University of Huddersfield Repository

Alviano, Mario, Faber, Wolfgang, Leone, Nicola and Manna, Marco

Query answering over disjunctive datalog with existential quantifiers

Original Citation

Alviano, Mario, Faber, Wolfgang, Leone, Nicola and Manna, Marco (2013) Query answering over disjunctive datalog with existential quantifiers. In: 21st Italian Symposium on Advanced Database Systems, 30 June - 3 July 2013, Roccella Jonica, Italy.

This version is available at http://eprints.hud.ac.uk/21033/

Query Answering over Disjunctive Datalog with Existential Quantifiers

Mario Alviano and Wolfgang Faber and Nicola Leone and Marco Manna

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

Abstract. The paper discusses the impact of adding existential quantification in the head of positive disjunctive Datalog rules. After introducing syntax and semantics of the resulting language, we provide a notion of instantiation, which has been proven to be adequate for query answering purposes. Although on the one hand this new formalism is attractive for knowledge management, especially in domains where ontology-based reasoning is needed, on the other hand the presence of existential quantifiers does not preserve decidability of the typical reasoning tasks even if disjunction is forbidden. Therefore, we consider many decidable fragments of the language that naturally extend analogous classes defined in the non-disjunctive case, and we give a complete picture of the complexity of answering conjunctive queries in this setting.

1 Introduction

Datalog based languages have their origins as query languages in Database Systems, and are now used in a variety of applications. For example, disjunctive Datalog has been widely applied in data-integration [25], semantic information extraction [24], workforce management [26], ontological reasoning [21] and other real world scenarios in Artificial Intelligence [18, 19]. In particular, the field of ontology-based Query Answering (QA) is thriving in data and knowledge management [7, 9, 11], and companies such as Oracle are adding ontological reasoning modules on top of their existing software. In this context, queries are not merely evaluated on an extensional relational database D, but over a logical theory combining D with an *ontological theory* Σ describing rules for inferring intensional knowledge from D [22]. Thus, for a Boolean conjunctive query q, it is not only checked whether D entails q, but rather whether $D \cup \Sigma$ does.

A key issue in ontology-based QA is the design of the language used for specifying the ontological theory Σ. To this end, Datalog$^{\pm}$ [7] generalizes well-known ontology specification languages by extending Datalog with existential quantifiers in rule heads. Complexity of QA has been analyzed for many fragments of Datalog with existential quantifiers [6, 8, 14, 20, 23]. The present paper is related to this line of research and reports results presented in [2] concerning QA over Datalog$^{\exists,\vee}$, an extension of Datalog that allows for disjunctions and existential quantifiers in rule heads. More specifically, the chase procedure [22], universal models [14], and some decidability and complexity results [7] are lifted from the non-disjunctive case to Datalog$^{\exists,\vee}$.

The discussed language combines the capability of disjunction to deal with incomplete information, with the power of existential quantification to deal with unnamed individuals. More specifically, it allows one to naturally encode advanced ontology properties such as role transitivity, role hierarchy, role inverse, concept products and union of concepts. For example, consider a scenario where each animal is either a carnivore or a herbivore, and any carnivore preys on at least one other animal. This knowledge can be modeled by the following Datalog$^{\exists,\vee}$ rules (in the left-hand side) or in equivalent ontological terms (in the right-hand side):

```
carnivore(X) v herbivore(X) ← animal(X)    Animal ⊑ Carnivore ⊔ Herbivore
∃Y preys(X,Y) ← carnivore(X)               Carnivore ⊑ ∃preys.⊤
animal(Y) ← preys(X,Y)                     ∃preys⁻.⊤ ⊑ Animal
```

