Datalog+/-: A Family of Logical Knowledge Representation and Query Languages for New Applications
Keynote Lecture

Andrea Calì[3,2] Georg Gottlob[1,2,4] Thomas Lukasiewicz[1,5] Bruno Marnette[1] Andreas Pieris[1]

[1]*Computing Laboratory, University of Oxford, UK*
[2]*Oxford-Man Institute of Quantitative Finance, University of Oxford, UK*
[3]*Department of Information Systems and Computing, Brunel University, UK*
e-mail: firstname.lastname@comlab.ox.ac.uk

***Abstract*—This paper summarizes results on a recently introduced family of Datalog-based languages, called Datalog+/-, which is a new framework for tractable ontology querying, and for a variety of other applications. Datalog+/- extends plain Datalog by features such as existentially quantified rule heads and, at the same time, restricts the rule syntax so as to achieve decidability and tractability. In particular, we discuss three paradigms ensuring decidability: chase termination, guardedness, and stickiness.**

***Keywords*-Knowledge Representation and Reasoning; Query Answering; Ontologies.**

I. Introduction

This paper is a survey of recently introduced variants of Datalog. On the one hand, Datalog is extended by allowing features such as existential quantifiers, the equality predicate, and the truth constant false (denoted $\bot$) to appear in rule heads. On the other hand, the resulting language is syntactically restricted, so to achieve decidability and in some relevant cases even tractability. The family of all such (existing and future) variants was dubbed *Datalog*$^{\pm}$ (also written Datalog+/- whenever appropriate). Before delving into this new language family, let us very briefly review the well-known Datalog language.

Datalog (see, e.g., [1], [2]) has been used as a paradigmatic database programming and query language for over three decades. While Datalog is rarely used directly as a query language in corporate application contexts, the language has influenced the development of popular query languages such as SQL, whose newer versions allow one to express recursive queries. Moreover, Datalog has been 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 [3], [4], [5], source code querying and program analysis [6], and modeling distributed systems [7].

A basic Datalog program consists of a set of universally quantified function-free Horn clauses. When writing a Datalog program, as usual in logic programming, we consider sets of rules to be conjunctions, use the comma for conjoining atoms, and assume all variables of a rule are universally quantified, while omitting the universal quantifiers. 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 may appear in rule bodies only.

Example 1: As an example, consider a program that 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 following recursive Datalog program computes the set r of all vertices in the graph reachable via a directed path of nonnegative length from special vertices:

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

Example 2: The following recursive program computes the transitive closure c of the binary relation e:

$$\begin{aligned} e(X,Y) &\rightarrow c(X,Y), \\ e(X,Y), c(Y,Z) &\rightarrow c(X,Z). \end{aligned}$$

A *Boolean conjunctive query (BCQ)* is an existentially quantified conjunction of atoms. For example, the BCQ q of whether a directed triangle is reachable in the graph e of Example 1 from the set s of special vertices can be written as

$$\exists X \exists Y \exists Z \, r(X), r(Y), r(Z), e(X,Y), e(Y,Z), e(Z,X).$$

Alternatively, a BCQ can be represented as a Datalog rule with a head predicate of arity 0, i.e., a Boolean head predicate, for example,

$$r(X), r(Y), r(Z), e(X,Y), e(Y,Z), e(Z,X) \rightarrow \mathit{triangle}.$$

A *conjunctive query (CQ)* is defined similarly to a BCQ but has free variables defining the output tuples (see Section II).

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. It is well-known that

[4] Keynote speaker.
[5] Alternative affiliation: Inst. f. Informationssysteme, TU Wien, Austria.

$D \cup \Sigma$ has a unique least Herbrand model $LHM(D \cup \Sigma)$, which consists of all ground atoms $\underline{a}$ such that $D \cup \Sigma \models \underline{a}$. This model can be computed by a least fixpoint iteration starting from the EDB D and adding at each iteration step all new facts generated by a single rule application. We say that a BCQ q evaluates to true over D and Σ iff $D \cup \Sigma \models q$. This is equivalent to the existence of a homomorphism from (the atoms of) q to $LHM(D \cup \Sigma)$.

Note that the unique least Herbrand model of a Datalog program and a database D is always finite and all values appearing in it are from the universe of the EDB given as input, which is usually defined to be the *active domain* of the EDB, i.e., all values that appear as arguments of EDB facts or that are explicitly mentioned in the Datalog program. For a number of applications, however, it would be desirable that a Datalog extension could be able to express the existence of certain values that are not necessarily from the EDB universe. This can be achieved by allowing existentially quantified variables in rule heads. Let us give a few brief examples of such applications and refer to Section IX and to the references therein for a more detailed treatment.

Data Exchange: When data needs to be transposed or copied from one relational database to another one, the problem of heterogeneous schemas often arises. Imagine, for example, company ACME stores data about their employees in a relation *emp-ACME* with schema $(\mathsf{Emp\#}, \mathsf{Name}, \mathsf{Address}, \mathsf{Salary})$, while the FOO corporation does not store employees' addresses, but only phone numbers, keeping their employee data in a relation *emp-FOO* having schema $(\mathsf{Emp\#}, \mathsf{Name}, \mathsf{Phone}, \mathsf{Salary})$. Imagine ACME is acquired by FOO and the ACME employee data ought to be transferred into the FOO database, although the phone numbers of the ACME employees are not (currently) known. This could be achieved by a rule of the form:

$$\textit{emp-ACME}(E, N, A, S) \rightarrow \exists P\, \textit{emp-FOO}(E, N, P, S),$$

where phone numbers are simply existentially quantified. In practice, each phone number is stored by a different (labeled) *null value*, representing a globally existentially quantified variable (i.e., a kind of Skolem constant). There are currently advanced data management systems such as Clio [8] that effectively manage such data-exchange mappings, handle such existential nulls, and allow one to query relations with nulls. In database theory, a rule of the above form is actually called a *tuple-generating dependency (TGD)*. In addition to TGDs, *equality-generating dependencies (EGDs)* are often used. They cover the well-known key constraints and functional dependencies that have been studied for a long time [2]. For example, we may impose that every ACME employee has only one phone number stored. This may be expressed as a Datalog rule with an equality in the head:

$$\textit{emp-FOO}(E, N, P, S), \textit{emp-FOO}(E, N', P', S') \rightarrow P = P'.$$

The data exchange literature insists on *finite target relations* because it is assumed that these relations are actually stored. It is thus important in this context to restrict our syntax so make sure that only a *finite number of different null values* be added.

Ontology Querying: Description logics (DLs) [9] are used to formalize so-called ontological knowledge about relationships between objects, entities, and classes in a certain application domain. For example, we could express that every person has exactly one father who, moreover, is himself a person, by the following DL clauses, where *person* is a set of objects whose initial value is specified in the form of an EDB relation, called *concept*, and where *father* is a binary relation, a so-called *role* in DL terminology: *(i)* $person \sqsubseteq \exists father$, *(ii)* $\exists father^- \sqsubseteq person$, *(iii)* $(\mathsf{funct}\ father)$. In an appropriate version of Datalog$^\pm$, the same can be expressed as:

$$\begin{aligned} person(X) &\rightarrow \exists Y\, father(X, Y), \\ father(X, Y) &\rightarrow person(Y), \\ father(X, Y), father(X, Y') &\rightarrow Y = Y'. \end{aligned}$$

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.

DLs usually rely on classical first-order (FO) semantics, and so arbitrary models (finite or infinite) are considered. In the above example, models with infinite chains of ancestors are perfectly legal. Rather than "materializing" such models, i.e., computing and storing them, we are interested in reasoning and query answering. For example, clearly, whenever the initial value of *person* is nonempty, then the BCQ

$$\exists X \exists Y \exists Z\, father(X, Y), father(Y, Z)$$

will evaluate to *true*, while the query

$$\exists X \exists Y\, father(X, Y), father(Y, X)$$

will evaluate to *false*, because it is false in some models.

Web Data Extraction: Another application of rules with existentially quantified heads is automatic web data extraction. Here, Datalog rules can identify objects on a web page and group them together to a compound object. The latter needs a new identifier, which can be achieved through an existential quantifier. An example is given in Section IX.

In summary, as we have briefly tried to sketch, all these applications could possibly profit from appropriate forms of Datalog extended by the possibility of using rules with existential quantifiers in their heads (TGDs), and by several additional features (such as, for example, EGDs).

Unfortunately, already for sets Σ of TGDs alone, most basic reasoning and query answering problems are undecidable. In particular, checking whether $D \cup \Sigma \models q$ for a ground fact q is undecidable [10]. Worse than that, undecidability

holds even in case both Σ and q are *fixed*, and only D is given as input, because, one can design a set Σ that simulates a universal Turing machine [11].

It is thus important to single out large classes of formalisms for rule sets Σ that

- are based on Datalog, and thus enable a modular rule-based style of knowledge representation;
- are syntactical fragments of first-order logic so that answering a BCQ q under Σ for an input database D is equivalent to the classical entailment check $D \cup \Sigma \models q$;
- are expressive enough for being useful in real applications in the above mentioned areas;
- have decidable query answering;
- have good query answering complexity properties in case Σ and q are fixed. This type of complexity is called *data complexity*, and is an important measure, because we can realistically assume that the EDB D is the only really large object in the input.

