TLP **12** (4–5): 701–718, 2012. © Cambridge University Press 2012
doi:10.1017/S1471068412000257

Disjunctive datalog with existential quantifiers: Semantics, decidability, and complexity issues

MARIO ALVIANO, WOLFGANG FABER, NICOLA LEONE
and MARCO MANNA*

Department of Mathematics, University of Calabria, Italy
(*e-mail:* {alviano,faber,leone,manna}@mat.unical.it)

Abstract

Datalog is one of the best-known rule-based languages, and extensions of it are used in a wide context of applications. An important *Datalog* extension is Disjunctive *Datalog*, which significantly increases the expressivity of the basic language. Disjunctive *Datalog* is useful in a wide range of applications, ranging from Databases (e.g., Data Integration) to Artificial Intelligence (e.g., diagnosis and planning under incomplete knowledge). However, in recent years an important shortcoming of *Datalog*-based languages became evident, e.g. in the context of data-integration (consistent query-answering, ontology-based data access) and Semantic Web applications: The language does not permit any generation of and reasoning with unnamed individuals in an obvious way. In general, it is weak in supporting many cases of existential quantification. To overcome this problem, $Datalog^{\exists}$ has recently been proposed, which extends traditional *Datalog* by existential quantification in rule heads. In this work, we propose a natural extension of Disjunctive *Datalog* and $Datalog^{\exists}$, called $Datalog^{\exists,\vee}$, which allows both disjunctions and existential quantification in rule heads and is therefore an attractive language for knowledge representation and reasoning, especially in domains where ontology-based reasoning is needed. We formally define syntax and semantics of the language $Datalog^{\exists,\vee}$, and provide a notion of instantiation, which we prove to be adequate for $Datalog^{\exists,\vee}$. A main issue of $Datalog^{\exists}$ and hence also of $Datalog^{\exists,\vee}$ is that decidability is no longer guaranteed for typical reasoning tasks. In order to address this issue, we identify many decidable fragments of the language, which extend, in a natural way, analog classes defined in the non-disjunctive case. Moreover, we carry out an in-depth complexity analysis, deriving interesting results which range from Logarithmic Space to Exponential Time.

KEYWORDS: datalog, non-monotonic reasoning, decidability, complexity

1 Introduction

Datalog has its origins as a query language in Database Systems, but the language, and in particular its extensions, have well gone beyond this original scope, and are now used in a variety of applications, see for example (De Moor *et al.* 2011). $Datalog^{\vee}$ (Eiter *et al.* 1997), an extension of *Datalog* in which rule heads may be disjunctions of atoms, proved to be especially rewarding in the context of AI,

* Marco Manna's work was supported by the European Commission through the European Social Fund and by Calabria Region.

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

as it allows for the representation of concepts like incomplete knowledge and nondeterministic effects in a simple and intuitive way. Examples for the use of $Datalog^{\vee}$ span from planning (Eiter *et al.* 2004), to data-integration (Leone *et al.* 2005), to reasoning with ontologies (Hustadt *et al.* 2004).

Concerning ontologies, we observe that the field of ontology-based Query Answering (QA) is thriving in data and knowledge management (Calvanese *et al.* 2007; Calì *et al.* 2009; Calì *et al.* 2011; Kollia *et al.* 2011), 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 against a logical theory combining D with an *ontological theory* Σ. More specifically, Σ describes rules and constraints for inferring intensional knowledge from the data stored in D (Johnson and Klug 1984). Thus, for a conjunctive query (CQ) 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, $\text{Datalog}^{\pm}$, a family of extensions of *Datalog* proposed by Calì *et al.* (2009) for tractable QA over ontologies, has recently gained increasing interest (Mugnier 2011). This family generalizes well-known ontology specification languages, and is mainly based on $Datalog^{\exists}$, an extension of *Datalog* that allows existentially quantified variables in rule heads.

In this paper we propose an extension of *Datalog* that allows for both disjunctions and existentially quantified variables in rule heads, called $Datalog^{\exists,\vee}$. This language is highly expressive and enables easy and powerful knowledge-modeling, combining the ability of disjunction to deal with incomplete information, with the power of existential quantifiers to generate unnamed individuals and to deal with them. For example, consider a scenario where each animal is either a carnivore or a herbivore, and any carnivore preys at least one other animal. This knowledge can be modeled by the following $Datalog^{\exists,\vee}$ rules (on the left-hand side) or in equivalent ontological terms (on the right-hand side):

`carnivore(X) v herbivore(X) ← animal(X)`	$\texttt{Animal} \sqsubseteq \texttt{Carnivore} \sqcup \texttt{Herbivore}$
`∃Y preys(X,Y) ← carnivore(X)`	$\texttt{Carnivore} \sqsubseteq \exists \texttt{preys}.\top$
`animal(Y) ← preys(X,Y)`	$\exists \texttt{preys}^{-1}.\top \sqsubseteq \texttt{Animal}$

In general, $Datalog^{\exists,\vee}$ allows to naturally encode advanced ontology properties such as role transitivity, role hierarchy, role inverse, concept products and union of concepts. We define the syntax of the language and provide a formal semantics for QA over $Datalog^{\exists,\vee}$ programs. Since QA over $Datalog^{\exists,\vee}$ is undecidable in the general case (as it is undecidable already on its subclass $Datalog^{\exists}$), we identify a number of $Datalog^{\exists,\vee}$ fragments where QA is decidable, lifting to the disjunctive case several decidability results proved by Calì *et al.* (2009). Moreover, we analyze the complexity of QA in $Datalog^{\exists,\vee}$ by varying different parameters. More specifically, our main contributions are the following:

► We define the novel language $Datalog^{\exists,\vee}$, extending both $Datalog^{\exists}$ and $Datalog^{\vee}$, and provide a formal definition for QA over this language. We also specify the notion of universal model set, which generalizes the concept of universal model to the disjunctive case. A universal model set allows for answering any query.

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

► We define the new concept of instantiation $\mathsf{inst}(P)$ of a $Datalog^{\exists,\vee}$ program P, and show that it is adequate for QA. The finiteness of $\mathsf{inst}(P)$ is a sufficient condition to ensure the decidability of QA over P, since one can compute a finite model set of P from $\mathsf{inst}(P)$ in this case. We design a procedure for computing $\mathsf{inst}(P)$ and prove that it generalizes the oblivious chase procedure introduced by Maier *et al.* (1979) and Johnson and Klug (1984).
► We define the classes of guarded, linear, and weakly guarded $Datalog^{\exists,\vee}$ programs. We show that: (i) they extend the corresponding classes of $Datalog^{\exists}$ programs, (ii) checking membership in these classes is doable in polynomial time, and (iii) QA is decidable in these classes.
► We carry out a complexity analysis to determine the data complexity of QA in all cases that are obtained by varying the following three parameters: (i) the query (atomic, conjunctive, or acyclic), (ii) the class of the underlying $Datalog^{\exists,\vee}$ program (guarded, linear, weakly guarded, monadic-linear, or multi-linear), (iii) the allowed *Datalog* extension (disjunction, existential variables, or both).

