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_³, a powerful reasoner over _Shy_ programs. Finally, we carry out an experimental analysis comparing _DLV_³ with some state-of-the-art systems for ontology-based query answering. The results confirm the effectiveness of _DLV_³, 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_± [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_³, the natural extension of _Datalog_ [1] that allows _∃_-quantified variables in rule heads. For example, the rules _∃Y_ _father_(X,Y) ← _person_(X) and _person_(Y) ← _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_± 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
In this work, we single out Shy, a new powerful, yet QA-decidable $Datalog^3$ class that combines positive aspects of different $Datalog^2$ 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 $Datalog^3$ 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 $Linear-Datalog^3$ [6], and that it is incomparable to all other known $Datalog^3$ 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 $DLV^3$, a powerful reasoner over Shy programs. To the best of our knowledge, $DLV^3$ 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 AC0), such as, role transitivity, role hierarchy, role inverse, and concept products [14].
- We perform an experimental analysis comparing $DLV^3$ with some state-of-the-art systems for QA. The results demonstrate that $DLV^3$ 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 $Datalog^3$ programs and CQs, equips such structures with a formal semantics and defines the query answering problem.

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

A substitution is a mapping $\sigma : \Delta_V \to \Delta_C \cup \Delta_N$. For a set $X \subseteq \Delta_V$, the application of $\sigma$ to $X$ is the set $\sigma(X) = \{ \sigma(x) \mid x \in X \}$. Moreover, the restriction of $\sigma$ to $X$, is the substitution $\sigma|_X$ from $X$ to $\Delta_C \cup \Delta_N$ s.t. $\sigma'(x) = \sigma(x)$ for each $x \in X$. For an atom $a = p(t_1, \ldots, t_k)$, $\sigma(a)$ denotes the atom obtained from $a$ by replacing each variable $x$ of $a$ with $\sigma(x)$. For a structure $\varsigma$ containing atoms, we denote by $\sigma(\varsigma)$ the structure obtained by replacing each atom $a$ of $\varsigma$ with $\sigma(a)$.
Programs and Queries. A rule $r$ is a finite expression of the form:

$$\forall X \exists Y \text{ atom}_{[X \cup Y]} \leftarrow \text{conj}_{[X]}$$

where $X$ and $Y$ are disjoint sets of variables (next called $\forall$-variables and $\exists$-variables, respectively), and $X' \subseteq X$. In the following, $\text{head}(r) = \text{atom}_{[X' \cup Y]}$ and $\text{body}(r) = \text{atoms}((\text{conj}_{[X]}))$. If $\text{body}(r) = \emptyset$, then $r$ is also called fact. A $\text{Datalog}^3$ program $P$ is a finite set of rules. We denote by data($P$) all the atoms constituting the ground facts of $P$.

Example 1. Let $P$-Jungle be a $\text{Datalog}^3$ program including the following rules:

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

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 $\text{escapes}(\text{gazelle})$, $\text{fast}(\text{gazelle})$, $\text{prey}(\text{antelope})$, $\text{strongerThan}(\text{lion}, \text{antelope})$, and possibly also $\text{pursues}(\text{lion}, \text{gazelle})$. We will use $P$-Jungle as a running example. \hfill $\square$

Given a $\text{Datalog}^3$ program $P$, a conjunctive query (CQ) $q$ over $P$ is a first-order (FO) expression of the form $\exists Y \text{ conj}_{[X \cup Y]}$, where $X$ is its free variables. To highlight the free variables, we write $q(X)$ instead of $q$. Query $q$ is a Boolean CQ (BCQ) if $X = \emptyset$. Moreover, $q$ is called atomic if $\text{conj}$ is an atom.

Example 2. Animals pursued by a $\text{lion}$ and stronger than some other animal can be retrieved by means of a CQ $\exists Y \text{ pursues}(\text{lion}, X)$, $\text{strongerThan}(X,Y)$.

