Efficient Query Answering over *Datalog* with Existential Quantifiers*

Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri

Department of Mathematics, University of Calabria, Italy
{leone,manna,terracina,veltri}@mat.unical.it

Abstract. This paper faces the problem of answering conjunctive queries over *Datalog* programs allowing existential quantifiers in rule heads. Such an extension of *Datalog* is highly expressive, enables easy yet powerful ontology-modelling, but leads to undecidable query answering in general. To overcome undecidability, we first define *Shy*, a subclass of *Datalog* with existential quantifiers preserving not only decidability but also the same complexity of query answering over *Datalog*. Next, we design and implement a bottom-up evaluation strategy for *Shy* programs. Our computation strategy includes a number of optimizations resulting in DLV$^{\exists}$, a powerful reasoner over *Shy* programs. Finally, we carry out an experimental analysis comparing DLV$^{\exists}$ with some state-of-the-art systems for ontology-based query answering. The results confirm the effectiveness of DLV$^{\exists}$, which outperforms all other systems in the benchmark.

1 Introduction

In the field of data and knowledge management, *query answering over ontologies* (QA) is becoming more and more a challenging task [9,6,18]. In this context, a conjunctive query (CQ) q is not merely evaluated on a extensional relational database D, but over a logical theory combining D with an ontology Σ describing rules for inferring intensional knowledge from D. A key issue here is the design of the language provided for specifying Σ. It should balance expressiveness and complexity, and in particular it should possibly be: (1) intuitive and easy-to-understand; (2) QA-decidable; (3) efficiently computable; (4) expressive enough; and (5) suitable for an efficient implementation. In this regard, Datalog$^{\pm}$ [6], the family of *Datalog*-based languages recently proposed for tractable QA, is arousing increasing interest. This family, generalizing well known ontology specification languages, is mainly based on *Datalog*$^{\exists}$, the natural extension of *Datalog* [1] that allows $\exists$-quantified variables in rule heads. For example, the rules $\exists$`Y father(X,Y)` $\leftarrow$ `person(X)` and `person(Y)` $\leftarrow$ `father(X,Y)` state that if `X` is a person, then `X` must have a father `Y`, which has to be a person as well. However, even if all known QA-decidable Datalog$^{\pm}$ languages enjoy the simplicity of *Datalog* and are endowed with properties that are desired for ontology languages, none of them fully satisfy conditions (1)–(5) above (see Section 5).

* Extended Abstract

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

In this work, we single out *Shy*, a new powerful, yet QA-decidable *Datalog*$^\exists$ class that combines positive aspects of different Datalog$^\pm$ languages. Concerning properties (1)–(5) above, *Shy*: (1) inherits the simplicity and naturalness of *Datalog*; (2) is QA-decidable; (3) is efficiently computable; (4) offers good expressiveness strictly generalizing *Datalog*; and (5) is suitable for an efficient implementation. From a technical viewpoint, our contribution is as follows:

– We define *Shy*, a subclass of *Datalog*$^\exists$ that preserves not only decidability but also the same data (resp., combined) complexity of *Datalog* for unrestricted conjunctive queries (resp., atomic queries).
– We show that *Shy* strictly encompasses both *Datalog* and *Linear-Datalog*$^\exists$ [6], and that it is uncomparable to all other known *Datalog*$^\exists$ classes.
– We design and implement a bottom-up evaluation strategy for *Shy* programs inside the DLV system. Our computation strategy includes a number of optimizations resulting in DLV$^\exists$, a powerful reasoner over *Shy* programs. To the best of our knowledge, DLV$^\exists$ is the first system supporting the standard first-order semantics for unrestricted CQs with $\exists$-variables over ontologies using advanced properties (some of these beyond $\mathbf{AC}_0$), such as, role transitivity, role hierarchy, role inverse, and concept products [14].
– We perform an experimental analysis comparing DLV$^\exists$ with some state-of-the-art systems for QA. The results demonstrate that DLV$^\exists$ is the most effective system for QA in dynamic environments where the ontology frequently changes making pre-computations and static optimizations inapplicable.

2 The Framework

This section, after introducing *Datalog*$^\exists$ programs and CQs, equips such structures with a formal semantics and defines the query answering problem.

