# Datalog$^{\pm}$: A Unified Approach to Ontologies and Integrity Constraints

Andrea Calì[2,1] Georg Gottlob[1,2] Thomas Lukasiewicz[1,*]

[1]Computing Laboratory
University of Oxford
Wolfson Building, Parks Road
Oxford OX1 3QD
United Kingdom

[2]Oxford-Man Institute of Quantitative Finance
University of Oxford
Blue Boar Court, 9 Alfred Street
Oxford OX1 4EH
United Kingdom

{andrea.cali, georg.gottlob, thomas.lukasiewicz}@comlab.ox.ac.uk

## ABSTRACT

We report on a recently introduced family of expressive extensions of Datalog, called Datalog$^{\pm}$, which is a new framework for representing ontological axioms in form of integrity constraints, and for query answering under such constraints. Datalog$^{\pm}$ is derived from Datalog by allowing existentially quantified variables in rule heads, and by enforcing suitable properties in rule bodies, to ensure decidable and efficient query answering. We first present different languages in the Datalog$^{\pm}$ family, providing tight complexity bounds for all cases but one (where we have a low complexity $\text{AC}_0$ upper bound). We then show that such languages are general enough to capture the most common tractable ontology languages. In particular, we show that the *DL-Lite* family of description logics and *F-Logic Lite* are expressible in Datalog$^{\pm}$. We finally show how stratified negation can be added to Datalog$^{\pm}$ while keeping ontology querying tractable in the data complexity. Datalog$^{\pm}$ is a natural and very general framework that can be successfully employed in different contexts such as data integration and exchange. This survey mainly summarizes two recent papers.

## Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; H.2.4 [**Database Management**]: Systems—*Relational databases, rule-based databases, query processing*

## General Terms

Algorithms, Theory, Databases

## Keywords

Datalog, ontologies, Semantic Web, conjunctive queries, query evaluation, chase, dependencies, constraints, complexity, tractability

## 1. INTRODUCTION

In this paper, we survey the results of two recent works [9, 10] on a family of expressive extensions of Datalog, called Datalog$^{\pm}$, towards query answering over ontologies. Rules in Datalog$^{\pm}$ are rules in Datalog that additionally admit existentially quantified variables in the head, but on which restrictions are enforced on the body to guarantee desirable decidability and tractability properties.

Ontologies are fundamental to the development of the Semantic Web [7]. Ontologies are also gaining importance in databases, since they provide the necessary expressive power for tasks such as data modeling and data integration [31]. Among formalisms for representing ontologies, *description logics* (DLs) have played a prominent role in the last decade, especially in the Semantic Web. Currently, much research on DLs is directed towards scalable and efficient query answering over ontologies. In particular, the DLs of the *DL-Lite* family [13, 35] are the most common DLs in the Semantic Web and databases that allow for tractable query answering.

*Example 1.* A DL knowledge base consists of an ABox and a TBox. The knowledge that John is a manager can be expressed by the axiom $\mathit{Manager}(\mathit{john})$ in the ABox, while the knowledge that (i) every manager is an employee, (ii) every manager supervises someone, (iii) employees are not employers, and (iv) every employee is supervised by at most one manager, can be expressed by the axioms $\mathit{Manager} \sqsubseteq \mathit{Employee}$, $\mathit{Manager} \sqsubseteq \exists \mathit{Supervises}$, $\mathit{Employee} \sqsubseteq \neg \mathit{Employer}$, and $(\mathsf{funct}\, \mathit{Supervises}^{-})$ in the TBox, respectively. A Boolean conjunctive query (BCQ) asking whether John supervises someone is $\exists X\, \mathit{Supervises}(\mathit{john}, X)$.

The ABox and the TBox of DL knowledge bases are closely related to extensional databases and intensional sets of rules in Datalog [37, 15]. Datalog is a language with simple syntax and natural semantics, which make it easily un-

[*]Alternative address: Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, 1040 Wien, Austria; email: lukasiewicz@kr.tuwien.ac.at.

**Table 1: Summary of complexity results: variable set of TGDs.**

| BCQ type | Linear TGDs | Guarded TGDs | Weakly Guarded TGDs |
|---|---|---|---|
| **general** | PSPACE-complete | 2EXPTIME-complete | 2EXPTIME-complete |
| **bounded width, fixed, and atomic** | PSPACE-complete | 2EXPTIME-complete | 2EXPTIME-complete |

**Table 2: Summary of complexity results: fixed set of TGDs.**

| BCQ type | Linear TGDs | Guarded TGDs | Weakly Guarded TGDs |
|---|---|---|---|
| **general** | NP-complete | NP-complete | EXPTIME-complete |
| **bounded width, fixed, and atomic** | in $AC_0$ | PTIME-complete | EXPTIME-complete |

derstandable and usable. It has also been successful in many applications such as Web data extraction [25]. Furthermore, there are several optimization techniques for Datalog, which improve its applicability. However, not all ontological statements can be expressed in plain Datalog, as discussed in [34] and illustrated below.

*Example 2.* The knowledge that every manager is an employee can be expressed in Datalog as

$$manager(X) \rightarrow employee(X).$$

However, we cannot express in Datalog that every manager supervises someone, which requires an existential quantification in the head of Datalog rules:

$$manager(X) \rightarrow \exists Y\ supervises(X, Y).$$

Such assertions are special *tuple-generating dependencies (TGDs)*, which generalize *inclusion dependencies (IDs)*. Similarly, we cannot express in plain Datalog that employees are not employers, which requires a rule of the form:

$$employee(X), employer(X) \rightarrow \bot.$$

We also cannot express that every employee is supervised by at most one manager, which requires a rule of the form:

$$supervises(X, Y), supervises(X', Y) \rightarrow X = X'.$$

Such assertions are special *equality-generating dependencies (EGDs)*, which generalize *functional dependencies (FDs)*.

As the above example shows, an extension of Datalog by TGDs, negative constraints, and EGDs allows to express forms of ontological knowledge beyond plain Datalog. However, the interaction among TGDs and EGDs leads to undecidability of query answering [18, 33, 29, 16, 12]. Interesting subclasses of TGDs and EGDs have been studied in the literature, e.g., in the fundamental work [29]. All these works make use of a technique called *chase* [16, 6, 29], which amounts to "repairing" violations of TGDs and EGDs starting from a database, until a fixpoint is reached. The chase can be seen as a tableaux technique. One of the main difficulties behind all these approaches is the fact that such a fixpoint may be infinite. Several works in the literature consider classes of TGDs for which the chase terminates and therefore generates a finite instance; for example, the *weakly acyclic TGDs*, first introduced in [22] and extensively studied in [23].

The Datalog$^{\pm}$ family is an expressive extension of Datalog by integrity constraints as described in the above example towards query answering over ontologies. Datalog$^{\pm}$ deals with certain TGDs as rules (as well as negative constraints and EGDs) for which the chase does not terminate, but for which query answering is nonetheless decidable in general and tractable in many cases in the data complexity.

Datalog$^{\pm}$ is divided into the sublanguages of *guarded*, *linear*, and *weakly guarded* Datalog$^{\pm}$, which have *guarded*, *linear*, and *weakly guarded* TGDs as rules, respectively:

- Guarded TGDs are characterized by the presence of a *guard*, i.e., an atom in the body that contains all the (universally quantified) variables in the rule body.
- Weakly guarded TGDs have instead a *weak guard*, i.e., an atom in the body that contains all the (universally quantified) variables in the body that appear in the so-called *affected positions*, where a position is identified by the argument of an atom; informally, the affected positions are the only ones where newly invented values can appear during the chase process.
- Linear TGDs have exactly one atom in the body and one in the head; they are the class which is closest to inclusion dependencies among the three, and they correspond to inclusion dependencies with repetition of columns; however, we will show that they (along with negative constraints and certain EGDs) are enough to express the languages of the *DL-Lite* family.

We characterize the complexity of query answering for all three sublanguages of Datalog$^{\pm}$. The results are compactly summarized in Tables 1 and 2. In detail, query answering is complete for PSPACE and 2EXPTIME in the linear and the guarded / weakly guarded case, respectively, for variable sets of TGDs. It is complete for NP and EXPTIME in the linear / guarded and the weakly guarded case, respectively, for fixed sets of TGDs. Finally, query answering is in $AC_0$ (which is the complexity of evaluating fixed first-order formulas over a database or finite structure), PTIME-complete, and EXPTIME-complete when the query is of bounded width, fixed, or atomic in the linear, the guarded, and the weakly guarded case, respectively, for fixed sets of TGDs.

We then further enrich Datalog$^{\pm}$ with additional features, which serve to represent ontology languages. In particular, we add 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. Negative constraints are easy to handle, and we actually show that their introduction does not increase the complexity of query answering in Datalog$^{\pm}$. As a second 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 [12].

We next show that the Datalog$^\pm$ family is able to express the most common tractable ontology languages. More concretely, linear Datalog$^\pm$ with negative constraints and non-conflicting keys, called *Datalog*$_0^\pm$, can be used for query answering in *DL-Lite*$_\mathcal{A}$ in a natural and unified way. Here, Datalog$_0^\pm$ is strictly more expressive than *DL-Lite*$_\mathcal{A}$. Furthermore, weakly guarded Datalog$^\pm$ with a single non-conflicting key can be used for query answering in F-Logic Lite ontologies. Other DLs of the *DL-Lite* family [13] (such as *DL-Lite*$_\mathcal{F}$ and *DL-Lite*$_\mathcal{R}$) can be similarly translated to Datalog$_0^\pm$. Since *DL-Lite*$_\mathcal{R}$ is able to fully capture (the DL fragment of) RDF Schema [8], Datalog$_0^\pm$ is also able to fully capture (the DL fragment of) RDF Schema. Furthermore, note that the F-Logic formalism [30] is able to offer the *meta-querying* facility, i.e., the possibility of querying the schema as well as the data, in a homogeneous fashion. This is considered important for service discovery and in information integration in the Semantic Web, for example; however, the facilities for meta-querying have been usually awkward to the user, for example, the SQL system catalogue, or the Java Reflection API. In F-Logic Lite [11], rules are indeed TGDs, and our goal is to generalize the F-Logic Lite framework, and to have the results of [11] as a special case of Datalog$^\pm$.

We finally describe 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. By the results of Section 7, this also provides a natural stratified negation for the *DL-Lite* family.

## 2. PRELIMINARIES

In this section, we briefly recall some basics on databases, queries, (tuple- and equality-generating) dependencies, and the chase. We also provide some notions about the treewidth of a graph and of a database instance.

### 2.1 Databases and Queries

We assume (i) an infinite universe of *data constants* $\Delta$ (which constitute the "normal" domain of a database), (ii) an infinite set of *(labeled) nulls* $\Delta_N$ (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 $\Delta_N$ following all symbols in $\Delta$. We denote by $\mathbf{X}$ sequences of variables $X_1, \ldots, X_k$ with $k \geqslant 0$.

A *relational schema* $\mathcal{R}$ is a finite set of *relation names* (or *predicates*). A *position* $p[i]$ identifies the $i$-th argument of a predicate $p$. A *term* $t$ is a constant, null, or variable. An *atomic formula* (or *atom*) $\mathbf{a}$ has the form $p(t_1, \ldots, t_n)$, where $p$ is an $n$-ary predicate, and $t_1, \ldots, t_n$ are terms. We denote by *dom*$(\mathbf{a})$, *pred*$(\mathbf{a})$, and *vars*$(\mathbf{a})$ the sets of all arguments, the predicate symbol, and the set of all variables of an atom $\mathbf{a}$, respectively. This notation naturally extends to sets 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 $\Delta$. 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 having as arguments variables $\mathbf{X}$ and $\mathbf{Y}$ and constants (but no nulls). A *Boolean CQ (BCQ)* over $\mathcal{R}$ is a CQ having head predicate $q$ of arity 0 (i.e., no variables in $\mathbf{X}$). BCQs are often identified with the sets of their atoms. Answers to CQs and BCQs are defined via *homomorphisms*, which are mappings $\mu\colon \Delta \cup \Delta_N \cup \mathcal{X} \to \Delta \cup \Delta_N \cup \mathcal{X}$ 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) $\mu$ 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 $\Delta$ 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 $Q(D) \neq \emptyset$.

### 2.2 Dependencies

Given a relational schema $\mathcal{R}$, a *tuple-generating dependency* (or *TGD*) $\sigma$ 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 $\sigma$, respectively. Such $\sigma$ is satisfied in a database $D$ for $\mathcal{R}$ iff, whenever there exists a homomorphism $h$ such that $h(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$, there exists an extension $h'$ of $h$ such that $h'(\Psi(\mathbf{X}, \mathbf{Y})) \subseteq D$.

The notion of *query answering* under TGDs is defined as follows. For a set of TGDs $\Sigma$ on $\mathcal{R}$, and a database $D$ for $\mathcal{R}$, the set of *models* of $D$ given $\Sigma$, 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 $\Sigma$, 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 $\Sigma$ is *Yes*, denoted $D \cup \Sigma \models Q$, iff *ans*$(Q, \Sigma, D) \neq \emptyset$.

We recall that the two problems of CQ and BCQ evaluation under TGDs are LOGSPACE-equivalent [17, 29, 23, 21]. 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 [9]. This is shown by means of a transformation from general TGDs to TGDs with single-atom heads [9]. Moreover, the transformation preserves the properties of the classes of TGDs that we are going to consider in the rest of the paper (*weakly-guarded*, *guarded*, and *linear* TGDs). Therefore, all our results carry over to TGDs with multiple atoms in the head. In the sequel, we thus always assume w.l.o.g. that every TGD has a singleton atom in its head.

An *equality-generating dependency* (or *EGD*) $\sigma$ 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 $\sigma$, 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 $\sigma$. Such $\sigma$ 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 $h(X_i) = h(X_j)$.