2 Syntax and Semantics

In the following, 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 domains; by φ, a null; 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 $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$, *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 an atom $\mathbf{a}$, we denote by $\mathsf{pred}(\mathbf{a})$ the predicate symbol of $\mathbf{a}$. For any structure ς over atoms (e.g., a conjunction of atoms), $\mathsf{atoms}(\varsigma)$ denotes the set of atoms in ς, and $\mathsf{terms}(\varsigma)$ denotes the set of terms occurring in $\mathsf{atoms}(\varsigma)$. If $\mathbf{X}$ are the variables of ς, then ς is also denoted by $\varsigma_{[\mathbf{X}]}$. A structure $\varsigma_{[\emptyset]}$ is called *ground*. Given a nonempty set T of terms, $\mathsf{base}(T)$ denotes the set of all atoms that can be formed with predicate symbols in Π and terms from T. A *mapping* is a function $\mu : \Delta \to \Delta$ s.t. $c \in \Delta_C$ implies $\mu(c) = c$, and $\varphi \in \Delta_N$ implies $\mu(\varphi) \in \Delta_C \cup \Delta_N$. Let T be a subset of Δ. The application of μ to T, denoted by $\mu(T)$, is the set $\{\mu(t) \mid t \in T\}$. The restriction of μ to T, denoted by $\mu|_T$, is the mapping μ' s.t. $\mu'(t) = \mu(t)$ for each $t \in T$, and $\mu'(t) = t$ for each $t \notin T$. In this case, we also say that μ is an *extension* of μ', denoted by $\mu \supseteq \mu'$. For an atom $\mathbf{a} = \mathtt{p}(t_1,\ldots,t_k)$, we denote by $\mu(\mathbf{a})$ the atom $\mathtt{p}(\mu(t_1),\ldots,\mu(t_k))$. For a structure ς over atoms, we denote by $\mu(\varsigma)$ the structure obtained by replacing each atom $\mathbf{a}$ of ς with $\mu(\mathbf{a})$. Let ς_1 and ς_2 be two structures over atoms. A *homomorphism* from ς_1 to ς_2 is a mapping h s.t. $h(\varsigma_1)$ is a substructure of ς_2 (for example, if ς_1 and ς_2 are sets of atoms, then $h(\varsigma_1) \subseteq \varsigma_2$). A *substitution* is a mapping σ s.t. $t \in \Delta_N$ implies $\sigma(t) = t$, and $t \in \Delta_V$ implies $\sigma(t) \in \Delta_C \cup \Delta_N \cup \{t\}$.

A Datalog$^{\exists,\vee}$ *rule* r is an expression of the form $\forall\mathbf{X}\exists\mathbf{Y}\ \mathbf{disj}_{[\mathbf{X}'\cup\mathbf{Y}]} \leftarrow \mathbf{conj}_{[\mathbf{X}]}$, where (i) $\mathbf{X}$ and $\mathbf{Y}$ are disjoint sets of variables; (ii) $\mathbf{X}' \subseteq \mathbf{X}$; (iii) $\mathbf{disj}$ is a nonempty disjunction of atoms; and (iv) $\mathbf{conj}$ is a conjunction of atoms. If $\mathbf{Y}$ is empty, then r coincides with a standard Datalog$^{\vee}$ rule. The sets $\mathsf{atoms}(\mathbf{disj})$ and $\mathsf{atoms}(\mathbf{conj})$ are denoted by $\mathsf{head}(r)$ and $\mathsf{body}(r)$, respectively. If $\mathsf{body}(r)$ is the empty set and $\mathsf{head}(r)$ is a singleton, then r is also referred to as a *fact.* A fact that contains no $\exists$-variable is called *ground.* A Datalog$^{\exists,\vee}$ program P is a set of Datalog$^{\exists,\vee}$ rules. Finally, we denote $\bigcup_{r \in P} \mathsf{head}(r)$ by $\mathsf{heads}(P)$. A *conjunctive query* (CQ) $q(\mathbf{X})$ is of the form $\exists \mathbf{Y}\ \mathbf{conj}_{[\mathbf{X} \cup \mathbf{Y}]}$, where $\mathbf{X}$ and $\mathbf{Y}$ are disjoint sets of variables, and $\mathbf{conj}_{[\mathbf{X} \cup \mathbf{Y}]}$ is a conjunction of atoms from $\mathsf{base}(\mathbf{X} \cup \mathbf{Y} \cup \Delta_C)$. Variables in $\mathbf{X}$ are called *free variables.* Query q is called *acyclic* (ACQ, for short) if its associated hypergraph is acyclic [12] or, equivalently, if it has hypertree-width 1 [15]. A *Boolean CQ* (BCQ) is a query where $\mathbf{X}$ is empty. An *atomic query* is a CQ such that $\mathbf{conj}$ consists of exactly one atom.

