A Logical Toolbox for Ontological Reasoning

Andrea Calì*

Department of Computer Science
and Information Systems
Birkbeck, University of London

andrea@dcs.bbk.ac.uk

Georg Gottlob*
Thomas Lukasiewicz
Andreas Pieris

Department of Computer Science
University of Oxford

name.surname@cs.ox.ac.uk

ABSTRACT

In ontology-enhanced database systems, an *ontology* on top of the extensional database expresses intensional knowledge that enhances the database schema. Queries posed to such systems are to be evaluated considering all the knowledge inferred from the data by means of the ontology; in other words, queries are to be evaluated against the logical theory constituted by the data and the ontology. In this context, tractability of query answering is a central issue, given that the data size is normally very large. This paper surveys results on a recently introduced family of Datalog-based languages, called Datalog+/-, which is a useful logical toolbox for ontology modeling and for ontology-based query answering. We present different Datalog+/- languages and related complexity results, showing that Datalog+/- can be successfully adopted due to its clarity, expressiveness and its good computational properties.

1. INTRODUCTION

Datalog (see, e.g., [1]) has been widely used as a database programming and query language for long time. It is rarely used directly as a query language in corporate application contexts. However, it is used as an inference engine for knowledge processing within several software tools, and has recently gained popularity in the context of various applications, such as web data extraction [6, 7, 25], source code querying and program analysis [28], and modeling distributed systems [2].

At the same time, Datalog has been shown to be too limited to be effectively used to model ontologies and expressive database schemata, as explained in [34]. In this respect, the main missing feature in Datalog is the possibility of expressing existential quantification in the head; this was addressed in the literature by introducing Datalog with *value invention* [10, 32].

This paper surveys recently introduced variants of Datalog, grouped in a family of languages that was named *Datalog*$^{\pm}$ (also written Datalog+/- whenever appropriate). Datalog$^{\pm}$ extends Datalog by allowing features such as existential quantifiers, the equality predicate, and the truth constant *false* to appear in rule heads. On the other hand, the resulting language has to be syntactically restricted, so as to achieve decidability, and in some relevant cases even tractability.

A basic Datalog program consists of a set of universally quantified function-free Horn clauses. The predicate symbols appearing in such a program either refer to *extensional database (EDB) predicates*, whose values are given via an input database, or to *intensional database (IDB) predicates*, whose values are computed by the program. In standard Datalog, EDB predicate symbols appear in rule-bodies only. A simple example is the program

$$\begin{array}{rcl} s(X) & \rightarrow & r(X), \\ r(X), e(X,Y) & \rightarrow & r(Y), \end{array}$$

which takes as input EDB a directed graph, given by a binary edge relation e, plus a set of special vertices of this graph given by a unary relation s. The above program computes the set r of all vertices in the graph reachable via a directed path of nonnegative length from special vertices.

Given an EDB D and a Datalog program Σ, let us denote by $D \cup \Sigma$ the logical theory containing both the facts (i.e., ground atoms) of D and the rules of Σ. We say that a BCQ q evaluates to true over D and Σ iff $D \cup \Sigma \models q$. For Datalog$^{\pm}$ languages, the notion of query answering is the same, with the difference that rules allow for existential quantification in the head; such rules, in database parlance, are also known as *tuple-generating dependencies (TGDs)*.

For example, with TGDs we are able to express that every person has a father who, moreover, is himself a

*Alternate address: Oxford-Man Institute of Quantitative Finance, University of Oxford; E-mail: name.surname@oxford-man.ox.ac.uk

Database Principles Column. Column editors: Leonid Libkin, School of Informatics, University of Edinburgh, Edinburgh, EH8 9AB, UK. E-mail: libkin@inf.ed.ac.uk. Pablo Barceló, Department of Computer Science, University of Chile. E-mail: pbarcelo@dcc.uchile.cl.

person:

$$\begin{array}{rcl} person(X) & \rightarrow & \exists Y father(X, Y), \\ father(X, Y) & \rightarrow & person(Y). \end{array}$$

Note that here the relation *person*, which is supplied in the input with an initial value, is actually modified. Therefore, we no longer require (as in standard Datalog) that EDB relation symbols cannot occur in rule-heads.

Ontology querying (and possibly a number of other applications such as data exchange and web data extraction) can profit from appropriate forms of Datalog extended by the possibility of using rules with existential quantifiers in their heads. Other useful features are, for example, equality in rule-heads —rules with the equality predicate in the head are known as *equality-generating dependencies (EGDs)*, and they capture the well-known *functional dependencies* (see, e.g., [1]). Unfortunately, already for sets of TGDs alone, most basic reasoning and query answering problems are undecidable. In particular, given a database D and a set Σ of TGDs, checking whether $D \cup \Sigma \models q$ for a ground fact q is undecidable [8]. Worse than that, undecidability holds even in case both q and Σ are *fixed*, and only D is given as input [11]. It is therefore important to identify large classes of Datalog-based rule sets Σ that are expressive enough for being useful in real applications, and allow for decidable query answering. A further desirable feature is the tractability of query answering in *data complexity*, that is, the complexity calculated by considering only the data as part of the input, whereas q and Σ are fixed; this type of complexity is an important measure, because we can realistically assume that the EDB D is the only really large object in the input. The languages in the Datalog$^{\pm}$ family fulfil the above criteria.

One of the main tools used for proving favorable results about a number of Datalog$^{\pm}$ languages is the *chase procedure* [29, 31]. The chase is an algorithm that, roughly speaking, executes the rules of a Datalog$^{\pm}$ program Σ on input D in a forward chaining manner, thus inferring new knowledge by either adding new atoms or unifying symbols. In general, the chase procedure may terminate or not. The most notable syntactic restriction guaranteeing chase termination is *weak acyclicity* of TGDs, for which we refer the reader to the landmark paper [24]; more general syntactic restrictions were studied in [23, 33]. However, none of these restrictions is appropriate for ontology querying. One of the challenges of studying Datalog$^{\pm}$ is therefore to tackle the possible non-finiteness of the chase, which complicates query answering. The first approach to query answering in case of infinite chase is found in the milestone paper by Johnson and Klug [29].