Semantics and Query Answering. Given a set $S$ of atoms and an atom $a$, we say $S$ entails $a$ ($S \models a$ for short) if there is a substitution $\sigma$ s.t. $\sigma(a) \in S$. Let $P \in \text{Datalog}^3$. A set $M \subseteq \text{base}(\Delta_C \cup \Delta_N)$ is a model for $P$ ($M \models P$) if $M \models \sigma|_X(\text{head}(r))$ for each $r \in P$ of the form (1) and substitution $\sigma$ s.t. $\sigma(\text{body}(r)) \subseteq M$. Let mods($P$) denote the set of models of $P$. Let $M \in \text{mods}(P)$. A BCQ $q$ is true w.r.t. $M$ ($M \models q$) if there is a substitution $\sigma$ s.t. $\sigma(\text{atoms}(q)) \subseteq M$. Analogously, the answer of a CQ $q(X)$ w.r.t. $M$ is the set $\text{ans}(q,M) = \{\sigma|_X : \sigma$ is a substitution $\land M \models \sigma|_X(q)\}$. The answer of a BCQ $q(X)$ w.r.t. a program $P$ is the set $\text{ans}_P(q) = \{\sigma : \sigma \in \text{ans}(q,M) \forall M \in \text{mods}(P)\}$. Note that for a BCQ $q$ either $\text{ans}_P(q) = \{\emptyset\}$ or $\text{ans}_P(q) = \emptyset$. 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 $\text{Datalog}^3$ 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 $\text{ans}_P(q)$ for a CQ $q(X)$ is Turing-reducible to QA. In fact, $\text{ans}_P(q)$ is the set of substitutions $\sigma$ s.t. the BCQ $\sigma(q)$ is true w.r.t. $P$. However, since $\sigma \in \text{ans}_P(q)$ implies $\sigma(X) \subseteq \text{dom}(P)$, only finitely many substitutions have to be considered [13].
Let $P$ be a $\mathsf{Datalog}^3$ 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 \textsc{chase} [17]. Intuitively, the \textsc{chase} first sets $U$ to 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 \textsc{chase} always constructs a (possibly infinite) universal model of $P$, QA remains undecidable in general [13].

## 3 Shy: A novel QA-decidable $\mathsf{Datalog}^3$ class

The key idea behind this class intuitively relays on the following shyness property: During a \textsc{chase}-run on a Shy program $P$, nulls propagated body-to-head do not meet each other to join. In this regard, given a $\mathsf{Datalog}^3$ program $P$, to detect whether a \textsc{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, $\varphi^r_X$ denotes the “representative” null introduced (during a \textsc{chase}-run on $P$) due to the $\exists$-variable $x$ of the rule $r \in P$. (If $(r, x) \neq (r', x')$, then $\varphi^r_X \neq \varphi^{r'}_{X'}$.) Let $a$ be an atom, and $X$ a variable occurring in $a$ at position $i$. The null-set of position $i$ in $a$ w.r.t. $P$, denoted by nullset($i$, $a$), is inductively defined as follows. If $a$ is the head atom of some rule $r \in P$, then nullset($i$, $a$) is: (1) either the set \{ $\varphi^r_X$ \} if $X$ is $\exists$-quantified in $r$; or (2) the intersection of every nullset($j$, $b$) s.t. $b \in$ body($r$) and $X$ occurs at position $j$ in $b$, if $X$ is $\forall$-quantified in $r$. If $a$ is not a head atom, then nullset($i$, $a$) is the union of nullset($i$, head($r$)) for each $r \in P$ s.t. pred(head($r$)) = pred($a$). A representative null $\varphi$ invades a variable $x$ that occurs at position $i$ in an atom $a$ if $\varphi$ is contained in nullset($i$, $a$). A variable $x$ occurring in a conjunction of atoms conj is attacked in conj by a null $\varphi$ if each occurrence of $x$ in conj is invaded by $\varphi$. A variable $x$ is protected in conj if it is attacked by no null. We are now ready to define the new $\mathsf{Datalog}^3$ class.

**Definition 1 ([19])**. Let Shy denote the class of all $\mathsf{Datalog}^3$ 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 body($r$); (ii) If two distinct $\forall$-variables are not protected in body($r$) but occur both in head($r$) and in two different body atoms, then they are not attacked by the same null. \hfill $\square$

Resuming program $P$-Jungle of Example 1 we now prove its shyness. Let $a_1, \ldots, a_{12}$ be the atoms of rules $r_1$–$r_4$ in left-to-right/top-to-bottom order, and nullset($1$, $a_1$) = $\{ \varphi^{r_1}_{Z_1} \}$. First, we propagate $\varphi^{r_1}_{Z_1}$ (head-to-body) to nullset($1$, $a_1$), nullset($1$, $a_7$), and nullset($1$, $a_{10}$). Next, this null is propagated (body-to-head) from $a_1$, $a_7$, and $a_3$ to nullset($1$, $a_3$), nullset($1$, $a_6$) and nullset($1$, $a_{11}$), respectively. Finally, we observe that rules $r_1$–$r_3$ are trivially shy, and that $r_4$ also is because variable $y$ is not invaded in $a_{12}$ even if $\varphi^{r_1}_{Z_1}$ invades $y$ both in $a_{10}$ and $a_{11}$.

According to Definition 1, we observe that a program is Shy regardless its ground facts. Finally, we mention notable computational properties of this class.
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 $\text{EXP}$-complete in combined complexity for atomic queries.

4 Implementation and Experiments

We implemented a system, called DLV$^3$, 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$^3$. 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$^3$.