This paper reports on some recent languages that fulfill these criteria. We dubbed the family of such languages Datalog$^{\pm}$, because, as explained, they add features to Datalog, and on the other hand make some syntactical restrictions. In what follows, we will always assume that D is a database of ground atoms, and Σ a set of rules or clauses in a Datalog$^{\pm}$ language.

One of the main tools used for proving favorable results about a number of Datalog$^{\pm}$ languages is the *chase procedure* [12], [13], of which we discuss two different versions in Section III. 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 by inferring new atoms, creating null values (Skolem constants) whenever an existential quantifier needs to be satisfied, and unifying such nulls with other nulls or with non-null values whenever required by an equality atom in the head of a rule whose body has become satisfied. The nice thing about the chase procedure is that, independently of the order, in which rules are processed, the result $chase(D, \Sigma)$ of the chase is a universal model of $D \cup \Sigma$, i.e., an "initial" model which can be homomorphically embedded into every other model (see, e.g., [14]). As a consequence, for each BCQ q, $D \cup \Sigma \models q$ iff $chase(D, \Sigma) \models q$ iff there is a homomorphism from (the atoms of) q into $chase(D, \Sigma)$. The chase procedure may terminate or not. Even in case the chase does not terminate and has an infinite result, it is a useful tool for studying query answering, because in relevant cases, it is sufficient to execute the chase up to a certain finite level (or derivation depth) for being able to answer a BCQ.

As already explained, for data exchange applications, one is usually interesting in *finite* models, and therefore in languages and settings that guarantee *chase termination*. Section III discusses chase termination and reports on useful Datalog$^{\pm}$ classes for which the chase is guaranteed to terminate. The classes and techniques discussed in Section III were mainly developed in the area of data exchange, but fit the Datalog$^{\pm}$ framework very well.

Section IV, instead, reports on classes of Datalog$^{\pm}$ formalisms that are related to the Guarded Fragment of first-order logic (GF) [15]. Guardedness [15] is a well-known restriction of first-order logic that ensures decidability. We start Section IV with a recall of very recent results [16] for the setting where Σ belongs to GF. To obtain better complexity results, we then study the class of *guarded TGDs*, where each rule body is required to have an atom that covers all body variables of the rule. For instance, the Datalog program in Example 1 is guarded, while the one in Example 2 is not. Guarded TGDs ensure polynomial-time data complexity of query answering, even though the chase may be infinite. We then consider the even more restricted class of *linear TGDs*, for which query answering is *first-order rewritable* which means that Σ and q can be transformed into a first-order query q_Σ such that $D \models q_\Sigma$ iff $D \cup \Sigma \models q$. This property, introduced in [17] in the context of DLs, is essential if D is a very large database. It means that query answering can be deferred to a standard query language such as (basic, non-recursive) SQL. We also show how guarded TGDs can be enriched by *stratified negation*, a simple nonmonotonic form of negation often used in the context of Datalog.

Section V discusses *weakly guarded* (sets of) TGDs, a useful generalization of the class of guarded TGDs, where the guardedness condition for rule bodies is somewhat relaxed, so that only those variables need to be guarded that occur in positions that may eventually contain nulls.

Stickiness, a completely different paradigm for decidable and tractable query answering is discussed in Section VI. Let us give a very informal explanation. First, stickiness requires that every TGD σ that has a double occurrence of a variable X in the rule body, has at least one occurrence of X in the rule head. Further, whenever such a TGD fires and produces a new atom $\underline{a}$ that has a value v in place of the variable X, then the value v is never lost by any derivation sequence that uses chase steps (i.e., forward chaining) for producing new atoms, and that involves $\underline{a}$. In other words, every value that arises in a new atom $\underline{a}$ through a *join* in a rule body must be present in all further atoms derived from $\underline{a}$. We will introduce stickiness by a syntactic criterion that is easily testable and equivalent to the above characterization.

In Section VII, 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 come for free, and can be used without any increase of complexity. The reason is that checking whether a rule ρ: $body \rightarrow \bot$ is satisfied by a database D given a Datalog$^{\pm}$ program Σ is tantamount to showing that $D \cup \Sigma \not\models body$, i.e., to the evaluation of a BCQ. We then proceed by drawing our attention to equality-generating dependencies (EGDs) that we would like to use together with TGDs. Unfortunately, as well-known in

database theory, query answering becomes undecidable even when putting together some extremely week forms of TGDs and EGDs such as inclusion dependencies and functional dependencies [18]. In this paper, whenever chase termination is not guaranteed, we therefore mainly concentrate 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.

In Section VIII, we report on interesting results by Baget et al. [19], [20] about high-level criteria for decidability and relate them to the specific logics dealt-with in this paper.

Section IX briefly describes various applications ranging from data exchange to reasoning with extended Entity-Relationship schemata. Importantly, we show how highly relevant DLs such as *DL-Lite* and F-Logic Lite can be modeled in the Datalog$^{\pm}$ framework.

We conclude with a brief outlook on further research.

II. Preliminaries

We now briefly recall some basics on databases, queries, and (tuple- and equality-generating) dependencies.

A. Databases and Queries

We assume (i) an infinite universe of *data constants* Δ (which constitute the "normal" domain of a database), (ii) an infinite set of *(labeled) nulls* Δ_N (used as "fresh" Skolem terms, which are placeholders for unknown values, and can thus be seen as variables), and (iii) an infinite set of variables Δ_V (used in dependencies and queries). Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. We assume a lexicographic order on $\Delta \cup \Delta_N$, with every symbol in Δ_N following all symbols in Δ. We denote by $\mathbf{X}$ sequences of variables $X_1, \ldots, X_k$ with $k \geq 0$.

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*) $\underline{a}$ 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 $dom(\underline{a})$, $pred(\underline{a})$, and $vars(\underline{a})$ the sets of all arguments, the predicate symbol, and the set of all variables of an atom $\underline{a}$, 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 Δ. A *conjunctive query (CQ)* over $\mathcal{R}$ has the form $q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$, where $\Phi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms 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 $\mu\colon \Delta \cup \Delta_N \cup \Delta_V \to \Delta \cup \Delta_N \cup \Delta_V$ such that (i) $c \in \Delta$ implies $\mu(c) = c$, (ii) $c \in \Delta_N$ implies $\mu(c) \in \Delta \cup \Delta_N$, and (iii) μ is naturally extended to atoms, sets of atoms, and conjunctions of atoms. The set of all *answers* to a CQ $q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$ over a database D, denoted $q(D)$, is the set of all tuples $\mathbf{t}$ over Δ for which there exists a homomorphism $\mu\colon \mathbf{X} \cup \mathbf{Y} \to \Delta \cup \Delta_N$ such that $\mu(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$ and $\mu(\mathbf{X}) = \mathbf{t}$. The *answer* to a BCQ q over D is *Yes*, denoted $D \models q$, iff $q(D) \neq \varnothing$.

B. Dependencies

Given a relational schema $\mathcal{R}$, a *tuple-generating dependency* (or *TGD*) σ is a first-order formula of the form $\forall \mathbf{X} \forall \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$, where $\Phi(\mathbf{X}, \mathbf{Y})$ and $\Psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$, called the *body* and the *head* of σ, respectively. Such σ is satisfied in a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h such that $h(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$, there exists an extension h' of h such that $h'(\Psi(\mathbf{X}, \mathbf{Y})) \subseteq D$. A TGD of the form $r_1(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, r_2(\mathbf{X}, \mathbf{Z})$, where no variable appears more than once in the body nor in the head, is called an *inclusion dependency (ID)* (see, e.g., [13]).

The notion of *query answering* under TGDs is defined as follows. For a set of TGDs Σ on $\mathcal{R}$, and a database D for $\mathcal{R}$, the set of *models* (or *solutions*) of D given Σ, denoted $sol(D, \Sigma)$, is the set of all databases B such that $B \models D \cup \Sigma$. The set of *answers* to a CQ q on D given Σ, denoted $ans(q, D, \Sigma)$, is the set of all tuples $\mathbf{t}$ such that $\mathbf{t} \in q(B)$ for all $B \in sol(D, \Sigma)$. The *answer* to a BCQ q over D given Σ is *Yes*, denoted $D \cup \Sigma \models q$, iff $ans(q, D, \Sigma) \neq \varnothing$. The *combined complexity* of query answering is the complexity of determining whether a given tuple is among the answers to a query, given a database D, a set of TGDs Σ, and a query q as input. The *data complexity* is the complexity of the same problem, where Σ and q are considered fixed, and only D is considered as input. The latter complexity is the most important in the context of data-oriented settings, where the data size is usually much larger than the size of the constraints and of the query.

The two problems of CQ and BCQ evaluation under TGDs are LOGSPACE-equivalent [21], [13], [22], [23]. 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 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 Sections IV, V, and VI (*guarded*, *linear*, *weakly-guarded*, and *sticky* TGDs). Therefore, all results for TGDs with only singleton atoms in the head carry over to TGDs with multiple

head-atoms. Thus, in Sections IV and V, w.l.o.g., every TGD has a singleton atom in its head.

