Tractable Query Answering over Ontologies with Datalog$^{\pm}$

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

Abstract. We present a family of expressive extensions of Datalog, called Datalog$^{\pm}$, as a new paradigm for query answering over ontologies. The Datalog$^{\pm}$ family admits existentially quantified variables in rule heads, and has suitable restrictions to ensure highly efficient ontology querying. In particular, we show that query answering under so-called *guarded Datalog$^{\pm}$* is PTIME-complete in data complexity, and that query answering under so-called *linear Datalog$^{\pm}$* is in AC_0 in data complexity. We also show how negative constraints and a general class of key constraints can be added to Datalog$^{\pm}$ while keeping ontology querying tractable. We then show that *linear Datalog$^{\pm}$*, enriched with a special class of key constraints, generalizes the well-known *DL-Lite* family of tractable description logics. Furthermore, the Datalog$^{\pm}$ family is of interest in its own right and can, moreover, be used in various contexts such as data integration and data exchange. This work is a short version of [8].

1 Introduction

Ontologies play a key role in the Semantic Web [4], data modeling, and information integration [19]. 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) [12, 22]. 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 [12]. As observed in [21], 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*

[‡] Alternative address: Institut für Informationssysteme, Technische Universität Wien, Austria.

Datalog$^{\pm}$, rules are *guarded TGDs (GTGDs)*, i.e., TGDs with a single atom in the body that contains all 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 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 and the set of GTGDs and LTGDs are fixed, respectively. We then enrich Datalog$^{\pm}$ by adding negative constraints, which are Horn clauses with a (not necessarily guarded) conjunction of atoms in their body and the truth constant *false*, denoted $\bot$, in the head. We show that the introduction of such constraints does not increase the complexity of query answering in Datalog$^{\pm}$. As a further extension, we 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 [10].

Further results. We have several other results that, for space reasons, we do not include here. We refer the reader to [8] for more details. A central result is that the Datalog$^{\pm}$ family is able to express the most common tractable ontology languages, in particular, the *DL-Lite* family [12] and F-Logic Lite [9]. In particular, it is interesting to see that linear Datalog$^{\pm}$ enriched with non-conflicting keys is expressive enough to capture the expressive power of the description logics *DL-Lite*$_{\mathcal{F}}$, *DL-Lite*$_{\mathcal{R}}$, and *DL-Lite*$_{\mathcal{A}}$, which are highly suitable for ontological modeling and reasoning. Finally, we are able to deal with 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. Note that the results of the present paper are related to but very different from the results in [7], where complexity issues of guarded TGDs 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 guarded TGDs is 2-EXPTIME complete, and it was noted that the data complexity is polynomial, which is now much 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, 11, 10]; the introduction of such values is usually called *value invention*. Other works integrate Datalog with description logic knowledge bases [16, 6, 23], 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 n-ary predicate, and $t_1, ..., t_n$ are terms. We denote by $dom(\mathbf{a})$ the set of all its arguments. 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 $\mu\colon \mathbf{X} \cup \mathbf{Y} \rightarrow \Delta \cup \Delta_N$ 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 $\forall \mathbf{X} \forall \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$, 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, 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 [7]. Query answering under a certain class $\mathcal{C}$ of TGDs is said to be *FO-rewritable* iff, 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 [13, 18, 17, 15]. 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 [7]. 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 the implication of dependencies [20], and later also for checking query containment [18]. It repairs a database relative to a set of dependencies, so that the result of the chase 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 [8]). The TGD chase rule comes in

two flavors: *oblivious* and *restricted*, where the restricted one repairs TGDs only when not satisfied. 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 $\Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$. 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 important 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 [15, 7], and thus $Q(chase(\Sigma, D)) = ans(Q, \Sigma, D)$.

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.

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 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 $\beta\colon A_1 \cup dom(A_1) \rightarrow A_2 \cup dom(A_2)$ 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.

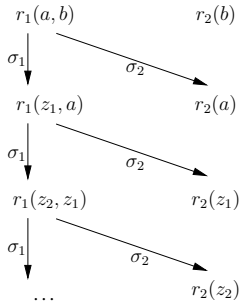

Fig. 1. Guarded chase forest in Example 3.2.