Given a rule r, a set $M \subseteq \mathsf{base}(\Delta_C \cup \Delta_N)$ is a *model* of r, denoted by $M \models r$, if for each substitution σ s.t. $\sigma(\mathsf{body}(r)) \subseteq M$, there is a substitution $\sigma' \supseteq \sigma|_{\mathbf{X}}$ s.t. $\sigma'(\mathsf{head}(r)) \cap M \neq \emptyset$. M is a model of a Datalog$^{\exists,\vee}$ program P, denoted by $M \models P$, if $M \models r$ for each $r \in P$. Let $\mathsf{mods}(P)$ denote the set of all the models of P. Two programs P and P' are called first-order equivalent (FO-equivalent, for short) if $\mathsf{mods}(P) = \mathsf{mods}(P')$. A BCQ q is *true* w.r.t. a model M, denoted by $M \models q$, if there is a substitution σ s.t. $\sigma(\mathsf{atoms}(q)) \subseteq M$. For a set of models $\mathcal{M}$, q is true w.r.t. $\mathcal{M}$, denoted by $\mathcal{M} \models q$, if $M \models q$ for each $M \in \mathcal{M}$. For a program P, q is true w.r.t. P, denoted by $P \models q$, if $\mathsf{mods}(P) \models q$. The answer of a CQ $q(\mathbf{X})$ w.r.t. a set of models $\mathcal{M}$, denoted by $\mathsf{ans}(q, \mathcal{M})$, is the set of substitutions $\sigma|_{\mathbf{X}}$ s.t. $M \models \sigma|_{\mathbf{X}}(q)$ for each $M \in \mathcal{M}$. The answer of $q(\mathbf{X})$ w.r.t. a program P, denoted by $\mathsf{ans}_P(q)$, is the set $\mathsf{ans}(q, \mathsf{mods}(P))$. Note that for a BCQ q, either $\mathsf{ans}_P(q) = \emptyset$ (if $P \not\models q$) or $\mathsf{ans}_P(q) = \{\sigma|_{\emptyset}\}$ (if $P \models q$; $\sigma|_{\emptyset}$ is the identity mapping). The same consideration also applies to $\mathsf{ans}(q, \mathcal{M})$.

Let $\mathcal{C}$ be a class of Datalog$^{\exists,\vee}$ programs whose terms belong to $\Delta_C \cup \Delta_V$. In this paper we call *query answering* (QA) over $\mathcal{C}$ the following decision problem:

Given a program $P \in \mathcal{C}$ and a BCQ q, determine whether $P \models q$ holds.

In the following, $\mathcal{C}$ 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 as defined above. In fact, $\mathsf{ans}_P(q)$ is defined as the set of substitutions $\sigma|_{\mathbf{X}}$ s.t. the BCQ $\sigma|_{\mathbf{X}}(q)$ is true w.r.t. P. Since $\sigma|_{\mathbf{X}} \in \mathsf{ans}_P(q)$ implies $\sigma|_{\mathbf{X}}(\Delta_V) \subseteq \mathsf{terms}(P) \cap \Delta_C$, only finitely many substitutions have to be considered.

3 Universal Model Sets and Instantiation

Let $P \in$ Datalog$^{\exists,\vee}$. A set $\mathcal{M} \subseteq \mathsf{mods}(P)$ is a *universal model set* (*UMS*) for P if for each $M \in \mathsf{mods}(P)$ there is $M' \in \mathcal{M}$ and a homomorphism h s.t. $h(M') \subseteq M$. Universal model sets are sufficient for QA over Datalog$^{\exists,\vee}$ programs, and generalize the the notion of *universal model*, which is widely used in the context of QA over Datalog$^{\exists}$ programs. In fact, if $\mathcal{M}$ is a UMS for P, then $P \models q$ iff $\mathcal{M} \models q$ for each BCQ q. We now design a strategy for identifying a UMS for a Datalog$^{\exists,\vee}$ program P. First, we introduce the notion of *fires* of a rule $r \in P$ on a set R of Datalog$^{\exists,\vee}$ ground rules. Next, we define an *instantiation procedure* for computing a ground program $\mathsf{inst}(P)$, the models of which form a UMS for P. Let r be a rule of the form $\forall \mathbf{X} \exists \mathbf{Y}\ \ \mathbf{disj}_{[\mathbf{X}' \cup \mathbf{Y}]} \leftarrow \mathbf{conj}_{[\mathbf{X}]}$, and R, R'