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

III. Chase and Termination

After presenting more formally the notion of a *universal* solution of a database given a set of TGDs, and the notion of *termination* of the *chase*, which computes such a solution, this section presents different ways of ensuring termination (of the *restricted chase* and the *oblivious chase*).

Universality and Termination: Intuitively, a *universal solution* U for a database D given a set of TGDs Σ is a solution containing sound and complete information. Given a conjunctive query q, we can then compute $ans(q, D, \Sigma)$ by simply evaluating q on the universal solution U, and discarding the answer tuples containing at least one value in Δ_N. A natural way of ensuring tractability is to make sure that a finite universal solution can be computed efficiently, with an algorithm typically called *a chase procedure* [13], [22] (and often referred to as *the chase*).

Definition 1 (Universality): A solution $U \in sol(D, \Sigma)$ is *universal*, and we let $U \in usol(D, \Sigma)$, iff for all solutions $K \in sol(D, \Sigma)$, there is a homomorphism from U to K.

Proposition 2 ([22], [23]): For all conjunctive queries q and universal solutions $U \in usol(D, \Sigma)$, the set $ans(q, D, \Sigma)$ coincides with the set of ground answers in $q(U)$.

Definition 3 (Termination): A set of TGDs Σ *ensures termination* iff there exists an algorithm that, given a finite database D, always returns a finite universal solution $U \in usol(D, \Sigma)$. We say that Σ ensures *polynomial* termination if this algorithm runs in polynomial time (data complexity).

A corollary of Proposition 2 is the following:

Proposition 4: If q is a CQ and Σ ensures polynomial termination, then the following problem is in PTIME: given a database D, compute $ans(q, D, \Sigma)$.

Restricted Chase: As mentioned above, a *chase procedure* is an algorithm to compute universal solutions. While many different chase procedures can be found in the literature (see, e.g., [12], [13], [23], [24]), one of the most widely adopted is the *restricted chase*. Given a set of TGDs Σ, the restricted chase consists intuitively in applying repeatedly the violated TGDs until a fixpoint is reached. More precisely, a TGD $\sigma = \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$ is *violated* for a tuple $\mathbf{t} \in dom(D)^{|\mathbf{X}|}$ iff $D \models \exists \mathbf{Z}\, \Phi(\mathbf{t}, \mathbf{Y})$ while $D \not\models \exists \mathbf{Z}\, \Psi(\mathbf{t}, \mathbf{Z})$. Then, *applying* σ to D (for the tuple $\mathbf{t}$) amounts to replacing D by $D' = D \cup \Psi(\mathbf{t}, \mathbf{u})$ for some tuple of fresh nulls $\mathbf{u} \in {\Delta_N}^{|\mathbf{Z}|}$ so that $D' \models \exists \mathbf{Z}\, \Psi(\mathbf{t}, \mathbf{Z})$.

Acyclicity: Several syntactic criteria of acyclicity have been identified that guarantee the termination of the restricted chase in polynomial time: a first criterion of *stratified witness* (SW) in [25]; a criterion of *weak acyclicity* (WA) in [22]; and, more recently, a criterion of *super-weak acyclicity* (SWA) in [24]. Each of these criteria can be decided in PTIME and consists intuitively in making sure that there is no cycle in the process of migration and creation of null values. The SWA criterion also achieves more generality by making use of efficient techniques (such as *unification*) for a more precise analysis. In fact, SW $\subset$ WA $\subset$ SWA. For instance, the following set of TGDs Σ_{swa} is super-weakly acyclic (but not weakly acyclic):

$$a(X) \rightarrow \exists Y\ b(X, Y), b(Y, X), c(Y),$$
$$b(X, X), c(Y) \rightarrow a(X), c(Y).$$

Theorem 5 ([22], [24]): For every (super-)weakly acyclic set of TGDs Σ, the restricted chase terminates in polynomial time (and Proposition 4 applies).

The criterion of weak acyclicity has been used in several papers as a building block for the design of larger tractable classes: in particular, a class based on *stratification* [23] and a class based on *inductive restriction* [26]. These criteria are incomparable with SWA. In particular, they do not capture Σ_{swa} above. Deciding whether a given set of TGDs is *stratified* or *inductively restricted* is co-NP-complete (while we can decide SWA in PTIME). Finally, the authors of [26] have recently shown in an online erratum (http://arxiv.org/abs/0906.4228) that these notions actually only ensured termination for *some* chase strategy (and not for *every* strategy, as initially claimed in [23] and [26]). It is however possible to combine the results obtained independently in [26] and [24] to design even larger classes of tractable constraints complying to Definition 3.

Oblivious Chase: While the restricted chase is a very intuitive algorithm, it is nondeterministic and may only behave well for *some* chase strategies. Also, the restricted chase is often less efficient than other chase procedures. Before applying a TGD σ, the restricted chase requires indeed to check whether the head of σ is already satisfied. In fact, it is often sufficient (and more efficient) to simply apply a TGD $\Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$ whenever a new tuple $\mathbf{t}$ is found that satisfies $D \models \exists \mathbf{Y}\, \Phi(\mathbf{t}, \mathbf{Y})$—without testing whether or not $D \models \exists \mathbf{Z}\, \Psi(\mathbf{t}, \mathbf{Z})$. The procedure obtained by removing this test is known as the *oblivious chase*.

It can be observed that the oblivious chase is deterministic (up to bijective renaming of the nulls) and in the following sections, we may simply denote by $chase(D, \Sigma)$ the universal solution computed by the oblivious chase for a database D and a set of TGDs Σ. Note that every universal solution U computed by the restricted chase is *homomorphically equivalent* to $chase(D, \Sigma)$, that is, there

exists a homomorphism from U to $chase(D, \Sigma)$, and one from $chase(D, \Sigma)$ to U [11].

With respect to termination, it has been shown in [24] that both the *restricted* and the *oblivious* chase terminate when Σ is (super-)weakly acyclic. More interestingly, one can observe the following dichotomy:

Theorem 6 ([24]): For every set of TGDs Σ, either

- $chase(D, \Sigma)$ is infinite for some database D; or
- the oblivious chase (for Σ) terminates in polynomial time (and Proposition 4 applies).

Unfortunately, there is no terminating procedure that decides in which of the two cases a given Σ falls [24]. Nonetheless, the following characterization can be used to guarantee termination in practice:

Theorem 7 ([24]): For every set of TGDs Σ, the oblivious chase terminates on all D iff it terminates on a specific critical D_Σ, which can be computed from Σ in EXPTIME.

IV. Guarded and Linear Datalog$^\pm$

As explained in the introduction, we do not want to limit our attention to cases where the chase terminates, but consider for many application cases where the chase produces an infinite universal solution, and where, in general, no finite universal solution exists. Unfortunately, as mentioned, query answering is undecidable in such cases, and we are looking for decidable subclasses. In this section, we describe the guarded fragment of first-order logic and its sub-fragments of guarded and linear Datalog$^\pm$, as well as the extension of the latter two by nonmonotonic negation.

A. Querying the Guarded Fragment

One very important and rather useful and general decidable class is the *guarded fragment* of first-order logic (GF) [15], which we assume the reader to be familiar with. The computational complexity of GF and a generalization of it, called the *clique-guarded fragment* was extensively analyzed in [27], [28]. Grädel [27] proved that satisfiability of GF-sentences is complete for 2EXPTIME, and is EXPTIME-complete for sentences involving relations of bounded arity. In the same paper, Grädel also showed that every satisfiable guarded first-order sentence has a finite model, i.e., that GF has the *finite model property (FMP)*.

In [16], the problem of evaluating a Boolean conjunctive query q over a guarded first-order theory Σ was studied. This is equivalent to checking whether $\Sigma \cup \{\neg q\}$ is unsatisfiable. Since q may not be guarded, well-known results about the decidability, complexity, and finite-model property of the guarded fragment do not obviously carry over to conjunctive query answering over guarded theories, and had been left open in general. But the following is shown in [16].

Theorem 8 ([16]): Let Σ be a guarded theory, and q be a union of conjunctive queries. Then:

1) $\Sigma \models q$ iff $\Sigma \models_{fin} q$, that is, iff q is true in each finite model of Σ (note that this result was already implicit in [29], but much better bounds on the size of finite models are given in [16]).
2) Determining whether $\Sigma \models q$ is 2EXPTIME-complete, even if the query q is fixed, and EXPTIME-complete in case of fixed arities.
3) If Σ and q are fixed, then deciding for an input conjunction of ground atoms D (i.e., for a database D) whether $D \cup \Sigma \models q$ is in co-NP, and there are certain purely universal theories Σ and atomic q, for which this problem is co-NP-complete.

Part 1 of Theorem 8 establishes the so-called *finite controllability* of the guarded fragment. This substantially generalizes an earlier result of Rosati [30], where a similar property was shown in case Σ consists of a conjunction of inclusion dependencies. Part 2 essentially settles the *combined complexity* of query answering over guarded theories. Finally, Part 3 deals with the *data complexity* of the same problem. Unfortunately, even for very simple fixed atomic queries taken together with fixed theories Σ without existential quantifiers, the problem is already intractable. For many applications involving large databases D, the latter is not acceptable. On the other hand, the guarded fragment GF does not allow us to express a number of practically relevant constraints such as functional dependencies and keys, see also Section VII. In the rest of this paper, we will thus focus on formalisms for query-answering having tractable data complexity, and later extend these classes by features that make them enough powerful for expressing relevant problems of ontological reasoning and querying. The first classes we consider are actually sub-fragments of GF and combine the Datalog paradigm with the one of guardedness.