Structure of the paper. After some technical definitions in Section 2, we report in Section 3 on the class of *guarded* TGDs, where each rule body is required to have an atom that covers all body variables of the rule. We then consider the even more restricted class of *linear* TGDs, for which query answering is *first-order rewritable* (*FO-rewritable*) which means that q and Σ can be transformed into a first-order query q_Σ such that $D \models q_\Sigma$ iff $D \cup \Sigma \models q$, for every extensional database D. This property, introduced in [18] in the context of DLs, is essential if D is a very large database. It implies that query answering can be deferred to a standard query language such as (non-recursive) SQL.

We present stickiness, a completely different paradigm for tractable query answering, in Section 4. Roughly speaking, the syntactic condition that defines sticky sets of TGDs, which will be given in detail in the following of the paper, guarantees that if we perform *resolution* [1] (or more precisely a variant of it that considers existential quantification in the head; see, e.g., [15]) starting from a subgoal, the *newly* introduced variables in all obtained subgoals appear at most once in each subgoal. The stickiness condition is easily testable. Stickiness also guarantees the desirable FO-rewritability property.

In Section 5, we first deal with *negative constraints*, i.e., rules whose head is the truth constant *false* denoted by $\bot$. It turns out that negative constraints can be introduced without any increase of complexity, as query answering can be computed in the same way as without negative constraints, after the evaluation of suitable queries. We then introduce equality-generating dependencies (EGDs), which are well-known to easily lead to undecidability of query answering in their simplest form, even when combined with simple classes of TGDs such as inclusion dependencies [16, 21]. We focus on a very simple, nevertheless extremely useful class of EGDs, namely *key dependencies* (or simply *keys*). We discuss semantic and syntactic conditions ensuring that keys are usable without destroying decidability and tractability.

Section 6 briefly describes how highly relevant tractable DL languages, in particular the main languages of the well-known *DL-Lite* family [18, 35], can be modeled in the Datalog$^{\pm}$ framework. Section 7 concludes the paper, and gives some directions for further research on the topic.

2. PRELIMINARIES

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

General. We define the following pairwise disjoint (infinite) sets of symbols: a set Γ of *constants* (constitute the "normal" domain of a database), a set Γ_N of *labeled nulls* (used as placeholders for unknown values, and thus can be also seen as variables), and a set Γ_V of variables (used in queries and constraints). Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. A lexicographic order is defined on $\Gamma \cup \Gamma_N$, such that every value of Γ_N follows all those of Γ. Sets of variables (or sequences, with a slight abuse of notation) are denoted as $\mathbf{X} = X_1, \ldots, X_k$, where $k \geqslant 0$. Let $[n]$ be the set $\{1, \ldots, n\}$, for any integer $n \geqslant 0$.

A *relational schema* $\mathcal{R}$ (or simply *schema*) is a set of *relational symbols* (or *predicates*), each with its associated arity. We write r/n to denote that the predicate r has arity n. A *position* $r[i]$ (in a schema $\mathcal{R}$) is identified by a predicate $r \in \mathcal{R}$ and its i-th argument (or attribute). A *term* t is a constant, null, or variable. An *atomic formula* (or simply *atom*) has the form $r(t_1, \ldots, t_n)$, where r/n is a relation, and $t_1, \ldots, t_n$ are terms. Conjunctions of atoms are often identified with the sets of their atoms.

A *substitution* from one set of symbols S_1 to another set of symbols S_2 is a function $h : S_1 \to S_2$ defined as follows: *(i)* $\varnothing$ is a substitution (the empty substitution), *(ii)* if h is a substitution, then $h \cup \{X \to Y\}$ is a substitution, where $X \in S_1$ and $Y \in S_2$, and h does not already contain some $X \to Z$ with $Y \neq Z$. If $X \to Y \in h$, then we write $h(X) = Y$. A *homomorphism* from a set of atoms A_1 to a set of atoms A_2, both over the same schema $\mathcal{R}$, is a substitution $h : \Gamma \cup \Gamma_N \cup \Gamma_V \to \Gamma \cup \Gamma_N \cup \Gamma_V$ such that: *(i)* if $t \in \Gamma$, then $h(t) = t$, and *(ii)* if $r(t_1, \ldots, t_n) \in A_1$, then $h(r(t_1, \ldots, t_n)) = r(h(t_1), \ldots, h(t_n)) \in A_2$. The notion of homomorphism naturally extends to conjunctions of atoms.

Databases and Queries. A *relational instance* (or simply *instance*) I for a schema $\mathcal{R}$ is a (possibly infinite) set of atoms of the form $r(\mathbf{t})$, where r/n is a predicate of $\mathcal{R}$, and $\mathbf{t} \in (\Gamma \cup \Gamma_N)^n$. We denote as $r(I)$ the set $\{\mathbf{t} \mid r(\mathbf{t}) \in I\}$. A *database* is a finite relational instance.

A *conjunctive query (CQ)* q of arity n over a schema $\mathcal{R}$, written as q/n, has the form $p(\mathbf{X}) \leftarrow \varphi(\mathbf{X}, \mathbf{Y})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms over $\mathcal{R}$, $\mathbf{X}$ and $\mathbf{Y}$ are sequences of variables of Γ_V or constants of Γ, and p is an n-ary predicate not occurring in $\mathcal{R}$. $\varphi(\mathbf{X}, \mathbf{Y})$ is called the *body* of q, denoted as $body(q)$. A *Boolean CQ (BCQ)* is a CQ of zero arity. The *answer* to a CQ q/n over an instance I, denoted as $q(I)$, is the set of all n-tuples $\mathbf{t} \in \Gamma^n$ for which there exists a homomorphism $h : \mathbf{X} \cup \mathbf{Y} \to \Gamma \cup \Gamma_N$ such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq I$ and $h(\mathbf{X}) = \mathbf{t}$. A BCQ has only the empty tuple $\langle\rangle$ as possible answer in which case it is said that has positive answer. Formally, a BCQ has *positive* answer over I, denoted as $I \models q$, if $\langle\rangle \in q(I)$.