be sets of ground rules. A *firing* substitution for r w.r.t. R is a substitution σ s.t. $\sigma = \sigma|_{\mathbf{X}}$ and $\sigma(\mathsf{body}(r)) \subseteq \mathsf{heads}(R)$. The *firing* of r on R' w.r.t. σ yields a ground rule $\hat{\sigma}(r)$, where $\hat{\sigma}$ is obtained by extending $\sigma|_{\mathbf{X}}$ as follows: $\exists$-variables in $\mathbf{Y}$ are assigned to the least $|\mathbf{Y}|$ nulls not occurring in $R \cup R'$. A firing substitution for a rule r is said to be *spent* if it has already been fired. The procedure that computes $\mathsf{inst}(P)$ consists of a fair exhaustive series of fires yielding a (possibly infinite) ground program $\mathsf{inst}(P)$.

Example 1. The instantiation of the program $\{\texttt{c(X) v h(X)} \leftarrow \texttt{a(X)}; \exists\texttt{Y p(X,Y)} \leftarrow \texttt{c(X)}; \texttt{a(Y)} \leftarrow \texttt{p(X,Y)}; \texttt{c(lion)} \leftarrow\}$ produces the following rules (in the given order): $\texttt{c(lion)}$, $\texttt{p(lion,}\varphi_1\texttt{)} \leftarrow \texttt{c(lion)}$; $\texttt{a(}\varphi_i\texttt{)} \leftarrow \texttt{p(lion,}\varphi_i\texttt{)}$; $\texttt{c(}\varphi_i\texttt{) v h(}\varphi_i\texttt{)} \leftarrow \texttt{a(}\varphi_i\texttt{)}$; $\texttt{p(}\varphi_i\texttt{,}\varphi_{i+1}\texttt{)} \leftarrow \texttt{c(}\varphi_i\texttt{)}$ $(i \geq 1)$. The (subset-minimal) models of $\mathsf{inst}(P)$ are as follows: $\bigcup_{i\in[1..k]}\{\texttt{c(}\varphi_i\texttt{), p(}\varphi_i\texttt{,}\varphi_{i+1}\texttt{), a(}\varphi_{i+1}\texttt{)}\} \cup I \cup \{\texttt{h(}\varphi_{k+1}\texttt{)}\}, \forall k \geq 1$ and $\bigcup_{i\geq 1}\{\texttt{c(}\varphi_i\texttt{), p(}\varphi_i\texttt{,}\varphi_{i+1}\texttt{), a(}\varphi_{i+1}\texttt{)}\} \cup I$, where $I = \{\texttt{c(lion), p(lion,}\varphi_1\texttt{), a(}\varphi_1\texttt{)}\}$.

The following result, by pointing out some relationships between the models of P and those of $\mathsf{inst}(P)$, states that $\mathsf{mods}(\mathsf{inst}(P))$ is a UMS for P [2].

Theorem 1. *Consider a Datalog$^{\exists,\vee}$ program P and its instantiation P'. It holds that $\mathcal{M} = \{M \in \mathsf{mods}(P') \mid M \subseteq \mathsf{heads}(P')\}$ is a UMS for P.*

The instantiation procedure is a generalization of the oblivious chase procedure [22] which associates every Datalog$^{\exists}$ program with a universal model. In fact, given a Datalog$^{\exists}$ program P, $\{\mathsf{heads}(\mathsf{inst}(P))\}$ is universal for P [2].