Preliminaries. Throughout this paper we denote by Δ_C, Δ_N and Δ_V, countably infinite domains of *terms* called *constants*, *nulls* and *variables*, respectively; by Δ, the union of these three domains; by t, a generic *term* in Δ; by $\mathtt{X}$ and $\mathtt{Y}$, variables; by $\mathbf{X}$ and $\mathbf{Y}$, sets of variables; by Π an alphabet of *predicate symbols* each of which, say $\mathtt{p}$, has a fixed nonnegative arity; by $\mathbf{a}$ and $\mathbf{b}$, *atoms* being expressions of the form $\mathtt{p}(t_1, \ldots, t_k)$, where $\mathtt{p}$ is a predicate symbol, and $t_1, \ldots, t_k$ is a *tuple* of terms. For a formal structure ς containing atoms, $\mathsf{atoms}(\varsigma)$ denotes the set of atoms in ς, and $\mathsf{dom}(\varsigma)$ denotes the set of terms from $\Delta_C \cup \Delta_N$ occurring in $\mathsf{atoms}(\varsigma)$. If $\mathbf{X}$ is the set of all variables of ς, then ς is also denoted by $\varsigma_{[\mathbf{X}]}$. A structure $\varsigma_{[\emptyset]}$ is called *ground*. If $T \subseteq \Delta$ and $T \neq \emptyset$, then $\mathsf{base}(T)$ denotes the set of all atoms that can be formed with predicate symbols in Π and terms from T.

A *substitution* is a mapping $\sigma : \Delta_V \to \Delta_C \cup \Delta_N$. For a set $\mathbf{X} \subseteq \Delta_V$, the application of σ to $\mathbf{X}$ is the set $\sigma(\mathbf{X}) = \{\sigma(\mathtt{X}) \mid \mathtt{X} \in \mathbf{X}\}$. Moreover, the restriction of σ to $\mathbf{X}$, is the substitution $\sigma|_{\mathbf{X}}$ from $\mathbf{X}$ to $\Delta_C \cup \Delta_N$ s.t. $\sigma'(\mathtt{X}) = \sigma(\mathtt{X})$ for each $\mathtt{X} \in \mathbf{X}$. For an atom $\mathbf{a} = \mathtt{p}(t_1, \ldots, t_k)$, $\sigma(\mathbf{a})$ denotes the atom obtained from $\mathbf{a}$ by replacing each variable $\mathtt{X}$ of $\mathbf{a}$ with $\sigma(\mathtt{X})$. For a structure ς containing atoms, we denote by $\sigma(\varsigma)$ the structure obtained by replacing each atom $\mathbf{a}$ of ς with $\sigma(\mathbf{a})$.

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6, Copyright (c) 2012 - Edizioni Libreria Progetto and the authors

Programs and Queries. A *rule* r is a finite expression of the form:

$$\forall \mathbf{X} \exists \mathbf{Y} \ \ \mathbf{atom}_{[\mathbf{X}' \cup \mathbf{Y}]} \leftarrow \mathbf{conj}_{[\mathbf{X}]} \tag{1}$$

where $\mathbf{X}$ and $\mathbf{Y}$ are disjoint sets of variables (next called $\forall$-variables and $\exists$-variables, respectively), and $\mathbf{X}' \subseteq \mathbf{X}$. In the following, $\mathsf{head}(r) = \mathbf{atom}_{[\mathbf{X}' \cup \mathbf{Y}]}$ and $\mathsf{body}(r) = \mathsf{atoms}(\mathbf{conj}_{[\mathbf{X}]})$. If $\mathsf{body}(r) = \emptyset$, then r is also called *fact*. A $Datalog^{\exists}$ program P is a finite set of rules. We denote by $\mathsf{data}(P)$ all the atoms constituting the ground facts of P.