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

We now define the notion of *query answering* under TGDs. Given a database D for a schema $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$, the *models* of D w.r.t. Σ, denoted as $mods(D, \Sigma)$, is the set of all instances I such that $I \models D \cup \Sigma$, i.e., $I \supseteq D$ and I satisfies Σ. The *answer* to a CQ q w.r.t. D and Σ, denoted as $ans(q, D, \Sigma)$, is the set $\{\mathbf{t} \mid \mathbf{t} \in q(I)$ for each $I \in mods(D, \Sigma)\}$. The *answer* to a BCQ q w.r.t. D and Σ is *positive*, denoted as $D \cup \Sigma \models q$, if $ans(q, D, \Sigma) \neq \varnothing$. Note that query answering under (general) TGDs is undecidable [8]. This holds even when the schema and the set of TGDs are fixed [11], and also in the case of singleton sets of TGDs [4].

Following Vardi's taxonomy [39], the *data complexity* of query answering is the complexity w.r.t. the database only, while the *combined complexity* is the complexity calculated by considering also the query and the set of TGDs as part of the input.

The two problems of CQ and BCQ answering under TGDs are LOGSPACE-equivalent [20, 23, 24, 29]. Henceforth, we thus focus only on the BCQ answering problem. We also recall that query answering under TGDs is equivalent to query answering under TGDs with singleton atoms in the head [11]. This is shown by means of a transformation from general TGDs to TGDs with single-atom heads [11]. Moreover, the transformation preserves the properties of the classes of TGDs that we consider in this paper. Therefore, all results for TGDs with singleton atoms in the head carry over to TGDs with multiple head-atoms. We thus always assume w.l.o.g. (unless stated otherwise) that every TGD has a singleton atom in its head.

The TGD Chase. The *chase procedure* (or simply *chase*) is a fundamental algorithmic tool introduced for checking implication of dependencies [31], and later for checking query containment [29]. Informally, the chase is a process of repairing a database w.r.t. a set of dependencies so that the resulted instance satisfies the dependencies. We shall use the term chase interchangeably for both the procedure and its result. The chase works on an instance through the so-called TGD *chase rule*. The TGD chase rule comes in two equivalent fashions: *oblivious* and *restricted* [11], where the restricted one repairs TGDs only when they are not satisfied. In the sequel, we focus on the oblivious one for technical clarity. The TGD chase rule defined below is the building block of the chase.

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

Given a database D and a set of TGDs Σ, the chase algorithm for D and Σ consists of an exhaustive application of the TGD chase, which leads to a (possibly infinite) instance, denoted as $chase(D, \Sigma)$. We assume that the chase algorithm is *fair*, i.e., each TGD that must be applied during the construction of $chase(D, \Sigma)$ eventually it is applied. Due to the fairness assumption, the (possibly infinite) chase for D and Σ is a *universal model* of D w.r.t. Σ, i.e., for each instance $I \in mods(D, \Sigma)$, there exists a homomorphism from $chase(D, \Sigma)$ to I [23, 24]. Using this fact it can be easily shown that for a BCQ q, $D \cup \Sigma \models q$ iff $chase(D, \Sigma) \models q$.

3. GUARDED & LINEAR DATALOG$^\pm$

For ontology querying purposes, we need to concentrate on cases where the chase produces an infinite universal solution, and also where no finite universal solution exists. Unfortunately, as already mentioned, query answering is undecidable in such cases. The recognition of expressive decidable classes of TGDs is a challenging problem. In this section we present the languages *guarded* and *linear Datalog*$^\pm$.

3.1 Guarded Datalog$^\pm$

We first discuss the class of *guarded* TGDs, which forms the language guarded Datalog$^\pm$, as a special class of TGDs which guarantees decidability of query answering, and even tractability w.r.t. data complexity. Queries relative to such TGDs can be evaluated over a finite part of the chase, whose size depends only on the query and the set of TGDs, but not on the database.

A TGD σ is *guarded* if it has a body-atom which contains all the universally quantified variables of σ. The leftmost such atom is the *guard atom* (or *guard*) of σ. The non-guard atoms are the *side atoms* of σ. For example, 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 [3].

Complexity Results. The next theorem, presented in [11], establishes combined complexity results for conjunctive query answering under guarded Datalog$^\pm$. The EXPTIME and 2EXPTIME-completeness results hold even if the input database is fixed. Note that an *atomic query* is a CQ with just one body-atom.

THEOREM 1. *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of guarded TGDs over $\mathcal{R}$. Let w be the maximum arity over all predicates of $\mathcal{R}$. Then, the following hold:*

a) *If q is an atomic query, then deciding whether $chase(D, \Sigma) \models q$ is* PTIME*-complete when Σ is fixed. The same problem is* EXPTIME*-complete if w is bounded, and* 2EXPTIME*-complete in general.*
b) *If q is a non-atomic query, then deciding whether $chase(D, \Sigma) \models q$ is* NP*-complete when Σ is fixed. The same problem is* EXPTIME*-complete if w is bounded, and* 2EXPTIME*-complete in general.*

The data complexity of query answering under guarded TGDs 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 Σ be a set of guarded TGDs over $\mathcal{R}$.

We first define the so-called *chase relation* for D and Σ, that is, a binary relation denoted $\xrightarrow{D,\Sigma}$, as follows. Suppose that during the construction of $chase(D, \Sigma)$ we apply a TGD $\sigma \in \Sigma$, with homomorphism h, and the atom $\underline{a}$ is obtained. Then, for each atom $\underline{b} \in body(\sigma)$, we have $h(\underline{b}) \xrightarrow{D,\Sigma} \underline{a}$. Intuitively, by exploiting $\xrightarrow{D,\Sigma}$, it is possible to extract all the chase derivations of $chase(D, \Sigma)$.

The *chase graph* for D and Σ is the directed graph with $chase(D, \Sigma)$ be the set of nodes, and having an edge from $\underline{a}$ to $\underline{b}$ if $\langle \underline{a}, \underline{b} \rangle \in \xrightarrow{D,\Sigma}$. We mark $\underline{a}$ as *guard* if $\underline{a}$ is the guard of σ. The *guarded chase forest* for D and Σ is the restriction of the chase graph for D and Σ to all atoms marked as guards and their children. The *guarded chase* of level up to $k \geqslant 0$ for D and Σ, denoted as $g\text{-}chase^k(D, \Sigma)$, is the set of all atoms in the forest of depth at most k.

It can be shown that (homomorphic images of) the query atoms are contained in a finite, initial part of the guarded chase forest, whose size is determined only by the query and the set of TGDs. However, this does not yet assure that also the whole derivation of the query atoms are contained in such a part of the guarded chase forest. This slightly stronger property is captured by the following definition.