4 QA-decidable fragments of Datalog$^{\exists,\vee}$

This section considers some Datalog$^{\exists,\vee}$ subclasses which rely on a well known paradigm called *guardedness* [6]. A Datalog$^{\exists,\vee}$ rule r is said to be *guarded* if its body contains at least one atom, the *guard* of r, that covers all the universally quantified variables of r. All other body atoms of r are called *side* atoms. For example, the rule $\texttt{p(Y) v q(X)} \leftarrow \texttt{r(X,Y), s(X)}$ is guarded, while the rule $\texttt{p(Y) v q(X)} \leftarrow \texttt{r(X,Y), s(X,Z)}$ is not. Moreover, a guarded rule is called *multi-linear* if each body atom can play the role of guard, *linear* if it contains only one body atom, and *monadic-linear* if it is linear and all of its head predicates are unary. Hereafter, a Datalog$^{\exists,\vee}$ program P is called Guarded (resp., Multi-Linear, Linear, Monadic-Linear) if each rule $r \in P$ either is guarded (resp., multi-linear, linear, monadic-linear) or has an empty body.

The Guarded-Datalog$^{\exists,\vee}$ class can be further extended by relaxing the notion of guard. To this end, we generalize the definition of affected positions of an atom already introduced for Weakly-Guarded-Datalog$^{\exists}$ programs [6]. Intuitively, these are the only positions where nulls might occur in the instantiation of a program. Let P be a Datalog$^{\exists,\vee}$ program, $\mathbf{a}$ be an atom, and $\texttt{X}$ a variable occurring in $\mathbf{a}$ at position i. Position i of $\mathbf{a}$ is (inductively) marked as *affected* w.r.t. P if there is a rule $r \in P$ with an atom $\mathbf{b} \in \mathsf{head}(r)$ s.t. $\mathsf{pred}(\mathbf{b}) = \mathsf{pred}(\mathbf{a})$ and $\texttt{X}$ is either an $\exists$-variable, or a $\forall$-variable s.t. $\texttt{X}$ occurs in $\mathsf{body}(r)$ in affected positions only. For example, consider the program $P_w = \{\exists \mathtt{Y}_1\ \mathtt{g(Y_1,X_1)} \leftarrow \mathtt{p(X_1)};\ \exists \mathtt{Y}_2\ \mathtt{s(Y_2,X_2)} \leftarrow \mathtt{u(X_2)};\ \mathtt{g(X_3,Y_3)} \leftarrow \mathtt{s(Y_3,X_3)};\ \mathtt{t(Y_4)} \leftarrow \mathtt{g(X_4,Y_4)},\ \mathtt{s(Y_4,Z_4)}\}$. One can verify that the first position of $\mathtt{g(Y_1,X_1)}$ is affected as well as the first position of $\mathtt{s(Y_2,X_2)}$ and the second position of $\mathtt{g(X_3,Y_3)}$. Also, all the positions of $\mathtt{g(X_4,Y_4)}$ are affected as well as the first position of $\mathtt{s(Y_4,Z_4)}$ and of $\mathtt{t(Y_4)}$. We can now define the class of weakly-guarded programs.

A rule r of a Datalog$^{\exists,\vee}$ program P is said to be *weakly-guarded* w.r.t. P if its body contains at least one atom, the *weak-guard* of r, that covers all the universally quantified variables of r that appear in affected positions only. By Weakly-Guarded-Datalog$^{\exists,\vee}$, we denote the set of Datalog$^{\exists,\vee}$ programs where each rule either is weakly-guarded or has an empty body. For example, program P_w is Weakly-Guarded since $\mathtt{g(X_4,Y_4)}$ is the weak-guard of the last rule.