To the best of our knowledge, this is the first paper proposing a dedicated extension of Disjunctive *Datalog* with existential quantifiers, and analyzing its decidability and complexity. There have been some proposals (for example, Ferraris *et al.* 2011) for interpreting arbitrary first-order formulas under the stable model semantics, which are more general than our approach, but have a rather different motivation and in particular do not address decidability issues. However, in the literature there are many studies concerning the decidability of (non-disjunctive) $Datalog^{\exists}$ fragments. The decidable subclasses of $Datalog^{\exists}$ rely on four main syntactic paradigms, called *guardedness* (Calì *et al.* 2008), *weak-acyclicity* (Fagin *et al.* 2005), *stickiness* (Calì *et al.* 2010a), and *shyness* (Leone *et al.* 2012). The guardedness paradigm will be discussed in depth in this paper and extended to the disjunctive case. Weak-acyclicity has originally been introduced in the context of data exchange, where programs are required to have finite universal models (Fagin *et al.* 2005). Further extensions have also been proposed in this context (Deutsch *et al.* 2008; Marnette 2009; Meier *et al.* 2009; Greco *et al.* 2011). Sticky $Datalog^{\exists}$ programs, defined more recently, have a low QA complexity and can express the well-known *inclusion dependencies*, but, since they are FO-rewritable, they have limited expressive power. Several generalizations of stickiness have been defined by Calì *et al.* (2010b). For example, the *Sticky-Join* class preserves the benign sticky complexity by also encompassing linear $Datalog^{\exists}$ programs. Finally, *Shy*, the newest among the syntactic $Datalog^{\exists}$ families, offers a good balance between expressivity and complexity. This class significantly extends both the class of *Datalog* and linear $Datalog^{\exists}$ programs, while preserving the same (data and combined) complexity of QA over *Datalog*, even though it includes existential quantifiers.

The results in this paper complement the above-mentioned works, and contribute to a more complete picture of the computational aspects of QA over extensions of *Datalog* with existential quantifiers, providing support for choosing the appropriate setting that fits particular needs in practical applications.

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

2 The disjunctive *Datalog*$^{\exists}$ language

In this section we introduce syntax and semantics of *Datalog*$^{\exists,\vee}$ programs and formally define the query answering problem.

2.1 Preliminaries

The following notation will be used throughout the paper. We always 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 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 a formal structure ς containing 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 $\mathbf{X}$ is the set of variables in ς, i.e., $\mathbf{X} = \mathsf{terms}(\varsigma) \cap \Delta_V$, 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.

2.1.1 Mappings

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 formal structure ς containing atoms, we denote by $\mu(\varsigma)$ the structure obtained by replacing each atom $\mathbf{a}$ of ς with $\mu(\mathbf{a})$. The *composition* of a mapping μ_1 with a mapping μ_2, denoted by $\mu_2 \circ \mu_1$, is the mapping associating each $t \in \Delta$ to $\mu_2(\mu_1(t))$.

Let ς_1 and ς_2 be two formal structures containing 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, $h(\varsigma_1) \subseteq \varsigma_2$). An *isomorphism* between ς_1 and ς_2 is a bijective homomorphism f from ς_1 to ς_2. If such an isomorphism exists, ς_1 and ς_2 are isomorphic, denoted by $\varsigma_1 \simeq \varsigma_2$. A *substitution* is a mapping σ s.t. $t \in \Delta_N$ implies $\sigma(t) = t$, and $t \in \Delta_V$ implies $\sigma(t) \in \Delta_C \cup \Delta_N \cup \{t\}$.

2.2 Programs and queries

A *Datalog*$^{\exists,\vee}$ *rule* r is a finite expression of the form:

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

where (i) $\mathbf{X}$ and $\mathbf{Y}$ are disjoint sets of variables (next called $\forall$-variables and $\exists$-variables, respectively); (ii) $\mathbf{X}' \subseteq \mathbf{X}$; (iii) $\mathbf{disj}_{[\mathbf{X}'\cup\mathbf{Y}]}$ is a nonempty disjunction of atoms;

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

and (iv) $\mathbf{conj}_{[\mathbf{X}]}$ is a conjunction of atoms. Universal quantifiers are usually omitted to lighten the syntax, while existential quantifiers are omitted only if $\mathbf{Y}$ is empty, in which case r coincides with a standard $Datalog^{\vee}$ rule. The sets $\mathsf{atoms}(\mathbf{disj}_{[\mathbf{X}'\cup\mathbf{Y}]})$ and $\mathsf{atoms}(\mathbf{conj}_{[\mathbf{X}]})$ are denoted by $\mathsf{head}(r)$ and $\mathsf{body}(r)$, respectively. If $\mathsf{body}(r)=\emptyset$ and $|\mathsf{head}(r)|=1$, then r is usually referred to as a *fact*. In particular, r is called *existential* or *ground* fact according to whether r contains some $\exists$-variable or not, respectively.

A $Datalog^{\exists,\vee}$ program P is a set of $Datalog^{\exists,\vee}$ rules. W.l.o.g., we assume that rules in P do not share any variable. We denote $\bigcup_{r\in P}\mathsf{head}(r)$ by $\mathsf{heads}(P)$.

A *conjunctive query* (CQ) q, also denoted by $q(\mathbf{X})$, is of the form:

$$\exists\mathbf{Y}\ \mathbf{conj}_{[\mathbf{X}\cup\mathbf{Y}]}, \tag{2}$$

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 (Chekuri and Rajaraman 2000) or, equivalently, if it has hypertree-width 1 (Gottlob *et al.* 1999). A *Boolean CQ* (BCQ) is a query of the form (2) s.t. $\mathbf{X}$ is empty. An *atomic query* is a CQ of the form (2) s.t. $\mathbf{conj}_{[\mathbf{X}\cup\mathbf{Y}]}$ consists of just one atom.

2.3 Semantics

Let $M\subseteq\mathsf{base}(\Delta_C\cup\Delta_N)$. M is a *model* of a rule r of the form (1), 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,P' are called FO-equivalent 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 $\mathscr{M}$, q is true w.r.t. $\mathscr{M}$, denoted by $\mathscr{M}\models q$, if $M\models q$ for each $M\in\mathscr{M}$. 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 $\mathscr{M}$, denoted by $\mathsf{ans}(q,\mathscr{M})$, is the set of substitutions $\sigma|_{\mathbf{X}}$ s.t. $M\models\sigma|_{\mathbf{X}}(q)$ for each $M\in\mathscr{M}$. 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 $\mathsf{ans}(q,\mathscr{M})$.

2.4 The query answering problem

Let $\mathscr{C}$ 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 $\mathscr{C}$ the following decision problem: Given a program $P\in\mathscr{C}$ and a BCQ q, determine whether $P\models q$ holds. In the following we will call class $\mathscr{C}$ QA-decidable if QA over $\mathscr{C}$ is decidable.

We observe that computing $\mathsf{ans}_P(q)$ for a CQ $q(\mathbf{X})$ of the form (2) 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.