Definition 1. We say that Σ has the *bounded guard-depth property (BGDP)* if, for each database D for $\mathcal{R}$ and for each BCQ q over $\mathcal{R}$, whenever there is a homomorphism μ that maps q into $chase(D, \Sigma)$, then there is a homomorphism λ of this kind such that all ancestors of $\lambda(body(q))$ in the chase graph for D and Σ are contained in $g\text{-}chase^k(D, \Sigma)$, where $k \geqslant 0$ depends only on q and Σ.

It is possible to show that guarded TGDs enjoy the BGDP. The proof is based on the observation 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 the set of TGDs (which is slightly larger than the one for the query atoms only). By this result, query answering under guarded TGDs is feasible in PTIME w.r.t. data complexity [11]. It is also hard for PTIME, as can be proved by reduction from propositional logic programming [12].

THEOREM 2. *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set of guarded TGDs over $\mathcal{R}$. Then, deciding whether $chase(D, \Sigma) \models q$ is* PTIME*-complete w.r.t. data complexity. If q is atomic, then* *the same problem is feasible in linear time w.r.t. data complexity.*

Extensions. It is important to say that guarded TGDs can be enriched by *stratified negation*, where non-monotonic negation may be used in TGD bodies and queries. A natural stratified negation for query answering over ontologies, which is in general based on several strata of infinite models, is proposed in [12].

An expressive language, which forms a generalization of guarded Datalog$^{\pm}$, is *weakly-guarded Datalog*$^{\pm}$ introduced in [11]. Roughly speaking, a set Σ of TGDs is weakly-guarded if, for each $\sigma \in \Sigma$, there exists an atom in *body*(σ), called a *weak-guard*, that contains only the universally quantified variables of σ that occur at positions where a "fresh" null of Γ_N can appear during the construction of the chase (and not all the universally quantified variables).

3.2 Linear Datalog$^{\pm}$

The class of *linear* TGDs, which forms the language linear Datalog$^{\pm}$, is a variant of the class of guarded TGDs, where query answering is highly tractable w.r.t. data complexity. A TGD is *linear* if it has only one atom in its body (which is automatically a guard). Notice that linear TGDs are strictly more expressive than inclusion dependencies, which are the simplest type of TGDs with just one body-atom and one head-atom, without repetition of variables. For example, the linear TGD $\mathit{supervises}(X, X) \rightarrow \mathit{manager}(X)$, which asserts that everyone supervising her/himself is a manager, is not expressible with inclusion dependencies.

Complexity Results. Query answering under linear TGDs is PSPACE-complete w.r.t. combined complexity. This result is obtained immediately by results in [19, 26, 29, 40].

THEOREM 3. *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of linear TGDs over $\mathcal{R}$. Then, deciding whether $\mathit{chase}(D, \Sigma) \models q$ is* PSPACE-*complete, even when the query is fixed.*

Let us now investigate the data complexity of query answering under linear TGDs. A class $\mathcal{C}$ of TGDs is *first-order rewritable* (henceforth abbreviated as *FO-rewritable*) if for every set Σ of TGDs in $\mathcal{C}$, and for every BCQ q, it is possible to construct a first-order query q_{Σ} such that, for every database D, $D \cup \Sigma \models q$ iff $D \models q_{\Sigma}$. Since answering first-order queries is in the highly tractable class AC$_0$ w.r.t. data complexity [41], it immediately follows that query answering under FO-rewritable classes of TGDs is in AC$_0$ w.r.t. data complexity.

We next recall the *bounded derivation-depth property*, introduced in [12], which is strictly stronger than the bounded guard-depth property. Informally, this property implies that (homomorphic images of) the query atoms along with their derivations are contained in a finite, initial part of the chase graph (rather than the guarded chase forest), whose size depends only on the query and and the set of TGDs. In the sequel, we denote by $\mathit{chase}^k(D, \Sigma)$ the *chase of level up to* $k \geqslant 0$ for D and Σ, that is, the set of all atoms of $\mathit{chase}(D, \Sigma)$ of derivation level at most k.

Definition 2. A set Σ of TGDs over a schema $\mathcal{R}$ has the *bounded derivation-depth property (BDDP)* if, for every database D for $\mathcal{R}$, and for every BCQ q over $\mathcal{R}$, whenever $\mathit{chase}(D, \Sigma) \models q$, then $\mathit{chase}^k(D, \Sigma) \models q$, where $k \geqslant 0$ depends only on q and Σ.

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

It is known that if a class of TGDs enjoys the BDDP, then it is also FO-rewritable [12]. The main ideas behind the proof of this result are informally as follows. Since the derivation depth and the number of body-atoms in TGDs are 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. As an immediate consequence we get the following result.

THEOREM 4. *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of linear TGDs over $\mathcal{R}$. Then, deciding whether $\mathit{chase}(D, \Sigma) \models q$ is in* AC$_0$ *w.r.t. data complexity.*

Small Query Rewritings. The rewriting algorithm employed in [12] in order to prove that the BDDP implies FO-rewritability is not very well-suited for practical implementations. In particular, the rewritten query obtained by applying this algorithm is of exponential size w.r.t. the given query and set of TGDs. The question that comes up is whether, under linear TGDs, a polynomially sized first-order query can be constructed. Interestingly, Gottlob and Schwentick shown recently that the answer to the above question is affirmative [27]. The key property underlying the proof of this result is the so-called *polynomial witness property* [27].

Definition 3. A class of TGDs $\mathcal{C}$ has the *polynomial witness property (PWP)* if, for every BCQ q over a schema $\mathcal{R}$, for every database D for $\mathcal{R}$, and for every set $\Sigma \in \mathcal{C}$ of TGDs over $\mathcal{R}$, the following holds: if $chase(D,\Sigma) \models q$, then there is a sequence of at most $k \geqslant 0$ chase steps whose atoms already entail q, where k is polynomial with respect to q and Σ, and independent from D. $\square$

As established in [27], given a BCQ q over a schema $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$ which falls in a class that enjoys the PWP, one can compute in polynomial time a non-recursive Datalog program P of polynomial size with respect to q and Σ such that, for every database D for $\mathcal{R}$, $chase(D,\Sigma) \models q$ iff $D \models P$. Moreover, it was shown that the class of linear TGDs enjoys the PWP.