The above definitions generalize Guarded-Datalog$^{\exists}$, Linear-Datalog$^{\exists}$, and Weakly-Guarded-Datalog$^{\exists}$ [6]. Checking whether a program belongs to the class Guarded-Datalog$^{\exists,\vee}$, Linear-Datalog$^{\exists,\vee}$ or Weakly-Guarded-Datalog$^{\exists,\vee}$ is doable in polynomial time. Decidability of QA over these classes can be proven by drawing from recent results [5] established on the *guarded fragment of first-order logic* [17], here denoted by Guarded-FOL. To this end, we first introduce the notion of *weak instantiation* for compiling a Weakly-Guarded-Datalog$^{\exists,\vee}$ program P into a FO-equivalent Guarded-Datalog$^{\exists,\vee}$ program $\mathsf{winst}(P)$. For each $r \in P$, let $\mathsf{winst}(r)$ denote the set of partially ground rules associated to r and consisting of the set $\{r\}$ if r has an empty body, or of the set $\{\sigma(r) \mid \sigma$ is a substitution from $\mathbf{X}$ to $\mathsf{terms}(P) \cap \Delta_C\}$ if r is weakly-guarded and $\mathbf{X}$ are its body variables that are not affected. Then, $\mathsf{winst}(P) = \bigcup_{r \in P} \mathsf{winst}(r)$.

Decidability of QA over Guarded-Datalog$^{\exists,\vee}$ follows by a compilation into FO-equivalent Guarded-FOL formulas, for which QA is know to be decidable [5]. Decidability extends also to Weakly-Guarded-Datalog$^{\exists,\vee}$ by means of the weakly instantiation [2].

Theorem 2. *Weakly-Guarded-Datalog$^{\exists,\vee}$ is QA-decidable.*

We now consider the data complexity of QA over the previously introduced Datalog$^{\exists,\vee}$ fragments, by also varying the structure of the query. As usual, we assume that a Datalog$^{\exists,\vee}$ program P is paired with a database $D \subset \mathsf{base}(\Delta_C)$. The set of ground facts $\{\mathbf{a} \leftarrow\ \mid \mathbf{a} \in D\}$ is denoted by $\overleftarrow{D}$. We start by considering the class Guarded-Datalog$^{\exists,\vee}$.

Theorem 3. *The data complexity of QA over Guarded-Datalog$^{\exists,\vee}$ programs is **coNP**-complete, and it is **coNP**-hard already both in case of (1) an acyclic CQ over a Monadic-Linear-Datalog$^{\vee}$ program, and (2) an atomic query over a Multi-Linear-Datalog$^{\vee}$ program.*

Proof. The membership follows from statement 5 of Theorem 19 in [5] (considering the complexity of QA over Guarded-FOL), and since there exists a logspace transducer that associates each Guarded-Datalog$^{\exists,\vee}$ program with a FO-equivalent Guarded-FOL formula. The fist hardness can be established by considering a database and an acyclic CQ involving only unary and binary atoms, and a single (nonrecursive) Monadic-Linear-Datalog$^{\vee}$ rule containing two head atoms. This result is implicit in the proof of Theorem 6.4 of [10]. The second hardness can be obtained by encoding the **coNP**-complete problem 3-UNSAT by means of an atomic query over a Multi-Linear-Datalog$^{\vee}$ program [2]. □

As in the non-disjunctive case, the complexity of answering conjunctive queries over Weakly-Guarded-Datalog$^{\exists,\vee}$ is harder that the one over Guarded-Datalog$^{\exists}$.

Theorem 4. *The data complexity of QA over Weakly-Guarded-Datalog*$^{\exists,\vee}$ *is* ***EXP****-complete, and already* ***EXP****-hard for atomic queries and no disjunction.*

Proof. Hardness comes from Weakly-Guarded-Datalog$^{\exists}$ [6]. For the membership, let P be a Weakly-Guarded-Datalog$^{\exists,\vee}$ program and $P' = \mathsf{winst}(P \cup \overleftarrow{D})$ be its compilation into Guarded-Datalog$^{\exists,\vee}$. The result follows by considering that: $P \cup \overleftarrow{D} \models q$ iff $P' \models q$, the size of P' is polynomial (in data complexity) in the cardinality of D, P' can be translated in logarithmic space into a FO-equivalent Guarded-FOL formula, and QA over a Guarded-FOL formula is in **EXP** if the formula and the query are considered fixed [5]. □

We now consider a disjunctive case guaranteeing tractability of QA. The result needs a technical construction designed in [2] and improved in [16].