B. Guarded Datalog$^\pm$

Query answering under general TGDs is undecidable [10], even when the schema and the TGDs are fixed [11]. We now 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.

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

Example 3: 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 GF [15]. 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.

Theorem 9 ([11]): There exists a fixed set of TGDs Σ_u, where all TGDs but one of Σ_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 $q \in chase(D, \Sigma_u)$, is undecidable.

2) Combined Complexity: The next theorem establishes combined complexity results for conjunctive query evaluation under guarded Datalog$^{\pm}$. The EXPTIME and 2EXPTIME-completeness results hold even if the input database is fixed.

Theorem 10 ([11]): Let Σ 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:

a) If q is an atomic query, then deciding 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 Σ is fixed. This problem is EXPTIME-complete if w is bounded; and 2EXPTIME-complete in general, even when $|\mathcal{R}|$ is bounded.

b) If q is a general conjunctive query, deciding 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 Σ. Checking whether $D \cup \Sigma \models q$ is EXPTIME-complete if w is bounded; and 2EXPTIME-complete in general, even when $|\mathcal{R}|$ is bounded.

3) Data Complexity: The data complexity of evaluating BCQs relative to guarded TGDs turns out to be polynomial in general and linear in the case of atomic queries.

We first give some preliminary definitions. 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 on $\mathcal{R}$. The *chase graph* for Σ and D is the directed graph consisting of $chase(D, \Sigma)$ as the set of nodes and having an arrow from $\underline{a}$ to $\underline{b}$ iff $\underline{b}$ is obtained from $\underline{a}$ and possibly other atoms by a one-step application of a TGD $\sigma \in \Sigma$. Here, we mark $\underline{a}$ as *guard* iff $\underline{a}$ is the guard of σ. The *guarded chase forest* for Σ and D is the restriction of the chase graph for Σ and D to all atoms marked as guards and their children. The *guarded chase* of level up to $k \geq 0$ for Σ and D, denoted $g\text{-}chase^k(D, \Sigma)$, is the set of all atoms in the forest of depth at most k.

Example 4: Consider the two TGDs

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

applied on $D = \{r_1(a,b), r_2(b)\}$. 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.

It can be shown 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 $\mathcal{R}$. However, this does not yet assure that also the whole derivation of the query atoms are contained in such a portion

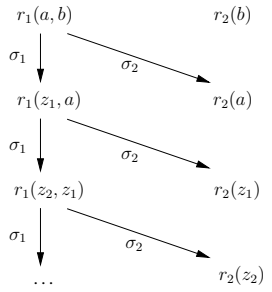

Figure 1. Guarded chase forest for Example 4.

of the guarded chase forest. This slightly stronger property is captured by the following definition.

Definition 11: We say that Σ 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 μ that maps q into $chase(D, \Sigma)$, then there is a homomorphism λ of this kind such that all ancestors of $\lambda(q)$ in the chase graph for Σ and D are contained in $g\text{-}chase^{\gamma_g}(D, \Sigma)$, where γ_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. Its 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 $\mathcal{R}$ (which is slightly larger than the one for the query atoms only).

Theorem 12 ([31]): 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) [11]. It is also hard for P, as can be proved by reduction from propositional logic programming [31].

Theorem 13 ([11], [31]): Let $\mathcal{R}$ be a relational schema, D be a database for $\mathcal{R}$, Σ 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.

Theorem 14 ([31]): Let $\mathcal{R}$ be a relational schema, D be a database for $\mathcal{R}$, Σ be a finite set of guarded TGDs on $\mathcal{R}$, and q be a Boolean atomic query over $\mathcal{R}$. Then, deciding $D \cup \Sigma \models q$ is possible in linear time in the data complexity.

C. Linear Datalog$^{\pm}$

Linear Datalog$^{\pm}$ is a variant of guarded Datalog$^{\pm}$, where query answering is even FO-rewritable in the data complexity. A TGD is *linear* iff it contains only a singleton body atom. Linear Datalog$^{\pm}$ generalizes the well-known class of *inclusion dependencies*, and is more expressive, e.g., the following linear TGD, which is not expressible with inclusion dependencies, asserts that everyone supervising her/himself is a manager: $supervises(X, X) \rightarrow manager(X)$.

1) Combined Complexity: Query answering with linear Datalog$^{\pm}$ is PSPACE-complete when the program is not fixed, which can be seen by results in [13], [32], [33], [34].

Theorem 15 ([13], [32], [33], [34]): Let $\mathcal{R}$ be a relational schema, Σ be a set of linear TGDs over $\mathcal{R}$, D be a database for $\mathcal{R}$, and q be a BCQ over $\mathcal{R}$. Then, deciding $D \cup \Sigma \models q$ is PSPACE-complete, even when q is fixed.

2) Data Complexity: Towards the data complexity, we start from some preliminaries. A class $\mathcal{C}$ of TGDs is *first-order rewritable* (or *FO-rewritable*) iff for every set of TGDs Σ in $\mathcal{C}$, and for every BCQ q, there exists a first-order query q_Σ 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 [35], it immediately follows that for FO-rewritable TGDs, BCQ answering is in AC_0 in the data complexity. The *chase of level up to* $k \geq 0$ for Σ and D, denoted $chase^k(D, \Sigma)$, is the set of all atoms in $chase(D, \Sigma)$ of derivation level at most k.

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

Clearly, in the case of linear TGDs, for every $\underline{a} \in chase(D, \Sigma)$, the subtree of $\underline{a}$ in the guarded chase forest is now 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. That is, the guarded chase forest coincides with the chase graph. By this observation, as an immediate consequence of Theorem 12, we obtain that linear TGDs have the bounded derivation-depth property.

Corollary 17 ([31]): Linear TGDs enjoy the BDDP.

The next result shows that BCQs q relative to TGDs Σ with the bounded derivation-depth property are FO-rewritable. The main ideas behind its proof are informally as follows. Since the derivation depth and the number of body atoms in TGDs in Σ 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 18 ([31]): Consider a class of TGDs $\mathcal{C}$. If $\mathcal{C}$ enjoys the BDDP, then $\mathcal{C}$ is FO-rewritable.

As an immediate consequence of Corollary 17 and Theorem 18, BCQs are FO-rewritable in the linear case.

Corollary 19 ([31]): Linear TGDs are FO-rewritable.

D. Nonmonotonic Negation

We now describe an extension of Datalog$^{\pm}$ with stratified negation, where nonmonotonic negations may be used in TGD bodies and queries. 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.

1) Normal TGDs and BCQs: 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 $\forall \mathbf{X} \forall \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{X}, \mathbf{Z})$, 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 $\Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{X}, \mathbf{Z})$. As in the case of standard TGDs, we can assume that $\Psi(\mathbf{X}, \mathbf{Z})$ is a singleton atom. Denote by $head(\sigma)$ the atom in the head of σ, and by $body^+(\sigma)$ and $body^-(\sigma)$ the sets of all positive and negative atoms (without "$\neg$") in the body of σ, respectively. We say σ is *guarded* iff it contains a positive atom in its body that contains all universally quantified variables of σ. We say σ is *linear* iff σ 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

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

$m, n \geq 1$. Denote by q^+ and q^- the positive and negative atoms (without "$\neg$") of q, respectively. We say q is *safe* iff every variable in a negative atom also occurs in a positive atom.

Example 5: Consider the following set of guarded normal TGDs Σ, 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:

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

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 $q_2 = \exists D \exists I\ \mathit{viol}(D, I), \neg \mathit{drives}(D)$, respectively.

2) Semantics and Complexity: The semantics of safe NBCQs is defined via canonical models relative to a stratification of normal TGDs. The notion of stratification of a set of normal TGDs is a generalization of the classical notion of stratification for Datalog with negation but without existentially quantified variables [36]. The canonical model semantics is then defined via iterative universal models along such a stratification, generalizing the iterative minimal model semantics for classical Datalog with negation. In general, there are several canonical models, which are all

homomorphically equivalent. We refer to [31] for details on stratifications and canonical models of normal TGDs.

There also exists a perfect model semantics of guarded Datalog$^{\pm}$ with stratified negation, which coincides with the canonical model semantics. Hence, the canonical model semantics is independent from the selected stratification.

A BCQ q *evaluates to true* in D given a set of guarded normal TGDs Σ, denoted $D \cup \Sigma \models_{strat} q$, iff there exists a homomorphism that maps q into a canonical model S_k of D given Σ. A safe NBCQ q *evaluates to true* in D given Σ, denoted $D \cup \Sigma \models_{strat} q$, iff there exists a homomorphism from q^+ to a canonical model of D given Σ, which cannot be extended to a homomorphism from some $q^+ \cup \{\underline{a}\}$, where $\underline{a} \in q^-$, to a canonical model of D given Σ.