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.*

Proof. Let $k = (2w)^w \cdot 2^{(2w)^{w \cdot |\mathcal{R}|}}$, where w is the maximal arity of a predicate in $\mathcal{R}$. Suppose $depth(\mathbf{b}) > depth(\mathbf{a}) + k$ for one atom $\mathbf{b}$ in the type of $\mathbf{a}$. That is, the path P in the guarded chase forest leading to $\mathbf{b}$ has a length greater than k from the depth of $\mathbf{a}$. Suppose first $\mathbf{b}$ contains no nulls. By Lemma 3.2, there are two isomorphic atoms $\mathbf{h}$ and $\mathbf{h}'$ (in this order) on P with isomorphic types (see Fig. 3). By Lemma 3.1, the subtree of $\mathbf{h}$ is isomorphic to the subtree of $\mathbf{h}'$. But then $\mathbf{b}$ is also in the subtree of $\mathbf{h}$, on a path Q that is at least one edge shorter than P, which contradicts the assumption

that P is the path leading to $\mathbf{b}$. Suppose next the set of nulls N in $\mathbf{b}$ is nonempty, and consider the common predecessor $\mathbf{c}$ of $\mathbf{a}$ and $\mathbf{b}$ of largest depth. By Lemma 3.2, there are two N-isomorphic atoms $\mathbf{h}$ and $\mathbf{h}'$ (in this order) on P with N-isomorphic types (see Fig. 3). Since $\mathbf{b}$ is in the type of $\mathbf{a}$, it cannot contain new nulls compared to $\mathbf{a}$.

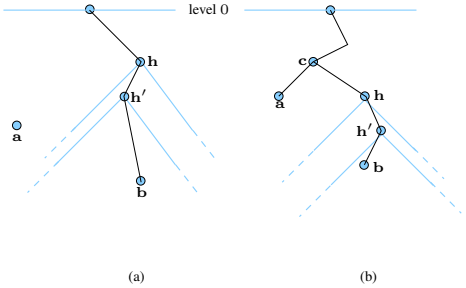

Fig. 2. Construction in Lemma 3.3's proof.

Since $\mathbf{c}$ is the common predecessor of $\mathbf{a}$ and $\mathbf{b}$ of largest depth, $\mathbf{b}$ also cannot contain new nulls compared to $\mathbf{c}$, and thus compared to $\mathbf{h}$ and $\mathbf{h}'$. So, $\mathbf{b}$ is in the types of both $\mathbf{h}$ and $\mathbf{h}'$. By Lemma 3.1, the subtree of $\mathbf{h}$ is N-isomorphic to the subtree of $\mathbf{h}'$. But then $\mathbf{b}$ is also in the subtree of $\mathbf{h}$, on a path Q that is at least one edge shorter than P, which contradicts the assumption that P is the path leading to $\mathbf{b}$. In summary, all atoms in the type of $\mathbf{a}$ can be obtained on paths of length at most k. □

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 $\mathcal{R}$.*

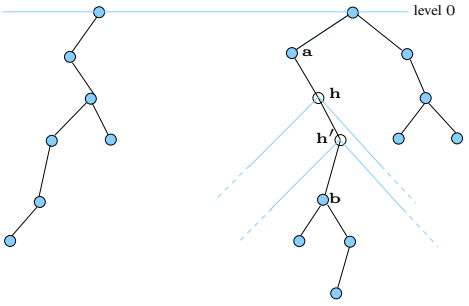

Fig. 3. Construction in Lemma 3.4's proof.