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

3 Universal model sets for *Datalog*$^{\exists,\vee}$ programs

In this section we generalize the notion of *universal model* widely used in the context of QA over *Datalog*$^{\exists}$ programs. Intuitively, a universal model M of a *Datalog*$^{\exists}$ program P is such that each model of P is homomorphic to a subset of M.

Definition 1
Let $P \in$ *Datalog*$^{\exists,\vee}$. A set $\mathscr{M} \subseteq \mathsf{mods}(P)$ is a *universal model set* for P if for each $M \in \mathsf{mods}(P)$ there is $M' \in \mathscr{M}$ and a homomorphism h s.t. $h(M') \subseteq M$. □

Universal model sets are sufficient for QA over *Datalog*$^{\exists,\vee}$ programs.

Theorem 1
If $\mathscr{M}$ is a universal model set for P, then $P \models q$ iff $\mathscr{M} \models q$ for each BCQ q.

Proof
($\Rightarrow$) Immediate because $\mathscr{M} \subseteq \mathsf{mods}(P)$ by Definition 1.
($\Leftarrow$) Assume $\mathscr{M} \models q$. Let M be a model of P. We have to show that $M \models q$. By Definition 1, there exist $M' \in \mathscr{M}$ and a homomorphism h s.t. $h(M') \subseteq M$. Since $\mathscr{M} \models q$ by assumption, $M' \in \mathscr{M}$ implies that there is a substitution σ s.t. $\sigma(\mathsf{atoms}(q)) \subseteq M'$. Therefore, $h \circ \sigma(\mathsf{atoms}(q)) \subseteq h(M')$, and combining with $h(M') \subseteq M$ we obtain $h \circ \sigma(\mathsf{atoms}(q)) \subseteq M$, i.e., $M \models q$. □

We now design a strategy for identifying a universal model set 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 universal model set for P.

Let r be a rule of the form (1), 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 $R \cup R'$. (We assume a fixed well-ordering of Δ_N and that variables in $\mathbf{Y}$ are processed according to their order in r.) A firing substitution for a rule r is said to be *spent* if it has already been fired.

Procedure 1 illustrates the overall instantiation procedure. It consists of an exhaustive series of fires in a breadth-first (level-saturating) fashion yielding a (possibly infinite) ground program $\mathsf{inst}(P)$.

Procedure 1: PROGRAM-INSTANTIATION

Input : A *Datalog*$^{\exists,\vee}$ program P
Output: The ground program $\mathsf{inst}(P)$

```
1 R := ∅;
2 repeat
3 |   R' := ∅;
4 |   foreach r ∈ P and foreach unspent firing substitution σ for r w.r.t. R do
5 |   |   R' := R' ∪ {σ̂(r)};
6 |   R := R ∪ R';
7 until R' = ∅;
8 return R;
```

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

Example 1

Let $\varphi_1 < \varphi_2 < \cdots$ be a well-ordering of Δ_N. A run of Procedure 1 on the following program (obtained from the one given in the introduction by predicate renaming):

r_1 : c(X) v h(X) ← a(X) r_3 : a(Y) ← p(X,Y)
r_2 : ∃Y p(X,Y) ← c(X) r_4 : c(lion) ←

starts by setting R and R' to the empty set. The only firing substitution w.r.t. R is the identity substitution for r_4, whose fire yields r_4 itself, which is then added into R'. Rules in R' are moved into R (lines 6 and 3). There is a new firing substitution for r_2, namely σ_1 s.t. $\sigma_1 = \sigma_1|_{\{\mathtt{X}\}}$ and $\sigma_1(\mathtt{X}) = \mathtt{lion}$. The fire of σ_1 yields $\mathtt{p(lion},\varphi_1\mathtt{)} \leftarrow \mathtt{c(lion)}$, which is added into R', and then moved into R. Now there is a firing substitution for r_3, namely σ_2 s.t. $\sigma_2 = \sigma_2|_{\{\mathtt{X},\mathtt{Y}\}}$, $\sigma_2(\mathtt{X}) = \mathtt{lion}$ and $\sigma_2(\mathtt{Y}) = \varphi_1$, whose fire yields $\mathtt{a}(\varphi_1) \leftarrow \mathtt{p(lion},\varphi_1\mathtt{)}$. After adding this rule into R', and then moving it into R, there is a new firing substitution for r_1, namely σ_3 s.t. $\sigma_3 = \sigma_3|_{\{\mathtt{X}\}}$ and $\sigma_3(\mathtt{X}) = \varphi_1$. The fire of σ_3 yields $\mathtt{c}(\varphi_1)\ \mathtt{v}\ \mathtt{h}(\varphi_1) \leftarrow \mathtt{a}(\varphi_1)$, which is added into R', and then moved into R. Now there is a new firing substitution for r_2, namely σ_4 s.t. $\sigma_4 = \sigma_4|_{\{\mathtt{X}\}}$ and $\sigma_4(\mathtt{X}) = \varphi_1$, whose fire yields $\mathtt{p}(\varphi_1,\varphi_2) \leftarrow \mathtt{c}(\varphi_1)$. The procedure thus go on, indefinitely. Let $I = \{\mathtt{c(lion)},\ \mathtt{p(lion},\varphi_1\mathtt{)},\ \mathtt{a}(\varphi_1)\}$. Subset-minimal models of $\mathsf{inst}(P)$ have the following forms:

- $\bigcup_{i\in[1..k]}\ \{\mathtt{c}(\varphi_i),\ \mathtt{p}(\varphi_i,\varphi_{i+1}),\ \mathtt{a}(\varphi_{i+1})\} \cup I \cup \{\mathtt{h}(\varphi_{k+1})\},\quad \forall k \geqslant 1;$
- $\bigcup_{i\geqslant 1}\ \{\mathtt{c}(\varphi_i),\ \mathtt{p}(\varphi_i,\varphi_{i+1}),\ \mathtt{a}(\varphi_{i+1})\} \cup I.$ □

In order to show that $\mathsf{mods}(\mathsf{inst}(P))$ is a universal model set for P, we first point out some relationships between the models of P and those of $\mathsf{inst}(P)$.

Lemma 1

Let P be a *Datalog*$^{\exists,\vee}$ program and $P' = \mathsf{inst}(P)$. For each $M \in \mathsf{mods}(P)$ there exist $M' \in \mathsf{mods}(P')$ and a homomorphism h s.t.: (i) $M' \subseteq \mathsf{heads}(P')$; (ii) $h(M') \subseteq M$; and (iii) $h = h|_{\mathsf{terms}(P')}$.

Proof

Let $M \in \mathsf{mods}(P)$ and $P_i = \{r_1,\ldots,r_i\}$ be the first i rules in P' (w.r.t. the order induced by Procedure 1). We prove by induction that, for each $i \geqslant 0$, there exist $M_i \in \mathsf{mods}(P_i)$ and a homomorphism h_i s.t.: $M_i \subseteq \mathsf{heads}(P_i)$; $h_i(M_i) \subseteq M$; and $h_i = h_i|_{\mathsf{terms}(P_i)}$.

The base case, for $i = 0$, is vacuously true by choosing $M_0 = \emptyset$ and h_0 the identity mapping. Let us assume that the claim holds for some $i \geqslant 0$, and let us extend M_i and h_i in order to show that the claim holds for $i + 1$.