The body (resp., head) of a TGD or EGD $\sigma$ is denoted by $body(\sigma)$ (resp., $head(\sigma)$). We usually omit the universal quantifiers in TGDs and EGDs, and all sets of TGDs and EGDs are finite here.

## 2.3 The Chase

The *chase* was introduced to enable checking implication of dependencies [32], and later also for checking query containment [29]. It is a procedure for repairing 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 chase works on a database through so-called TGD and EGD *chase rules*. The TGD chase rule comes in two flavors: *oblivious* and *restricted*, where the restricted one repairs TGDs only when they are 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 $\sigma$ on $\mathcal{R}$ of the form $\Phi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X},\mathbf{Z})$. Then, $\sigma$ is *applicable* to $D$ if there exists a homomorphism $h$ such that $h(\Phi(\mathbf{X},\mathbf{Y})) \subseteq D$. Let $\sigma$ 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 $\sigma$ 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 $k+1$.

Given a set of TGDs $\Sigma$ and a database $D$, the chase algorithm for $\Sigma$ 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 $\Sigma$ and $D$. Formally, the *chase* of $D$ relative to $\Sigma$, denoted $chase(\Sigma, D)$, is the database built by an iterative application of the TGD chase rule as follows: let $I_1, \ldots, I_k$ be all possible images of bodies of TGDs in $\Sigma$ relative to some homomorphism, and $\mathbf{a}_i$ be the atom with highest level in $I_i$; let $M$ be such that $\text{level}(\mathbf{a}_M) = \min_{1 \leq i \leq k}\{\text{level}(\mathbf{a}_i)\}$: among the possible applications of TGDs, choose the lexicographically first among those that utilize a homomorphism from the body of a TGD to $I_M$. For brevity, the application of the chase rule with a TGD $\sigma$ on a database $D$ is called application of $\sigma$ on $D$. The *chase of level up to* $k \geqslant 0$ for $\Sigma$ and $D$, denoted $chase^k(\Sigma, D)$, is the set of all atoms in $chase(\Sigma, D)$ of level at most $k$.

*Example 3.* Consider the two TGDs

$$\begin{array}{lrcl} \sigma_1\colon & r(X,Y), r(Z,X) & \rightarrow & \exists W\, s(W,Z,X) \\ \sigma_2\colon & r(X,Y), s(Z,X,Y) & \rightarrow & \exists W\, r(W,X) \end{array}$$

and the database $D$ consisting of the two atoms $r(a,b)$ and $s(c,a,b)$. Then, in the construction of $chase(\{\sigma_1,\sigma_2\}, D)$, we apply first the TGD $\sigma_2$ on $r(a,b)$ and $s(c,a,b)$, adding $r(z_1,a)$ (where $z_1$ is a new null), and we apply then $\sigma_1$ on $r(a,b)$ and $r(z_1,a)$, adding $s(z_2,z_1,a)$ (where $z_2$ is a new null).

The (possibly infinite) chase relative to a set of TGDs is a *universal model*, i.e., for every $B \in mods(\Sigma, D)$, there exists a homomorphism that maps $chase(\Sigma, D)$ onto $B$ [21, 9].

EGD Chase Rule. Consider a database $D$ for a relational schema $\mathcal{R}$, and an EGD $\sigma$ on $\mathcal{R}$ of the form $\Phi(\mathbf{X}) \rightarrow X_i = X_j$. Such an EGD $\sigma$ 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 $\sigma$, and the chase *fails*. Otherwise, the result of the application of $\sigma$ 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). ■

The *chase* of a database $D$, in the presence of two sets $\Sigma_T$ and $\Sigma_E$ 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 they are applicable (i.e., until a fixpoint is reached).

## 2.4 Treewidth

We now come to the notions of treewidth, tree decomposition, and bounded-treewidth model property (see, e.g., [24]).

Consider a graph $G = (V, E)$; a *tree decomposition of* $G$ consists of a tree $T = (N, A)$ and a labeling function $\lambda : N \rightarrow 2^V$ with the following properties: *(i)* for every $v \in V$, there exists $n \in N$ such that $v \in \lambda(N)$; *(ii)* for every arc $(v_1, v_2) \in E$, there exists $n \in N$ such that $\{v_1, v_2\} \subseteq \lambda(n)$; *(ii)* for every $v \in V$, the set $\{n \in N \mid v \in \lambda(n)\}$ induces a (connected) subtree in $T$.

The *Gaifman graph* of a database $D$ (or of any set of atoms) is a nondirected graph defined as follows: *(1)* the nodes are the symbols in $dom(D)$ (in general, constants in $\Delta$ and nulls in $\Delta_N$); *(ii)* there exists an arc $(c_1, c_2)$ between $c_1$ and $c_2$ if there exist some atom in $D$ that has both $c_1$ and $c_2$ as arguments.

The *width* of a tree decomposition $(T, \lambda)$, with $T = (N, A)$, of graph $G = (V, E)$ is the integer $\max_{n \in N} |\lambda(n)|$. The *treewidth* of a graph $G = (V, E)$, denoted $tw(G)$, is the minimum width among all tree decompositions. Given a relational instance $D$ (or any set of atoms), its treewidth $tw(D)$ is defined as the treewidth of its Gaifman graph.

A class $\mathcal{C}$ of formulas has the *bounded treewidth model property* if for each $\phi \in \mathcal{C}$, whenever $\phi$ is satisfiable, then it is possible to compute a number $f(\phi)$ such that $\phi$ has a model of treewidth at most $f(\phi)$.

## 3. GUARDED DATALOG$^\pm$

Query answering under general TGDs is undecidable [5], even when the schema and the TGDs are fixed [9]. In this section, we discuss *guarded* TGDs, also called *guarded Datalog*$^\pm$, as a special class of TGDs relative to which query answering is decidable in the general case and even tractable in the data complexity. Queries relative to such TGDs can be evaluated on a finite part of the chase, which is of constant size when the query and the TGDs are fixed.

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

*Example 4.* 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]. Guardedness is a truly fundamental class ensuring decidability. As the following theorem shows, adding a single unguarded Datalog rule to a guarded Datalog$^{\pm}$ program may destroy decidability. The proof of this theorem in [9] hinges on the fact that with appropriate input facts $D$, using a fixed set of TGDs comprising guarded TGDs and a single unguarded TGD, it is possible to force an infinite grid to appear in $chase(\Sigma_u, D)$. By a further set of guarded rules, one can then easily simulate the behavior of a deterministic Turing machine (TM) $\mathcal{M}$ with an empty input tape. This is done by using the infinite grid, where the $i$-th horizontal line of the grid represents the tape content at instant $i$. For details see [9].

**Theorem 1** **[9]** *There is a fixed set of TGDs $\Sigma_u$ such that all TGDs but one of $\Sigma_u$ are guarded, such that for instances $D$ for a schema $\mathcal{R}$ and atomic queries $Q$, determining whether $D \cup \Sigma_u \models Q$, or, equivalently, whether $Q \in chase(\Sigma_u, D)$, is undecidable.*

In the following two subsections, we report on the combined complexity and on the data complexity of query answering with guarded Datalog$^{\pm}$, respectively.

## 3.1 Combined Complexity

The next theorem establishes combined complexity results for conjunctive query evaluation under guarded Datalog$^{\pm}$.

**Theorem 2** **[9]** *Let $\Sigma$ be a guarded Datalog$^{\pm}$ program (i.e., a set of guarded TGDs) over a schema $\mathcal{R}$, and let $D$ be an instance for $\mathcal{R}$. Let, moreover, $w$ denote the maximum arity of any predicate appearing in $\mathcal{R}$, and let $|\mathcal{R}|$ denote the total number of predicate symbols. Then:*

*(1) If $Q$ is an atomic query, then checking whether $D \cup \Sigma \models Q$ is* PTIME*-complete in case both $w$ and $|\mathcal{R}|$ are bounded, and remains* PTIME*-complete even in case $\Sigma$ is fixed**. *This problem is* EXPTIME*-complete if $w$ is bounded and* 2EXPTIME*-complete in general. It remains* 2EXPTIME*-complete even when $|\mathcal{R}|$ is bounded.*

*(2) If $Q$ is a general conjunctive query, checking whether $D \cup \Sigma \models Q$ is* NP*-complete in case both $w$ and $|\mathcal{R}|$ are bounded, and thus also in case of a fixed set $\Sigma$. Checking whether $D \cup \Sigma \models Q$ is* EXPTIME*-complete if $w$ is bounded and* 2EXPTIME*-complete in general. It remains* 2EXPTIME*-complete even when $|\mathcal{R}|$ is bounded.*

*(3) Query containment under guarded TGDs is* NP*-complete if both $w$ and $|\mathcal{R}|$ are bounded, and even in case the set of guarded TGDs $\Sigma$ is fixed.*

*(4) Query containment under guarded TGDs is* EXPTIME*-complete if $w$ is bounded and* 2EXPTIME*-complete in general. It remains* 2EXPTIME*-complete even when $|\mathcal{R}|$ is bounded.*

For unguarded Datalog$^{\pm}$, and, in particular, for plain Datalog, even when we fix both, the width $w$ and the number of predicates $|\mathcal{R}|$ that are allowed to occur in a program, we can still formulate an infinity of mutually non-equivalent programs. In contrast to this, it is not hard to see that in case both $w$ and $|\mathcal{R}|$ are bounded, there are – up to isomorphism – only constantly many guarded Datalog$^{\pm}$ programs. It is thus not very astonishing that conjunctive query answering on the basis of such drastically limited programs is not harder than the standard task of evaluating a query over an extensional database. The EXPTIME and 2EXPTIME upper bounds of Theorem 2 can be established via alternating computations in a similar way as those for the more general *weakly guarded* Datalog$^{\pm}$ programs, which will be dealt with in Section 5. The EXPTIME-hardness results of the above theorem are achieved via simulations of an alternating PSPACE Turing machine, and an alternating EXPSPACE Turing machine, respectively. Simulating an alternating PSPACE Turing machine is not difficult. The (polynomially many) rules that simulate such a machine can contain explicit symbols for each of the polynomially many worktape cell positions. Simulating an EXPSPACE Turing machine via guarded rules is much harder. Here the problem is that to this aim we can can no longer explicitly address each worktape cell $i$, or each pair of cells $i, j$, given that there is now an *exponential* number of worktape cells. The idea is thus to encode tape cell indexes as *vectors* of variables $(v_1, \ldots, v_k)$ where the value of each $v_i$ ranges over 0 and 1. We can then define a successor relation $succ(v_1, \ldots, v_k, w_1, \ldots, w_k)$ with a polynomial number of Datalog rules. However, there is a further difficulty. We now have two different types of variables, the $v_i, w_j$ variables representing the bits in the above-described bit vectors, and other variables say, $x, y, z$, for denoting configurations. A major difficulty is to ensure that we do not violate guardedness, when using both types of variables in a rule body. In our proof in the full version of [10], we employ sophisticated tricks in order to achieve this.

The next result, which tightens parts of Theorem 2, shows that the above EXPTIME and 2EXPTIME-completeness results hold even in case of a fixed input database.

**Theorem 3** **[9]** *Let $\Sigma$ be a guarded Datalog$^{\pm}$ program over a schema $\mathcal{R}$. Let $w$ denote the maximum arity of any predicate appearing in $\mathcal{R}$, and let $|\mathcal{R}|$ denote the total number of predicate symbols. Then, for fixed databases $D$ and for both fixed atomic and variable queries $Q$, checking whether $chase(\Sigma, D) \models Q$ is* EXPTIME*-complete if $w$ is bounded, and* 2EXPTIME*-complete in general. This problem remains* 2EXPTIME*-complete even when $|\mathcal{R}|$ is bounded.*

From the above 2EXPTIME-completeness result, it follows that the satisfiability problem for very simple guarded theories is already at the same complexity level as decidability for the entire guarded fragment GFO of first order logic, for which Grädel [27] has shown 2EXPTIME-completeness. In fact, atomic queries and their (universally quantified) negation are guarded. If $\Sigma$ is a guarded Datalog$^{\pm}$ program, i.e., a set of guarded TGDs with single right hand sides, and $Q$ an (existentially quantified) atomic Boolean conjunctive query, then $D \wedge \Sigma \wedge \neg Q$ is a guarded FO theory of a particularly simple form. It is clear that $chase(\Sigma, D) \models Q$ iff $D \cup \Sigma \models Q$ iff $D \wedge \Sigma \wedge \neg Q$ is satisfiable. It is quite astonishing that the satisfiability of theories in such simple disjunction-free subfragments of GFO is as hard to decide as deciding whether an arbitrary GFO theory is satisfiable.

*This is just a mild extension of the data complexity case presented in Section 3.2, where also $Q$ is fixed.

On the other hand, it is less surprising that deciding the satisfiability of guarded $\Pi_2$-theories, i.e., guarded universal-existential first-order theories, is 2EXPTIME hard. In fact, Grädel [27] has shown a $\Pi_2$ normal form for satisfiability in the guarded fragment. Note, however, that this normal form is not disjunction-free.

## 3.2 Data Complexity

We next focus on the data complexity of evaluating BCQs relative to guarded TGDs, which turns out to be polynomial in general and linear in the case of atomic queries. In the sequel, let $\mathcal{R}$ be a relational schema, $D$ be a database for $\mathcal{R}$, and $\Sigma$ be a set of guarded TGDs on $\mathcal{R}$. We first give some preliminary definitions.

The *chase graph* for $\Sigma$ and $D$ is the directed graph consisting of $chase(\Sigma, D)$ as the set of nodes and having an arrow 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 $\sigma$. The *guarded chase forest* for $\Sigma$ and $D$ is the restriction of the chase graph for $\Sigma$ 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 5.* Consider the two TGDs

$$\begin{array}{lrcl} \sigma_1: & r_1(X,Y), r_2(Y) & \to & \exists Z\, r_1(Z,X), \\ \sigma_2: & r_1(X,Y) & \to & r_2(X), \end{array}$$

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. 1, where every arrow is labeled with the applied TGD.