Proof. Let $k = n \cdot \delta$, where $n = |Q|$, $\delta = (2w)^w \cdot 2^{(2w)^{w \cdot |\mathcal{R}|}}$, and w is the maximal arity of a predicate in $\mathcal{R}$. Suppose there exists a homomorphism that maps Q into $chase(\Sigma, D)$. Let μ be a homomorphism of this kind such that $depth(\mu) = \sum_{q \in Q} depth(\mu(q))$ is minimal. We now show that $\mu(Q)$ is contained in $g\text{-}chase^k(\Sigma, D)$. Towards a contradiction, suppose the contrary. Consider the tree consisting of all atoms in $\mu(Q)$ and their ancestors in the guarded chase forest for Σ and D. As $\mu(Q)$ is not contained in $g\text{-}chase^k(\Sigma, D)$, this tree must contain a path P of length greater than δ of which all inner nodes (i.e., without start and end node) do not belong to $\mu(Q)$ and have no branches (i.e., have exactly one outgoing edge). Let $\mathbf{a}$ be the start node of P. By Lemma 3.2, there are two $dom(\mathbf{a})$-isomorphic atoms $\mathbf{h}$ and $\mathbf{h}'$ on P with $dom(\mathbf{a})$-isomorphic types. By Lemma 3.1, $subtree(\mathbf{h})$ is $dom(\mathbf{a})$-isomorphic to $subtree(\mathbf{h}')$. Thus, we can remove $\mathbf{h}$ and the path to $\mathbf{h}'$, obtaining a path that is at least one edge shorter. Let ι be the homomorphism mapping $subtree(\mathbf{h}')$ to $subtree(\mathbf{h})$, and let $\mu' = \mu \circ \iota$. Then, μ' is a homomorphism that maps Q into $chase(\Sigma, D)$ such that $depth(\mu') < depth(\mu)$, which contradicts μ being a homomorphism of this kind such that $depth(\mu)$ is minimal. This shows that $\mu(Q)$ is contained in $g\text{-}chase^k(\Sigma, D)$. □

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 [8].

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

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 (and thus in AC_0) in the data complexity. Nonetheless, linear Datalog$^{\pm}$ is still expressive enough for representing ontologies [8] in many cases. 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 Σ be a set of TGDs over a relational schema $\mathcal{R}$, 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 Extensions

In this section, we extend Datalog$^{\pm}$ with negative constraints and EGDs, which are both important when representing ontologies.

Negative Constraints. A *negative constraint* (or *constraint*) is a first-order formula of the form $\forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow \bot$, where $\Phi(\mathbf{X})$ is a (not necessarily guarded) conjunction of atoms. It is often also written as $\forall \mathbf{X} \Phi'(\mathbf{X}) \rightarrow \neg p(\mathbf{X})$, where $\Phi'(\mathbf{X})$ is obtained from $\Phi(\mathbf{X})$ by removing the atom $p(\mathbf{X})$. We usually omit the universal quantifiers.

Example 5.1. If the unary predicates c and c' represent two classes, we may use the constraint $c(X), c'(X) \rightarrow \bot$ to assert that the two classes have no common instances. Similarly, if additionally the binary predicate r represents a relationship, we may use $c(X), r(X, Y) \rightarrow \bot$ to enforce that no member of the class c participates to r.

Query answering on a database D under TGDs Σ_T and constraints Σ_C can be done effortlessly by additionally checking that every constraint $\sigma = \Phi(\mathbf{X}) \rightarrow \bot \in \Sigma_C$ is satisfied in D given Σ_T, each of which can be done by checking that the BCQ $Q_\sigma = \Phi(\mathbf{X})$ evaluates to false in D given Σ_T. We write $D \cup \Sigma_T \models \Sigma_C$ iff every $\sigma \in \Sigma_C$ is false in D given Σ_T. We thus obtain immediately the following result. Here, a BCQ Q is *true* in D given Σ_T and Σ_C, denoted $D \cup \Sigma_T \cup \Sigma_C \models Q$, iff (i) $D \cup \Sigma_T \models Q$ or (ii) $D \cup \Sigma_T \not\models \Sigma_C$ (as usual in DLs).

Theorem 5.1. *Let $\mathcal{R}$ be a relational schema, Σ_T and Σ_C be sets of TGDs and constraints on $\mathcal{R}$, respectively, D be a database for $\mathcal{R}$, and Q be a BCQ on $\mathcal{R}$. Then, $D \cup \Sigma_T \cup \Sigma_C \models Q$ iff (i) $D \cup \Sigma_T \models Q$ or (ii) $D \cup \Sigma_T \models Q_\sigma$ for some $\sigma \in \Sigma_C$.*

The next theorem shows that constraints do not increase the data and combined complexity of answering BCQs in the guarded and linear case. It follows from Theorem 5.1, by which the additional effort of deciding $D \cup \Sigma_T \models \Sigma_C$ can be done by answering $|\Sigma_C|$ BCQs Q_σ without constraints, where each query Q_σ has a size of $\|\sigma\| \leqslant \|\Sigma_C\|$, and in the data complexity, $|\Sigma_C|$ and $\|\sigma\|$ are constants. Here, $|\Sigma_C|$ denotes the cardinality of Σ_C, while $\|\Sigma_C\|$ denotes the input size of Σ_C. This additional effort does not increase the complexity of answering BCQs without constraints.