4. STICKY DATALOG$^{\pm}$

Unfortunately, none of the formalisms presented in the previous section is expressive enough to be able to express simple cases that allow for joins in rule-bodies such as the rule $r(X,Y), r(Z,X) \rightarrow s(X)$; clearly, the above rule is non-guarded since there is no body-atom that contains all the universally quantified variables. In this section, we present another Datalog$^{\pm}$ language which hinges on a paradigm, called *stickiness*, which is very different from guardedness and allows for joins in rule-bodies (with some realistic restriction to ensure decidability).

Formal Definition. The formal definition of the class of *sticky* sets of TGDs (which forms the language *sticky Datalog*$^{\pm}$) is based heavily on a variable-marking procedure called SMarking. This procedure accepts as input a set of TGDs Σ, and marks the variables that occur in the body of the TGDs of Σ. Formally, SMarking(Σ) works as follows. First, we apply the so-called *initial marking* step: for each TGD $\sigma \in \Sigma$, and for each variable V in $body(\sigma)$, if there exists an atom $\underline{a}$ in $head(\sigma)$ such that V does not appear in $\underline{a}$, then we mark each occurrence of V in $body(\sigma)$. Then, we apply exhaustively (i.e., until a fixpoint is reached) the *propagation* step: for each pair of TGDs $\langle \sigma, \sigma' \rangle \in \Sigma \times \Sigma$ (including the case $\sigma = \sigma'$), if a universally quantified variable V occurs in $head(\sigma)$ at positions $\pi_1, \ldots, \pi_m$, for $m \geqslant 1$, and there exists an atom $\underline{a} \in body(\sigma')$ such that at each position $\pi_1, \ldots, \pi_m$ a marked variable occurs, then we mark each occurrence of V in $body(\sigma)$. We are now ready to give the formal definition of sticky sets of TGDs.

Definition 4. Consider a set Σ of TGDs over a schema $\mathcal{R}$. Σ is *sticky* if there is no TGD $\sigma \in$ SMarking(Σ) such that a marked variable occurs in $body(\sigma)$ more than once.

Example 1. Consider the following set of TGDs. We mark the body-variables, according to the SMarking procedure, with hat, e.g., $\hat{X}$:

$$\begin{array}{rcl} r(\hat{X},\hat{Y}) & \rightarrow & \exists Z\, r(Y,Z) \\ r(X,\hat{Y}) & \rightarrow & s(X) \\ s(X), s(Y) & \rightarrow & t(X,Y) \\ r(X,\hat{Y}), r(\hat{Z},X) & \rightarrow & s(X). \end{array}$$

The only variable that occurs more than once in the body of a TGD, i.e., the variable X in the body of the last TGD, is non-marked. Therefore, Σ is a sticky set. $\blacksquare$

Consider the simple database $D = \{r(a,a)\}$ and the set Σ of TGDs given in the above example. It is easy to verify that $chase(D,\Sigma)$ is infinite; recall that for ontology querying purposes we need to focus on cases where the chase is (in general) infinite. In fact, the first rule of Σ by itself leads to an infinite chase. Moreover, the third rule of Σ is a prime example of non-guardedness.

It is straightforward to see that the problem of identifying sticky sets of TGDs, that is, given a set Σ of TGDs, decide whether Σ is sticky, is feasible in polynomial time. This follows by observing that at each application of the propagation step, during the execution of the SMarking procedure, at least one body-variable is marked. Thus, after polynomially many steps the SMarking procedure terminates.

Sticky Property. It is interesting to see that the chase constructed under a sticky set of TGDs enjoys a syntactic property called *sticky property*.

Definition 5. Consider a database D for a schema $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$. Suppose that in the construction of $chase(D,\Sigma)$ we apply $\sigma \in \Sigma$, with homomorphism h, that has a variable V appearing more than once in its body, and the atoms $\underline{a}_1, \ldots, \underline{a}_k$, for $k \geqslant 1$, are generated. We say that $chase(D,\Sigma)$ has the *sticky property* if, for each atom $\underline{a} \in \{\underline{a}_1, \ldots, \underline{a}_k\}$, $h(V)$ occurs in $\underline{a}$, and also in every atom $\underline{b}$ such that $\langle \underline{a}, \underline{b} \rangle$ is in the transitive closure of $\xrightarrow{D,\Sigma}$.

Intuitively speaking, the sticky property implies that, during the construction of the chase, whenever a rule σ is applied, then the symbols (constants or nulls) which are associated (via homomorphism) to the join body-variables of σ appear in the generated atom $\underline{a}$, and also in all atoms resulting from some chase derivation involving $\underline{a}$, "sticking" to them (hence the name "sticky sets of TGDs").

As established in [14], stickiness is a sufficient condition for the sticky property of the chase.

THEOREM 5. *Consider a set Σ of TGDs over a schema $\mathcal{R}$. If Σ is sticky, then $chase(D,\Sigma)$ enjoys the sticky property, for every database D for $\mathcal{R}$.*

Complexity Results. The next theorem, presented in [14], establishes combined complexity results for conjunctive query answering under sticky Datalog$^{\pm}$.

THEOREM 6. *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a sticky set Σ of TGDs over $\mathcal{R}$. Then, deciding whether chase$(D, \Sigma) \models q$ is* NP*-complete if Σ is fixed, and* EXPTIME*-complete in general.*

As shown in [14], the class of sticky sets of TGDs enjoys the BDDP (see Definition 2), and thus sticky sets of TGDs are FO-rewritable. The next result follows immediately.

THEOREM 7. *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of linear TGDs over $\mathcal{R}$. Then, deciding whether chase$(D, \Sigma) \models q$ is in* AC_0 *w.r.t. data complexity.*

Interestingly, the class of sticky sets of TGDs enjoys the PWP (see Definition 3)[13]. Therefore, sticky sets of TGDs are not only FO-rewritable, but also the constructed first-order query can be of polynomial size.