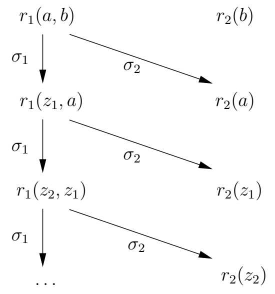

**Figure 1: Guarded chase forest for Example 5.**

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 $S = \emptyset$) iff there exists a bijection $\beta \colon A_1 \cup dom(A_1) \to A_2 \cup dom(A_2)$ such that (i) $\beta$ and $\beta^{-1}$ 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 $S = \emptyset$) iff $\{a_1\}$ and $\{a_2\}$ are $S$-isomorphic. The notion of $S$-isomorphism (or isomorphism if $S = \emptyset$) 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 key lemma shows that for each set of guarded TGDs $\Sigma$, there exists a constant $k$ such that for every database $D$ and every atom $\mathbf{a}$ generated at some depth level $d$ while chasing $D$ with $\Sigma$, such that whenever the same chase generates an atom $\mathbf{b}$ whose arguments are among those of $\mathbf{a}$, then $\mathbf{b}$ must be generated at depth at most $d+k$. Here, the *(guarded) depth* of an atom $\mathbf{a}$ in the guarded chase forest for $\Sigma$ 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. The main idea behind the proof is that the subtree of an atom $\mathbf{a}$ depends only on $\mathbf{a}$ and the type of $\mathbf{a}$, and the number of non-*dom*$(\mathbf{c})$-isomorphic pairs consisting of an atom and its type is bounded by a constant, depending only on $\mathcal{R}$.

**Lemma 4 [10]** *Let $\mathcal{R}$ be a relational schema, $D$ be a database for $\mathcal{R}$, and $\Sigma$ be a set of guarded TGDs on $\mathcal{R}$. Let $\mathbf{a}$ be a guard in the chase graph for $\Sigma$ and $D$, and $\mathbf{b} \in type(\mathbf{a})$. Then, $depth(\mathbf{b}) \leqslant depth(\mathbf{a}) + k$, where $k$ depends only on $\mathcal{R}$.*

The next lemma shows that query evaluation can be done on only a finite, initial portion of the guarded chase forest, whose size is determined by the query $Q$ and $\mathcal{R}$ only. Here, the *guarded chase* of level up to $k \geqslant 0$ for $\Sigma$ and $D$, denoted $g\text{-}chase^k(\Sigma, D)$, is the set of all atoms in the forest of depth at most $k$. The result is proved similarly to Lemma 4, showing that any path of a certain minimal length (depending on $Q$ and $\mathcal{R}$) from $D$ to (the image of) a query atom in the guarded chase forest has two atoms with *dom*$(\mathbf{c})$-isomorphic subtrees (since two atoms and their types are *dom*$(\mathbf{c})$-isomorphic), and thus $Q$ can also be evaluated "closer" to $D$.

**Lemma 5 [10]** *Let $\mathcal{R}$ be a relational schema, $D$ be a database for $\mathcal{R}$, $\Sigma$ be a set of guarded TGDs on $\mathcal{R}$, and $Q$ be a BCQ over $\mathcal{R}$. If there is a homomorphism $\mu$ that maps $Q$ into $chase(\Sigma, D)$, then there is a homomorphism $\lambda$ that maps $Q$ into $g\text{-}chase^k(\Sigma, D)$, where $k$ depends only on $Q$ and $\mathcal{R}$.*

The above lemma says that (homomorphic images of) the query atoms are contained in a finite, initial portion of the guarded chase forest, whose size is determined only by the query and $\mathcal{R}$. But it does not yet assure that also the whole derivation of the query atoms are contained in such a portion of the guarded chase forest. This slightly stronger property is captured by the following definition.

*Definition 1.* We say $\Sigma$ has the *bounded guard-depth property (BGDP)* iff, for each database $D$ for $\mathcal{R}$ and for each BCQ $Q$, whenever there is a homomorphism $\mu$ that maps $Q$ into $chase(\Sigma, D)$, then there is a homomorphism $\lambda$ of this kind such that all ancestors of $\lambda(Q)$ in the chase graph for $\Sigma$ and $D$ are contained in $g\text{-}chase^{\gamma_g}(\Sigma, D)$, where $\gamma_g$ depends only on $Q$ and $\mathcal{R}$.

In fact, the following result shows that guarded TGDs have also this stronger bounded guard-depth property. The proof of this result is based on Lemmas 4 and 5, where the former now also assures that all side atoms that are necessary in the derivation of the query atoms are contained in a finite, initial portion of the guarded chase forest, whose size is determined only by the query and $\mathcal{R}$ (which is slightly larger than the one for the query atoms only).

**Theorem 6** **[10]** *Guarded TGDs enjoy the BGDP.*

By this result, deciding BCQs in the guarded case is in P in the data complexity (where all but the database is fixed) [9]. It is also hard for P, as can be proved by reduction from propositional logic programming [10]. This is expressed by the following theorem.

**Theorem 7** **[9, 10]** *Let $\mathcal{R}$ be a relational schema, $D$ be a database for $\mathcal{R}$, $\Sigma$ be a set of guarded TGDs on $\mathcal{R}$, and $Q$ be a BCQ over $\mathcal{R}$. Then, deciding $D \cup \Sigma \models Q$ is* P*-complete in the data complexity.*

Deciding atomic BCQs in the guarded case can even be done in linear time in the data complexity, as the next result shows, which can be proved by reduction to propositional logic programming.

**Theorem 8** **[10]** *Let $\mathcal{R}$ be a relational schema, $D$ be a database for $\mathcal{R}$, $\Sigma$ be a finite set of guarded TGDs on $\mathcal{R}$, and $Q$ be a Boolean atomic query over $\mathcal{R}$. Then, deciding $D \cup \Sigma \models Q$ can be done in linear time in the data complexity.*

## 4. LINEAR DATALOG$^\pm$

Linear Datalog$^\pm$ is a variant of guarded Datalog$^\pm$, where query answering is even FO-rewritable in the data complexity. Nonetheless, linear Datalog$^\pm$ is still expressive enough for representing ontologies, as we will show in Section 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*.

Note that linear TGDs are more expressive than inclusion dependencies. For example, the following linear TGD, not expressible with inclusion dependencies, asserts that everyone supervising him/herself is a manager:

$$\mathit{supervises}(X, X) \rightarrow \mathit{manager}(X).$$

### 4.1 Combined Complexity

Query answering with linear Datalog$^\pm$ is PSPACE-complete when the Datalog$^\pm$ program is not fixed, which can be seen by results in [29, 38, 14, 26].

**Theorem 9** **[29, 38, 14, 26]** *Let $\mathcal{R}$ be a relational schema, $\Sigma$ 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, deciding $D \cup \Sigma \models Q$ is* PSPACE*-complete. It remains* PSPACE*-complete when $Q$ is fixed.*

### 4.2 Data Complexity

We start from some preliminary definitions. We say that a class $\mathcal{C}$ of TGDs is *first-order rewritable* (or *FO-rewritable*) iff for every set of TGDs $\Sigma$ in $\mathcal{C}$, and for every BCQ $Q$, there exists a first-order query $Q_\Sigma$ such that, for every database instance $D$, it holds $D \cup \Sigma \models Q$ iff $D \models Q_\Sigma$. Since answering first-order queries is in the class AC$_0$ in the data complexity [39], it immediately follows that for FO-rewritable TGDs, BCQ answering is in AC$_0$ in the data complexity.

We next define the bounded derivation-depth property, which is strictly stronger than the bounded guard-depth property. Informally, this property says that (homomorphic images of) the query atoms along with their derivations are contained in a finite, initial portion of the chase graph (rather than the guarded chase forest), whose size is determined only by the query and $\mathcal{R}$.

*Definition 2.* A set of TGDs $\Sigma$ 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 $\mathit{chase}^{\gamma_d}(\Sigma, D) \models Q$, where $\gamma_d$ depends only on $Q$ and $\mathcal{R}$.

Clearly, in the case of linear TGDs, for every $\mathbf{a} \in \mathit{chase}(\Sigma, D)$, the subtree of $\mathbf{a}$ is now determined only by $\mathbf{a}$ itself, while in the case of guarded TGDs it depends on $\mathit{type}(\mathbf{a})$. Therefore, for a single atom, its depth coincides with the number of applications of the TGD chase rule that are necessary to generate it. That is, the guarded chase forest coincides with the chase graph. By this observation, as an immediate consequence of Theorem 6, we obtain that linear TGDs have the bounded derivation-depth property.

**Corollary 10** **[10]** *Linear TGDs enjoy the BDDP.*

The next result shows that BCQs $Q$ relative to TGDs $\Sigma$ with the bounded derivation-depth property are FO-rewritable. The main ideas behind its proof are informally summarized as follows. Since the derivation depth and the number of body atoms in TGDs in $\Sigma$ is bounded, the number of all database ancestors of query atoms is also bounded. Thus, the number of all non-isomorphic sets of potential database ancestors with variables as arguments is also bounded. Take the existentially quantified conjunction of every such ancestor set where the query $Q$ is answered positively. Then, the FO-rewriting of $Q$ is the disjunction of all these formulas.

**Theorem 11** **[10]** *Let $\mathcal{R}$ be a relational schema, $\Sigma$ be a set of TGDs over $\mathcal{R}$, $D$ be a database for $\mathcal{R}$, and $Q$ be a BCQ over $\mathcal{R}$. If $\Sigma$ enjoys the BDDP, then $Q$ is FO-rewritable.*

As an immediate consequence of Corollary 10 and Theorem 11, BCQs are FO-rewritable in the linear case.

**Corollary 12** **[10]** *Let $\mathcal{R}$ be a relational schema, $\Sigma$ 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. WEAKLY GUARDED DATALOG$^\pm$

In this section, we introduce *weakly-guarded Datalog*$^\pm$, or *weakly guarded TGDs (WGTGDs)*, which is a generalization of guarded Datalog$^\pm$.

We first give the notion of *affected* position of a relational schema, given a set of TGDs $\Sigma$, as follows. *(a)* If an existentially quantified variable appears in $\pi$ in some TGD in $\Sigma$, then $\pi$ is affected wrt $\Sigma$. *(b)* If, for some $\sigma \in \Sigma$, the same universally quantified variable appears both in $\pi$ in $\mathit{head}(\sigma)$, and *only* in affected positions in $\mathit{body}(\sigma)$, then $\pi$ is affected wrt $\Sigma$. It is not difficult to see that affected positions are the only ones where a null can appear, during the chase construction.

*Example 6.* Consider the following set of TGDs:

$$\begin{array}{rrcl} \sigma_1: & p_1(X.Y), p_2(X, Y) & \rightarrow & \exists Z\, p_2(Y, Z), \\ \sigma_2: & p_2(X.Y), p_2(W, X) & \rightarrow & p_1(Y, X). \end{array}$$

Notice that $p_2[2]$ is affected since $Z$ in $\sigma_1$ is existentially quantified in $\sigma_1$. Considering again $\sigma_1$, the variable $Y$ appears in $p_2[2]$ but also in $p_1[2]$, therefore it does not make the position $p_2[1]$ affected. In $\sigma_2$, $X$ appears in the affected position $p_2[2]$ but also in $p_2[1]$, which is not affected; therefore, it does not make the position $p_1[2]$ affected. On the contrary, in $\sigma_2$, $Y$ appears in $p_2[2]$ and nowhere else, thus causing $p_1[1]$ to be affected.

A TGD is *weakly-guarded* wrt a set $\Sigma$ of TGDs if there is an atom in its body that contains all the universally quantified variables that appear *only* in positions that are affected wrt $\Sigma$.

*Example 7.* Consider the two TGDs in Example 6. In $\sigma_1$ both atoms are guards (and obviously weak guards), since they contain all universally quantified variables in the TGD. In $\sigma_2$, the only variable that appears only in affected positions is $Y$; therefore, the first atom is a weak guard.

We now give some preliminary definitions, which serve for the subsequent complexity analysis.

Let $D$ be a possibly infinite relational instance for a schema $\mathcal{R}$, and let $S$ be a set of symbols. An $S$-join forest of $D$ is an undirected labeled forest $T = (V, E, \lambda)$, with labeling function $\lambda : V \to D$ such that: *(i)* $D \subseteq \lambda(V)$, and *(ii)* $T$ is $S$-connected, i.e., for each $c \in \mathit{dom}(D) - S$, the set $\{v \in V \mid c \text{ occurs in } \lambda(v)\}$ induces a connected subtree in $T$. We say that $D$ is $S$-acyclic iff $D$ has an $S$-join forest. This generalizes the classical notion of hypergraph acyclicity [4] of an instance (or, equivalently, of a query). In fact, an instance or a query (seen as an instance) is hypergraph-acyclic iff it is $\emptyset$-acyclic.

From the definitions of $S$-acyclicity, we straightforwardly get that, if an instance $D$ for a schema $\mathcal{R}$ is $S$-acyclic, then $tw(D) \leqslant |S| + w$, where $w$ is the maximum arity of any predicate symbol in $\mathcal{R}$.

Notice that $\mathit{chase}(\Sigma, D)$ can be partitioned into two sets: $\mathit{chase}^{\perp}(\Sigma, D)$, the set of the atoms in $\mathit{chase}(\Sigma, D)$ that have *only* values in $\mathit{dom}(D)$ as arguments, and $\mathit{chase}^{+}(\Sigma, D)$, the rest of the atoms. An important fact is that $\mathit{chase}^{+}(\Sigma, D)$ is $\mathit{dom}(D)$-acyclic.

The proof of the following lemma is easily derived from the above results.