Example 1. Let *P-Jungle* be a $Datalog^{\exists}$ program including the following rules:

r_1 : $\exists$`Z pursues(Z,X)` $\leftarrow$ `escapes(X)`
r_2 : `hungry(Y)` $\leftarrow$ `pursues(Y,X), fast(X)`
r_3 : `pursues(X,Y)` $\leftarrow$ `pursues(X,W), prey(Y)`
r_4 : `afraid(X)` $\leftarrow$ `pursues(Y,X), hungry(Y), strongerThan(Y,X).`

The program describes a funny scenario where an escaping, yet fast animal `X` may induce other animals to be afraid. Data for *P-Jungle* could be `escapes(gazelle)`, `fast(gazelle)`, `prey(antelope)`, `strongerThan(lion,antelope)`, and possibly also `pursues(lion,gazelle)`. We will use *P-Jungle* as a running example. □

Given a $Datalog^{\exists}$ program P, a *conjunctive query* (CQ) q over P is a first-order (FO) expression of the form $\exists \mathbf{Y}\ \mathbf{conj}_{[\mathbf{X} \cup \mathbf{Y}]}$, where $\mathbf{X}$ are its free variables. To highlight the free variables, we write $q(\mathbf{X})$ instead of q. Query q is a *Boolean CQ* (BCQ) if $\mathbf{X} = \emptyset$. Moreover, q is called *atomic* if $\mathbf{conj}$ is an atom.

Example 2. Animals pursed by a *lion* and stronger than some other animal can be retrieved by means of a CQ $\exists$`Y pursues(lion,X), strongerThan(X,Y).` □

Semantics and Query Answering. Given a set S of atoms and an atom $\mathbf{a}$, we say S entails $\mathbf{a}$ ($S \models \mathbf{a}$ for short) if there is a substitution σ s.t. $\sigma(\mathbf{a}) \in S$. Let $P \in Datalog^{\exists}$. A set $M \subseteq \mathsf{base}(\Delta_C \cup \Delta_N)$ is a *model* for P ($M \models P$) if $M \models \sigma|_{\mathbf{X}}(\mathsf{head}(r))$ for each $r \in P$ of the form (1) and substitution σ s.t. $\sigma(\mathsf{body}(r)) \subseteq M$. Let $\mathsf{mods}(P)$ denote the set of models of P. Let $M \in \mathsf{mods}(P)$. A BCQ q is *true* w.r.t. M ($M \models q$) if there is a substitution σ s.t. $\sigma(\mathsf{atoms}(q)) \subseteq M$. Analogously, the answer of a CQ $q(\mathbf{X})$ w.r.t. M is the set $\mathsf{ans}(q, M) = \{\sigma|_{\mathbf{X}} : \sigma \text{ is a substitution } \wedge\ M \models \sigma|_{\mathbf{X}}(q)\}$. The answer of a CQ $q(\mathbf{X})$ w.r.t. a program P is the set $\mathsf{ans}_P(q) = \{\sigma : \sigma \in \mathsf{ans}(q, M)\ \forall M \in \mathsf{mods}(P)\}$. Note that for a BCQ q either $\mathsf{ans}_P(q) = \{\sigma|_{\emptyset}\}$ or $\mathsf{ans}_P(q) = \emptyset$. In the former case we say that q is (cautiously) *true* w.r.t. P, denoted by $P \models q$.

Let $\mathcal{C}$ be a class of $Datalog^{\exists}$ programs. *Query answering* (QA) over $\mathcal{C}$ is the following decision problem: *Given a program P in $\mathcal{C}$ and a BCQ q, determine whether $P \models q$ holds.* Class $\mathcal{C}$ is called QA-decidable if QA over $\mathcal{C}$ is decidable. We observe that computing $\mathsf{ans}_P(q)$ for a CQ $q(\mathbf{X})$ is Turing-reducible to QA. In fact, $\mathsf{ans}_P(q)$ is the set of substitutions σ s.t. the BCQ $\sigma(q)$ is true w.r.t. P. However, since $\sigma \in \mathsf{ans}_P(q)$ implies $\sigma(\mathbf{X}) \subseteq \mathsf{dom}(P)$, only finitely many substitutions have to be considered [13].

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

Let P be a *Datalog*$^{\exists}$ program. It is well-known that QA can be carried out by using a *universal model* of P, namely a model U of P s.t., for each BCQ q, $P \models q$ iff $U \models q$ [13]. A well-known procedure for constructing such a model U is the CHASE [17]. Intuitively, the CHASE first sets U to $\mathsf{data}(P)$. Next, it exhaustively repairs rules that are not satisfied in U by adding to U new atoms having "fresh" nulls in the positions of $\exists$-variables. Eventually, it terminates if $U \models P$. (See [19] for a formal definition.) Unfortunately, although the CHASE always constructs a (possibly infinite) universal model of P, QA remains undecidable in general [13].