Note that rule r_{i+1} has been obtained by a substitution $\hat{\sigma}$ and a rule $r \in P$ of the form (1). Note also that $h_i \circ \hat{\sigma}$ is a substitution because $h_i = h_i|_{\mathsf{terms}(P_i)}$ by the induction hypothesis. If $h_i \circ \hat{\sigma}(\mathsf{body}(r)) \subseteq M$, there is a substitution $\sigma' \supseteq (h_i \circ \hat{\sigma})|_{\mathbf{X}}$ s.t. $\sigma'(\mathsf{head}(r)) \cap M \neq \emptyset$ (because M is a model of P by assumption). Otherwise, if $h_i \circ \hat{\sigma}(\mathsf{body}(r)) \nsubseteq M$, let $\sigma' = h_i \circ \hat{\sigma}$. Let h_{i+1} be the homomorphism s.t. $t \in \hat{\sigma}(\mathbf{Y})$ implies $h_{i+1}(t) = \sigma'(t)$, and $t \notin \hat{\sigma}(\mathbf{Y})$ implies $h_{i+1}(t) = h_i(t)$. Let M_{i+1} be the following set of atoms: $M_i \cup \hat{\sigma}(\{\mathbf{a} \in \mathsf{atoms}(r) \mid \sigma'(\mathbf{a}) \in M\})$.

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

The following properties hold by construction: $M_{i+1} \subseteq \mathsf{heads}(P_{i+1})$; $h_{i+1}(M_{i+1}) \subseteq M$; and $h_{i+1} = h_{i+1}|_{\mathsf{terms}(P_{i+1})}$. Hence, to complete the proof, we have just to prove that M_{i+1} is a model of P_{i+1}. In fact, this is the case because: r_{i+1} is satisfied by construction of M_{i+1}; rules of P_i are satisfied by M_{i+1} because they are satisfied by M_i, and atoms in $M_{i+1} \setminus M_i$ do not occur in P_i by construction of M_{i+1}. □

A universal model set for P can be obtained from $\mathsf{mods}(\mathsf{inst}(P))$, which allows for answering queries on P by performing the reasoning on $\mathsf{inst}(P)$.

Theorem 2
Let P be a $Datalog^{\exists,\vee}$ program and $P' = \mathsf{inst}(P)$. Model set $\mathscr{M} = \{M \in \mathsf{mods}(P') \mid M \subseteq \mathsf{heads}(P')\}$ is universal for P.

Proof
By Lemma 1, for each $M \in \mathsf{mods}(P)$ there is $M' \in \mathscr{M}$ and a homomorphism h s.t. $h(M') \subseteq M$. It remains to show that $\mathscr{M} \subseteq \mathsf{mods}(P)$, i.e., $M \in \mathsf{mods}(P')$ s.t. $M \subseteq \mathsf{heads}(P')$ implies $M \in \mathsf{mods}(P)$. Let $r \in P$ and σ be a substitution s.t. $\sigma(\mathsf{body}(r)) \subseteq M$, so σ is a firing substitution for P'. Let $\hat{\sigma}(r)$ be the rule of P' obtained by the firing of r. Thus, $\mathsf{head}(\hat{\sigma}(r)) \cap M \neq \emptyset$, i.e., $M \models \hat{\sigma}(r)$. □

The program produced by Procedure 1 is a generalization of the oblivious chase procedure (Maier *et al.* 1979; Johnson and Klug 1984), which associates every $Datalog^{\exists}$ program with a universal model. In fact, the oblivious chase procedure can be obtained from Procedure 1 by replacing line 5 with $R' := R' \cup \hat{\sigma}(\mathsf{head}(r))$, which is enough for $Datalog^{\exists}$ programs.

Corollary 1
Let P be a $Datalog^{\exists}$ program. Then, $\{\mathsf{heads}(\mathsf{inst}(P))\}$ is universal for P.

4 Extending guards-based classes to $Datalog^{\exists,\vee}$

We next define subclasses of $Datalog^{\exists,\vee}$ relying on a well known paradigm, called *guardedness*, first introduced by Andréka *et al.* (1998) in the definition of the guarded fragment of first-order logic and further revisited by Calì *et al.* (2008) for defining $Datalog^{\exists}$ subclasses. In the next section, we show that all these new classes both depend on (easily) checkable syntactic properties, and are QA-decidable.

Definition 2
A $Datalog^{\exists,\vee}$ rule r is said to be *guarded* if it is of the form:

$$\forall \mathbf{X} \exists \mathbf{Y}\ \mathbf{disj}_{[\mathbf{X}' \cup \mathbf{Y}]} \leftarrow\ \mathbf{guard}_{[\mathbf{X}]},\ \mathbf{s\text{-}conj}_{[\mathbf{X}'']}, \tag{3}$$

where $\mathbf{X}'$ and $\mathbf{X}''$ are subsets of $\mathbf{X}$, $\mathbf{guard}_{[\mathbf{X}]}$ is an atom called *guard* and denoted by $\mathsf{guard}(r)$, $\mathbf{s\text{-}conj}_{[\mathbf{X}'']}$ is a conjunction of atoms called *sides* and denoted by $\mathsf{sides}(r)$. Moreover, a guarded rule r is called: *multi-linear* if each side atom could be chosen as guard; *linear* if $\mathsf{sides}(r) = \emptyset$; *monadic-linear* if $\mathsf{sides}(r) = \emptyset$ and all 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. □

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

We now introduce the notion of affected positions of an atom, which are the only positions where nulls might occur in the output of Procedure 1.

Definition 3
Let P be a *Datalog*$^{\exists,\vee}$ program, $\mathbf{a}$ be an atom, and $\mathtt{x}$ 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 $\mathtt{x}$ is either an $\exists$-variable, or a $\forall$-variable s.t. $\mathtt{x}$ occurs in $\mathsf{body}(r)$ in affected positions only. A variable $\mathtt{x}$ occurring in the body of a rule is *unaffected* if it is not affected. □

The above definition is now used to define the class of weakly-guarded programs.

Definition 4
Let P be a *Datalog*$^{\exists,\vee}$ program, and $r \in P$ be a rule of the form:

$$\forall \mathbf{X} \exists \mathbf{Y}\ \mathbf{disj}_{[\mathbf{X}' \cup \mathbf{Y}]} \leftarrow\ \mathbf{wguard}_{[\mathbf{X}'']}, \mathbf{s\text{-}conj}_{[\mathbf{X}''']}, \tag{4}$$

where $\mathbf{X}' \subseteq \mathbf{X} = \mathbf{X}'' \cup \mathbf{X}'''$. Rule r is said to be *weakly-guarded* w.r.t. P, if each variable in $\mathbf{X}''' \setminus \mathbf{X}''$ is unaffected in r. Here, $\mathsf{guard}(r)$ and $\mathsf{sides}(r)$ still denote the (weak) guard and the side atoms of r, respectively. In the following, *Weakly-Guarded-Datalog*$^{\exists,\vee}$ will denote the set of *Datalog*$^{\exists,\vee}$ programs where each rule either is weakly-guarded or has an empty body. □

The new *Datalog*$^{\exists,\vee}$ subclasses introduced in this section generalize important fragments of *Guarded-Datalog*$^{\exists}$ already analyzed in the literature. (Note that *Weakly-Guarded-Datalog*$^{\exists,\vee}$ generalized *Weakly-Guarded-Datalog*$^{\exists}$ because for disjunction-free programs Definition 3 coincides with the the notion of affected position introduced by Calì *et al.* 2008.)