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 $\text{lubm-10}$, $\text{lubm-30}$ and $\text{lubm-50}$. 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$^3$ 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>
<thead>
<tr>
<th></th>
<th>$q_1$</th>
<th>$q_2$</th>
<th>$q_3$</th>
<th>$q_4$</th>
<th>$q_5$</th>
<th>$q_6$</th>
<th>$q_7$</th>
<th>$q_8$</th>
<th>$q_9$</th>
<th>$q_{10}$</th>
<th>$q_{11}$</th>
<th>$q_{12}$</th>
<th>$q_{13}$</th>
<th>$q_{14}$</th>
<th>$q_{all}$</th>
<th>G.Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{lubm-10}$</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td>5</td>
<td>&lt;1</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>17</td>
<td>2.87</td>
</tr>
<tr>
<td>DLV$^3$</td>
<td>82</td>
<td>84</td>
<td>84</td>
<td>82</td>
<td>80</td>
<td>88</td>
<td>81</td>
<td>89</td>
<td>95</td>
<td>82</td>
<td>82</td>
<td>89</td>
<td>82</td>
<td>84</td>
<td>27</td>
<td>84.48</td>
</tr>
<tr>
<td>Pellet</td>
<td>33</td>
<td>–</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>4909</td>
<td>70</td>
<td>–</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>53.31</td>
<td></td>
</tr>
<tr>
<td>OWLIM-Lite</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OWLIM-SE</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{lubm-30}$</td>
<td>16</td>
<td>13</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>3</td>
<td>21</td>
<td>12</td>
<td>25</td>
<td>18</td>
<td>&lt;1</td>
<td>5</td>
<td>23</td>
<td>8</td>
<td>55</td>
<td>9.70</td>
</tr>
<tr>
<td>DLV$^3$</td>
<td>82</td>
<td>84</td>
<td>84</td>
<td>82</td>
<td>80</td>
<td>88</td>
<td>81</td>
<td>89</td>
<td>95</td>
<td>82</td>
<td>82</td>
<td>89</td>
<td>82</td>
<td>84</td>
<td>27</td>
<td>84.48</td>
</tr>
<tr>
<td>Pellet</td>
<td>33</td>
<td>–</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>4909</td>
<td>70</td>
<td>–</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>53.31</td>
<td></td>
</tr>
<tr>
<td>OWLIM-Lite</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>123.18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OWLIM-SE</td>
<td>323</td>
<td>328</td>
<td>323</td>
<td>323</td>
<td>323</td>
<td>323</td>
<td>323</td>
<td>323</td>
<td>323</td>
<td>323</td>
<td>323</td>
<td>323</td>
<td>323</td>
<td>323.57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{lubm-50}$</td>
<td>27</td>
<td>23</td>
<td>12</td>
<td>23</td>
<td>35</td>
<td>6</td>
<td>34</td>
<td>22</td>
<td>42</td>
<td>31</td>
<td>&lt;1</td>
<td>9</td>
<td>33</td>
<td>14</td>
<td>93</td>
<td>16.67</td>
</tr>
<tr>
<td>DLV$^3$</td>
<td>82</td>
<td>84</td>
<td>84</td>
<td>82</td>
<td>80</td>
<td>88</td>
<td>81</td>
<td>89</td>
<td>95</td>
<td>82</td>
<td>82</td>
<td>89</td>
<td>82</td>
<td>84</td>
<td>27</td>
<td>84.48</td>
</tr>
<tr>
<td>Pellet</td>
<td>33</td>
<td>–</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>4909</td>
<td>70</td>
<td>–</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>53.31</td>
<td></td>
</tr>
<tr>
<td>OWLIM-Lite</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OWLIM-SE</td>
<td>536</td>
<td>547</td>
<td>536</td>
<td>536</td>
<td>536</td>
<td>536</td>
<td>536</td>
<td>536</td>
<td>536</td>
<td>536</td>
<td>536</td>
<td>536</td>
<td>536</td>
<td>123.72</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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\(^3\) 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 \(\text{lubm-10}\), but it answered no tested queries in the allotted time on \(\text{lubm-30}\) and \(\text{lubm-50}\). 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\(^3\) is specifically designed for QA over frequently changing ontologies, it outperformed the competitors also in performing fact inference. Indeed, DLV\(^3\) 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\(^3\)* classes relying 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 \(C_1\) and \(C_2\) of classes, there is a direct path from \(C_1\) to \(C_2\) iff \(C_1 \subset C_2\); also, \(C_1\) and \(C_2\) are not linked by any directed path iff they are incomparable \[19\]. 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,
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 $\exists$-quantifiers in Datalog is Nyaya [12], which performs QA over Linear-Datalog$^3$. (We could not compare DLV$^3$ 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 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$^3$ is the first system supporting the standard FO semantics for unrestricted CQs with $\exists$-variables over ontologies using advanced properties (even beyond AC$^0$), such as, role transitivity, role hierarchy, role inverse, and concept products. The experiments confirm the efficiency of DLV$^3$, which constitutes a powerful system for a fully-declarative QA.

References