**Lemma 13 [9]** *If $\Sigma$ is a set of WGTGDs and $D$ an instance of a schema $\mathcal{R}$, then $tw(\mathit{chase}(\Sigma, D)) \leq |D| + w$, where $w$ is the maximum arity of a predicate in $\mathcal{R}$.*

We now come to the decidability of query answering under WGTGDs. From [19] (see also [24, 27]) we have that if a set of first-order formulas has the bounded-treewidth model property, then checking satisfiability for such formulas is decidable. Observing that both $\mathit{chase}(\Sigma, D) \wedge Q$ and $\mathit{chase}(\Sigma, D) \wedge \neg Q$ have a (possibly infinite) model of bounded treewidth, when they are satisfiable, we get the following result.

**Theorem 14 [9]** *Query answering under WGTGDs is decidable.*

This theorem establishes decidability of query answering under WGTGDs; however, nothing is said about the complexity. This will be the subject of the following sections.

## 5.1 Combined Complexity

We now tackle the complexity of query answering under WGTGDs. We start from determining the lower bound.

**Theorem 15 [9]** *Query answering under WGTGDs is* 2EXPTIME*-hard. The same problem is* EXPTIME*-hard in case the arity of predicates in the schema is fixed. The same complexity bound holds in the case of atomic queries and even fixed queries.*

Note that this result was already shown for the more restricted class of guarded TGDs (see Theorems 2 and 3).

Now, we come to the upper bound for query answering under WGTGDs. We start by defining notion of *squid decomposition*, and prove a lemma called "Squid Lemma" which will be a useful tool for proving the upper complexity bound of the query answering problem.

We first define the notion of $\mathcal{R}$-cover. Given a BCQ $Q$ on a schema $\mathcal{R}$ having $n$ body atoms, an *$\mathcal{R}$-cover* of $Q$ is a Boolean conjunctive query $Q^+$ on $\mathcal{R}$ that contains in its body all atoms of $Q$ and that may, in addition, contain at most $n$ further $\mathcal{R}$-atoms whose variables can be either from $\mathit{vars}(Q)$ or new.

For example, consider the schema $\mathcal{R} = \{r/2, s/3, t/3\}$, and let $Q$ be the Boolean conjunctive query $r(X, Y), r(Y, Z), t(Z, X, X)$. The following query $Q^+$ is an $\mathcal{R}$-cover of $Q$: $Q^+ = \{r(X, Y), r(Y, Z), t(Z, X, X), t(Y, Z, Z), s(Z, U, U)\}$.

It is possible to show, given an instance (finite or infinite) $D$ for a schema $\mathcal{R}$ and a BCQ $Q$, that $D \models Q$ iff there exists an $\mathcal{R}$-cover $Q^+$ of $Q$ such that $D \models Q^+$. We refer the reader to [9] for the proof of this result.

The following definition amends and replaces the one in [9], which was correct for relational schemas with binary predicates only; the present definition and the subsequent results are general and can deal with predicates of arbitrary arity.

*Definition 3.* Let $Q$ be a BCQ over a schema $\mathcal{R}$. A *squid decomposition* $\delta = (Q^+, h, H, T)$ of $Q$ consists of an $\mathcal{R}$-cover $Q^+$ of $Q$, a mapping $h : \mathit{vars}(Q^+) \to \mathit{vars}(Q^+)$, and a decomposition of $h(Q^+)$ into two sets $H$ and $T$, with $T = h(Q^+) - H$, such that: *(i)* there exists $V_\delta \subseteq \mathit{vars}(Q^+)$ such that $H = \{\mathbf{a} \in h(Q^+) \mid \mathit{vars}(\mathbf{a}) \subseteq V_\delta\}$; *(ii)* $T$ is $V_\delta$-acyclic. We refer to $H$ as the *head* of $\delta$, and to $T$ as the *tentacles* of $\delta$. The set of all squid decompositions of $Q$ is referred to as $\mathit{squidd}(Q)$.

One may imagine the set $H$ in a squid decomposition as the head of a squid, and the set $T$ as a forest of tentacles attached to that head. Note that a squid decomposition $\delta = (Q^+, h, H, T)$ of $Q$ does not not necessarily define a query folding [17, 36] of $Q^+$, because $h$ does not need to be an endomorphism of $Q^+$: in other terms, we do not require that $h(Q^+) \subseteq Q^+$. Of course, $h$ is a homomorphism.

*Example 8.* Consider the following Boolean conjunctive query (the schema is omitted for brevity):

$$\begin{aligned}
&Q = \{r(X, Y), r(X, Z), r(Y, Z),\\
&r(Z, V_1), r(V_1, V_2), r(V_2, V_3), r(V_3, V_4), r(V_4, V_5),\\
&r(V_1, V_6), r(V_6, V_5), r(V_5, V_7), r(Z, U_1), s(U_1, U_2, U_3),\\
&r(U_3, U_4), r(U_3, U_5), r(U_4, U_5)\}
\end{aligned}$$

Let $Q^+$ be the Boolean query where we add the atom $s(U_3, U_4, U_5)$ to the body. A possible squid decomposition $(Q^+, h, H, T)$ can be based on the homomorphism $h$, where $h(V_6) = V_2$, $h(V_4) = h(V_5) = h(V_7) = V_3$, and where $h(\xi) = \xi$ for each other variable $\xi$. The result of the decomposition with $V_\delta = \{X, Y, Z\}$ is the query shown in Fig. 2, where its join graph is depicted, in order to better distinguish the (cyclic) head from the (acyclic) tentacles. Note that if we eliminated the additional atom $s(U_3, U_4, U_5)$, the original atoms $r(U_3, U_4), r(U_3, U_5), r(U_4, U_5)$ would form an uncovered cycle, and could therefore not simultaneously be part of the tentacles, as $T$ would then no longer be $V_\delta$-acyclic.

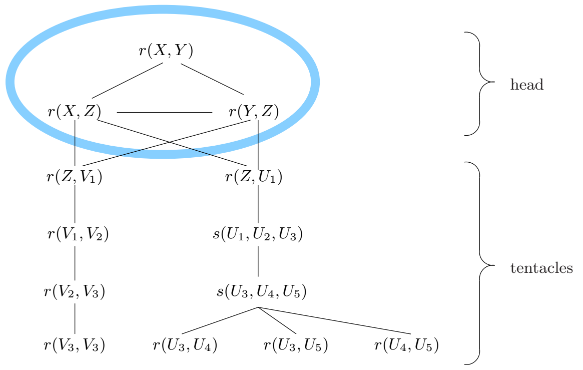

**Figure 2: Squid decomposition from Example 8**

The following lemma serves as a main tool in the complexity analysis of the problem of query answering under WGTGDs.

**Lemma 16 (Squid Lemma)** *Let $\Sigma$ be a set of WGTGDs on a schema $\mathcal{R}$, $D$ a ground database instance for $\mathcal{R}$, and $Q$ a conjunctive query, then $chase(\Sigma, D) \models Q$ iff there is a squid decomposition $\delta = (Q^+, h, H, T) \in squidd(Q)$, and a homomorphism $\theta : dom(h(Q^+)) \to dom(chase(\Sigma, D))$ such that: (i) $\theta(H) \subseteq chase^\perp(\Sigma, D)$, and (ii) $\theta(T) \subseteq chase^+(\Sigma, D)$.*

We now come to our main result.

**Theorem 17** [9] *Query answering under WGTGDs is decidable in 2*EXPTIME*. The same problem is decidable in* EXPTIME *in case the arity of the predicates in the schema is bounded.*

The above result is proved by exhibiting an alternating algorithm that decides the query answering problem. We preliminarily define the notion of *cloud* of an atom in the chase: $cloud(\Sigma, D, \mathbf{a})$ is the set of atoms in $chase(\Sigma, D)$ that have as arguments symbols from $dom(\mathbf{a}) \cup dom(D)$. Notice that the notion of cloud is similar to the one of type, with the difference that $cloud(\Sigma, D, \mathbf{a})$ can have as arguments symbols in $dom(D)$ that do not occur in $\mathbf{a}$. First of all, given a BCQ $Q$, the algorithm can guess a path in the guarded chase forest from some atom in $D$ to an atom $\mathbf{b}$ which is a homomorphic image of some atom in $Q$. For each atom $\mathbf{a}$ in such a path, a subset $S$ of $cloud(\Sigma, D, \mathbf{a})$ and a configuration are generated, where the configuration is basically an ordering on the atoms of $S$; by guessing that the atoms in $S$ until the $i$-th are derived, and the others are yet to be derived in the chase, the algorithm checks whether the set $S$ is actually in the chase. Now, given a query $Q$, the alternating algorithm guesses a squid decomposition of it such that the head maps onto $chase^\perp(\Sigma, D)$ via a homomorphism $\theta_0$. Then, the algorithm tries to find an extension $\theta$ of $\theta_0$ that maps the tentacles of the squid decomposition to $chase^+(\Sigma, D)$. By using a technique similar to the one described above, that checks whether the image of some atom is in the chase, the algorithm is then able, based on Lemma 16, to check whether $\theta$ exists.

By combining Theorems 15 and 17 we get the following complexity characterization for reasoning under WGTGDs.

**Theorem 18** [9] *Query answering under WGTGDs is* 2-EXPTIME *complete. It is* EXPTIME *complete in case of bounded predicate arities, and even in case the WGTGDs are fixed.*

## 5.2 Data Complexity

We now give some results regarding data complexity. With a reduction from a PSPACE alternating Turing Machine

**Theorem 19** [9] *Query answering under WGTGDs is* EXPTIME*-hard in the data complexity.*

Note that this result is quite surprising, in the light of the fact that the data complexity of query answering under guarded TGDs is polynomial. The proof of the above theorem needs to (ab)use the constants from $dom(D)$ in order to simulate tape cell positions.

By adapting the algorithm of Theorem 17, we can show the following result on the upper complexity bound.

**Theorem 20** [9] *Query answering under WGTGDs is in* EXPTIME *in the data complexity.*

## 6. EXTENSIONS

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

*Example 9.* If the unary predicates $c$ and $c'$ represent two classes, we may use the constraint $c(X), c'(X) \to \perp$ to assert that the two classes have no common instances. Similarly, if additionally the binary predicate $r$ represents a relationship, we may use $c(X), r(X, Y) \to \perp$ to enforce that no member of the class $c$ participates to $r$. In a similar way, we may use the EGD $r(X, Y), r(X, Y') \to Y = Y'$ to express that the second argument of the binary predicate $r$ functionally depends on the first argument of $r$.

However, while adding negative constraints is effortless from a computational perspective, 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 [18], or keys and inclusion dependencies (see, e.g., [12], where the proof of undecidability is done in the style of Vardi as in [29]). 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, after dealing with negative constraints, 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 modeling ontologies.

## 6.1 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 or weakly 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.

Query answering on a database $D$ under TGDs $\Sigma_T$ and constraints $\Sigma_C$ can be done effortless by additionally checking that every constraint $\sigma = \Phi(\mathbf{X}) \rightarrow \bot \in \Sigma_C$ is satisfied in $D$ given $\Sigma_T$, each of which can be done by checking that the BCQ $Q_\sigma = \Phi(\mathbf{X})$ evaluates to false in $D$ given $\Sigma_T$. We write $D \cup \Sigma_T \models \Sigma_C$ iff every $\sigma \in \Sigma_C$ is false in $D$ given $\Sigma_T$. We thus obtain immediately the following result. Here, a BCQ $Q$ is *true* in $D$ given $\Sigma_T$ and $\Sigma_C$, 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 21** **[10]** *Let $\mathcal{R}$ be a relational schema, $\Sigma_T$ and $\Sigma_C$ 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, linear, and weakly guarded case. It follows from Theorem 21, by which the additional effort of deciding $D \cup \Sigma_T \models \Sigma_C$ can be done by answering $|\Sigma_C|$ BCQs $Q_\sigma$ without constraints, where each query $Q_\sigma$ 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 $\Sigma_C$, while $\|\Sigma_C\|$ denotes the input size of $\Sigma_C$. This additional effort does not increase the complexity of answering BCQs without constraints. For the data complexity in the guarded and linear case, the result is known from [10].

**Theorem 22** **[10]** *Let $\mathcal{R}$ be a relational schema, $\Sigma_T$ and $\Sigma_C$ 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, weakly guarded) 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, weakly guarded) case.*

## 6.2 Non-Conflicting Keys

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 [12] 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 4. [10]* Let $\mathcal{R}$ be a relational schema, and $\Sigma_T$ and $\Sigma_E$ be sets of TGDs and EGDs on $\mathcal{R}$, respectively. Then, $\Sigma_E$ is *separable* from $\Sigma_T$ 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 $\Sigma_E$, 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, linear, and weakly guarded case. It follows immediately from the fact that the separability implies that chase failure can be directly evaluated on $D$. For fixed (resp., variable) $\Sigma_E$, this can be done by evaluating a first-order formula on $D$ (resp., in polynomial time in the size of $D$ and $\Sigma_E$), which clearly does not increase the data (resp., combined) complexity of answering BCQs in the three cases. For the data complexity in the guarded and linear case, the result is known from [10].

**Theorem 23** **[10]** *Let $\mathcal{R}$ be a relational schema, $\Sigma_T$ and $\Sigma_C$ be sets of TGDs and constraints on $\mathcal{R}$, respectively, and $D$ be a database for $\mathcal{R}$. Let $\Sigma_E$ be a set of EGDs that is separable from $\Sigma_T$, and $Q$ be a BCQ. Then, deciding $D \cup \Sigma_T \cup \Sigma_C \cup \Sigma_E \models Q$ in the guarded (resp., linear, weakly guarded) 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, weakly guarded) case.*

We next provide a sufficient syntactic condition for the separability of EGDs. We assume that the reader is familiar with the notions of *functional dependency (FD)* and *key* [1]. Clearly, FDs are special types of EGDs. A key $\kappa$ of a relation $r$ can be written as a set of FDs that specify that $\kappa$ determines each other attribute of $r$. Thus, keys can be identified with sets of EGDs. It will be clear from the context when we regard a key as a set of attribute positions, and when as a set of EGDs. The following definition generalizes the notion of “non-key-conflicting” dependency relative to a set of keys, introduced in [12], to the context of arbitrary TGDs.