Theorem 5. *Data complexity of atomic QA over Linear-Datalog*$^{\exists,\vee}$ *is in* $\mathbf{AC}_0$.

5 Discussion

Table 1 provides an overview of complexity results that follow from the results obtained in this section and in the literature. Each row reports the complexity of QA for each of the classes defined in Section 4 together with either atomic queries (AQ), acyclic conjunctive queries (ACQ) or conjunctive queries (CQ). In each row we differentiate between the presence or absence of existential variables and disjunction: $\exists$-variables in rule heads (column $\{\exists\}$), disjunctive heads (column $\{\vee\}$), and both (column $\{\exists,\vee\}$). Results in the $\{\exists\}$-column are from [6, 7], results for Weakly-Guarded-Datalog$^{\vee}$ (last cell in column $\{\vee\}$) follow from [13], since this class coincides with Datalog$^{\vee}$. All the remaining **coNP**-completeness results

Table 1. Data complexity of QA in Datalog$^{\exists,\vee}$.

Datalog Restrictions	**Query Structure**	**Datalog Extensions**		
		$\{\exists\}$	$\{\vee\}$	$\{\exists,\vee\}$
(Monadic-)Linear	AQ	in $\mathbf{AC}_0$	in $\mathbf{AC}_0$	in $\mathbf{AC}_0$
	ACQ/CQ	in $\mathbf{AC}_0$	**coNP**-complete	**coNP**-complete
Multi-Linear	AQ/ACQ/CQ	in $\mathbf{AC}_0$	**coNP**-complete	**coNP**-complete
Guarded	AQ/ACQ/CQ	**P**-complete	**coNP**-complete	**coNP**-complete
Weakly-Guarded	AQ/ACQ/CQ	**EXP**-complete	**coNP**-complete	**EXP**-complete

follow from Theorem 3, the remaining **EXP**-completeness results follow from Theorem 4, and the $\mathbf{AC}_0$ upper bounds follow from Theorem 5.

Let us first consider the impact of allowing disjunction in the presence of existential quantifiers in rule heads, i.e. columns $\{\exists\}$ versus $\{\exists, \vee\}$. We can see that in most considered cases, the problem becomes (potentially) harder, except for the class Weakly-Guarded. Indeed, for this case the problem is provably intractable already without disjunctions, and turns out to remain so when including them. In most other cases, we actually identify a tractability boundary, passing from $\mathbf{AC}_0$ to **coNP**-completeness. Notable exceptions are Monadic-Linear and Linear with atomic queries, in which case the problem remains tractable (but may be slightly more complex). It is interesting to observe that in the presence of disjunction the nature of the query has a huge impact on complexity for classes Monadic-Linear and Linear, while this is not the case in the absence of disjunction.

Let us now discuss the impact of adding existential quantification in the presence of disjunction in rule heads, i.e. columns $\{\vee\}$ versus $\{\exists, \vee\}$. We can see that in all considered classes except for Weakly-Guarded, adding existential quantifiers does not alter complexity. This is a notable result, since having existential quantification is a powerful construct for knowledge representation. Only for Weakly-Guarded we obtain a significant rise from **coNP**-completeness to **EXP**-completeness and thus provable intractability.

Concerning expressivity, we point out that Guarded-Datalog$^{\exists,\vee}$ is strictly more expressive than $\mathcal{ELU}$ [4], the extension of the well-known Description Logic $\mathcal{EL}$ that allows for union of concepts. Actually, Guarded-Datalog$^{\exists,\vee}$ even generalizes the $\mathcal{ELU}$ extension which is enhanced by role inclusions and inverse roles.

In future work, we intend to study the impact of disjunction on other tractable fragments of Datalog$^{\exists,\vee}$ based on different paradigms, for example *stickiness* [8], *shyness* [23] and *weak-acyclicity* [14]. Moreover, it would also be interesting to broaden the study to combined complexity or to limit it to fixed or bounded predicate arities. Finally, also investigating on implementation issues is on our agenda. In fact, the extension of DLV [3] handling Datalog$^{\exists}$ is based on decidability results on disjunctive Datalog with uninterpreted function symbols [1].