3 *Shy*: A novel QA-decidable *Datalog*$^{\exists}$ class

The key idea behind this class intuitively relays on the following *shyness* property: *During a* CHASE*-run on a Shy program P, nulls propagated body-to-head do not meet each other to join.* In this regard, given a *Datalog*$^{\exists}$ program P, to detect whether a CHASE-run on P might produce some atom containing a null at a given position, we define the *null-set* of a position in an atom. More precisely, $\varphi^r_{\mathtt{X}}$ denotes the "representative" null introduced (during a CHASE-run on P) due to the $\exists$-variable $\mathtt{X}$ of the rule $r \in P$. (If $(r, \mathtt{X}) \neq (r', \mathtt{X}')$, then $\varphi^r_{\mathtt{X}} \neq \varphi^{r'}_{\mathtt{X}'}$.) Let $\mathbf{a}$ be an atom, and $\mathtt{X}$ a variable occurring in $\mathbf{a}$ at position i. The *null-set* of position i in $\mathbf{a}$ w.r.t. P, denoted by $\mathsf{nullset}(i, \mathbf{a})$, is inductively defined as follows. If $\mathbf{a}$ is the head atom of some rule $r \in P$, then $\mathsf{nullset}(i, \mathbf{a})$ is: (1) either the set $\{\varphi^r_{\mathtt{X}}\}$, if $\mathtt{X}$ is $\exists$-quantified in r; or (2) the intersection of every $\mathsf{nullset}(j, \mathbf{b})$ s.t. $\mathbf{b} \in \mathsf{body}(r)$ and $\mathtt{X}$ occurs at position j in $\mathbf{b}$, if $\mathtt{X}$ is $\forall$-quantified in r. If $\mathbf{a}$ is not a head atom, then $\mathsf{nullset}(i, \mathbf{a})$ is the union of $\mathsf{nullset}(i, \mathsf{head}(r))$ for each $r \in P$ s.t. $\mathsf{pred}(\mathsf{head}(r)) = \mathsf{pred}(\mathbf{a})$. A representative null φ *invades* a variable $\mathtt{X}$ that occurs at position i in an atom $\mathbf{a}$ if φ is contained in $\mathsf{nullset}(i, \mathbf{a})$. A variable $\mathtt{X}$ occurring in a conjunction of atoms **conj** is *attacked* in **conj** by a null φ if each occurrence of $\mathtt{X}$ in **conj** is invaded by φ. A variable $\mathtt{X}$ is *protected* in **conj** if it is attacked by no null. We are now ready to define the new *Datalog*$^{\exists}$ class.

Definition 1 ([19]). *Let Shy denote the class of all Datalog$^{\exists}$ programs containing only shy rules, where a rule r is called* shy *w.r.t. a program P if the following conditions are satisfied:* (i) *If a variable occurs in more than one body atom, then this variable is protected in $\mathsf{body}(r)$;* (ii) *If two distinct $\forall$-variables are not protected in $\mathsf{body}(r)$ but occur both in $\mathsf{head}(r)$ and in two different body atoms, then they are not attacked by the same null.* □

Resuming program *P-Jungle* of Example 1 we now prove its shyness. Let $\mathbf{a}_1, \ldots, \mathbf{a}_{12}$ be the atoms of rules r_1–r_4 in left-to-right/top-to-bottom order, and $\mathsf{nullset}(1, \mathbf{a}_1)$ be $\{\varphi^{r_1}_{\mathtt{Z}}\}$. First, we propagate $\varphi^{r_1}_{\mathtt{Z}}$ (head-to-body) to $\mathsf{nullset}(1, \mathbf{a}_4)$, $\mathsf{nullset}(1, \mathbf{a}_7)$, and $\mathsf{nullset}(1, \mathbf{a}_{10})$. Next, this null is propagated (body-to-head) from $\mathbf{a}_4$, $\mathbf{a}_7$ and $\mathbf{a}_3$ to $\mathsf{nullset}(1, \mathbf{a}_3)$, $\mathsf{nullset}(1, \mathbf{a}_6)$ and $\mathsf{nullset}(1, \mathbf{a}_{11})$, respectively. Finally, we observe that rules r_1–r_3 are trivially shy, and that r_4 also is because variable $\mathtt{Y}$ is not invaded in $\mathbf{a}_{12}$ even if $\varphi^{r_1}_{\mathtt{Z}}$ invades $\mathtt{Y}$ both in $\mathbf{a}_{10}$ and $\mathbf{a}_{11}$.