Theorem 5.2. *Let $\mathcal{R}$ be a relational schema, Σ_T and Σ_C be sets of TGDs and constraints on $\mathcal{R}$, respectively, D be a database for $\mathcal{R}$, and Q be a BCQ. Then, deciding $D \cup \Sigma_T \cup \Sigma_C \models Q$ in the guarded (resp., linear) case has the same data and the same combined complexity as deciding $D \cup \Sigma_T \models Q$ in the guarded (resp., linear) case.*

EGDs and Non-Conflicting Keys. An *equality-generating dependency* (or *EGD*) σ is a first-order formula of the form $\forall \mathbf{X}\, \Phi(\mathbf{X}) \rightarrow X_i = X_j$, where $\Phi(\mathbf{X})$, called the *body* of σ, is a conjunction of atoms, and X_i and X_j are variables from $\mathbf{X}$. We call $X_i = X_j$ the *head* of σ. Such σ is satisfied in a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h such that $h(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$, it holds that $h(X_i) = h(X_j)$.

Example 5.2. The EGD $r(X, Y), r(X, Y') \rightarrow Y = Y'$ expresses that the second argument of the binary predicate r functionally depends on the first argument of r.

The chase is naturally extended to databases under EGDs in addition to TGDs: The *chase* of a database D in the presence of two sets Σ_T and Σ_E of TGDs and EGDs, respectively, denoted $chase(\Sigma_T \cup \Sigma_E, D)$, is computed by iteratively applying (1) a single TGD once, according to the order specified above, and (2) the EGDs, as long as applicable (i.e., until a fixpoint is reached), where EGDs are applied as follows:

EGD CHASE RULE. Consider a database D for a relational schema $\mathcal{R}$, and an EGD σ on $\mathcal{R}$ of the form $\Phi(\mathbf{X}) \rightarrow X_i = X_j$. Such an EGD σ is *applicable* to D iff there exists a homomorphism $\eta\colon \Phi(\mathbf{X}) \rightarrow D$ such that $\eta(X_i)$ and $\eta(X_j)$ are different and not both constants. If $\eta(X_i)$ and $\eta(X_j)$ are different constants, then there is a *hard violation* of σ, and the chase *fails*. Otherwise, the result of the application of σ to D is the database $h(D)$ obtained from D by replacing every occurrence of a non-constant element $e \in \{\eta(X_i), \eta(X_j)\}$ in D by the other element e' (if e and e' are both nulls, then e precedes e' in the lexicographic order). ■

While adding negative constraints is computationally effortless, adding EGDs is more problematic. The interaction of TGDs and EGDs leads to undecidability of query answering even in simple cases, such that of functional and inclusion dependencies [14], or keys and inclusion dependencies (see, e.g., [10], where the proof of undecidability is done in the style of Vardi as in [18]). It can even be seen that a *fixed* set of EGDs and guarded TGDs can simulate a universal Turing machine, and thus query answering and even propositional ground atom inference is undecidable with such fixed sets of dependencies. For this reason, we consider a restricted class of EGDs, namely, *non-conflicting key dependencies* (or *NC keys*), which show a controlled interaction with TGDs (and negative constraints), such that they do not increase the complexity of answering BCQs. Nonetheless, this class is sufficient for ontology modeling.

We now first concentrate on the semantic notion of separability for EGDs, which formulates a controlled interaction between EGDs and TGDs (and negative constraints), such that the EGDs do not increase the complexity of answering BCQs. We then provide a sufficient syntactic condition for the separability of EGDs, where we transfer a result by [10] about *non-key-conflicting* inclusion dependencies to the more general setting of Datalog$^{\pm}$. In the context of description logics, general EGDs cannot be formulated, but only keys. Therefore, we mainly focus on keys here.