Proposition 1
Definitions 2 and 4 generalize the classes *Guarded-Datalog*$^{\exists}$, *Linear-Datalog*$^{\exists}$, and *Weakly-Guarded-Datalog*$^{\exists}$ defined by Calì *et al.* (2008).

We now pinpoint the complexity of recognizing programs in these classes.

Theorem 3
Checking whether a program belongs to *Guarded-Datalog*$^{\exists,\vee}$, *Linear-Datalog*$^{\exists,\vee}$, or *Weakly-Guarded-Datalog*$^{\exists,\vee}$ is decidable, and doable in polynomial-time.

Proof
Checking whether a program is guarded (resp., linear or multi-linear) is doable in linear time by inspection of the rule bodies. Concerning a *Weakly-Guarded-Datalog*$^{\exists,\vee}$ program P, we observe that Definition 3 introduces a monotone operator for determining affected positions, and the number of such positions is linear in the size of P. Hence, all affected positions in P can be determined in quadratic time. □

5 Decidability results

We now show that all classes introduced in the previous section are QA-decidable. In particular, we use results recently established by Barany *et al.* (2010) on the

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

guarded fragment of first-order logic (Andréka *et al.* 1998; Grädel 1999), here denoted by *Guarded-FOL* and inductively defined as follows: (i) $\mathsf{base}(\Delta_C \cup \Delta_V) \subset$ *Guarded-FOL*; (ii) if $\psi_1, \psi_2 \in$ *Guarded-FOL*, then $\neg\psi_1$, $\psi_1 \vee \psi_2$, $\psi_1 \wedge \psi_2$, and $\psi_1 \leftarrow \psi_2$ also belong to *Guarded-FOL*; and (iii) if $\mathbf{a}_{[\mathbf{X}\cup\mathbf{Y}]} \in \mathsf{base}(\Delta_C \cup \Delta_V)$, $\psi(\mathbf{X}' \cup \mathbf{Y}') \in$ *Guarded-FOL*, and the (free) variables of $\mathbf{a}$ include all the free variables $\mathbf{X}' \cup \mathbf{Y}'$ of ψ, then $\exists\mathbf{Y}(\mathbf{a}_{[\mathbf{X}\cup\mathbf{Y}]} \wedge \psi(\mathbf{X}' \cup \mathbf{Y}'))$ and $\forall\mathbf{X}(\psi(\mathbf{X}' \cup \mathbf{Y}') \leftarrow \mathbf{a}_{[\mathbf{X}\cup\mathbf{Y}]})$ are also in *Guarded-FOL*.

Any *Guarded-Datalog*$^{\exists,\vee}$ program can be viewed as a *Guarded-FOL* formula.

Proposition 2
There is a logarithmic space transducer associating each *Guarded-Datalog*$^{\exists,\vee}$ program with a FO-equivalent *Guarded-FOL* formula.

Proof
For a guarded *Datalog*$^{\exists,\vee}$ rule r of the form (3), let $\mathbf{h}^i_{[\mathbf{X}'_i\cup\mathbf{Y}_i]}$ be the i-th atom in $\mathbf{disj}_{[\mathbf{X}'\cup\mathbf{Y}]}$, with $i \in [1..k]$, $\mathbf{X}'_i \subseteq \mathbf{X}'$, and $\mathbf{Y}_i \subseteq \mathbf{Y}$. Rule r is translated into the following FO-equivalent formula:

$$\forall\mathbf{X}(\exists\mathbf{Y}_1\mathbf{h}^1_{[\mathbf{X}'_1\cup\mathbf{Y}_1]} \vee \cdots \vee \exists\mathbf{Y}_k\mathbf{h}^k_{[\mathbf{X}'_k\cup\mathbf{Y}_k]} \vee \neg\mathbf{s\text{-}conj}_{[\mathbf{X}'']} \leftarrow \mathbf{guard}_{[\mathbf{X}]}).$$

The whole disjunction is an expression $\psi(\mathbf{X}' \cup \mathbf{X}'')$ in *Guarded-FOL* because each $\exists\mathbf{Y}_i\mathbf{h}^i_{[\mathbf{X}'_i\cup\mathbf{Y}_i]}$ is equivalent to $\exists\mathbf{Y}_i(\mathbf{h}^i_{[\mathbf{X}'_i\cup\mathbf{Y}_i]} \wedge \mathbf{h}^i_{[\mathbf{X}'_i\cup\mathbf{Y}_i]}) \in$ *Guarded-FOL*, and since $\neg\mathbf{s\text{-}conj}_{[\mathbf{X}'']}$ trivially belongs to *Guarded-FOL*. Moreover, the expression $\forall\mathbf{X}(\psi(\mathbf{X}' \cup \mathbf{X}'') \leftarrow \mathbf{guard}_{[\mathbf{X}]})$ is in *Guarded-FOL* since $\mathbf{X}' \cup \mathbf{X}'' \subseteq \mathbf{X}$.Finally, a similar construction applies to rules having empty bodies. □

QA-decidability of *Guarded-Datalog*$^{\exists,\vee}$ and its subclasses can now be established.

Theorem 4
Conjunctive QA is decidable under *Guarded*, *Multi-Linear* and *Linear Datalog*$^{\exists,\vee}$.

Proof
The result follows from Proposition 2 and from the fact that conjunctive QA is decidable under *Guarded-FOL* (Barany *et al.* 2010). □

In order to prove that *Weakly-Guarded-Datalog*$^{\exists,\vee}$ is QA-decidable as well, we first introduce the notion of *weak instantiation*.

Definition 5
Let $P \in$ *Weakly-Guarded-Datalog*$^{\exists,\vee}$. For each $r \in P$, let $\mathsf{winst}(r)$ denote the set of partially ground rules associated to r and consisting of the set $\{r\}$ or of the set $\{\sigma(r) \mid \sigma \text{ is a substitution from } \mathbf{X}''' \setminus \mathbf{X}'' \text{ to } \mathsf{terms}(P) \cap \Delta_C\}$ according to whether rule r has an empty body or is of the form (4), respectively. The weak instantiation of P, denoted by $\mathsf{winst}(P)$, is defined as the union of $\mathsf{winst}(r)$ for each $r \in P$. □

The above definition transforms any *Weakly-Guarded-Datalog*$^{\exists,\vee}$ program into a FO-equivalent *Guarded-Datalog*$^{\exists,\vee}$ program.

Lemma 2
Let P be a *Weakly-Guarded-Datalog*$^{\exists,\vee}$ program and $P' = \mathsf{winst}(P)$. Then, both $P' \in$ *Guarded-Datalog*$^{\exists,\vee}$ and $\mathsf{inst}(P) \simeq \mathsf{inst}(P')$ hold.