Extensions. Several convincing arguments for the usefulness of sticky sets of TGDs are given in [14]. However, sticky sets of TGDs are not expressive enough for being able to model simple cases such as the TGD $r(X, Y, X) \rightarrow \exists Z\, s(Y, Z)$; clearly, the variable X is marked, and thus the stickiness condition is violated. Note that the above rule falls in the FO-rewritable class of linear TGDs (see Subsection 3.2). A language that captures both linear and sticky Datalog$^{\pm}$, without losing the desirable property of FO-rewritability (and also the PWP), is *sticky-join Datalog*$^{\pm}$ introduced in [14].

A more general class of TGDs, which is called *weakly-sticky* sets of TGDs, and which constitute *weakly-sticky Datalog*$^{\pm}$, is studied in [14]. Roughly, in a weakly-sticky set of TGDs, the variables that occur more than once in the body of a TGD are non-marked or occur at positions where a finite number of symbols can appear during the construction of the chase.

5. ADDITIONAL FEATURES

In this section we discuss how Datalog$^{\pm}$ can be extended with negative constraints and key dependencies.

5.1 Negative Constraints

A *negative constraint* (or simply *constraint*) is a first-order sentence of the form $\forall \mathbf{X}\, \phi(\mathbf{X}) \rightarrow \bot$, where $\phi(\mathbf{X})$ is a conjunction of atoms (with no syntactic restrictions) and $\bot$ denotes the truth constant *false*; the universal quantifier is omitted for brevity. As we shall see in Section 6, constraints are vital when representing ontologies.

Example 2. Suppose that the unary predicates c and c' represent two classes. The fact that these two classes have no common instances can be expressed by the constraint $c(X), c'(X) \rightarrow \bot$. Moreover, if the binary predicate r represents a relationship, the fact that no instance of the class c participates to the relationship r (as the first component) can be stated by the constraint $c(X), r(X, Y) \rightarrow \bot$. ∎

Checking whether a set of constraints is satisfied by a database given a set of TGDs is tantamount to query answering [12]. In particular, given a set of TGDs Σ_T, a set of constraints $\Sigma_\bot$, and a database D, for each constraint $\nu : \phi(\mathbf{X}) \rightarrow \bot$ we evaluate the BCQ $p \leftarrow \phi(\mathbf{X})$ over $D \cup \Sigma_T$. If at least one of such queries answers positively, then $D \cup \Sigma_T \cup \Sigma_\bot \models \bot$ (i.e., the theory is inconsistent), and thus $D \cup \Sigma_T \cup \Sigma_\bot$ entails every BCQ; otherwise, given a BCQ q, we have that $D \cup \Sigma_T \cup \Sigma_\bot \models q$ iff $D \cup \Sigma_T \models q$ (or equivalently *chase*$(D, \Sigma) \models q$), i.e., we can answer q by ignoring the constraints.

THEOREM 8. *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, a set Σ_T of TGDs over $\mathcal{R}$, and a set $\Sigma_\bot$ of constraints over $\mathcal{R}$. Then, $D \cup \Sigma_T \cup \Sigma_\bot \models q$ iff (i) chase$(D, \Sigma_T) \models q$ or (ii) chase$(D, \Sigma_T) \models q_\nu$, for some constraint $\nu \in \Sigma_\bot$.*

As an immediate consequence, constraints do not increase the complexity of BCQ answering under TGDs alone [12].

5.2 Key Dependencies

The addition of key dependencies (KDs) [1] is more problematic than that of constraints, since the former easily leads to undecidability of query answering (see, e.g., [16]). For this reason, the restricted class of *non-conflicting KDs*, which has a controlled interaction with TGDs, and thus decidability of query answering is guaranteed, was proposed in [12]. Nonetheless, as we shall see in Section 6, this class is expressive enough for modeling ontologies.

A *key dependency (KD)* κ is an assertion of the form *key*$(r) = \mathbf{A}$, where r is a predicate symbol and $\mathbf{A}$ is a set of attributes of r. It is equivalent to the set of EGDs $\{r(\mathbf{X}, Y_1, \ldots, Y_m), r(\mathbf{X}, Y'_1, \ldots, Y'_m) \rightarrow Y_i = Y'_i\}_{i \in [m]}$, where the variables $\mathbf{X} = X_1, \ldots, X_n$ appear exactly at the attributes of $\mathbf{A}$ (w.l.o.g., the first n attributes of r). Such a KD κ is *applicable* to a set of atoms B iff there exist two (distinct) tuples $\mathbf{t_1}, \mathbf{t_2} \in \{\mathbf{t} \mid r(\mathbf{t}) \in B\}$ such that $\mathbf{t_1}[\mathbf{A}] = \mathbf{t_2}[\mathbf{A}]$, where $\mathbf{t}[\mathbf{A}]$ is the projection of tuple $\mathbf{t}$ over $\mathbf{A}$. If there exists an attribute $i \notin \mathbf{A}$ of r such that $\mathbf{t_1}[i]$ and $\mathbf{t_2}[i]$ are two (distinct) constants of Γ, then there is a *hard violation* of κ, and the chase *fails*. Otherwise, the result of the application of κ to B is the set of tuples obtained by either replacing each occurrence of $\mathbf{t_1}[i]$ in B with $\mathbf{t_2}[i]$, if $\mathbf{t_1}[i]$ follows lexicographically $\mathbf{t_2}[i]$, or vice-versa otherwise.

The chase of a database D, in the presence of two sets Σ_T and Σ_K of TGDs and KDs, respectively, is computed by iteratively applying: *(i)* a single TGD once, and *(ii)* the KDs as long as they are applicable.

We continue by introducing the semantic notion of separability [16, 12], which formulates a controlled interaction of TGDs and KDs, so that the KDs do not increase the complexity of query answering.

Definition 6. Let $\mathcal{R}$ be a relational schema. Consider a set $\Sigma = \Sigma_T \cup \Sigma_K$ over $\mathcal{R}$, where Σ_T and Σ_K are sets of TGDs and KDs, respectively. Then, Σ is *separable* iff for every database D for $\mathcal{R}$ the following conditions are satisfied: *(i)* if $chase(D, \Sigma)$ fails, then there is a hard violation of some KD $\kappa \in \Sigma_K$, when κ is applied directly on D, and *(ii)* if there is no chase failure, then for every BCQ q over $\mathcal{R}$, $chase(D, \Sigma) \models q$ iff $chase(D, \Sigma_T) \models q$.