Definition 5.1. Let $\mathcal{R}$ be a relational schema, and Σ_T and Σ_E be sets of TGDs and EGDs on $\mathcal{R}$, respectively. Then, Σ_E is *separable* from Σ_T iff for every database D for $\mathcal{R}$, the following conditions (i) and (ii) are both satisfied:

(i) If there is a hard violation of an EGD in $chase(\Sigma_T \cup \Sigma_E, D)$, then there is also a hard violation of some EGD of Σ_E, when this EGD is directly applied to D.
(ii) If there is no chase failure, then for every BCQ Q, $chase(\Sigma_T \cup \Sigma_E, D) \models Q$ iff $chase(\Sigma_T, D) \models Q$.

The following result shows that adding separable EGDs to TGDs and constraints does not increase the data and combined complexity of answering BCQs in the guarded and linear case. It follows immediately from the fact that the separability implies that chase failure can be directly evaluated on D. For fixed (resp., variable) Σ_E, this can be done by evaluating a first-order formula on D (resp., in polynomial time in the size of D and Σ_E), which clearly does not increase the data (resp., combined) complexity of answering BCQs in the three cases.

Theorem 5.3. *Let $\mathcal{R}$ be a relational schema, Σ_T and Σ_C be sets of TGDs and constraints on $\mathcal{R}$, respectively, and D be a database for $\mathcal{R}$. Let Σ_E be a set of EGDs that is separable from Σ_T, and Q be a BCQ. Then, deciding $D \cup \Sigma_T \cup \Sigma_C \cup \Sigma_E \models Q$ in the guarded (resp., linear) case has the same data complexity and also the same combined complexity as deciding $D \cup \Sigma_T \models Q$ in the guarded (resp., linear) case.*

We next provide a sufficient syntactic condition for the separability of EGDs. We focus on *key dependencies (KDs, or simply keys)*: a key κ on a predicate r specifies a set $\mathbf{K}$ of key attribute positions of r; it is satisfied in a database D if no two atoms in D of the form $r(\mathbf{t})$ have the same values in all positions in $\mathbf{K}$. Clearly, KDs are special types of EGDs. The EGD chase rule in case of a KD applies to pairs of tuples that violate the KD. For example, a KD κ asserting that the key attribute of a ternary predicate r is the first one is applicable to the instance $D = \{r(a, b, z_3), r(a, z_2, z_1)\}$ (where the z_i's are nulls), and its application yields the new instance $\{r(a, b, z_1)\}$. The following definition generalizes the notion of "non-key-conflicting" dependency relative to a set of keys, introduced in [10], to the context of arbitrary TGDs.

Definition 5.2. Let κ be a key, and σ be a TGD of the form $\Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, r(\mathbf{X}, \mathbf{Z})$. Then, κ is *non-conflicting (NC)* with σ iff either (i) the key does not apply to the predicate r in the head of σ or (ii) the positions of κ in r are not a proper subset of the $\mathbf{X}$-positions in r in the head of σ, and every existentially quantified variable in σ appears only once in the head of σ. We say κ is *non-conflicting (NC)* with a set of TGDs Σ_T iff κ is NC with every $\sigma \in \Sigma_T$. We say a set of keys Σ_K is *non-conflicting (NC)* with Σ_T iff every $\kappa \in \Sigma_K$ is NC with Σ_T.

Example 5.3. Consider the four keys κ_1, κ_2, κ_3, and κ_4 defined by the key attribute sets $\mathbf{K}_1 = \{r[1], r[2]\}$, $\mathbf{K}_2 = \{r[1], r[3]\}$, $\mathbf{K}_3 = \{r[3]\}$, and $\mathbf{K}_4 = \{r[1]\}$, respectively, and the TGD $\sigma = p(X, Y) \rightarrow \exists Z\, r(X, Y, Z)$. Then, the head predicate of σ is r, and the set of positions in r with universally quantified variables is $\mathbf{H} = \{r[1], r[2]\}$. Observe that all keys but κ_4 are NC with σ, since only $\mathbf{K}_4 \subset \mathbf{H}$. Roughly, every atom added in a chase by applying σ would have a fresh null in some position in $\mathbf{K}_1$, $\mathbf{K}_2$, and $\mathbf{K}_3$, thus never firing κ_1, κ_2, and κ_3, respectively.