A canonical model can be determined via iterative chases, where every chase may be infinite. But, for answering NBCQs, it is sufficient to consider only finite parts of these chases, and we obtain that answering safe NBCQs in guarded Datalog$^{\pm}$ with stratified negation is data tractable.

Theorem 20 ([31]): Let $\mathcal{R}$ be a relational schema, Σ 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 $D \cup \Sigma \models_{strat} q$ 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 21 ([31]): Stratified linear NTGDs are FO-rewritable.

V. Weakly Guarded Datalog$^{\pm}$

This section introduces the class of *weakly guarded TGDs*, also called *weakly guarded Datalog*$^{\pm}$, which is a generalization of guarded Datalog$^{\pm}$. We first give the notion of *affected* position of a schema w.r.t. a set of TGDs.

Definition 22: Given a relational schema $\mathcal{R}$ and a set of TGDs Σ over $\mathcal{R}$, an *affected position* in $\mathcal{R}$ w.r.t. Σ is defined inductively as follows. Let π_h be a position in the head of a TGD $\sigma \in \Sigma$. If an $\exists$-variable appears in π_h, then π_h is affected w.r.t. Σ. If the same $\forall$-variable X appears both in position π_h, and in the body of σ in affected positions *only*, then π_h is affected w.r.t. Σ.

It is easy to see that affected positions are the only ones where a "fresh" null of Δ_N can appear during the construction of the chase.

Definition 23: Consider a set Σ of TGDs over $\mathcal{R}$. A TGD $\sigma = \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$ in Σ is a *weakly guarded TGD (WGTGD)* w.r.t. Σ if there exists an atom in $body(\sigma)$, called a *weak guard*, that contains all the $\forall$-variables of σ that appear *only* in affected positions of $\mathcal{R}$ w.r.t. Σ.

Clearly, guarded TGDs are trivially WGTGDs since the guard atom in the body of a guarded TGD contains all the universally quantified variables, and therefore all the universally quantified variables that appear only at affected positions. The following theorem, established in [11], characterizes the complexity of reasoning under WGTGDs.

Theorem 24 ([11]): Let Σ be a set of WGTGDs over a schema $\mathcal{R}$, let D be an instance for $\mathcal{R}$, and let q be a BCQ over $\mathcal{R}$. Determining whether $D \cup \Sigma \models q$ is EXPTIME-complete in case of bounded predicate arities, and even in case Σ is fixed; it is 2EXPTIME-complete in general.

VI. Sticky Datalog$^{\pm}$

In this section, we present another language in the Datalog$^{\pm}$ family, which hinges on a paradigm that is very different from guardedness, and that we call *stickiness*.

Stickiness, formally defined below by an efficiently testable condition involving variable-marking, has also an equivalent, more intuitive definition, which is as follows. For every instance D, assume that during the chase of D under a set Σ of TGDs, we apply a TGD $\sigma \in \Sigma$ that has a variable V appearing more than once in its body; assume also that V maps (via homomorphism) on the symbol z, and that by virtue of this application the atom $\underline{a}$ is introduced. In this case, for each atom $\underline{b}$ in $body(\sigma)$, we say that $\underline{a}$ is *derived* from $\underline{b}$. Then, we have that z appears in $\underline{a}$ and in all atoms resulting from some chase derivation sequence starting from $\underline{a}$, "sticking" to them (hence the name "sticky TGDs") [37]. We now come to the formal definition.

Definition 25: Consider a set Σ of TGDs over a schema $\mathcal{R}$. We mark the variables that occur in the body of the TGDs of Σ according to the following marking procedure. First, 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 $body(\sigma)$. Now, we apply exhaustively (i.e., until a fixpoint is reached) the following step: for each TGD $\sigma \in \Sigma$, if a marked variable in $body(\sigma)$ appears at position π, then for every TGD $\sigma' \in \Sigma$ (including the case $\sigma' = \sigma$), we mark each occurrence of the variables in $body(\sigma')$ that appear in $head(\sigma')$ at the same position π. We say that Σ is a set of *sticky TGDs (STGDs)* if there is no TGD $\sigma \in \Sigma$ such that a marked variable occurs in $body(\sigma)$ more than once.

Example 6: Consider the following set Σ of TGDs:

$$\begin{aligned} p(X,Y) &\rightarrow \exists Z\, p(Y,Z) \\ p(X,Y) &\rightarrow q(X) \\ q(X), q(Y) &\rightarrow r(X,Y) \\ p(X,Y), p(Z,X) &\rightarrow q(X). \end{aligned}$$

Obviously, this set is not weakly acyclic: the first rule by itself violates weak acyclicity. On an input database as simple as $\{p(a,a)\}$, the chase does not terminate. Moreover, Σ is non-guarded. In fact, the third rule is a prime example of non-guardedness. Also, Σ is not weakly guarded, since the positions $q[1]$ and $q[2]$ are affected (see Definition 22), and thus the third rule is not weakly guarded w.r.t. Σ. However, Σ is sticky since 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.

Observe that in the chase under the database $D = \{p(a, a)\}$ and the set Σ of sticky TGDs given in the above example, the extension of the relation r is an infinite clique, and thus $chase(D, \Sigma)$ has infinite treewidth. The next theorem establishes combined complexity results for BCQ answering under STGDs.

Theorem 26 ([37]): BCQ answering under STGDs is NP-complete for fixed Σ, and EXPTIME-complete in general.

As shown in [37], sticky TGDs enjoy the BDDP (see Definition 16). Therefore, from Theorem 18, we immediately get the following result.

Corollary 27 ([37]): Sticky TGDs are FO-rewritable.

A more general class of TGDs, which we call *weakly sticky TGDs*, and which constitute *weakly sticky Datalog*$^{\pm}$, is discussed in [37]. Roughly, in a set of weakly sticky 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 chase.

VII. Negative Constraints and Keys

In this section we extend Datalog$^{\pm}$ with negative constraints and key dependencies.

A. 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 restrictions) and $\bot$ is the constant *false*; the universal quantifier is omitted for brevity. As we shall see in Section IX, constraints are vital when representing ontologies.

Example 7: 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 [31]. 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 $q_\nu = \exists \mathbf{X}\, \Phi(\mathbf{X})$ 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 for every BCQ q it holds that $D \cup \Sigma_T \cup \Sigma_\bot \models q$; otherwise, given a BCQ q, we have that $D \cup \Sigma_T \cup \Sigma_\bot \models q$ iff $D \cup \Sigma_T \models q$, i.e., we can answer q by ignoring the constraints.

Theorem 28 ([31]): Let $\mathcal{R}$ be a relational schema. Consider a set Σ_T of TGDs over $\mathcal{R}$, a set $\Sigma_\bot$ of constraints over $\mathcal{R}$, a database D for $\mathcal{R}$, and a BCQ q over $\mathcal{R}$. Then, $D \cup \Sigma_T \cup \Sigma_\bot \models q$ iff (i) $D \cup \Sigma_T \models q$ or (ii) $D \cup \Sigma_T \models q_\nu$, for some constraint $\nu \in \Sigma_\bot$.

As an immediate consequence, constraints do not increase the complexity of BCQ answering under guarded (resp., linear, weakly guarded, sticky) TGDs alone [31], [37].

B. Key Dependencies

The addition of keys is more problematic than that of constraints, since the former easily makes answering undecidable (see, e.g., [38]). For this reason, we consider a restricted class of keys, namely, *non-conflicting KDs*, which have a controlled interaction with TGDs, and thus decidability of query answering is guaranteed. Nonetheless, as we shall see in Section IX, 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 $\{r(\mathbf{X}, Y_1, \ldots, Y_m), r(\mathbf{X}, Y'_1, \ldots, Y'_m) \rightarrow Y_i = Y'_i\}_{1 \leq i \leq m}$, where the $\mathbf{X} = X_1, \ldots, X_n$ appear exactly in the attributes in $\mathbf{A}$ (w.l.o.g., the first n 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 Δ, 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, which formulates a controlled interaction of TGDs and KDs, so that the KDs do not increase the complexity of BCQ answering.

Definition 29 ([38], [31]): 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 guarded (resp., linear, weakly guarded, sticky) TGDs and KDs, the complexity of query answering is the same as in the presence of the TGDs alone. This is proved in [31], generalizing [38], by showing that in such a case we can first perform a chase failure check, which has the same complexity as BCQ answering, and then, if is negative, proceed with query answering under the TGDs alone.

We now give a sufficient syntactic condition for separability. The next definition generalizes the notion of *non-key-*

conflicting IDs introduced in [38]. This condition is crucial for using TGDs to capture ontology languages, as we will show in Section IX. Notice that, in the following definition, TGDs are assumed to have single-atom heads; this is, as stated in Section II, without loss of generality.

Definition 30 ([31]): Let $\mathcal{R}$ be a relational schema. Consider a TGD $\sigma = \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, r(\mathbf{X}, \mathbf{Z})$ over $\mathcal{R}$, and a set Σ_K of KDs over $\mathcal{R}$. We say 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 $\forall$-variable occurs is not a strict superset of $\mathbf{A}$, and (ii) each $\exists$-variable in σ occurs just once. We say Σ_K is NC relative to a set Σ_T of TGDs iff Σ_K is NC relative to every $\sigma \in \Sigma_T$.