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

Proof
Assume that Procedure 1 builds isomorphic sets of rules for P and P' up to a given iteration of the repeat-until loop. We shall show that this isomorphism can be extended to the succeeding iteration. For each firing substitution σ for a rule $r \in P$, there are σ_1, σ_2 s.t. $\sigma = \sigma_2 \circ \sigma_1$, where σ_1 is a substitution from $\mathbf{X}''' \setminus \mathbf{X}''$ to $\mathsf{terms}(P) \cap \Delta_C$. Let $r' = \sigma_1(r)$. Therefore, $r' \in P'$ and σ_2 is a firing substitution for r'. Consider now the other direction. Let σ' be a firing substitution for $r' \in P'$. Let $r' = \sigma(r)$, where $r \in P$ and σ is a substitution from $\mathbf{X}''' \setminus \mathbf{X}''$ to $\mathsf{terms}(P) \cap \Delta_C$. Therefore, $\sigma' \circ \sigma$ is a firing substitution for r. The isomorphism can thus be extended by opportunely mapping new nulls. □

We can thus conclude that *Weakly-Guarded-Datalog*$^{\exists,\vee}$ is QA-decidable.

Theorem 5
Conjunctive QA is decidable under *Weakly-Guarded-Datalog*$^{\exists,\vee}$.

Proof
The statement directly follows from Lemma 2 and Theorem 4. □

6 Complexity analysis

In this section we study data complexity of QA under different classes of *Datalog*$^{\exists,\vee}$ and queries. As usual in this setting, we assume that a *Datalog*$^{\exists,\vee}$ program P is paired with a (finite) database $D \subset \mathsf{base}(\Delta_C)$. The set of ground facts $\{\mathbf{a} \leftarrow \mid \mathbf{a} \in D\}$ is denoted by $\overleftarrow{D}$. Similarly, $\overleftarrow{\mathbf{a}}$ denotes the singleton $\{\mathbf{a} \leftarrow\}$ for some atom $\mathbf{a} \in D$. Finally, whenever P contains a rule r of the form $\mathbf{disj} \leftarrow$ (even if $|\mathbf{disj}| = 1$), we replace it in P by $\mathbf{disj} \leftarrow \mathtt{edb}$ and we add to D the extra (propositional) atom $\mathtt{edb}$ of arity zero. Hereafter, we assume $D = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\}$.

6.1 *Guarded-Datalog*$^{\exists,\vee}$

We start by providing an upper bound for QA over *Guarded-Datalog*$^{\exists,\vee}$.

Theorem 6
Data complexity of QA over *Guarded-Datalog*$^{\exists,\vee}$ programs is in **coNP**.

Proof
From statement 5 of Theorem 19 in Barany *et al.* (2010), data complexity of deciding whether a CQ is true w.r.t. a *Guarded-FOL* formula is in **coNP**. The claim therefore follows from Proposition 2. □

We now pinpoint the complexity of QA over *Guarded-Datalog*$^{\exists,\vee}$.

Theorem 7
Data complexity of QA over *Guarded-Datalog*$^{\exists,\vee}$ programs is **coNP**-complete in general, and it is **coNP**-hard already in the following cases:

1. A *Monadic-Linear-Datalog*$^{\vee}$ program under an acyclic CQ.
2. A *Multi-Linear-Datalog*$^{\vee}$ program under an atomic query.

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

Proof
(1) QA is **coNP**-hard already in the following setting: 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 follows from Theorem 6.4 (and its proof) of Calvanese *et al.* (2009): Let ϕ be a 2+2-CNF formula, namely a CNF formula where each clause has exactly two positive and two negative literals. Let D be a database containing an atom $\texttt{lit}(x)$ for each propositional variable x, and atoms $\texttt{p}_1(c,x_1)$, $\texttt{p}_2(c,x_2)$, $\texttt{n}_1(c,x_3)$, $\texttt{n}_2(c,x_4)$ for each clause $x_1 \wedge x_2 \wedge \neg x_3 \wedge \neg x_4$ having c as identifier. Let P be a *Monadic-Linear-Datalog*$^{\vee}$ program consisting of the following rule: $\texttt{t(X) v f(X)} \leftarrow \texttt{lit(X)}$, and q be the following acyclic CQ: $\exists\ \texttt{C,P}_1\texttt{,P}_2\texttt{,N}_1\texttt{,N}_2$ $\texttt{p}_1\texttt{(C,P}_1\texttt{),f(P}_1\texttt{),p}_2\texttt{(C,P}_2\texttt{),f(P}_2\texttt{),n}_1\texttt{(C,N}_1\texttt{),t(N}_1\texttt{),n}_2\texttt{(C,N}_2\texttt{),t(N}_2\texttt{)}$. Hence, ϕ is satisfiable if and only if $P \cup \overleftarrow{D} \models q$.

(2) The **coNP**-complete problem 3-UNSAT can be encoded by means of an atomic query $\texttt{wrongAssignment}$ over the following *Multi-Linear-Datalog*$^{\vee}$ program P:

$$\texttt{sel(L}_1\texttt{,N}_1\texttt{) v sel(L}_2\texttt{,N}_2\texttt{) v sel(L}_3\texttt{,N}_3\texttt{)} \leftarrow \texttt{clause(L}_1\texttt{,L}_2\texttt{,L}_3\texttt{,N}_1\texttt{,N}_2\texttt{,N}_3\texttt{).}$$
$$\texttt{wrongAssignment} \leftarrow \texttt{sel(L,N), sel(N,L).}$$

As far as database D is concerned, each clause $\ell_1 \vee \ell_2 \vee \ell_3$ of a given 3-CNF formula ϕ is encoded in D by the ground atom $\texttt{clause}(\text{“}\ell_1\text{”}, \text{“}\ell_2\text{”}, \text{“}\ell_3\text{”}, n(\ell_1), n(\ell_2), n(\ell_3))$, where $n(\ell) = \text{“}\neg x\text{”}$ if ℓ is a positive propositional variable x, and $n(\ell) = \text{“}x\text{”}$ if ℓ is a negative propositional variable $\neg x$. If there is a satisfying assignment for ϕ, then there is a model of $P \cup \overleftarrow{D}$ not containing $\texttt{wrongAssignment}$. □

6.2 *Weakly-Guarded-Datalog*$^{\exists,\vee}$

As in the disjunction-free case, the complexity of QA over *Weakly-Guarded-Datalog*$^{\exists,\vee}$ is harder than QA over *Guarded-Datalog*$^{\exists,\vee}$.

Theorem 8
Data complexity of QA over *Weakly-Guarded-Datalog*$^{\exists,\vee}$ is **EXP**-complete in general, and it is **EXP**-hard already for atomic queries over *Weakly-Guarded-Datalog*$^{\exists}$.

