

$$\begin{aligned}&\mathit{ConPaper} \sqsubseteq \mathit{Article},\ \mathit{JouPaper} \sqsubseteq \mathit{Article},\\ &\mathit{ConPaper} \sqsubseteq \neg \mathit{JouPaper},\ \mathit{Scientist} \sqsubseteq \exists \mathit{isAuthorOf},\\ &\exists \mathit{isAuthorOf} \sqsubseteq \mathit{Scientist},\ \exists \mathit{isAuthorOf}^- \sqsubseteq \mathit{Article}.\end{aligned}$$

The following role inclusion and functionality axioms express that (v) *isAuthorOf* is the inverse of *hasAuthor*, and (vi) *hasFirstAuthor* is a functional binary relationship:

$$\mathit{isAuthorOf}^- \sqsubseteq \mathit{hasAuthor},\ \mathit{hasAuthor}^- \sqsubseteq \mathit{isAuthorOf},\ (\mathsf{funct}\ \mathit{hasFirstAuthor}).$$

The following concept and role memberships express that the individual i_1 is a scientist who authors the article i_2:

$$\mathit{Scientist}(i_1),\ \mathit{isAuthorOf}(i_1, i_2),\ \mathit{Article}(i_2).$$

Then, the concept inclusion axioms are translated to the following TGDs and constraints (where we identify atomic concepts and roles with their predicates):

$$\begin{aligned}&\mathit{ConPaper}(X) \rightarrow \mathit{Article}(X),\ \mathit{JouPaper}(X) \rightarrow \mathit{Article}(X),\\ &\mathit{ConPaper}(X) \rightarrow \neg \mathit{JouPaper}(X),\ \mathit{Scientist}(X) \rightarrow \exists Z\, \mathit{isAuthorOf}(X, Z),\\ &\mathit{isAuthorOf}(X, Y) \rightarrow \mathit{Scientist}(X),\ \mathit{isAuthorOf}(Y, X) \rightarrow \mathit{Article}(X).\end{aligned}$$

The role inclusion and functionality axioms are translated to these TGDs and EGDs:

$$\begin{aligned}&\mathit{isAuthorOf}(Y, X) \rightarrow \mathit{hasAuthor}(X, Y),\ \mathit{hasAuthor}(Y, X) \rightarrow \mathit{isAuthorOf}(X, Y),\\ &\mathit{hasFirstAuthor}(X, Y), \mathit{hasFirstAuthor}(X, Y') \rightarrow Y = Y'.\end{aligned}$$

The concept and role membership axioms are translated to the following database atoms (where we also identify individuals with their constants):

$$\mathit{Scientist}(i_1),\ \mathit{isAuthorOf}(i_1, i_2),\ \mathit{Article}(i_2).$$

Acknowledgments. The work of Andrea Calì and Georg Gottlob was supported by the EPSRC grant EP/E010865/1 "Schema Mappings and Automated Services for Data Integration". G. Gottlob, whose work was partially carried out at the Oxford-Man Institute of Quantitative Finance, gratefully acknowledges support from the Royal Society as the holder of a Royal Society-Wolfson Research Merit Award. Thomas Lukasiewicz's work was supported by the German Research Foundation under the Heisenberg Programme.

References

1. S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *J. Comput. Syst. Sci.*, 43(1):62–124, 1991.
2. H. Andréka, I. Németi, and J. van Benthem. Modal languages and bounded fragments of predicate logic. *J. Phil. Log.*, 27(3):217–274, 1998.
3. C. Beeri and M. Y. Vardi. The implication problem for data dependencies. In *Proc. ICALP-1981*, *LNCS* 115, pp. 73–85. Springer, 1981.
4. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Sci. Amer.*, 284:34–43, 2001.
5. L. Cabibbo. The expressive power of stratified logic programs with value invention. *Inf. Comput.*, 147(1):22–56, 1998.
6. A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *Proc. KR-2008*, pp. 70–80. AAAI Press, 2008. Revised version: http://benner.dbai.tuwien.ac.at/staff/gottlob/CGK.pdf.
7. A. Calì, G. Gottlob, and T. Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. In *Proc. PODS-2009*. ACM Press, 2009 (in press).
8. A. Calì and M. Kifer. Containment of conjunctive object meta-queries. In *Proc. VLDB-2006*, pp. 942–952, 2006.
9. A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. PODS-2003*, pp. 260–271, 2003.
10. F. Calimeri, S. Cozza, and G. Ianni. External sources of knowledge and value invention in logic programming. *Ann. Math. Artif. Intell.*, 50(3/4):333–361, 2007.
11. 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.
12. A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. STOC-1977*, pp. 77–90. ACM Press, 1977.
13. A. Deutsch, A. Nash, and J. B. Remmel. The chase revisited. In *Proc. PODS-2008*, pp. 149–158. ACM Press, 2008.
14. 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.
15. D. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28:167–189, 1984.
16. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. PODS-2002*, pp. 233–246. ACM Press, 2002.
17. D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.
18. P. F. Patel-Schneider and I. Horrocks. Position paper: A comparison of two modelling paradigms in the Semantic Web. In *Proc. WWW-2006*, pp. 3–12. ACM Press, 2006.
19. 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.