In the presence of separable sets of TGDs and KDs, the complexity of query answering is the same as in the presence of the TGDs alone. This was established in [12] (generalizing [16]) by showing that in such a case we can first perform a preliminary check to see whether the chase fails, which has the same complexity as BCQ answering, and if the chase does not fail, then proceed with query answering under the TGDs alone.

We now give the formal definition of the class of non-conflicting KDs, as defined in [12], which is actually a sufficient syntactic condition for separability. Let us say that the class of non-conflicting KDs generalizes the class of *non-key-conflicting inclusion dependencies* introduced in [16]. This condition is crucial for using TGDs to capture ontology languages, as we shall see in Section 6. Notice that, in the following definition, TGDs are assumed w.l.o.g. to have single-atom heads.

Definition 7. Let $\mathcal{R}$ be a relational schema. Consider a TGD $\sigma : \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, r(\mathbf{X}, \mathbf{Z})$ over $\mathcal{R}$, and a set Σ_K of KDs over $\mathcal{R}$. We say that Σ_K is *non-conflicting (NC)* relative to σ if for each $\kappa \in \Sigma_K$ of the form $key(r) = \mathbf{A}$, the following conditions are satisfied: *(i)* the set of the attributes of r in $head(\sigma)$ where a universally quantified variable occurs is not a strict superset of $\mathbf{A}$, and *(ii)* each existentially quantified variable in σ occurs just once. We say that Σ_K is NC relative to a set Σ_T of TGDs if Σ_K is NC relative to each TGD $\sigma \in \Sigma_T$.

Example 3. Consider the TGD σ of the form $p(X, Y) \rightarrow \exists Z\, r(X, Y, Z)$, and the KDs $\kappa_1 : key(r) = \{1, 2\}$ and $\kappa_2 : key(r) = \{1\}$. Clearly, the set of the $\forall$-attributes of r in $head(\sigma)$ is $\mathbf{U} = \{1, 2\}$. Observe that $\{\kappa_1\}$ is NC relative to σ; roughly, every atom generated during the chase by applying σ will have a "fresh" null of Γ_N in some key attribute of κ_1, thus never firing this KD. On the contrary, $\{\kappa_2\}$ is not NC relative to σ since $\mathbf{U} \supset \{1\}$. ■

6. ONTOLOGY QUERYING

In this section we briefly describe how the main languages of the well-known DL-Lite family of DLs [18, 35], namely, DL-Lite$_\mathcal{F}$, DL-Lite$_\mathcal{R}$ and DL-Lite$_\mathcal{A}$ can all of them be reduced to linear (resp., sticky) Datalog$^\pm$ with (negative) constraints and non-conflicting KDs, called *linear* (resp., *sticky*) *Datalog*$^\pm[\bot, =]$, and that the former are strictly less expressive than the latter. Let us recall that DL-Lite$_\mathcal{R}$ is able to fully capture the (DL fragment of) RDF Schema [9], the vocabulary description language for RDF; see [22] for a translation.

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. For instance, a DL knowledge base (or ontology) in DL-Lite$_\mathcal{F}$ encodes subset relationships between concepts and between roles, the membership of individuals to concepts and of pairs of individuals to roles, and functional dependencies on roles. The following example, taken from [12], illustrates some DL axioms in DL-Lite$_\mathcal{F}$ and their translation into Datalog$^\pm$ rules.

Example 4. The following are some concept inclusion axioms, which informally express that *(i)* conference and journal papers are articles, *(ii)* conference papers are not journal papers, *(iii)* every scientist has a publication, *(iv)* *isAuthorOf* relates scientists and articles:

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

They are translated into the following TGDs and constraints (we identify atomic concepts and roles with their predicates):

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

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

$$\begin{array}{rcl} \mathit{isAuthorOf}^- & \sqsubseteq & \mathit{hasAuthor}, \\ \mathit{hasAuthor}^- & \sqsubseteq & \mathit{isAuthorOf}, \\ & (\mathsf{funct}\ \mathit{hasFirstAuthor}). & \end{array}$$

They are translated into the following TGDs and KDs:

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

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

$$Scientist(i_1),\ isAuthorOf(i_1, i_2),\ Article(i_2).$$

They are translated to identical database atoms (where we also identify individuals with their constants). ■

Formally speaking, every knowledge base $\mathcal{K}$ in DL-Lite$_X$, where $X \in \{\mathcal{F}, \mathcal{R}, \mathcal{A}\}$, is translated into a database $D_\mathcal{K}$, a set of TGDs Σ_T, a set of KDs Σ_K, and a set of constraints $\Sigma_\perp$. Notice that Σ_T is a set of linear TGDs and also a sticky set of TGDs. Moreover, Σ_K is non-conflicting relative to Σ_T. The next result, established in [12, 14], shows that query answering under DL-Lite$_X$, where $X \in \{\mathcal{F}, \mathcal{R}, \mathcal{A}\}$, knowledge bases can be reduced to query answering under linear and sticky Datalog$^\pm[\perp, =]$.

THEOREM 9. *Let $\mathcal{K}$ be a knowledge base in DL-Lite$_X$, where $X \in \{\mathcal{F}, \mathcal{R}, \mathcal{A}\}$, and q be a BCQ for $\mathcal{K}$. If $D_\mathcal{K} \models \Sigma_K$, then q holds in $\mathcal{K}$ iff either (i) chase$(D_\mathcal{K}, \Sigma_T) \models q$, or (ii) chase$(D_\mathcal{K}, \Sigma_T) \models q_\nu$, for some constraint $\nu \in \Sigma_\perp$.*

The next result follows immediately from the fact that the simple linear and sticky TGD $r(X) \rightarrow s(X, X)$ is not expressible in DL-Lite$_X$, where $X \in \{\mathcal{F}, \mathcal{R}, \mathcal{A}\}$ [12].

THEOREM 10. *Both linear and sticky Datalog$^\pm[\perp, =]$ are strictly more expressive than DL-Lite$_X$, where $X \in \{\mathcal{F}, \mathcal{R}, \mathcal{A}\}$.*

By observing that concept products [38], that is, rules of the form $p(X), q(Y) \rightarrow r(X, Y)$ which express the cartesian product of two concepts (unary relations) p and q, are very special cases of sticky sets of TGDs, it is possible to show that the above DL-Lite languages can be extended with concept product, without increasing the complexity of query answering.