Example 8: 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 Δ_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 $\mathbf{U} \supset \{1\}$.

VIII. Combining Decidability Paradigms

We recall the known paradigms for ensuring decidability of query answering under constraints, and how they can be combined in order to obtain more general decidable classes.

Finite Expansion Set: A set of TGDs is called *finite expansion set (FES)* if it is guaranteed that, for every instance, after finitely many application of the TGD chase rule, all further applications are superfluous; for the formal definition see [19]. It is straightforward to see that query answering under FESs is decidable, since we just need to compute the initial (finite) part of the chase that is sufficient for query answering, and then evaluate the given query over it. Every class of TGDs ensuring chase termination, in particular those discussed in Section III, is trivially a FES.

Finite Unification Set: Given a set of TGDs Σ and a BCQ q, a backward chaining mechanism is a procedure that constructs a rewriting q_Σ of q w.r.t. Σ, also called Σ-rewriting of q, such that for every database D, $D \cup \Sigma \models q$ iff $D \models q_\Sigma$. The key operation in backward chaining is the unification between the set atoms in the body of q and the head of some TGD in Σ; for the precise definitions see [20]. *Finite unification sets (FUSs)* ensure that the constructed rewriting is finite. Thus, query answering under FUSs is decidable, as we just have to build the (finite) rewriting, and then evaluate it over the given database. Interestingly, linearity and stickiness are sufficient syntactic properties which ensure that the TGDs are FUSs [31], [37].

Bounded Treewidth Set: A set of TGDs Σ is called *bounded treewidth set (BTS)* if for every database D, the chase graph of $chase(D, \Sigma)$ has bounded treewidth. Intuitively, this means that the chase graph is a "tree-like" graph. Decidability of query answering under BTSs was established in [11]. A FES is trivially a BTS, since the finite saturated graph generated by a FES has bounded treewidth. However, a FUS is not necessarily a BTS, e.g., sticky TGDs. Notice that every set of linear, guarded and weakly guarded TGDs is a BTS [11].

Extend Decidable Classes: Roughly, a TGD σ' *depends* on a TGD σ if the application of σ during the chase may cause a new application of σ' [20]. The *graph of rule dependencies (GRD)* of a set of TGDs Σ is as follows: the set of nodes is Σ, and if σ' depends on σ, then we have an edge from σ to σ' [20]. Using the GRD, we can combine the classes presented above in order to obtain more general decidable classes. Consider a set Σ of TGDs that can be partitioned into a FES Σ_1 and a FUS Σ_2, and in the GRD of Σ there is no edge from a TGD of Σ_2 to a TGD of Σ_1. Then, query answering under Σ is decidable; in particular, for every BCQ q, $D \cup \Sigma \models q$ iff $chase(D, \Sigma) \models q_{\Sigma_2}$, where q_{Σ_2} is a Σ_2-rewriting of q [20]. Query answering under Σ is still decidable even in the case where Σ_1 is a BTS [20].

From the above discussion, we immediately get that BCQ answering under a set Σ of TGDs as above, where Σ_1 is a set of guarded (resp., linear, weakly guarded) TGDs and Σ_2 is a set of sticky TGDs, is decidable.

IX. Applications

A. Data Exchange

As discussed in the introduction, the goal of *data exchange* is to automatically transfer data between heterogeneous and constrained schemas. A high-level specification for a particular data-exchange scenario is called a *schema mapping* [22] and typically consists of a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{T}, \Sigma_{st}, \Sigma_t)$ where $\mathcal{S}$ is a source schema, $\mathcal{T}$ is a target schema, Σ_{st} is a set of source-to-target TGDs, and Σ_t is a set of target TGDs and target EGDs. The corresponding *data-exchange problem* is the following: given a source instance I, compute a target instance J such that $(I \cup J) \in usol(I, \Sigma_{st} \cup \Sigma_t)$.

The results presented in Section III are of clear interest to data exchange. In particular, when $\Sigma_{st} \cup \Sigma_t$ is a set of TGDs ensuring the termination of the oblivious chase, Theorem 6 guarantees a PTIME complexity for the data-exchange problem. A technique of *substitution-free simulation* was also introduced in [24] that allows for capturing schema mappings with target EGDs. Similarly, positive results on the static analysis of schema mappings were obtained in [39] under relevant assumptions of terminations.

The results on *guarded* TGDs are also relevant to data exchange. In particular, if Σ_t is a set of guarded TGDs, then $\Sigma_{st} \cup \Sigma_t$ is a set of weakly guarded TGDs. By results of [16], we have that for each positive integer k, we can compute a so-called *quasi-universal* instance J_k, such that J_k can be used for correct query answering for conjunctive queries of size at most k. More formally, for each database D and query

size k, there exists a finite instance J_k , such that for each conjunctive query q of size at most k, $D \cup \Sigma_{st} \cup \Sigma_t \models q$ iff $J_k \models q$. Moreover, J_k is effectively computable from D; for size bounds on J_k, see [16]. We can thus always materialize a target instance that correctly serves for query answering as long as queries of a bounded size are considered. Moreover, one can decide whether a given set of TGDs (or another schema mapping) is logically implied by $\Sigma_{st} \cup \Sigma_t$.

B. Ontologies and DL-Lite

We now briefly describe how the description logics (DLs) *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$ [17] can both be reduced to linear Datalog$^{\pm}$ with (negative) constraints and NC keys, called Datalog$_0^{\pm}$, and that the former are strictly less expressive than the latter. Note that *DL-Lite*$_{\mathcal{R}}$ is able to fully capture the (DL fragment of) RDF Schema [40], the vocabulary description language for RDF; see [41] for a translation. Note also that the other DLs of the *DL-Lite* family [17] can be similarly translated to Datalog$_0^{\pm}$. In particular, the translation for *DL-Lite*$_{\mathcal{A}}$ is given in [42].

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) in *DL-Lite*$_{\mathcal{F}}$ encodes in particular 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 illustrates some DL axioms in *DL-Lite*$_{\mathcal{F}}$ and their translation to Datalog$_0^{\pm}$.

Example 9: 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:

$$\begin{array}{l} \mathit{CPaper} \sqsubseteq \mathit{Article},\ \mathit{JPaper} \sqsubseteq \mathit{Article},\\ \mathit{CPaper} \sqsubseteq \neg \mathit{JPaper},\ \mathit{Scientist} \sqsubseteq \exists \mathit{isAuthorOf},\\ \exists \mathit{isAuthorOf} \sqsubseteq \mathit{Scientist},\ \exists \mathit{isAuthorOf}^- \sqsubseteq \mathit{Article}. \end{array}$$

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

$$\begin{array}{l} \mathit{CPaper}(X) \rightarrow \mathit{Article}(X),\ \mathit{JPaper}(X) \rightarrow \mathit{Article}(X),\\ \mathit{CPaper}(X), \mathit{JPaper}(X) \rightarrow \bot,\\ \mathit{Scientist}(X) \rightarrow \exists Z\ \mathit{isAuthorOf}(X, Z),\\ \mathit{isAuthorOf}(X, Y) \rightarrow \mathit{Scientist}(X),\\ \mathit{isAuthorOf}(Y, X) \rightarrow \mathit{Article}(X). \end{array}$$

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

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

They are translated to the following TGDs and EGDs:

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

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

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

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

Formally, every knowledge base KB in *DL-Lite*$_{\mathcal{F}}$ or *DL-Lite*$_{\mathcal{R}}$ is translated into a database D_{KB}, set of TGDs Σ_{KB}, and set of queries Q_{KB} representing a set of EGDs, which are in fact linear TGDs and NC keys, respectively. The next result shows that BCQs from knowledge bases in *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$ can be reduced to BCQs in Datalog$_0^{\pm}$.

Theorem 31 ([31]): Let KB be a knowledge base in *DL-Lite*$_{\mathcal{F}}$ or *DL-Lite*$_{\mathcal{R}}$, and q be a BCQ for KB. Then, q holds in KB iff either *(i)* $D_{KB} \cup \Sigma_{KB} \models q_c$ for some $q_c \in Q_{KB}$, or *(ii)* $D_{KB} \cup \Sigma_{KB} \models q$.

Consequently, the satisfiability of knowledge bases in *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$ can be reduced to BCQs in Datalog$_0^{\pm}$.

Corollary 32 ([31]): Let KB be a knowledge base in *DL-Lite*$_{\mathcal{F}}$ or *DL-Lite*$_{\mathcal{R}}$. Then, KB is unsatisfiable iff $D_{KB} \cup \Sigma_{KB} \models q_c$ for some $q_c \in Q_{KB}$.

A further result on Datalog$_0^{\pm}$ follows.

Theorem 33 ([31]): Datalog$_0^{\pm}$ is strictly more expressive than both *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$.

Note that the TGDs used in our translation are in fact IDs (see Section II). Since a set of IDs is also a sticky set of TGDs, we have that also sticky TGDs (plus negative constraints and non-conflicting keys) are strictly more general than both *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$.