According to Definition 1, we observe that a program is *Shy* regardless its ground facts. Finally, we mention notable computational properties of this class.

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6, Copyright (c) 2012 - Edizioni Libreria Progetto and the authors

Theorem 1 ([19]). *Checking whether a program P is shy is decidable. In particular, it is doable in polynomial-time.*

Theorem 2 ([19]). *QA over Shy is decidable. In particular, it is **P**-complete in data complexity for unrestricted conjunctive queries, and **EXP**-complete in combined complexity for atomic queries.*

4 Implementation and Experiments

We implemented a system, called DLV$^\exists$, that computes a very succinct set of atoms for answering CQs over *Shy* programs. Let P be a *Shy* program and q be a CQ. First, $\exists$-variables in rule heads are managed by skolemization. In particular, every rule $r \in P$ is skolemized, and skolemized terms are interpreted as functional symbols [8] within DLV$^\exists$. Next, the system singles out the set of predicates that are relevant for answering q by recursively traversing top-down (head-to-body) the rules in P, starting from the query predicates. This information is used to filter out, at loading time, the facts belonging to predicates irrelevant for answering the input query. At this point, the computation is further optimized rewriting P by a variant of the well-known Magic-Sets technique [11,3], that we adapted to *Datalog*$^\exists$.

We carried out an experimental analysis considering the well-known LUBM benchmark (see `http://swat.cse.lehigh.edu/projects/lubm/`). It refers to a university domain and includes a synthetic data generator, which we used to generate three increasing data sets, namely `lubm-10`, `lubm-30` and `lubm-50`. LUBM incorporates a set of 14 queries referred to as q_1–q_{14}. Tests were performed on an Intel Xeon X3430, 2.4 GHz, with 4 Gb Ram, running Linux Operating System. For each query, we allowed a maximum running time of 7200 seconds and a maximum memory usage of 2 Gb.

We compared DLV$^\exists$ with three state-of-the-art reasoners called Pellet [25], OWLIM-SE [4] and OWLIM-Lite [4]. Results are reported in Table 1, where

Table 1. Systems comparison: running time (sec.) and average time (G.Avg)

	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}	q_{11}	q_{12}	q_{13}	q_{14}	q_{all}	G.Avg
lubm-10																
DLV$^\exists$	5	4	2	4	6	1	6	4	8	5	<1	1	6	2	17	2.87
Pellet	82	84	84	82	80	88	81	89	95	82	82	89	82	84	27	84.48
OWLIM-Lite	33	–	33	33	33	33	4909	70	–	33	33	33	33	33	33	53.31
OWLIM-SE	105	105	105	105	105	105	105	106	106	105	105	105	105	105	105	105.14
lubm-30																
DLV$^\exists$	16	13	7	14	21	3	21	12	25	18	<1	5	23	8	55	9.70
Pellet	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
OWLIM-Lite	107	–	107	106	107	106	–	528	–	107	106	106	107	106	106	123.18
OWLIM-SE	323	328	323	323	323	323	323	323	326	323	323	323	323	323	323	323.57
lubm-50																
DLV$^\exists$	27	23	12	23	35	6	34	22	42	31	<1	9	33	14	93	16.67
Pellet	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
OWLIM-Lite	188	–	190	187	189	188	–	1272	–	189	187	187	189	187	187	223.79
OWLIM-SE	536	547	536	536	536	537	536	536	542	536	536	536	536	537	536	537.35

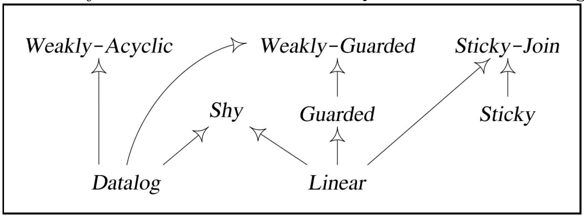
Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

times include the *total time* required for QA. We measured the total time, including data parsing and loading, because we are interested in ontology reasoning contexts where data and knowledge rapidly vary, even within hours. DLV$^\exists$ significantly outperforms all other systems in all tested queries and data sets. Comparing the other systems, OWLIM-Lite is in general faster than Pellet and OWLIM-SE. Pellet is faster than OWLIM-SE on `lubm-10`, but it answered no tested queries in the allotted time on `lubm-30` and `lubm-50`. Finally, column q_{all} shows the time taken by the systems to perform *fact inference*, namely to compute all atomic consequences required to answer any atomic query. Interestingly, even if DLV$^\exists$ is specifically designed for QA over frequently changing ontologies, it outperformed the competitors also in performing fact inference. Indeed, DLV$^\exists$ took about 17% and 51% of the time taken by OWLIM-SE and OWLIM-Lite.