*Definition 5. [10]* Let $\kappa$ be a key, and $\sigma$ be a TGD of the form $\Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, r(\mathbf{X}, \mathbf{Z})$. Then, $\kappa$ is *non-conflicting (NC)* with $\sigma$ iff either (i) the relational predicate on which $\kappa$ is defined is different from $r$, or (ii) the positions of $\kappa$ in $r$ are not a proper subset of the $\mathbf{X}$-positions in $r$ in the head of $\sigma$, and every variable in $\mathbf{Z}$ appears only once in the head of $\sigma$. We say $\kappa$ is *non-conflicting (NC)* with a set of TGDs $\Sigma_T$ iff $\kappa$ is NC with every $\sigma \in \Sigma_T$. We say a set of keys $\Sigma_K$ is *non-conflicting (NC)* with $\Sigma_T$ iff every $\kappa \in \Sigma_K$ is NC with $\Sigma_T$.

*Example 10.* Consider the four keys $\kappa_1$, $\kappa_2$, $\kappa_3$, and $\kappa_4$ 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 $\sigma$ 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 $\kappa_4$ are NC with $\sigma$, since only $\mathbf{K}_4 \subset \mathbf{H}$. Roughly, every atom added in a chase by applying $\sigma$ would have a fresh null in some position in $\mathbf{K}_1$, $\mathbf{K}_2$, and $\mathbf{K}_3$, thus never firing $\kappa_1$, $\kappa_2$, and $\kappa_3$, respectively.

The following theorem shows that the property of being NC between keys and TGDs implies their separability. This generalizes a useful result of [12] on inclusion dependencies to the much larger class of all TGDs. The main idea behind the proof can be roughly described as follows. The NC condition between a key $\kappa$ and a TGD $\sigma$ assures that either (a) the application of $\sigma$ in the chase generates an atom with a fresh null in a position of $\kappa$, and so the fact does not violate $\kappa$ (see also Example 10), or (b) the $\mathbf{X}$-positions in the predicate $r$ in the head of $\sigma$ coincide with the key positions of $\kappa$ 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 24 [10]** *Let $\mathcal{R}$ be a relational schema, $\Sigma_T$ and $\Sigma_K$ be sets of TGDs and keys, respectively, such that $\Sigma_K$ is NC with $\Sigma_T$. Then, $\Sigma_K$ is separable from $\Sigma_T$.*

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, weakly guarded) TGDs and constraints. This result follows immediately from Theorems 24 and 23. For the data complexity in the guarded and linear case, this result is known from [10].

**Corollary 25** *Let $\mathcal{R}$ be a relational schema, $\Sigma_T$ and $\Sigma_C$ be sets of TGDs and constraints on $\mathcal{R}$, respectively, and $D$ be a database for $\mathcal{R}$. Let $\Sigma_E$ be a set of EGDs that is NC with $\Sigma_T$, and $Q$ be a BCQ. Then, deciding $D \cup \Sigma_T \cup \Sigma_C \cup \Sigma_E \models Q$ in the guarded (resp., linear, weakly guarded) 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, weakly guarded) case.*

## 7. ONTOLOGY QUERYING

In this section, we show how Datalog$^\pm$ can be used for query answering in *DL-Lite*$_\mathcal{A}$ and F-Logic Lite ontologies.

### 7.1 DL-Lite$_\mathcal{A}$

We now describe how the DL *DL-Lite*$_\mathcal{A}$ [35] can be translated to linear Datalog$^\pm$ with (negative) constraints and NC keys, called *Datalog*$_0^\pm$, and that the former is strictly less expressive than the latter. We first recall the syntax and the semantics of *DL-Lite*$_\mathcal{A}$. We then define the translation and provide the expressivity result.

Note that other DLs of the *DL-Lite* family [13] can be similarly translated to Datalog$_0^\pm$. In particular, the translation for *DL-Lite*$_\mathcal{F}$ and *DL-Lite*$_\mathcal{R}$ is given in [10]. Note also that *DL-Lite*$_\mathcal{R}$ is able to fully capture (the DL fragment of) RDF Schema [8], the vocabulary description language for RDF; see [20]. Consequently, Datalog$_0^\pm$ is also able to fully capture (the DL fragment of) RDF Schema.

Intuitively, DLs model a domain of interest in terms of concepts and roles, which represent classes of individuals and binary relations on classes of individuals, respectively. A DL knowledge base (or ontology) encodes in particular subset relationships between concepts, subset relationships between roles, the membership of individuals to concepts, the membership of pairs of individuals to roles, and functional dependencies on roles. Important additional ingredients of *DL-Lite*$_\mathcal{A}$ are datatypes and attributes, which are collections of data values and binary relations between individuals and data values, respectively, along with the possibility to encode subset relationships between datatypes, subset relationships between attributes, the membership of individual-value pairs to attributes, and functional dependencies on attributes.

*Syntax.* We first describe the elementary ingredients of *DL-Lite*$_\mathcal{A}$. We assume a finite set $\mathbf{D}$ of *atomic datatypes* $d$, which are associated with pairwise disjoint sets of *data values* $\mathbf{V}_d$. We also assume pairwise disjoint sets $\mathbf{A}$, $\mathbf{R}_A$, $\mathbf{R}_D$, and $\mathbf{I}$ of *atomic concepts*, *atomic roles*, *atomic attributes*, and *individuals*, respectively. We denote by $\mathbf{V}$ the union of all $\mathbf{V}_d$ with $d \in \mathbf{D}$.

These elementary ingredients are used to construct roles, concepts, attributes, and datatypes, which are defined as follows:

- A *basic role* $Q$ is either an atomic role $P \in \mathbf{R}_A$ or its inverse $P^-$. A *(general) role* $R$ is either a basic role $Q$ or the negation of a basic role $\neg Q$.
- A *basic concept* $B$ is either an atomic concept $A \in \mathbf{A}$, or an existential restriction on a basic role $Q$, denoted $\exists Q$, or the domain of an atomic attribute $U$, denoted $\delta(U)$. A *(general) concept* $C$ is either the *universal concept* $\top_C$, or a basic concept $B$, or the negation of a basic concept $\neg B$, or an existential restriction on a basic role $Q$ of the form $\exists Q.C$, where $C$ is a concept.
- A *(general) attribute* $V$ is either an atomic attribute $U$ or the negation of an atomic attribute $\neg U$.
- A *basic datatype* $E$ is the range of an atomic attribute $U$, denoted $\rho(U)$. A *(general) datatype* $F$ is either the *universal datatype* $\top_D$ or an atomic datatype.

Statements about roles, concepts, attributes, and datatypes are expressed via *axioms*, which have the following forms: (1) $B \sqsubseteq C$ (*concept inclusion axiom*), where $B$ is a basic concept, and $C$ is a concept; (2) $Q \sqsubseteq R$ (*role inclusion axiom*), where $Q$ is a basic role, and $R$ is a role; (3) $U \sqsubseteq V$ (*attribute inclusion axiom*), where $U$ is an atomic attribute, and $V$ is an attribute; (4) $E \sqsubseteq F$ (*datatype inclusion axiom*), where $E$ is a basic datatype, and $F$ is a datatype; (5) $(\mathsf{funct}\ Q)$ (*role functionality axiom*), where $Q$ is a basic role; (6) $(\mathsf{funct}\ U)$ (*attribute functionality axiom*), where $U$ is an atomic attribute; (7) $A(a)$ (*concept membership axiom*), where $A$ is an atomic concept and $a \in \mathbf{I}$, (8) $P(a, b)$ (*role membership axiom*), where $P$ is an atomic role and $a, b \in \mathbf{I}$; (9) $U(a, v)$ (*attribute membership axiom*), where $U$ is an atomic attribute, $a \in \mathbf{I}$, and $v \in \mathbf{V}$.

We next define knowledge bases, which consist of a restricted finite set of inclusion and functionality axioms, called TBox, and a finite set of membership axioms, called ABox. We also describe CQs and BCQs to such knowledge bases.

We first define the restriction on inclusion and functionality axioms. A basic role $Q$ (resp., atomic attribute $U$) is an *identifying property* in a set of axioms $\mathcal{S}$ iff $\mathcal{S}$ contains a functionality axiom $(\mathsf{funct}\ Q)$ (resp., $(\mathsf{funct}\ U)$). Given an inclusion axiom $\alpha$ of the form $X \sqsubseteq Y$ (resp., $X \sqsubseteq \neg Y$), a basic role (resp., atomic attribute) $Y$ appears *positively* (resp., *negatively*) in the right-hand side of $\alpha$. A basic role (resp., atomic attribute) is *primitive* in $\mathcal{S}$ iff it does not appear positively in the right-hand side of an inclusion axiom in $\mathcal{S}$ and it does not appear in an expression of the form $\exists Q.C$ in $\mathcal{S}$.

We can now define knowledge bases. A *TBox* is a finite set $\mathcal{T}$ of inclusion and functionality axioms such that every identifying property in $\mathcal{T}$ is primitive. Intuitively, identifying properties cannot be specialized in $\mathcal{T}$, i.e., they cannot appear positively in the right-hand side of inclusion axioms in $\mathcal{T}$. An *ABox* $\mathcal{A}$ is a finite set of membership axioms. A *knowledge base* $KB = (\mathcal{T}, \mathcal{A})$ consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. *Conjunctive queries (CQs)* and *Boolean CQs (BCQs)* are defined as usual, where concept, role, and attribute membership axioms (over variables, individuals, and values as arguments) are used as atoms .

*Example 11.* We use a knowledge base $KB = (\mathcal{T}, \mathcal{A})$ in *DL-Lite*$_{\mathcal{A}}$ to specify some simple information about scientists and their publications. Consider the following sets of atomic concepts, atomic roles, atomic attributes, individuals, and data values:

$$\begin{aligned}
\mathbf{A} &= \{\mathit{Scientist}, \mathit{Article}, \mathit{ConferencePaper}, \mathit{JournalPaper}\},\\
\mathbf{R}_A &= \{\mathit{hasAuthor}, \mathit{isAuthorOf}, \mathit{hasFirstAuthor}\},\\
\mathbf{R}_D &= \{\mathit{name}, \mathit{title}, \mathit{yearOfPublication}\},\\
\mathbf{I} &= \{i_1, i_2\},\\
\mathbf{V} &= \{\text{“}\mathit{mary}\text{”}, \text{“}\mathit{Semantic\ Web\ search}\text{”}, 2008\}.
\end{aligned}$$

The TBox $\mathcal{T}$ contains the subsequent axioms, which informally express that (i) conference and journal papers are articles, (ii) conference papers are not journal papers, (iii) every scientist has at least one publication, (iv) *isAuthorOf* relates scientists and articles, (v) *isAuthorOf* is the inverse of *hasAuthor*, and (vi) *hasFirstAuthor* is a functional binary relationship:

$$\begin{aligned}
&\mathit{ConferencePaper} \sqsubseteq \mathit{Article},\\
&\mathit{JournalPaper} \sqsubseteq \mathit{Article},\\
&\mathit{ConferencePaper} \sqsubseteq \neg \mathit{JournalPaper},\\
&\mathit{Scientist} \sqsubseteq \exists \mathit{isAuthorOf},\\
&\exists \mathit{isAuthorOf} \sqsubseteq \mathit{Scientist},\\
&\exists \mathit{isAuthorOf}^- \sqsubseteq \mathit{Article},\\
&\mathit{isAuthorOf}^- \sqsubseteq \mathit{hasAuthor},\\
&\mathit{hasAuthor}^- \sqsubseteq \mathit{isAuthorOf},\\
&(\mathsf{funct}\ \mathit{hasFirstAuthor}).
\end{aligned}$$

The ABox $\mathcal{A}$ contains the following axioms, which express that the individual $i_1$ is a scientist whose name is “*mary*” and who is the author of article $i_2$, which is entitled “*Semantic Web search*” and has been published in the year 2008:

$$\begin{aligned}
&\mathit{Scientist}(i_1),\ \mathit{name}(i_1, \text{“}\mathit{mary}\text{”}), \mathit{isAuthorOf}(i_1, i_2),\\
&\mathit{Article}(i_2),\ \mathit{title}(i_2, \text{“}\mathit{Semantic\ Web\ search}\text{”}),\\
&\mathit{yearOfPublication}(i_2, 2008).
\end{aligned}$$

Querying for all scientists who published an article in 2008 can be expressed by the following CQ:

$$\begin{aligned}
Q(x) = \exists y\, (&\mathit{Scientist}(x) \wedge \mathit{isAuthorOf}(x, y) \wedge\\
&\mathit{Article}(y) \wedge \mathit{yearOfPublication}(y, 2008)).
\end{aligned}$$