Proof
Hardness comes from the **EXP**-hardness of *Weakly-Guarded-Datalog*$^{\exists}$ (Calì *et al.* 2008). As for the membership, let P be a *Weakly-Guarded-Datalog*$^{\exists,\vee}$ program and $P' = \mathsf{winst}(P \cup \overleftarrow{D})$ be the *Guarded-Datalog*$^{\exists,\vee}$ program built according to Definition 5. By Lemma 2, $P \cup \overleftarrow{D} \models q$ iff $P' \models q$. Moreover, let k be the maximum number of unguarded (thus unaffected) variables appearing in some rule of P, γ be the number of constants occurring in P, and w be the maximum arity over all predicate symbols in $P \cup \overleftarrow{D}$. We point out that $|P'| \leqslant |D| + |P| \cdot (w \cdot |D| + \gamma)^k$. Hence, in data complexity, the size of P' is polynomial in the cardinality of D. Barany *et al.* (2010) have shown that QA over a *Guarded-FOL* formula is in **2EXP** in the general case. However, this double exponential dependence is only in terms of q and w. If P and q are considered fixed, then the complexity is simply exponential in the size of P'. Moreover, since P' can be translated in logarithmic space into a FO-equivalent *Guarded-FOL* formula by Proposition 2, then we have an **EXP** (w.r.t. the cardinality of D) algorithm deciding whether $P' \models q$. □

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

6.3 Atomic queries over Linear-Datalog$^{\exists,\vee}$

In the following, let P be a *Linear-Datalog*$^{\exists,\vee}$ program and q be Boolean atomic query. As before, $D = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\}$ is the input database. We first introduce a decomposition property relying on the structure of P.

Lemma 3
Let C be the set $\mathsf{mods}(P \cup \overleftarrow{\mathbf{a}_1}) \times \cdots \times \mathsf{mods}(P \cup \overleftarrow{\mathbf{a}_n})$, and $\mathscr{M}$ be $\{M_1 \cup \cdots \cup M_n \mid \langle M_1, \ldots, M_n \rangle \in C\}$. It holds that $\mathscr{M} = \mathsf{mods}(P \cup \overleftarrow{D})$.

Proof
($\subseteq$) Let $\langle M_1, \ldots, M_n \rangle \in C$, and $M = M_1 \cup \cdots \cup M_n$. To prove that M is a model of $P \cup \overleftarrow{D}$, we have to show that whenever for a rule $r \in P$ there exists a substitution σ s.t. $\sigma(\mathsf{body}(r)) \subseteq M$, then $M \models \sigma(\mathsf{head}(r))$. Let us fix a pair (r, σ) s.t. $\sigma(\mathsf{body}(r)) \subseteq M$. Since P is linear, there is $i \in [1..n]$ s.t. $\sigma(\mathsf{body}(r)) \subseteq M_i$. But since M_i is a model of $P \cup \overleftarrow{\mathbf{a}}$, then $M_i \models \sigma(\mathsf{head}(r))$. Finally, the implication holds since $M_i \subseteq M$.

($\supseteq$) Let M be a model of $P \cup \overleftarrow{D}$. For each $i \in [1..n]$, M is also a model of $P \cup \overleftarrow{\mathbf{a}_i}$. Consequently, the n-tuple $\langle M, \ldots, M \rangle$ belongs to C, entailing that $M \in \mathscr{M}$. □

The following lemma represents a logspace Turing reduction from the problem of evaluating q over $P \cup \overleftarrow{D}$ to the problem of evaluating q over $P \cup \overleftarrow{\mathbf{a}}$ for some $\mathbf{a} \in D$.

Lemma 4
$P \cup \overleftarrow{D} \models q$ if and only if $\exists i \in [1..n]$ s.t. $P \cup \overleftarrow{\mathbf{a}_i} \models q$.

Proof
($\Rightarrow$) We prove the contrapositive. Let us assume that $\forall i \in [1..n]$ $P \cup \overleftarrow{\mathbf{a}_i} \not\models q$. Thus, $\forall i \in [1..n]$ there exists a model M_i s.t. $M_i \not\models q$. Therefore, $M_1 \cup \cdots \cup M_n \not\models q$ and by Lemma 3 we obtain $P \cup \overleftarrow{D} \not\models q$.

($\Leftarrow$) Since $\exists i \in [1..n]$ s.t. $P \cup \overleftarrow{\mathbf{a}}_i \models q$, then $M \models q$ for each $M \in \mathsf{mods}(P \cup \overleftarrow{\mathbf{a}}_i)$. By Lemma 3, $P \cup \overleftarrow{D} \models q$. □

Lemma 4 allows for focussing the analysis on a single database atom, say $\mathbf{a} \in D$. The *instantiation-tree* for $P \cup \overleftarrow{\mathbf{a}}$ is the directed acyclic graph $T = \mathsf{tree}(P \cup \overleftarrow{\mathbf{a}})$ inductively constructed as follows: (i) the root of T is a node labeled with $\overleftarrow{\mathbf{a}}$; (ii) for each node m of T and for each rule $r \in \mathsf{inst}(P \cup \overleftarrow{\mathbf{a}})$ s.t. $\mathsf{body}(r)$ appears in the head of the rule labeling m, we add a node n labeled with r along with an arc from m to n. (See Example 2.) Let $\mathsf{nodes}(T)$ and $\mathsf{arcs}(T)$ denote the nodes and arcs of T, respectively; $\mathsf{label}(n)$ denotes the ground rule used as label for n; $n \in T$ is short for $n \in \mathsf{nodes}(T)$; $\mathsf{subtree}(n)$ is the tree below n; finally, $\mathsf{depth}(n)$ is the *depth* of n in T, defined as the length of the path leading from the root of T to n.

Definition 6
The *stem* of $P \cup \overleftarrow{\mathbf{a}}$, denoted by $\mathsf{stem}(P \cup \overleftarrow{\mathbf{a}})$, is the maximal subtree that can be obtained starting from the root of $\mathsf{tree}(P \cup \overleftarrow{\mathbf{a}})$ in such a way that each path contains no nodes labelled with rules with isomorphic bodies. Finally, $\mathsf{sinst}(P \cup \overleftarrow{\mathbf{a}})$ denotes the set $\{\mathsf{label}(n) \mid n \in \mathsf{stem}(P \cup \overleftarrow{\mathbf{a}})\}$. □

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

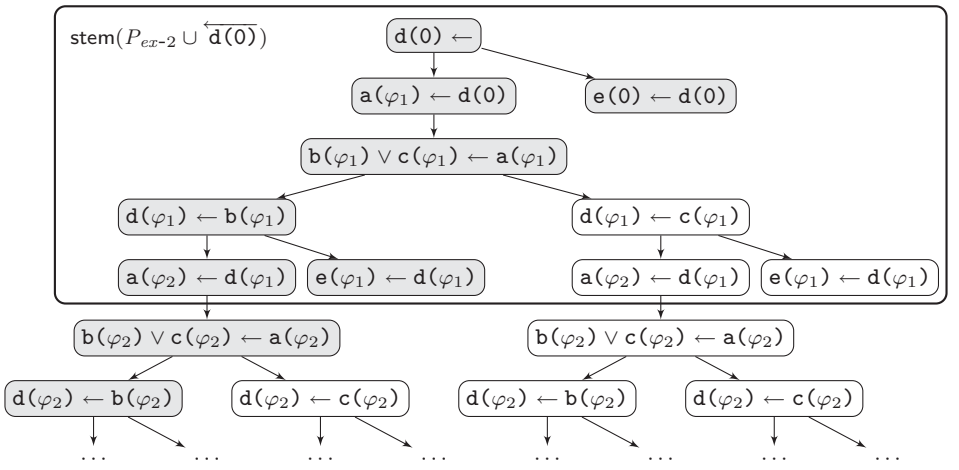


Fig. 1. The instantiation-tree for $P_{ex\text{-}2}$ and d(0), and their stem.