7. DISCUSSION AND FUTURE WORK

In this paper we have surveyed some of the key languages in the Datalog$^\pm$ famliy. From a database point of view, these languages are in fact syntactically defined sets of TGDs (possibly enriched with non-monotonic negation and other features) that are especially suited for ontological query answering. We believe that the Datalog$^\pm$ family is a useful logical toolbox for tackling ontology reasoning tasks. Datalog$^\pm$ languages have a simple syntax and are easy to understand; they are decidable and enjoy good complexity properties. They can easily express very popular DL languages, and in addition they can be extended by non-monotonic stratified negation, a desirable feature which is not expressible in current DL languages.

Datalog$^\pm$ is still the subject of active research, and there are many challenging research problems to be tackled, some of wich we list below.

- We would like to find more general decidable fragments; the first goal is to combine the two tractability paradigms guardedness and stickiness in a natural way.
- *Transitive closure*, introduced in some expressive DL languages in a limited form, is easily expressible in Datalog, but only through non-guarded rules, whose addition to decidable sets of rules may easily lead to undecidability. We would like to study whethere there are limited forms of transitive closure that can be safely added to various versions of Datalog$^\pm$.
- A class of TGDs is *finite controllable* if it guarantees that query answering under arbitrary (finite or infinite) models coincide with query answering under finite models only. Finite controllability was shown recently for the guarded fragment of first-order logic [5], and thus holds for guarded TGDs (and it easily extends to weakly-guarded TGDs). We plan to study this property in the context of sticky sets of TGDs.
- For non-finitely-controllable Datalog$^\pm$ languages, we would like to study the complexity of query answering under finite models. Pioneering work on finite model reasoning was done in [17, 30, 36, 37].
- We plan to study optimizations of rewritings obtained for FO-rewritable Datalog$^\pm$ languages. As for now, such rewritings are in general quite large and therefore not very efficient in practice.

Acknowledgments. This work was supported by the European Research Council under the European Community's 7-th Framework Programme (FP7/2007-2013)/ERC grant no. 246858 – DIADEM.

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] P. Alvaro, W. Marczak, N. Conway, J. M. Hellerstein, D. Maier, and R. C. Sears. Towards scalable architectures for clickstream data warehousing. Technical report, EECS Department, University of California, Berkeley, 2009.

[3] H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *J. Philosophical Logic*, 27:217–274, 1998.

[4] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011.

[5] V. Bárány, G. Gottlob, and M. Otto. Querying the guarded fragment. In *Proc. of LICS*, pages 1–10, 2010.

[6] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with Lixto. In *Proc. of VLDB*, pages 119–128, 2001.

[7] R. Baumgartner, W. Gatterbauer, and G. Gottlob. Monadic Datalog and the expressive

power of web information extraction languages. In L. Liu and M. T. Özsu, editors, *Encyclopedia of Database Systems*, pages 3465–3471. Springer-Verlag New York, Inc., 2009.

[8] C. Beeri and M. Y. Vardi. The implication problem for data dependencies. In *Proc. of ICALP*, pages 73–85, 1981.

[9] D. Brickley and R. V. Guha. RDF vocabulary description language 1.0: RDF Schema. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/, 2004. W3C Recommendation.

[10] L. Cabibbo. The expressive power of stratified logic programs with value invention. *Inf. Comput.*, 147(1):22–56, 1998.

[11] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *Proc. of KR*, pages 70–80, 2008.

[12] A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *Proc. of PODS*, pages 77–86, 2009.

[13] A. Calì, G. Gottlob, and A. Pieris. Towards more expressive ontology langugaes: The query answering problem. Unpublished Manuscript.

[14] A. Calì, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. *PVLDB*, 3(1):554–565, 2010.

[15] A. Calì, G. Gottlob, and A. Pieris. Query rewriting under non-guarded rules. In *Proc. AMW*, 2010.

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

[17] D. Calvanese. Finite model reasoning in description logics. In *Proc. of KR*, pages 292–303, 1996.

[18] 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.

[19] M. A. Casanova, R. Fagin, and C. H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. *J. Comput. Syst. Sci.*, 28:29–59, 1984.

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

[21] A. K. Chandra and M. Y. Vardi. The implication problem for functional and inclusion dependencies. *SIAM Journal of Computing*, 14:671–677, 1985.

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

[23] A. Deutsch, A. Nash, and J. B. Remmel. The chase revisisted. In *Proc. of PODS*, pages 149–158, 2008.

[24] 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.

[25] G. Gottlob and C. Koch. Monadic Datalog and the expressive power of web information extraction languages. *J. ACM*, 51(1):71–113, 2004.

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

[27] G. Gottlob and T. Schwentick. Rewriting ontological queries into small non-recursive Datalog programs. In *Proc. of DL*, 2011.

[28] E. Hajiyev, M. Verbaere, and O. de Moor. *codeQuest*: scalable source code queries with Datalog. In *Proc. of ECOOP*, pages 2–27, 2006.

[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] C. Lutz, U. Sattler, and L. Tendera. The complexity of finite model reasoning in description logics. *Inf. Comput.*, 199(1-2):132–171, 2005.

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

[32] D. Mailharrow. A classification and constraint-based framework for configuration. *Artif. Intell. for Engineering Design, Analysis and Manufacturing*, 12(4):383–397, 1998.

[33] B. Marnette. Generalized schema-mappings: from termination to tractability. In *Proc. of PODS*, pages 13–22, 2009.

[34] P. F. Patel-Schneider and I. Horrocks. A comparison of two modelling paradigms in the semantic web. *J. Web Semantics*, 5(4):240–250, 2007.

[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] R. Rosati. Finite model reasoning in DL-lite. In *Semantic Web Conf.*, pages 215–229, 2008.

[37] R. Rosati. On the finite controllability of conjunctive query answering in databases under open-world assumption. *J. Comput. Syst. Sci.*, 77(3):572–594, 2011.

[38] S. Rudolph, M. Krötzsch, and P. Hitzler. All elephants are bigger than all mice. In *Description Logics*, 2008.

[39] M. Y. Vardi. The complexity of relational query languages. In *Proc. of STOC*, pages 137–146, 1982.

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

[41] M. Y. Vardi. On the complexity of bounded-variable queries. In *Proc. of PODS*, pages 266–276, 1995.