***Semantics.*** The semantics of *DL-Lite*$_{\mathcal{A}}$ is defined in terms of standard first-order interpretations as usual. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of (i) a nonempty domain $\Delta^{\mathcal{I}} = (\Delta_O^{\mathcal{I}}, \Delta_V^{\mathcal{I}})$, which is the disjoint union of the *domain of objects* $\Delta_O^{\mathcal{I}}$ and the *domain of values* $\Delta_V^{\mathcal{I}} = \bigcup_{d \in \mathbf{D}} \Delta_d^{\mathcal{I}}$, where the $\Delta_d^{\mathcal{I}}$'s are pairwise disjoint domains of values for the datatypes $d \in \mathbf{D}$, and (ii) a mapping $\cdot^{\mathcal{I}}$ that assigns to each datatype $d \in \mathbf{D}$ its domain of values $\Delta_d^{\mathcal{I}}$, to each data value $v \in \mathbf{V}_d$ an element of $\Delta_d^{\mathcal{I}}$ (such that $v \neq w$ implies $v^{\mathcal{I}} \neq w^{\mathcal{I}}$), to each atomic concept $A \in \mathbf{A}$ a subset of $\Delta_O^{\mathcal{I}}$, to each atomic role $P \in \mathbf{R}_A$ a subset of $\Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}$, to each atomic attribute $P \in \mathbf{R}_D$ a subset of $\Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}$, to each individual $a \in \mathbf{I}$ an element of $\Delta_O^{\mathcal{I}}$ (such that $a \neq b$ implies $a^{\mathcal{I}} \neq b^{\mathcal{I}}$). Note that different data values (resp., individuals) are associated with different elements of $\Delta_V^{\mathcal{I}}$ (resp., $\Delta_O^{\mathcal{I}}$) (*unique name assumption*). The extension of $\cdot^{\mathcal{I}}$ to all concepts, roles, attributes, and datatypes, and the *satisfaction* of an axiom $\alpha$ in $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, denoted $\mathcal{I} \models \alpha$, are defined by:

- $(\top_D)^{\mathcal{I}} = \Delta_V^{\mathcal{I}}$ and $(\top_C)^{\mathcal{I}} = \Delta_O^{\mathcal{I}}$;
- $(\neg U)^{\mathcal{I}} = (\Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}) - U^{\mathcal{I}}$;
- $(\neg Q)^{\mathcal{I}} = (\Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}) - Q^{\mathcal{I}}$;
- $(\rho(U))^{\mathcal{I}} = \{v \in \Delta_V^{\mathcal{I}} \mid \exists o\colon (o, v) \in U^{\mathcal{I}}\}$;
- $(\delta(U))^{\mathcal{I}} = \{o \in \Delta_O^{\mathcal{I}} \mid \exists v\colon (o, v) \in U^{\mathcal{I}}\}$;
- $(P^-)^{\mathcal{I}} = \{(o, o') \in \Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}} \mid (o', o) \in P^{\mathcal{I}}\}$;
- $(\exists Q)^{\mathcal{I}} = \{o \in \Delta_O^{\mathcal{I}} \mid \exists o'\colon (o, o') \in Q^{\mathcal{I}}\}$;
- $(\exists Q.C)^{\mathcal{I}} = \{o \in \Delta_O^{\mathcal{I}} \mid \exists o'\colon (o, o') \in Q^{\mathcal{I}},\ o' \in C^{\mathcal{I}}\}$;
- $(\neg B)^{\mathcal{I}} = \Delta_O^{\mathcal{I}} \setminus B^{\mathcal{I}}$.

The *satisfaction* of an axiom $\alpha$ in the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, denoted $\mathcal{I} \models \alpha$, is defined as follows: (1) $\mathcal{I} \models B \sqsubseteq C$ iff $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$, (2) $\mathcal{I} \models Q \sqsubseteq R$ iff $Q^{\mathcal{I}} \subseteq R^{\mathcal{I}}$, (3) $\mathcal{I} \models E \sqsubseteq F$ iff $E^{\mathcal{I}} \subseteq F^{\mathcal{I}}$, (4) $\mathcal{I} \models U \sqsubseteq V$ iff $U^{\mathcal{I}} \subseteq V^{\mathcal{I}}$, (5) $\mathcal{I} \models (\mathsf{funct}\ Q)$ iff $(o, q), (o, q') \in Q^{\mathcal{I}}$ implies $q = q'$, (6) $\mathcal{I} \models (\mathsf{funct}\ U)$ iff $(o, v), (o, v') \in U^{\mathcal{I}}$ implies $v = v'$, (7) $\mathcal{I} \models A(a)$ iff $a^{\mathcal{I}} \in A^{\mathcal{I}}$, (8) $\mathcal{I} \models P(a, b)$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$, (9) $\mathcal{I} \models U(a, v)$ iff $(a^{\mathcal{I}}, v^{\mathcal{I}}) \in U^{\mathcal{I}}$. The interpretation $\mathcal{I}$ *satisfies* the axiom $\alpha$, or $\mathcal{I}$ is a *model* of $\alpha$, iff $\mathcal{I} \models \alpha$. We say $\mathcal{I}$ *satisfies* a knowledge base $KB = (\mathcal{T}, \mathcal{A})$, or $\mathcal{I}$ is a *model* of $KB$, denoted $\mathcal{I} \models KB$, iff $\mathcal{I} \models \alpha$ for all $\alpha \in \mathcal{T} \cup \mathcal{A}$. We say $KB$ is *satisfiable* (resp., *unsatisfiable*) iff $KB$ has a (resp., no) model. The semantics of *CQs* and *BCQs* is as usual in first-order logic.

*Example 12.* The knowledge base $KB = (\mathcal{T}, \mathcal{A})$ of Example 11 is satisfiable, and the answer to the CQ $Q(x)$ of Example 11 contains the tuple $(i_1)$. Informally, *mary* published an article in 2008.

***Translation into Datalog$^{\pm}$.*** The translation $\tau$ from the elementary ingredients and axioms of *DL-Lite*$_{\mathcal{A}}$ into Datalog$^{\pm}$ with (negative) constraints and EGDs is defined as follows:

(1) Every data value $v$ has a constant $\tau(v) = c_v \in \Delta$ such that the $\tau(\mathbf{V}_d)$'s for all datatypes $d \in \mathbf{D}$ are pairwise disjoint. Every datatype $d \in \mathbf{D}$ has under $\tau$ a predicate $\tau(d) = p_d$ along with the constraint $p_d(X) \wedge p_{d'}(X) \rightarrow \bot$ for all pairwise distinct $d, d' \in \mathbf{D}$. Every atomic concept $A \in \mathbf{A}$ has a unary predicate $\tau(A) = p_A \in \mathcal{R}$, every abstract role $P \in \mathbf{R}_A$ has a binary predicate $\tau(P) = p_P \in \mathcal{R}$, every attribute $U \in \mathbf{R}_D$ has a binary predicate $\tau(U) = p_U \in \mathcal{R}$, and every individual $i \in \mathbf{I}$ has a constant $\tau(i) = c_i \in \Delta - \bigcup_{d \in \mathbf{D}} \tau(\mathbf{V}_d)$.

(2) Every concept inclusion axiom $B \sqsubseteq C$ is translated to the TGD or constraint $\tau(B \sqsubseteq C) = \tau'(B) \rightarrow \tau''(C)$, where

   (i) $\tau'(B)$ is defined as $p_A(X)$, $p_P(X, Y)$, $p_P(Y, X)$, and $p_U(X, Y)$, if $B$ is of the form $A$, $\exists P$, $\exists P^-$, and $\delta(U)$, respectively, and

(ii) $\tau''(C)$ is defined as $p_A(X)$, $\exists Z p_P(X,Z)$, $\exists Z p_P(Z,\,X)$, $\exists Z p_U(X,Z)$, $\neg p_A(X)$, $\neg p_P(X,Y)$, $\neg p_P(Y,X)$, $\neg p_U(X,Y)$, $\exists Z p_P(X,Z) \wedge p_A(Z)$, and $\exists Z p_P(Z,X) \wedge p_A(Z)$, if $C$ is of form $A$, $\exists P$, $\exists P^-$, $\delta(U)$, $\neg A$, $\neg\exists P$, $\neg\exists P^-$, $\neg\delta(U)$, $\exists P.A$, and $\exists P^-.A$, respectively.

Note that concept inclusion axioms $B \sqsubseteq \top_C$ can be safely ignored, and concept inclusion axioms $B \sqsubseteq \exists Q.C$ can be expressed by the two concept inclusion axioms $B \sqsubseteq \exists Q.A$ and $A \sqsubseteq C$, where $A$ is a fresh atomic concept. Note also that the TGDs with two atoms in their heads abbreviate their equivalent sets of TGDs with singleton atoms in the heads.

(3) The functionality axioms ($\mathsf{funct}\ P$) and ($\mathsf{funct}\ P^-$) are under $\tau$ translated to the EGDs $p_P(X,Y) \wedge p_P(X,Y') \rightarrow Y = Y'$ and $p_P(X,Y) \wedge p_P(X',Y) \rightarrow X = X'$, respectively. The functionality axiom ($\mathsf{funct}\ U$) is under $\tau$ translated to the EGD $p_U(X,Y) \wedge p_U(X,Y') \rightarrow Y = Y'$.

(4) Every concept membership axiom $A(a)$ is under $\tau$ translated to the database atom $p_A(c_a)$. Every role membership axiom $P(a,b)$ is under $\tau$ translated to the database atom $p_P(c_a, c_b)$. Every attribute membership axiom $U(a,v)$ is under $\tau$ translated to the database atom $p_U(c_a, c_v)$.

(5) Every role inclusion axiom $Q \sqsubseteq R$ is translated to the TGD or constraint $\tau(Q \sqsubseteq R) = \tau'(Q) \rightarrow \tau''(R)$, where

(i) $\tau'(Q)$ is defined as $p_P(X,Y)$ and $p_P(Y,X)$, if $Q$ is of the form $P$ and $P^-$, respectively, and

(ii) $\tau''(R)$ is defined as $p_P(X,Y)$, $p_P(Y,X)$, $\neg p_P(X,Y)$, and $\neg p_P(Y,X)$, if $R$ is of the form $P$, $P^-$, $\neg P$, and $\neg P^-$, respectively.

(6) Attribute inclusion axioms $U \sqsubseteq U'$ and $U \sqsubseteq \neg U'$ are under $\tau$ translated to the TGD $p_U(X,Y) \rightarrow p_{U'}(X,Y)$ and the constraint $p_U(X,Y) \rightarrow \neg p_{U'}(X,Y)$, respectively.

(7) Every datatype inclusion axiom $\rho(U) \sqsubseteq d$ is under $\tau$ translated to the TGD $p_U(Y,X) \rightarrow p_d(X)$. Note that datatype inclusion axioms $\rho(U) \sqsubseteq \top_D$ can be safely ignored.

*Example 13.* The concept inclusion axioms of Example 11 are translated to the following TGDs and constraints (where we identify atomic concepts and roles with their predicates):

$$\begin{aligned}&\mathit{ConferencePaper}(X) \rightarrow \mathit{Article}(X),\\ &\mathit{JournalPaper}(X) \rightarrow \mathit{Article}(X),\\ &\mathit{ConferencePaper}(X) \rightarrow \neg\mathit{JournalPaper}(X),\\ &\mathit{Scientist}(X) \rightarrow \exists Z\ \mathit{isAuthorOf}(X,Z),\end{aligned}$$

The role inclusion and functionality axioms of Example 11 are translated to the following TGDs and EGDs:

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

The concept, role, and attribute membership axioms of Example 11 are translated to the following database atoms (where we also identify individuals and values with their constants):

$$\begin{aligned}&\mathit{Scientist}(i_1),\ \mathit{name}(i_1, \text{“}\mathit{mary}\text{”}), \mathit{isAuthorOf}(i_1, i_2),\\ &\mathit{Article}(i_2),\ \mathit{title}(i_2, \text{“}\mathit{Semantic\ Web\ search}\text{”}),\\ &\mathit{yearOfPublication}(i_2, 2008).\end{aligned}$$

Every knowledge base $KB$ in $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ is then translated into a database $D_{KB}$, set of TGDs $\Sigma_{KB}$, and set of queries $\mathcal{Q}_{KB}$ as follows: (i) $D_{KB}$ is the set of all $\tau(\phi)$ such that $\phi$ is a membership axiom in $KB$ along with “type declarations” $p_d(v)$ for all their data values; (ii) $\Sigma_{KB}$ is the set of all TGDs resulting from $\tau(\phi)$ such that $\phi$ is an inclusion axiom in $KB$; and (iii) $\mathcal{Q}_{KB}$ is the set of all queries resulting from datatype constraints and from constraints and EGDs $\tau(\phi)$ such that $\phi$ is an inclusion or functionality axiom in $KB$.

The following lemma shows that the TGDs and the EGDs that are generated via $\tau$ from a $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ knowledge base are in fact linear TGDs and NC keys, respectively.

**Lemma 26** **[10]** *Let $KB$ be a knowledge base in $\mathit{DL\text{-}Lite}_{\mathcal{A}}$, and $\Sigma_K$ be the set of EGDs encoded in $\mathcal{Q}_{KB}$. Then, (a) every TGD in $\Sigma_{KB}$ is linear, (b) every EGD in $\Sigma_K$ is a key, and (c) $\Sigma_K$ is NC with $\Sigma_{KB}$.*

Proof. (a) The generated TGDs are clearly linear. (b), (c) Observe first that basic roles and atomic attributes in inclusion axioms and expressions of the form $\exists Q.C$ never interact with role and attribute functionality axioms, since such functionality axioms can be expressed only on primitive basic roles and atomic attributes. Hence, it only remains to consider TGDs that are derived from axioms of the types (i) $B \sqsubseteq \exists P$ or $B \sqsubseteq \exists P^-$, when a functionality axiom ($\mathsf{funct}\,P$) or ($\mathsf{funct}\,P^-$) is in $KB$, and (ii) $B \sqsubseteq \delta(U)$, when a functionality axiom ($\mathsf{funct}\,U$) is in $KB$. W.l.o.g. consider the case (i) and suppose that ($\mathsf{funct}\,P$) is in $KB$. The TGDs generated via $\tau$ then have a head of the form $\exists Y\, p_P(X,Y)$ or $\exists Y\, p_P(Y,X)$, and a key of the form $P(X,Y), P(X,Y) \rightarrow Y = Y'$ (the key consists of the position $p_P[1]$). In both cases, the key is clearly non-conflicting with the above TGDs (and also with all the others in $\Sigma_{KB}$). $\square$

The next result shows that BCQs to knowledge bases in $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ can be reduced to BCQs in $\text{Datalog}_0^\pm$. This result follows immediately from the above lemma and Theorem 24.