C. F-Logic Lite

F-Logic Lite, introduced in [43], is a small but expressive subset of F-Logic [44], a well-known formalism introduced for object-oriented deductive databases. For better clarity, we do not use the standard F-Logic notation; instead, we represent F-Logic Lite via the following predicates:

- $\mathsf{member}(O, C)$: object O is a member of class C.
- $\mathsf{sub}(C_1, C_2)$: class C_1 is a subclass of class C_2.
- $\mathsf{data}(O, A, V)$: attribute A has value V on object O.
- $\mathsf{type}(O, A, T)$: attribute A has type T for object O (recall that in F-Logic classes are also objects).
- $\mathsf{mandatory}(A, O)$: attribute A is mandatory for object (class) O, i.e., it must have at least one value for O.
- $\mathsf{funct}(A, O)$: A is a functional attribute for the object (class) O, i.e., it can have at most one value for O.

These predicates are related to each other by the following twelve deductive rules, which we denote as Σ_{FLL}.

ρ_1: $\mathsf{member}(V, T) \leftarrow \mathsf{type}(O, A, T), \mathsf{data}(O, A, V)$.
ρ_2: $\mathsf{sub}(C_1, C_2) \leftarrow \mathsf{sub}(C_1, C_3), \mathsf{sub}(C_3, C_2)$.

ρ_3: $\mathsf{member}(O, C_1) \leftarrow \mathsf{member}(O, C), \mathsf{sub}(C, C_1)$.
ρ_4: $V{=}W \leftarrow \mathsf{data}(O,A,V), \mathsf{data}(O,A,W), \mathsf{funct}(A,O)$.
Note that this is the only EGD in this axiomatization.
ρ_5: $\mathsf{data}(O, A, V) \leftarrow \mathsf{mandatory}(A, O)$.
Note that this is a TGD with an $\exists$-variable in the head (variable V; quantifiers are omitted).
ρ_6: $\mathsf{type}(O, A, T) \leftarrow \mathsf{member}(O, C), \mathsf{type}(C, A, T)$.
ρ_7: $\mathsf{type}(C, A, T) \leftarrow \mathsf{sub}(C, C_1), \mathsf{type}(C_1, A, T)$.
ρ_8: $\mathsf{type}(C, A, T) \leftarrow \mathsf{type}(C, A, T_1),\ \mathsf{sub}(T_1, T)$.
ρ_9: $\mathsf{mandatory}(A, C) \leftarrow$
$\mathsf{sub}(C, C_1), \mathsf{mandatory}(A, C_1)$.
ρ_{10}: $\mathsf{mandatory}(A, O) \leftarrow$
$\mathsf{member}(O, C), \mathsf{mandatory}(A, C)$.
ρ_{11}: $\mathsf{funct}(A, C) \leftarrow \mathsf{sub}(C, C_1),\ \mathsf{funct}(A, C_1)$.
ρ_{12}: $\mathsf{funct}(A, O) \leftarrow \mathsf{member}(O, C), \mathsf{funct}(A, C)$.

Membership in NP of BCQ answering under F-Logic Lite can be obtained as a special case of weakly guarded TGDs. Roughly, the *cloud* of an atom $\underline{a}$ in the chase is the set of all atoms in the chase whose arguments appear also in the given database or in $\underline{a}$. BCQ answering under a (fixed) set Σ of WGTGDs is in NP if Σ enjoys the *polynomial cloud criterion*, that is, for every instance D, the number of clouds in $chase(D, \Sigma)$ (up to D-isomorphism) is polynomial in the size of D, and also the cloud of every atom $\underline{a}$ in the chase can be computed in polynomial time in the size of D (see [11]).

It can be shown that *(1)* ρ_4 behaves analogously to a NC key, and therefore we can ignore it whenever the initial data satisfy it; *(2)* $\Sigma_{FLL} \setminus \{\rho_4\}$ is a set of WGTGDs and fulfills the polynomial cloud criterion. Thus, BCQ answering under F-Logic Lite is in NP. It is also shown that the same problem is NP-hard [11], and thus NP-complete (both with fixed and variable q).

D. RDF and Semantic Web

Many of the recent results on relational databases and ontologies proved to be applicable to RDF. One can indeed observe (see, e.g., [45]) that *blank nodes* in RDF graphs are very similar to *null values* and the notion of *lean graph* is closely related to the notion of *core* in data exchange [46], [47], [24]. It can also be seen that the rules used to specify the semantics of RDF generally consist of standard TGDs [48], [45]. Finally, an approach that fits particularly well with the Datalog$^{\pm}$ framework is the one of [49], [50], where an extension of Datalog is introduced that supports TGDs with quantifier alternations.

E. Web Data Extraction

Web data extraction deals with the automatic identification of relevant data objects on web pages, followed by the extraction of such objects and their transformation into structured data (formatted, for example, in XML or relational database format) so that the output data can be used and further processed by application programs [5]. A program that performs a data extraction task from specific web sites is called a *wrapper*. Wrappers can be hand-written or generated by tools. Datalog has been used successfully as a representation language for semi-automatic wrapper generation tools [3], [4]. In this context, an HTML page is encoded by Datalog facts whose domain constants represent the vertices of the page's HTML parsing tree (a.k.a. *DOM tree*), where the edges of the tree are represented by specific binary predicates such as *firstchild* and *nextsibling*, and where tags and other annotations are represented by monadic predicates. Datalog rules can then be written, which compute for each input page specified in this way the objects to be extracted from that page (for details, see [4]). A hot topic of current and future research is *domain-specific, fully-automated data extraction*. This means that for specific domains such as real estate or restaurants, wrappers could be automatically generated based on domain knowledge. In such a context, it is often necessary to create new (higher level) objects from existing objects on an HTML page; this is particularly useful in case automatic page analysis has to be done. For example, it is often the case that two separate HTML objects on a web page, say, two neighboring tables of the same color, should be taken together and considered a unique conceptual object, say, of type *tablebox*. We thus need to dynamically create a new object identifier, which could be achieved by rules with existential quantifiers in their heads, such as:

$$table(T_1), table(T_2), isNeighborRight(T_1, T_2),$$
$$sameColor(T_1, T_2) \rightarrow$$
$$\exists X\ tablebox(X), contains(X, T_1), contains(X, T_2)$$

Here, the $isNeighborRight$ and $sameColor$ facts are assumed to have been generated by some low-level page analysis, and $contains(U, V)$ expresses that V is a sub-object of U. This approach is currently considered by the DIADEM ERC project, which has just started at Oxford University[1]. In particular, we are currently trying to identify the most appropriate Datalog$^{\pm}$ fragment for data extraction applications.

F. Extended ER in Linear Datalog$^{\pm}$

We finally consider a conceptual modeling formalism, which is expressible by means of linear Datalog$^{\pm}$ plus KDs.

The formalism, which we call *Extended Entity-Relationship (EER)*, derives from the Entity-Relationship model, where for simplicity we assume all relationships to be binary (the general case, with arbitrary relationship arity, can be treated similarly [51]). It can be summarized as follows: *(1)* entities and relationships can have attributes; an attribute can be mandatory (instances have at least one value for it), and functional (instances have at most one value for it); *(2)* entities can participate in relationships; a participation of an entity E in a relationship R can be mandatory (instances of E participate at least once), and functional (instances of E participate at most once); *(3)* is-a relations can hold

[1]ERC Advanced Grant no. 246858 DIADEM (Domain-centric Intelligent Automated Data Extraction Methodology).

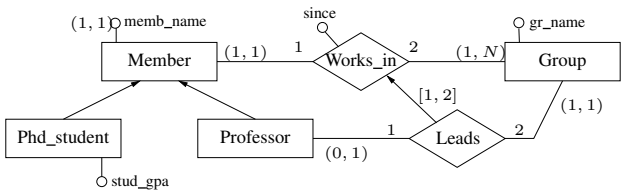

Figure 2. Example EER Schema.

between entities and between relationships; in the latter case, a permutation $[1, 2]$ or $[2, 1]$ specifies the correspondence among the components.

A knowledge base in the above formalism is called an *EER schema*. An example one is shown in Figure 2, where the graphical notation is obvious. It is shown in [51] that every EER schema can be expressed by means of a relational schema with linear TGDs and keys of a particular form; every set of dependencies in such a form is classified as *conceptual dependencies (CDs)*.

The class of CDs is not FO-rewritable [51]; this is due to the fact that in general CDs are not separable. We therefore provide a syntactic characterization of a class of CDs, called *non-conflicting CDs (NCCDs)*, which precisely captures the class of separable EER schemata—the formal definition is given in [51]. We immediately get that NCCDs are FO-rewritable, and thus BCQ answering under NCCDs is in AC_0 in data complexity.

X. Conclusion and Future Research

In this paper, we reported on the Datalog$^{\pm}$ family, and reviewed a number of languages in this family. These languages can be considered specifically-engineered (syntactic) fragments of first-order logic (possibly with nonmonotonic negation) that are suited for various tasks, for example, data exchange or ontological query answering. We find these languages rather attractive: they are simple, easy to understand, easy to analyze, decidable, and they have good complexity properties. Moreover, for ontological reasoning and query answering they turn out to be extremely versatile and expressive. In fact, we have shown that languages as simple as linear Datalog$^{\pm}$ with negative constraints and non-conflicting keys (both simple first-order features) can express very popular DLs. But unlike these DLs, the Datalog$^{\pm}$ languages are not restricted to a binary signature, and can be augmented – without problems and without additional complexity – by nonmonotonic stratified negation, a desirable expressive feature not present in DLs.