The following theorem shows that the NC property between keys and TGDs implies their separability. This generalizes a useful result of [10] on inclusion dependencies to the much larger class of all TGDs. The main idea behind the proof is roughly as follows. The NC condition between a key κ and a TGD σ assures that either (a) the application of σ in the chase generates an atom with a fresh null in a position of κ, and so the fact does not violate κ (see also Example 5.3), or (b) the $\mathbf{X}$-positions in the predicate r in the head of σ coincide with the key positions of κ in r, and thus any newly generated atom must have fresh distinct nulls in all but the key position, and may eventually be eliminated without violation. It then follows that the full chase does not fail. Since the new nulls are all distinct, it also contains a homomorphic image of the TGD chase. Therefore, the full chase is in fact homomorphically equivalent to the TGD chase.

Theorem 5.4. *Let $\mathcal{R}$ be a relational schema, Σ_T and Σ_K be sets of TGDs and keys, respectively, such that Σ_K is NC with Σ_T. Then, Σ_K is separable from Σ_T.*

We conclude this section by stating that in the NC case, keys do not increase the data and the combined complexity of answering BCQs under guarded (resp., linear) TGDs and constraints. This result is immediate by Theorems 5.4 and 5.3.

Corollary 5.1. *Let $\mathcal{R}$ be a relational schema, Σ_T and Σ_C be sets of TGDs and constraints on $\mathcal{R}$, respectively, and D be a database for $\mathcal{R}$. Let Σ_E be a set of EGDs that is NC with Σ_T, and Q be a BCQ. Then, deciding $D \cup \Sigma_T \cup \Sigma_C \cup \Sigma_E \models Q$ in the guarded (resp., linear) case has the same data complexity and also the same combined complexity as deciding $D \cup \Sigma_T \models Q$ in the guarded (resp., linear) case.*

6 Ontology Querying

In [8], 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 [12] into linear Datalog$^{\pm}$ with negative constraints and non-conflicting keys, called Datalog$_0^{\pm}$, and shown that the former are strictly less expressive than the latter. The other DLs of the *DL-Lite* family (including *DL-Lite*$_{\mathcal{F},\sqcap}$ and *DL-Lite*$_{\mathcal{R},\sqcap}$) can be similarly translated to Datalog$_0^{\pm}$. We now illustrate the translation.

Example 6.1. 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:

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

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:

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

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

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

Finally, the concept and role membership axioms $\mathit{Scientist}(i_1)$, $\mathit{isAuthorOf}(i_1, i_2)$, and $\mathit{Article}(i_2)$ express that the individual i_1 is a scientist who authors the 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):

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

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

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

Finally, the three concept and role membership axioms are translated to the three database atoms $\mathit{Scientist}(i_1)$, $\mathit{isAuthorOf}(i_1, i_2)$, and $\mathit{Article}(i_2)$, respectively (where we also identify individuals with their constants).

Acknowledgments. The work of A. Calì and G. Gottlob was supported by the EPSRC grant Number 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. T. 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. Philos. Logic*, 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. Am.*, 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. M. Cadoli, L. Palopoli, and M. Lenzerini. Datalog and description logics: Expressive power. In *Proc. DBPL-1997*, *LNCS* 1369, pp. 281–298. Springer, 1998.
7. 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.
8. 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.
9. A. Calì and M. Kifer. Containment of conjunctive object meta-queries. In *Proc. VLDB-2006*.
10. A. Calì, D. Lembo, R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. PODS-2003*, pp. 260–271. ACM Press, 2003.
11. 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.
12. 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.
13. A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. STOC-1977*, pp. 77–90. ACM Press, 1977.
14. A. K. Chandra and M. Y. Vardi. The implication problem for functional and inclusion dependencies is undecidable. *SIAM J. Comput.*, 14(3):671–677, 1985.
15. A. Deutsch, A. Nash, J. B. Remmel. The chase revisited. In *Proc. PODS-2008*, pp. 149–158.
16. F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. $\mathcal{AL}$-log: Integrating Datalog and description logics. *J. Intell. Inf. Syst.*, 10(3):227–252, 1998.
17. 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.
18. D. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.
19. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. PODS-2002*, pp. 233–246.
20. D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.
21. 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.
22. 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.
23. R. Rosati. On combining description logic ontologies and nonrecursive Datalog rules. In *Proc. RR-2008*, *LNCS* 5341, pp. 13–27. Springer, 2008.