**Theorem 27** **[10]** *Let $KB$ be a knowledge base in $\mathit{DL\text{-}Lite}_{\mathcal{A}}$, and let $Q$ be a BCQ for $KB$. Then, $Q$ is true in $KB$ iff (i) $D_{KB} \cup \Sigma_{KB} \models Q$, or (ii) $D_{KB} \cup \Sigma_{KB} \models Q_c$ for some $Q_c \in \mathcal{Q}_{KB}$.*

As an immediate consequence, the satisfiability of knowledge bases in $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ can also be reduced to BCQs in $\text{Datalog}_0^\pm$.

**Corollary 28** **[10]** *A knowledge base $KB$ in $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ is unsatisfiable iff $D_{KB} \cup \Sigma_{KB} \models Q_c$ for some $Q_c \in \mathcal{Q}_{KB}$.*

The following important result shows that $\text{Datalog}_0^\pm$ is strictly more expressive than $\mathit{DL\text{-}Lite}_{\mathcal{A}}$.

**Theorem 29** **[10]** *$\mathit{Datalog}_0^\pm$ is strictly more expressive than $\mathit{DL\text{-}Lite}_{\mathcal{A}}$.*

PROOF. As shown in [10], the TGD $p(X) \to q(X,X)$ can neither be expressed in *DL-Lite*$_{\mathcal{A}}$, since the TGDs of concept and role inclusion axioms can only project away arguments, introduce new nulls as arguments, and change the order of arguments in the predicates for atomic concepts and abstract roles, and the EGDs for functionality axioms can only produce an atom of form $q(c,c)$ from $q(n,c)$ and/or $q(c,n)$, where $n$ is a null, if $q(c,c)$ was already there before. □

## 7.2 F-Logic Lite

In this section, we briefly present a formalism for object-oriented schemas called F-Logic Lite, and we then show that it is a special case of weakly-guarded Datalog$^\pm$. We will also show, with a different proof, the same result appearing in [11], i.e., that query answering under F-Logic Lite rules is NP-complete.

F-Logic Lite is a smaller but still expressive version of F-logic [30], a well-known formalism introduced for object-oriented deductive databases. We refer the reader to [11] for details about F-Logic Lite. Roughly speaking, with respect to F-Logic, F-Logic Lite excludes negation and default inheritance, and allows only for a limited form of cardinality constraints.

We now show that F-Logic Lite can be expressed by using weakly-guarded Datalog$^\pm$ rules and a single EGD. We denote the Datalog$^\pm$ that we obtain in this translation with $\Sigma_{FLL}$, with $\Sigma_{FLL} = \{\rho_i\}_{1 \leq i \leq 12}$. The rules are shown below.

*(1)* $\mathsf{type}(O,A,T), \mathsf{data}(O,A,V) \to \mathsf{member}(V,T)$.
*(2)* $\mathsf{sub}(C_1,C_3), \mathsf{sub}(C_3,C_2) \to \mathsf{sub}(C_1,C_2)$.
*(3)* $\mathsf{member}(O,C), \mathsf{sub}(C,C_1) \to \mathsf{member}(O,C_1)$.
*(4)* $\mathsf{data}(O,A,V), \mathsf{data}(O,A,W), \mathsf{funct}(A,O) \to V = W$.
Note that this is the only EGD in this axiomatization.
*(5)* $\mathsf{mandatory}(A,O) \to \exists V\, \mathsf{data}(O,A,V)$.
Note that this is a TGD with an existential variable in the head (variable $V$).
*(6)* $\mathsf{member}(O,C), \mathsf{type}(C,A,T) \to \mathsf{type}(O,A,T)$.
*(7)* $\mathsf{sub}(C,C_1), \mathsf{type}(C_1,A,T) \to \mathsf{type}(C,A,T)$.
*(8)* $\mathsf{type}(C,A,T_1), \mathsf{sub}(T_1,T) \to \mathsf{type}(C,A,T)$.
*(9)* $\mathsf{sub}(C,C_1), \mathsf{mandatory}(A,C_1) \to \mathsf{mandatory}(A,C)$.
*(10)* $\mathsf{member}(O,C), \mathsf{mandatory}(A,C)$.
$\to \mathsf{mandatory}(A,O)$
*(11)* $\mathsf{sub}(C,C_1), \mathsf{funct}(A,C_1) \to \mathsf{funct}(A,C)$.
*(12)* $\mathsf{member}(O,C), \mathsf{funct}(A,C) \to \mathsf{funct}(A,O)$.

Now, we present a set $\Sigma'_{FLL}$ of TGDs and EGDs that is equivalent to $\Sigma_{FLL}$, but enjoys the desirable property that the single EGD in it is a key dependency. The set of rules $\Sigma'_{FLL}$ is obtained from $\Sigma_{FLL}$ by: *(1)* adding to $\mathcal{R}$ the ternary predicate $\mathsf{data}'$, thus obtaining $\mathcal{R}'$; *(2)* replacing $\rho_4$ with the rule $\mathsf{data}'(O,A,V), \mathsf{data}'(O,A,W) \to V = W$, that we denote with $\rho'_4$; *(3)* adding the rule $\mathsf{mandatory}(A,O), \mathsf{funct}(A,O) \to \exists V\, \mathsf{data}'(O,A,V)$, that we denote with $\rho_{13}$. It can be straightforwardly shown that: *(1)* all TGDs in $\Sigma'_{FLL}$ are weakly-guarded; *(2)* the single EGD $\rho'_4$ in $\Sigma'_{FLL}$ is a key dependency; *(3)* $\rho'_4$ is non-conflicting with all the other TGDs in $\Sigma'_{FLL}$. Moreover, it is not difficult to prove that, for every query $Q$ expressed on $\mathcal{R}'$ but without the predicate $\mathsf{data}'$ (i.e., with predicates in $\mathcal{R}$ only), and for every instance $D$, we have $D \cup \Sigma_{FLL} \models Q$ iff $D \cup \Sigma'_{FLL} \models Q$. By the above considerations, and by Theorem 24, we can restrict our attention solely on the TGDs in $\Sigma_{FLL}$.

By a polynomial reduction from the 3-COLORABILITY problem, we can show the following complexity result.

**Theorem 30 [11, 9]** *Query answering under F-Logic Lite rules is* NP*-hard.*

F-Logic Lite rules have also some interesting properties, which are formalized as *Polynomial Clouds Criterion* in [9]. First, for every instance $D$, the number of clouds in $\mathit{chase}(\Sigma_{FLL}, D)$ that are pairwise non-$D$-isomorphic is polynomial in the size $|D|$ of $D$. Second, for every atom $\mathbf{a}$, the set $\mathit{cloud}(\Sigma_{FLL}, D, \mathbf{a})$ can be computed in time polynomial in $|D|$ from $\mathbf{a}$ and $\mathit{cloud}(\Sigma, D, \mathbf{b})$ (whenever $\mathbf{b}$ exists), where $\mathbf{b}$ is the predecessor of $\mathbf{a}$ in the chase forest. Starting from the above considerations, it is possible to get the following result.

**Theorem 31 [11, 9]** *Conjunctive query answering under F-Logic Lite rules is in* NP.

From Theorems 30 and 31, we immediately get:

**Corollary 32** *Conjunctive query answering under F-Logic Lite rules is* NP*-complete.*

## 8. STRATIFIED NEGATION

In this section, we describe an extension of Datalog$^\pm$ with stratified negation. After adding negation to TGD bodies and queries, we define a semantics based on canonical models. Finally, we present a perfect model semantics and we show that it coincides with the previous one. 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. Note that by the results of Section 7, this also provides a natural stratified negation for the *DL-Lite* family.

## 8.1 Normal TGDs

We now define normal TGDs, which are informally TGDs that may also have negated atoms in their bodies. A *normal TGD (NTGD)* has the form $\forall\mathbf{X}\forall\mathbf{Y}\,\Phi(\mathbf{X},\mathbf{Y}) \to \exists\mathbf{Z}\Psi(\mathbf{X},\mathbf{Z})$, where $\Phi(\mathbf{X},\mathbf{Y})$ is a conjunction of atoms and negated atoms over $\mathcal{R}$, and $\Psi(\mathbf{X},\mathbf{Z})$ is a conjunction of atoms over $\mathcal{R}$. It is also abbreviated as $\Phi(\mathbf{X},\mathbf{Y}) \to \exists\mathbf{Z}\Psi(\mathbf{X},\mathbf{Z})$. As in the case of standard TGDs, we can assume that $\Psi(\mathbf{X},\mathbf{Z})$ is a singleton atom. We denote by $\mathit{head}(\sigma)$ the atom in the head of $\sigma$, and by $\mathit{body}^+(\sigma)$ and $\mathit{body}^-(\sigma)$ the sets of all positive and negative atoms (without "$\neg$") in the body of $\sigma$, respectively. We say that $\sigma$ is *guarded* iff it contains a positive atom in its body that contains all universally quantified variables of $\sigma$. We say that $\sigma$ is *linear* iff $\sigma$ is guarded and has exactly one positive atom in its body.

We extend BCQs by negation as follows. A *normal Boolean conjunctive query (NBCQ)* $Q$ is an existentially closed conjunction of atoms and negated atoms

$$\exists\mathbf{X}\, p_1(\mathbf{X}) \wedge \cdots \wedge p_m(\mathbf{X}) \wedge \neg p_{m+1}(\mathbf{X}) \wedge \cdots \wedge \neg p_{m+n}(\mathbf{X}),$$

where $m, n \geqslant 1$. We denote by $Q^+$ and $Q^-$ the positive resp. negative atoms (without "$\neg$") of $Q$. We say $Q$ is *safe* iff every variable in a negative atom also occurs in a positive atom.

*Example 14.* Consider the following set of guarded normal TGDs $\Sigma$, expressing that (1) if a driver has a non-valid license and drives, then he violates a traffic law, and (2) a license that is not suspended is valid:

$$\sigma\colon \mathit{hasLic}(D,L), \mathit{drives}(D), \neg \mathit{valid}(L) \rightarrow \exists I\, \mathit{viol}(D,I)\,;$$
$$\sigma'\colon \mathit{hasLic}(D,L), \neg \mathit{susp}(L) \rightarrow \mathit{valid}(L)\,.$$

Then, asking whether John commits a traffic violation and whether there exist traffic violations without driving can be expressed by the safe BCQs $Q_1 = \exists X\, \mathit{viol}(\mathit{john}, X)$ and $Q_2 = \exists D, I\, \mathit{viol}(D,I) \wedge \neg \mathit{drives}(D)$, respectively.

## 8.2 Canonical Models

We now define the concept of a stratification of normal TGDs and their canonical model semantics via iterative universal models along a stratification. We then define the semantics of safe NBCQs via such canonical models. We finally show how answering safe NBCQs can be done via an iterative chase procedure, which is tractable in the data complexity.

We define a *stratification* of a set of normal TGDs $\Sigma$ as a mapping $\mu\colon \mathcal{R} \rightarrow \{0, 1, \ldots, k\}$ such that for each $\sigma \in \Sigma$:

(i) $\mu(\mathit{pred}(\mathit{head}(\sigma))) \geqslant \mu(\mathit{pred}(\mathbf{a}))$ for all $\mathbf{a} \in \mathit{body}^+(\sigma)$;
(ii) $\mu(\mathit{pred}(\mathit{head}(\sigma))) > \mu(\mathit{pred}(\mathbf{a}))$ for all $\mathbf{a} \in \mathit{body}^-(\sigma)$.

We call $k \geqslant 0$ the *length* of $\mu$. For every $i \in \{0, \ldots, k\}$, we then define $\Sigma_i = \{\sigma \in \Sigma \mid \mu(\mathit{pred}(\mathit{head}(\sigma))) = i\}$ and $\Sigma_i^\star = \{\sigma \in \Sigma \mid \mu(\mathit{pred}(\mathit{head}(\sigma))) \leqslant i\}$. We say $\Sigma$ is *stratified* iff it has a stratification $\mu$ of some length $k \geqslant 0$. Note that the above notion of stratification generalizes the classical notion of stratification for Datalog with negation but without existentially quantified variables [3].

*Example 15.* The mapping $\mu$ where $\mu(\mathit{susp}) = \mu(\mathit{hasLic}) = \mu(\mathit{drives}) = 0$, $\mu(\mathit{valid}) = 1$, and $\mu(\mathit{viol}) = 2$ is a stratification of the set of guarded normal TGDs $\Sigma$ in Example 14.

We next define the notion of indefinite grounding, which extends the standard grounding (where rules are replaced by all their possible instances over constants) towards existentially quantified variables. The set of nulls $\Delta_N$ is partitioned into infinite sets of nulls $\Delta_{\sigma,X}$ (which can be seen as Skolem terms by which $X$ can be replaced), one for every TGD $\sigma \in \Sigma$ and every existentially quantified variable $X$ in $\sigma$. An *indefinite instance* of an NTGD $\sigma$ is obtained from $\sigma$ by replacing every universally quantified variable by an element from $\Delta \cup \Delta_N$ and every existentially quantified variable $X$ by an element from $\Delta_{\sigma,X}$. The *indefinite grounding* of $\Sigma$, denoted $\mathit{ground}(\Sigma)$, is the set of all its indefinite instances. We denote by $\mathit{HB}_\Sigma$ the set of all atoms built from predicate symbols from $\Sigma$ and arguments from $\Delta \cup \Delta_N$.

We are now ready to define canonical models.