References

1. M. Alviano, W. Faber, and N. Leone. Disjunctive ASP with functions: Decidable queries and effective computation. *Theory and Practice of Logic Programming*, 10(4–6):497–512, July 2010.
2. M. Alviano, W. Faber, N. Leone, and M. Manna. Disjunctive datalog with existential quantifiers: Semantics, decidability, and complexity issues. *Theory and Practice of Logic Programming*, 12(4-5):701–718, 2012.
3. M. Alviano, W. Faber, N. Leone, S. Perri, G. Pfeifer, and G. Terracina. The disjunctive datalog system DLV. In *Datalog 2.0*, volume 6702 of *LNCS*, pages 282–301. Springer, 2011.
4. F. Baader, S. Brandt, and C. Lutz. Pushing the EL envelope. In *Proc. of IJCAI*, pages 364–369, 2005.
5. V. Barany, G. Gottlob, and M. Otto. Querying the Guarded Fragment. In *Proc. of LICS*, pages 1–10, 2010.

6. A. Calì, G. Gottlob, and M. Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. In *Proc. of KR*, pages 70–80, 2008.
7. A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *Proc. of PODS*, pages 77–86, 2009.
8. A. Calì, G. Gottlob, and A. Pieris. Advanced Processing for Ontological Queries. *PVLDB*, 3(1):554–565, 2010.
9. A. Calì, G. Gottlob, and A. Pieris. New Expressive Languages for Ontological Query Answering. In *Proc. of AAAI*, pages 1541–1546, 2011.
10. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, and R. Rosati. Ontologies and Databases: The *DL-Lite* Approach. In *Reasoning Web*, volume 5689 of *LNCS*, pages 255–356. Springer, 2009.
11. 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.
12. C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *Theoretical Computer Science*, 239(2):211–229, 2000.
13. T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.
14. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.
15. G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. In *Proc. of PODS*, pages 21–32, 1999.
16. G. Gottlob, M. Manna, M. Morak, and A. Pieris. On the complexity of ontological reasoning under disjunctive existential rules. In *Proc. of MFCS*, volume 7464 of *LNCS*, pages 1–18. Springer, 2012.
17. E. Grädel. On the Restraining Power of Guards. *The Journal of Symbolic Logic*, 64(4):1719–1742, 1999.
18. G. Grasso, S. Iiritano, N. Leone, and F. Ricca. Some dlv applications for knowledge management. In *LPNMR*, volume 5753 of *LNCS*, pages 591–597. Springer, 2009.
19. G. Grasso, N. Leone, M. Manna, and F. Ricca. Asp at work: Spin-off and applications of the dlv system. In *LPNMR*, volume 6565 of *LNCS*, pages 432–451. Springer, 2011.
20. S. Greco, F. Spezzano, and I. Trubitsyna. Stratification Criteria and Rewriting Techniques for Checking Chase Termination. *PVLDB*, 4(11):1158–1168, 2011.
21. U. Hustadt, B. Motik, and U. Sattler. Reducing SHIQ- Descrption Logic to Disjunctive Datalog Programs. In *Proc. of KR*, pages 152–162, 2004.
22. D. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences*, 28(1):167–189, 1984.
23. N. Leone, M. Manna, G. Terracina, and P. Veltri. Efficiently Computable Datalog$^\exists$ Programs. In *Proc. of KR*, pages 13–23, 2012.
24. M. Manna, E. Oro, M. Ruffolo, M. Alviano, and N. Leone. The hilex system for semantic information extraction. *T. Large-Scale Data- and Knowledge-Centered Systems*, 5:91–125, 2012.
25. M. Manna, F. Ricca, and G. Terracina. Consistent query answering via asp from different perspectives: Theory and practice. *Theory and Practice of Logic Programming*, 13(2):227–252, 2013.
26. F. Ricca, G. Grasso, M. Alviano, M. Manna, V. Lio, S. Iiritano, and N. Leone. Team-building with answer set programming in the gioia-tauro seaport. *Theory and Practice of Logic Programming*, 12(3):361–381, 2012.