5 Related Work and Discussion

Comparison with the literature reveals that *Shy* offers the best balance between expressivity and complexity among all known QA-decidable *Datalog*$^\exists$ classes relaying on the three main paradigms called *weak-acyclicity* [13], *guardness* [5] and *stickiness* [7]. Figure 1 provides a taxonomy of the most representative of these classes. In particular, for each pair $\mathcal{C}_1$ and $\mathcal{C}_2$ of classes, there is a direct path from $\mathcal{C}_1$ to $\mathcal{C}_2$ iff $\mathcal{C}_1 \subset \mathcal{C}_2$; also, $\mathcal{C}_1$ and $\mathcal{C}_2$ are not linked by any directed path iff they are uncomparable [19]. Table 2 summarizes the complexity of QA over these classes [19]. In both diagrams, only *Datalog* is intended to be $\exists$-free. In fact, among these classes, *Shy* is the only one supporting advanced, yet relevant properties such as role-transitivity and concept-product [24] (besides standard ontological properties such as role-hierarchy, role-inverse, concept-hierarchy). More specifically, even if *Weakly-Guarded* [5] encompasses and generalizes both *Datalog* and *Linear* [5] as *Shy*, it has untractable data complexity and no implementation. *Weakly-Acyclic* [13] and *Guarded* [5] are tractable (although they suffer of higher combined complexity than *Shy*) but the former does not include *Linear* (even the basic "father-person" ontology cannot be represented), while the latter does not include *Datalog* and supports neither role-transitivity nor concept-product. Moreover, no efficient implementation of *Guarded* has been found so far since the natural termination condition on *Guarded* programs requires the generation of a huge set of atoms for answering CQs. *Sticky* and *Sticky-Join* [7] are suitable for an efficient implementation and capture some light-weight DL properties but,

Fig. 1. Taxonomy of the most attractive QA-decidable *Datalog*$^\exists$ classes



Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

Class $\mathcal{C}$	*Data Complexity*	*Combined Complexity*
Weakly-Guarded	**EXP**-complete	**2EXP**-complete
Guarded, *Weakly-Acyclic*	**P**-complete	**2EXP**-complete
Datalog, *Shy*	**P**-complete	**EXP**-complete
Sticky, *Sticky-Join*	in $\mathbf{AC}_0$	**EXP**-complete
Linear	in $\mathbf{AC}_0$	**PSPACE**-complete

Table 2. Complexity of atomic query answering

since they do not generalize *Datalog*, they cannot express important KR features such as role-transitivity.

Regarding related systems, to the best of our knowledge, the only one directly supporting ∃-quantifiers in *Datalog* is Nyaya [12], which performs QA over *Linear*-*Datalog*$^{\exists}$. (We could not compare DLV$^{\exists}$ with Nyaya since it still provides no API for data loading and querying.) Concerning ontology reasoners, we mention QuOnto [2], Presto [23], Quest [21], Mastro [10] and OBDA [22] which rewrite axioms and queries to SQL. Such systems support standard FO semantics for unrestricted CQs, but the expressivity of their languages is limited to $\mathbf{AC}_0$ and excludes, e.g., transitivity property or concept products. The systems FaCT++ [26], RacerPro [15], Pellet [25] and HermiT [20] materialize all inferences at loading-time, implement very expressive description logics, but they do not support the standard FO semantics for CQs [14]. Actually, the Pellet system enables first-order CQs but only in the acyclic case. OWLIM [4] and KAON2 [16] perform full-materialization and implement expressive DLs, but they still miss to support the standard FO semantics for CQs [14].