*Definition 6.* Given a database $D$ under a set of guarded normal TGDs $\Sigma$, we define the sets $S_i$ along a stratification $\mu\colon \mathcal{R} \rightarrow \{0, 1, \ldots, k\}$ of $\Sigma$ as follows:

(i) $S_0$ is a universal model of $D$ given $\Sigma_0$;
(ii) if $i > 0$, then $S_i$ is a universal model of $S_{i-1}$ given $\Sigma_i^{S_{i-1}}$, where $\Sigma_i^{S_{i-1}}$ is obtained from $\mathit{ground}(\Sigma_i)$ by (i) deleting all $\sigma$ such that $\mathit{body}^-(\sigma) \cap S_{i-1} \neq \emptyset$ and (ii) removing the negative body from the remaining $\sigma$'s.

Then, $S_k$ is a *canonical model* of $D$ given $\Sigma$.

*Example 16.* Consider again the guarded normal TGDs $\Sigma$ of Example 14 and the database $D = \{\mathit{susp}(l), \mathit{drives}(\mathit{john}, c), \mathit{hasLic}(\mathit{john}, l)\}$. Then, $\Sigma_0 = \emptyset$, $\Sigma_1 = \{\sigma'\}$, and $\Sigma_2 = \{\sigma\}$, and we obtain $S_0 = S_1 = D$, and $S_2$ is homomorphically equivalent to $D \cup \{\mathit{viol}(\mathit{john}, i)\}$, where $i$ is a null.

Canonical models of $D$ given $\Sigma$ are in fact also models of $D$ given $\Sigma$. In general, there are several canonical models, which are all homomorphically equivalent. This shows that all canonical models of $D$ given $\Sigma$ are universal relative to all canonical models. Note that they are generally not universal relative to *all* models of $D$ given $\Sigma$.

We finally define the semantics of safe NBCQs via canonical models. A BCQ $Q$ *evaluates to true* in $D$ given a set of guarded normal TGDs $\Sigma$, denoted $D \cup \Sigma \models_{\mathit{strat}} Q$, iff there exists a homomorphism that maps $Q$ into a canonical model $S_k$ of $D$ given $\Sigma$. A safe NBCQ $Q$ *evaluates to true* in $D$ given $\Sigma$, denoted $D \cup \Sigma \models_{\mathit{strat}} Q$, iff there exists a homomorphism from $Q^+$ to a canonical model of $D$ given $\Sigma$, which cannot be extended to a homomorphism from some $Q^+ \cup \{\mathbf{a}\}$, where $\mathbf{a} \in Q^-$, to a canonical model of $D$ given $\Sigma$.

*Example 17.* Consider again the guarded normal TGDs $\Sigma$ of Example 14 and the database $D$ of Example 16. By the canonical model shown in Example 16, the BCQs $Q_1$ and $Q_2$ are answered positively and negatively, respectively.

A canonical model can be determined via iterative chases, where every chase may be infinite. We next show that for answering NBCQs, it is sufficient to consider finite parts of these chases. We first give some preliminary definitions.

Given a set of atoms $S$, we denote by $\mathit{chase}^S(\Sigma, D)$ a slightly modified oblivious chase where the TGD chase rule is *applicable* on an NTGD $\sigma$ iff the homomorphism $h$ maps every atom in $\mathit{body}^-(\sigma)$ to an atom not from $S$, and in that case, the TGD chase rule is applied on the TGD obtained from $\sigma$ by removing the negative body of $\sigma$. Then, we denote by $\mathit{g\text{-}chase}^{\ell,S}(\Sigma, D)$ the set of all atoms of depth at most $\ell$ in the guarded chase forest.

The next result shows that safe NBCQs $Q$ can be evaluated on finite parts of iterative chases, namely, on iterative guarded chase forests of depths depending only on $Q$ and $\mathcal{R}$.

**Theorem 33 [10]** *Let $\mathcal{R}$ be a relational schema, $\Sigma$ be a set of stratified guarded NTGDs over $\mathcal{R}$, $D$ be a database for $\mathcal{R}$, and $Q$ be a safe NBCQ over $\mathcal{R}$. Then, there exists some $\ell \geqslant 0$, which depends only on $Q$ and $\mathcal{R}$, such that $D \cup \Sigma \models_{\mathit{strat}} Q$ implies that $Q$ can be evaluated on $S_k$, where the sets $S_i$, $i \in \{0, \ldots, k\}$, are defined as follows:*

*(i) $S_0 = \mathit{g\text{-}chase}^{\ell}(\Sigma_0, D)$;*
*(ii) if $i > 0$, then $S_i = \mathit{g\text{-}chase}^{\ell, S_{i-1}}(\Sigma_i, S_{i-1})$.*

The following result shows that answering safe NBCQs in guarded Datalog$^\pm$ with stratified negation is data tractable.

**Theorem 34 [10]** *Let $\mathcal{R}$ be a relational schema, $\Sigma$ a set of stratified guarded NTGDs over $\mathcal{R}$, $D$ a database for $\mathcal{R}$, and $Q$ a safe NBCQ over $\mathcal{R}$. Then, deciding $D \cup \Sigma \models_{\mathit{strat}} Q$ can be done in polynomial time in the data complexity.*

The next result shows that answering safe NBCQs in linear Datalog$^\pm$ with stratified negation is FO-rewritable.

**Theorem 35** [10] *Let $\mathcal{R}$ be a relational schema, $\Sigma$ be a set of stratified linear NTGDs over $\mathcal{R}$, $D$ be a database for $\mathcal{R}$, and $Q$ be a safe NBCQ over $\mathcal{R}$. Then, $Q$ is FO-rewritable.*

## 8.3 Perfect Models

We now introduce the perfect model semantics of guarded Datalog$^{\pm}$ with stratified negation, and show that it coincides with the canonical model semantics, which gives evidence that the semantics is quite natural.

We first define the strict and reflexive relations $\prec$ and $\preccurlyeq$ as follows. Given a database $D$ under a set of guarded normal TGDs $\Sigma$, the relations $\prec$ and $\preccurlyeq$ on the set of all indefinite atoms are the smallest relations that satisfy (i) to (iv):

(i) $\mu(\mathit{head}(\sigma)) \preccurlyeq \mu(\mathbf{a})$ for every $\sigma \in \mathit{ground}(\Sigma)$ and every $\mathbf{a} \in \mathit{body}^{+}(\sigma)$,
(ii) $\mu(\mathit{head}(\sigma)) \prec \mu(\mathbf{a})$ for every $\sigma \in \mathit{ground}(\Sigma)$ and every $\mathbf{a} \in \mathit{body}^{-}(\sigma)$,
(iii) $\prec$ and $\preccurlyeq$ are transitively closed, and
(iv) $\prec$ is a subset of $\preccurlyeq$.

We are now ready to define perfect models.

*Definition 7.* Let $D$ be a database under a set of guarded normal TGDs $\Sigma$. For sets $M, N \subseteq HB_{\Sigma}$, we say $M$ is *preferable* to $N$, denoted $M \leqslant N$, iff some homomorphism $h$ exists such that for every $a \in h(M) - N$, there exists some $b \in N - h(M)$ such that $a \prec b$. We say $M$ is a *perfect model* of $D$ given $\Sigma$ iff $M \leqslant N$ for all models $N$ of $D$ given $\Sigma$.

It is possible to show [10] that the perfect model semantics coincides with the canonical model semantics. Since the notion of perfect model is independent of a concrete stratification, this also implies the important result that the canonical model semantics is independent of a concrete stratification.

## 9. CONCLUSION

In this paper, we have surveyed recent results about Datalog$^{\pm}$, a family of expressive extensions of Datalog that allow for a general and natural formalization of ontologies. We have presented the different languages in the Datalog$^{\pm}$ family along with tight complexity bounds for all cases. We have then shown that the Datalog$^{\pm}$ family is capable of expressing the most common tractable ontology languages that are currently adopted in the Semantic Web and databases, namely, the *DL-Lite* family of DLs and *F-Logic Lite*. We have finally shown how stratified negation can be added to Datalog$^{\pm}$ while keeping ontology querying tractable in the data complexity.

Datalog$^{\pm}$ is a natural and very general framework that closes the gap between query answering in databases, on the one hand, and ontology querying in DLs and the Semantic Web, on the other hand. It is an exciting family of languages for ontology querying, which are important in their own right and well worth being studied more deeply. Datalog$^{\pm}$ allows to naturally satisfy the emerging need of incorporating ontologies in the database area. Furthermore, it paves the way for applying standard database technology in the areas of DLs and the Semantic Web, as it has been shown in this paper for stratified negation, but which is also highly interesting for areas such as data integration and exchange [23]. In particular, such a transfer of database technology may help to satisfy the current need of DLs and the Semantic Web for scalable and efficient techniques.

In future research, we aim especially at making Datalog$^{\pm}$ even more powerful, without destroying its nice computational properties, so to also allow for embedding further, more expressive ontology languages, especially further tractable ontology languages, such as e.g. Horn-$\mathcal{SHIQ}$ [28], but also those involving disjunctive knowledge. Furthermore, we work on generalizations of the class of non-conflicting keys, which does not interact with TGDs. We also intend to find a syntactic criterion to restrict the class of weakly guarded TGDs and to lower its complexity. A semantic criterion is found in [9], which ensures that there are polynomially many clouds relative to the size of the database. Finally, we plan to investigate relevant first-order fragments, and get for them precise time bounds for Courcelle's theorem.

## 10. ACKNOWLEDGMENTS

The work of Andrea Calì and Georg Gottlob was supported by the EPSRC grant Number EP/E010865/1 "Schema Mappings and Automated Services for Data Integration". Georg 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 (DFG) under the Heisenberg Programme.

## 11. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[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] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, 1988.
[4] C. Beeri, R. Fagin, D. Maier, A. O. Mendelzon, J. D. Ullman, and M. Yannakakis. Properties of acyclic database schemes. In *Proc. of STOC 1981*, pages 355–362. ACM Press, 1981.
[5] C. Beeri and M. Y. Vardi. The implication problem for data dependencies. In *Proc. of ICALP 1981*, volume 115 of *LNCS*, pages 73–85. Springer, 1981.
[6] C. Beeri and M. Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984.
[7] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284:34–43, 2001.
[8] D. Brickley and R. V. Guha. RDF vocabulary description language 1.0: RDF Schema, 2004. W3C Recommendation.
[9] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. Unpublished technical report, 2008. Available from the authors or at `http://benner.dbai.tuwien.ac.at/staff/gottlob/CGK.pdf`. This is the full and revised version of a preliminary short version which has appeared in *Proc. of KR 2008*, pages 70–80. AAAI Press, 2008.
[10] A. Calì, G. Gottlob, and T. Lukasiewicz. A general Datalog-based framework for tractable query

answering over ontologies. In *Proc. of PODS 2009*. ACM Press, 2009. To appear.

[11] A. Calì and M. Kifer. Containment of conjunctive object meta-queries. In *Proc. of VLDB 2006*, pages 942–952. ACM Press, 2006.

[12] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of PODS 2003*, pages 260–271. ACM Press, 2003.

[13] 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.

[14] M. A. Casanova, R. Fagin, and C. H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. *Journal of Computer and System Sciences*, 28:29–59, 1984.

[15] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer, New York, NY, 1990.

[16] A. K. Chandra, H. R. Lewis, and J. A. Makowsky. Embedded implicational dependencies and their inference problem. In *Proc. of STOC 1981*, pages 342–354. ACM Press, 1981.

[17] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of STOC 1977*, pages 77–90. ACM Press, 1977.

[18] A. K. Chandra and M. Y. Vardi. The implication problem for functional and inclusion dependencies is undecidable. *SIAM J. Comput.*, 14:671–677, 1985.

[19] B. Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[20] J. de Bruijn and S. Heymans. Logical foundations of (e)RDF(S): Complexity and reasoning. In *Proc. of ISWC 2007*, volume 4825 of *LNCS*, pages 86–99. Springer, 2007.

[21] A. Deutsch, A. Nash, and J. B. Remmel. The chase revisited. In *Proc. of PODS 2008*, pages 149–158. ACM Press, 2008.

[22] A. Deutsch and V. Tannen. Reformulation of XML queries and constraints. In *Proc. of ICDT 2003*, volume 2572 of *LNCS*, pages 225–241. Springer, 2003.

[23] 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.

[24] M. E. Goncalves and E. Grädel. Decidability issues for action guarded logics. In *Proc. of DL 2000*, pages 123–132, 2000.

[25] G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The Lixto data extraction project – back and forth between theory and practice. In *Proc. of PODS 2004*, pages 1–12. ACM Press, 2004.

[26] G. Gottlob and C. H. Papadimitriou. On the complexity of single-rule Datalog queries. *Inf. Comput.*, 183(1):104–122, 2003.

[27] E. Grädel. On the restraining power of guards. *J. Symb. Log.*, 64(4):1719–1742, 1999.

[28] U. Hustadt, B. Motik, and U. Sattler. Data complexity of reasoning in very expressive description logics. In *Proc. of IJCAI 2005*, pages 466–471, 2005.

[29] 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.

[30] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42:741–843, 1995.

[31] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS 2002*, pages 233–246. ACM Press, 2002.

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

[33] J. Mitchell. The implication problem for functional and inclusion dependencies. *Information and Control*, 56:154–173, 1983.

[34] P. F. Patel-Schneider and I. Horrocks. Position paper: A comparison of two modelling paradigms in the Semantic Web. In *Proc. of WWW 2006*, pages 3–12. ACM Press, 2006.

[35] 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.

[36] X. Qian. Query folding. In *Proc. of ICDE 1996*, pages 48–55. IEEE, 1996.

[37] J. D. Ullman. *Principles of Database and Knowledge Base Systems*. CS Press, Rockville, MD, 1989.

[38] M. Y. Vardi, 1984. Personal communication reported in [29].

[39] M. Y. Vardi. On the complexity of bounded-variable queries. In *Proc. of PODS 1995*, pages 266–176. ACM Press, 1995.