Datalog$^{\pm}$ is still a young research topic, and there are many challenging research problems to be tackled. Some of the issues that we want to address in the near future follow.

• In general, we would like to extend our decidable fragments as much as possible. As a first step, we plan to combine the two tractability paradigms guardedness and stickiness in a smart way, so to obtain a formalism that generalizes both in the best possible way.

• More expressive DLs allow for restricted forms of *transitive closure* or of transitivity constraints. Transitive closure is easily expressible in Datalog (see Example 2), but only through non-guarded rules, whose addition to decidable sets of rules may easily lead to undecidability. We would like to study under which conditions closure can be safely added to various versions of Datalog$^{\pm}$.

• Finite controllability was shown for the guarded fragment and thus holds for guarded TGDs (and it easily extends to the class of weakly guarded TGDs). We plan to study this property in the context of sticky 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 in the DL area was done in [52], [53], [54], [55].

• For those logics where query answering is FO-rewritable, the resulting FO-query is usually very large. We plan to study the optimization of such FO-rewritings from both a theoretical and a practical point of view.

• We have shown, how stratified negation can be added to Datalog$^{\pm}$. What about other forms of nonmonotonic negation such as negation under the well-founded and stable model semantics?

Acknowledgments: This research was supported by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013)/ERC grant no. 246858 – DIADEM. The authors also acknowledge support by the EPSRC project "Schema Mappings and Automated Services for Data Integration and Exchange" (EP/E010865/1) and by the German Research Foundation (DFG) under the Heisenberg Programme. Georg Gottlob's work was also supported by a Royal Society Wolfson Research Merit Award.

References

[1] S. Ceri, G. Gottlob, and L. Tanca, *Logic Programming and Databases*. Springer, 1990.

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

[3] R. Baumgartner, S. Flesca, and G. Gottlob, "Visual web information extraction with Lixto," in *Proc. of VLDB*, 2001, pp. 119–128.

[4] G. Gottlob and C. Koch, "Monadic Datalog and the expressive power of web information extraction languages," *J. ACM*, vol. 51, no. 1, pp. 71–113, 2004.

[5] R. Baumgartner, W. Gatterbauer, and G. Gottlob, "Monadic Datalog and the expressive power of web information extraction languages," in *Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, Eds. Springer, 2009, pp. 3465–3471.

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

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

[8] R. J. Miller, M. A. Hernández, L. M. Haas, L. Yan, C. T. Howard Ho, R. Fagin, and L. Popa, "The Clio project: managing heterogeneity," *SIGMOD Record*, vol. 30, no. 1, pp. 78–83, 2001.

[9] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[10] C. Beeri and M. Y. Vardi, "The implication problem for data dependencies," in *Proc. of ICALP*, 1981, pp. 73–85.

[11] A. Calì, G. Gottlob, and M. Kifer, "Taming the infinite chase: Query answering under expressive relational constraints," in *Proc. of KR*, 2008. Full version available from the authors.

[12] D. Maier, A. O. Mendelzon, and Y. Sagiv, "Testing implications of data dependencies," *ACM TODS*, vol. 4, no. 4, pp. 455–469, 1979.

[13] D. S. Johnson and A. C. Klug, "Testing containment of conjunctive queries under functional and inclusion dependencies," *J. Comput. Syst. Sci.*, vol. 28, no. 1, pp. 167–189, 1984.

[14] R. Fagin, P. G. Kolaitis, and L. Popa, "Data exchange: getting to the core," *ACM TODS*, vol. 30, no. 1, pp. 174–210, 2005.

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

[16] V. Barany, G. Gottlob, and M. Otto, "Querying the guarded fragment," in *Proc. of LICS*, 2010, this proceedings.

[17] 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*, vol. 39, no. 3, pp. 385–429, 2007.

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

[19] J.-F. Baget and M.-L. Mugnier, "Extensions of simple conceptual graphs: The complexity of rules and constraints," *J. Artif. Intell. Res.*, vol. 16, pp. 425–465, 2002.

[20] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat, "Extending decidable cases for rules with existential variables," in *Proc. of IJCAI*, 2009, pp. 677–682.

[21] A. K. Chandra and P. M. Merlin, "Optimal implementation of conjunctive queries in relational data bases," in *Proc. of STOC*, 1977, pp. 77–90.

[22] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa, "Data exchange: semantics and query answering," *Theor. Comput. Sci.*, vol. 336, no. 1, pp. 89–124, 2005.

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

[24] B. Marnette, "Generalized schema-mappings: from termination to tractability," in *Proc. of PODS*, 2009, pp. 13–22.

[25] A. Deutsch and V. Tannen, "Reformulation of XML queries and constraints," in *Proc. of ICDT*, 2003, pp. 225–241.

[26] M. Meier, M. Schmidt, G. Lausen, "On chase termination beyond stratification," *PVLDB*, vol. 2, no. 1, pp. 970–981, 2009.

[27] E. Grädel, "On the restraining power of guards," *J. Symb. Log.*, vol. 64, no. 4, pp. 1719–1742, 1999.

[28] ——, "Decision procedures for guarded logics," in *Proc. of CADE*, 1999, pp. 31–51.

[29] M. Otto, "Avoiding incidental homomorphisms into guarded covers," Technische Universität Darmstadt, Tech. Rep., 2009.

[30] R. Rosati, "On the decidability and finite controllability of query processing in databases with incomplete information," in *Proc. of PODS*, 2006, pp. 356–365.

[31] A. Calì, G. Gottlob, and T. Lukasiewicz, "A general Datalog-based framework for tractable query answering over ontologies," in *Proc. of PODS*, 2009, pp. 77–86.

[32] M. Y. Vardi, 1984, personal communication reported in [13].

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

[34] G. Gottlob and C. H. Papadimitriou, "On the complexity of single-rule Datalog queries," *Inform. Comput.*, vol. 183, no. 1, pp. 104–122, 2003.

[35] M. Y. Vardi, "On the complexity of bounded-variable queries," in *Proc. of PODS*, 1995, pp. 266–276.

[36] K. Apt, H. Blair, and A. Walker, "Towards a theory of declarative knowledge," in *Foundations of Deductive Databases and Logic Programming*, 1988, pp. 89–148.

[37] A. Calì, G. Gottlob, and A. Pieris, "Advanced processing for ontological queries," 2010, unpublished manuscript. Available at http://benner.dbai.tuwien.ac.at/staff/gottlob/CGP.pdf.

[38] 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, pp. 260–271.

[39] B. Marnette and F. Geerts, "Static analysis of schema-mappings ensuring oblivious termination," in *Proc. of ICDT*, 2010, to appear.

[40] 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.

[41] J. de Bruijn and S. Heymans, "Logical foundations of (e)RDF(S): Complexity and reasoning," in *Proc. of ISWC*, 2007, pp. 86–99.

[42] A. Calì, G. Gottlob, and T. Lukasiewicz, "Datalog$^{\pm}$: A unified approach to ontologies and integrity constraints," in *Proc. of ICDT*, 2009, pp. 14–30.

[43] A. Calì and M. Kifer, "Containment of conjunctive object meta-queries," in *Proc. of VLDB*, 2006, pp. 942–952.

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

[45] C. Gutierrez, C. Hurtado, and A. O. Mendelzon, "Foundations of semantic web databases," in *Proc. of PODS*, 2004.

[46] R. Fagin, P. G. Kolaitis, and L. Popa, "Data exchange: getting to the core," *ACM TODS*, vol. 30, no. 1, pp. 174–210, 2005.

[47] G. Gottlob and A. Nash, "Efficient core computation in data exchange," *J. ACM*, vol. 55, no. 2, 2008.

[48] P. Hayes, "RDF semantics," http://www.w3.org/TR/2004/REC-rdf-mt-20040210/, 2004, W3C Recommendation.

[49] F. Bry, T. Furche, C. Ley, B. Linse, and B. Marnette, "RDFLog: It's like Datalog for RDF," in *Proc. of WLP*, 2008.

[50] F. Bry, T. Furche, B. Marnette, C. Ley, B. Linse, and O. Poppe, "SPARQLog: SPARQL with rules and quantification," in *Semantic Web Information Management: A Model-Based Perspective*, 2010, pp. 341 – 369.

[51] A. Calì, G. Gottlob, and A. Pieris, "Tractable query answering over conceptual schemata," in *Proc. of ER*, 2009.

[52] R. Rosati, "Finite model reasoning in *DL-Lite*," in *Proc. of ESWC*, 2008, pp. 215–229.

[53] ——, "On the finite controllability of conjunctive query answering in databases under open-world assumption," *J. Comput. Syst. Sci.*, 2010, to appear.

[54] C. Lutz, U. Sattler, and L. Tendera, "The complexity of finite model reasoning in description logics," *Inform. Comput.*, vol. 199, no. 1–2, pp. 132–171, 2005.

[55] D. Calvanese, "Finite model reasoning in description logics," in *Proc. of KR*, 1996, pp. 292–303.