Summing up, it turns out that DLV$^{\exists}$ is the first system supporting the standard FO semantics for unrestricted CQs with ∃-variables over ontologies using advanced properties (even beyond $\mathbf{AC}_0$), such as, role transitivity, role hierarchy, role inverse, and concept products. The experiments confirm the efficiency of DLV$^{\exists}$, which constitutes a powerful system for a fully-declarative QA.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., 1995.
2. A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. QUONTO: querying ontologies. In *Proc. of the 20th AAAI Conf. on AI*, volume 4, pages 1670–1671, 2005.
3. M. Alviano, W. Faber, G. Greco, and N. Leone. Magic Sets for Disjunctive Datalog Programs. Technical Report 09/2009, Dept. of Math., Univ. of Calabria, IT, 2009. See `https://www.mat.unical.it/~faber/research/papers/TRMAT092009.pdf`.
4. B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov. OWLIM: A family of scalable semantic repositories. *Semant. Web*, 2:33–42, 2011.
5. A. Calì, G. Gottlob, and M. Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. In *Proc. of the 11th KR Conf.*, pages 70–80, 2008. See: `http://dbai.tuwien.ac.at/staff/gottlob/CGK.pdf`.

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

6. A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *Proc. of the 28th PODS Symp.*, pages 77–86, 2009.
7. A. Calì, G. Gottlob, and A. Pieris. Advanced Processing for Ontological Queries. *PVLDB*, 3(1):554–565, 2010.
8. F. Calimeri, S. Cozza, G. Ianni, and N. Leone. Enhancing ASP by Functions: Decidable Classes and Implementation Techniques. In *Proc. of the 24th AAAI Conf. on AI*, pages 1666–1670, 2010.
9. D. Calvanese, G. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *J. Autom. Reason.*, 39:385–429, 2007.
10. D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo. The MASTRO system for ontology-based data access. *Semant. Web*, 2(1):43–53, 2011.
11. C. Cumbo, W. Faber, G. Greco, and N. Leone. Enhancing the Magic-Set Method for Disjunctive Datalog Programs. In *Proc. of the 20th ICLP*, volume 3132, pages 371–385, 2004.
12. R. De Virgilio, G. Orsi, L. Tanca, and R. Torlone. Semantic Data Markets: A Flexible Environment for Knowledge Management. In *Proc. of the 20th Int. CIKM*, pages 1559–1564, 2011.
13. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *TCS*, 336(1):89–124, 2005.
14. B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive query answering for the description logic SHIQ. *JAIR*, 31(1):157–204, 2008.
15. V. Haarslev and R. Möller. RACER System Description. In *Proc. of the 6th IJCAR*, pages 701–705, 2001.
16. U. Hustadt, B. Motik, and U. Sattler. Reducing SHIQ- Descrption Logic to Disjunctive Datalog Programs. In *Proc. of the 9th KR Conf.*, pages 152–162, 2004.
17. D. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.
18. I. Kollia, B. Glimm, and I. Horrocks. SPARQL Query Answering over OWL Ontologies. In *Proc. of the 24th DL Int. Workshop*, volume 6643 of *LNCS*, pages 382–396. Springer, 2011.
19. N. Leone, M. Manna, G. Terracina, and P. Veltri. Efficiently Computable Datalog$^{\exists}$ Programs. In *Proc. of the 13th KR Int. Conf.*, 2012. To Appear. See `www.mat.unical.it/kr2012/`.
20. B. Motik, R. Shearer, and I. Horrocks. Hypertableau Reasoning for Description Logics. *JAIR*, 36:165–228, 2009.
21. M. Rodriguez-Muro and D. Calvanese. Dependencies: Making Ontology Based Data Access Work in Practice. In *Proc. of the 5th AMW on Foundations of Data Management*, volume 477, 2011.
22. M. Rodriguez-Muro and D. Calvanese. Dependencies to Optimize Ontology Based Data Access. In *Description Logics*, volume 745. CEUR-WS.org, 2011.
23. R. Rosati and A. Almatelli. Improving Query Answering over DL-Lite Ontologies. In *Proc. of the 12th KR Conf.*, pages 290–300, 2010.
24. S. Rudolph, M. Krötzsch, and P. Hitzler. All Elephants are Bigger than All Mice. In *Proc. of the 21st DL Int. Workshop*, volume 353, 2008.
25. E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semant.*, 5(2):51–53, 2007.
26. D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. of the 3rd IJCAR*, volume 4130, pages 292–297, 2006.

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,