Example 2
Consider a database atom d(0) for the following program $P_{ex\text{-}2}$:

```
∃Y a(Y) ← d(X)          d(X) ← b(X)          e(X) ← d(X)
b(X) v c(X) ← a(X)      d(X) ← c(X)
```

The instantiation-tree is reported in Figure 1, where we also highlighted the stem. Note also that there are many isomorphic subtrees. This is due to a structural property of $\mathsf{tree}(P \cup \overleftarrow{\mathbf{a}})$, which we highlight in the next lemma. □

Lemma 5
Let m, n be two nodes of $T = \mathsf{tree}(P \cup \overleftarrow{\mathbf{a}})$ s.t. $\mathsf{body}(\mathsf{label}(m)) \simeq \mathsf{body}(\mathsf{label}(n))$. There is a node $m' \in T$ among m and its siblings s.t. $\mathsf{subtree}(m') \simeq \mathsf{subtree}(n)$.

Proof
The statement holds if $m = \mathsf{root}(T)$ or $n = \mathsf{root}(T)$ because in this case $n = m$ as only the root of T can contain a rule with an empty body. Otherwise, let m_p, n_p be the parent nodes of m and n, respectively. By construction (relying on Procedure 1), $\mathsf{body}(\mathsf{label}(m)) \subseteq \mathsf{head}(\mathsf{label}(m_p))$ and $\mathsf{body}(\mathsf{label}(n)) \subseteq \mathsf{head}(\mathsf{label}(n_p))$. Let $\mathsf{label}(n) = \hat{\sigma}(r)$, where r is a rule and $\hat{\sigma}$ is a substitution. Let h be the isomorphism between $\mathsf{body}(\mathsf{label}(m))$ and $\mathsf{body}(\mathsf{label}(n))$. Thus, there is a child m' of m s.t. $\mathsf{label}(m') = h \,\hat{\circ}\, \sigma(r)$, which in turn implies $\mathsf{label}(m') \simeq \mathsf{label}(n)$. We now use induction. Let $n, n_1, \ldots, n_k$ and $m', m'_1, \ldots, m'_k$ $(k \geqslant 0)$ be two isomorphic paths in $\mathsf{subtree}(n)$ and $\mathsf{subtree}(m')$, respectively. Still by construction, there is a one-to-one mapping μ between the children of n_k and those of m'_k s.t. for each child n_{k+1} of n_k it holds that $\mathsf{label}(n_{k+1}) \simeq \mathsf{label}(\mu(n_{k+1}))$. □

Given a model M of $\mathsf{sinst}(P \cup \overleftarrow{\mathbf{a}})$, we shall show how to build a model M^* of $\mathsf{inst}(P \cup \overleftarrow{\mathbf{a}})$ s.t. $M^* \models q$ implies $M \models q$. Let $S = \mathsf{stem}(P \cup \overleftarrow{\mathbf{a}})$, and C_0 be the smallest subset of $\mathsf{nodes}(S)$ satisfying the following properties: (i) $\mathsf{root}(S) \in C_0$; (ii) $n \in C_0$ whenever its parent belongs to C_0 and $\mathsf{body}(\mathsf{label}(n)) \subseteq M$ holds. We can thus restrict model M as follows: $M_0 = \{\mathbf{b} \in M \mid \mathbf{b} \in \mathsf{head}(\mathsf{label}(n)) \wedge n \in C_0\}$.

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

Example 3
Consider again the instantiation-tree reported in Figure 1. Let $M = \{\mathtt{d(0)}\} \cup \{\mathtt{a}(\varphi_i) \mid i \geqslant 1\} \cup \{\mathtt{b}(\varphi_i) \mid i \geqslant 1\} \cup \{\mathtt{c}(\varphi_i) \mid i \geqslant 2\} \cup \{\mathtt{d}(\varphi_i) \mid i \geqslant 1\} \cup \{\mathtt{e}(\varphi_i) \mid i \geqslant 1\}$. Nodes in C_0 are those colored in gray, and $M_0 = M \setminus \{\mathtt{c}(\varphi_i) \mid i \geqslant 2\}$. Note that M_0 is still a model of the program, as formally established by the next lemma. □

Lemma 6
If M is a model of $\mathsf{sinst}(P \cup \overleftarrow{\mathbf{a}})$, then also M_0 is.

Proof
Let $n \in \mathsf{nodes}(S) \setminus C_0$, m be the parent of n, $\mathbf{b}$ be the unique atom in $\mathsf{body}(\mathsf{label}(n))$, and $\mathbf{b} \in M_0$. We claim that $\mathsf{head}(\mathsf{label}(n)) \cap M_0 \neq \emptyset$. By Procedure 1, since $\mathbf{b} \in M_0$, then $\mathbf{b} \in \mathsf{head}(\mathsf{label}(m))$. Moreover, according to the definition of C_0, if m belongs to C_0, then also n does. Hence, $m \notin C_0$ implying that there is a node m' in C_0 s.t. $\mathbf{b} \in \mathsf{head}(\mathsf{label}(m'))$. But this means, since $\mathbf{b} \in M_0$, that there is a child n' of m' s.t. $n' \in C_0$ and $\mathsf{label}(n') = \mathsf{label}(n)$. However, since by construction the head of each node in C_0 has a nonempty intersection with M, then $\mathsf{head}(\mathsf{label}(n')) = \mathsf{head}(\mathsf{label}(n))$ has a nonempty intersection with M_0. □

From $T = \mathsf{tree}(P \cup \overleftarrow{\mathbf{a}})$, we define a total function $f : \mathsf{nodes}(T) \to \mathsf{nodes}(T)$ as follows: For each node $n \in S = \mathsf{stem}(P \cup \overleftarrow{\mathbf{a}})$, $f(n) = n$. For the remaining nodes, let $n \in \mathsf{nodes}(T) \setminus \mathsf{nodes}(S)$ s.t. its parent belongs to S. Let m be the (unique) node in the path from $\mathsf{root}(T)$ to n s.t. $\mathsf{body}(\mathsf{label}(m)) \simeq \mathsf{body}(\mathsf{label}(n))$. Let m' be either m or one of its siblings according to whether $\mathsf{label}(m') \simeq \mathsf{label}(n)$. Function f thus maps $\mathsf{subtree}(n)$ into $\mathsf{subtree}(m')$; it is total by Lemma 5. As a remark, we have that $n \simeq f(n)$, for each $n \in T$. Moreover, $f(n) = n$ if and only if $n \in S$.

Finally, we build the set C^* and the model M^* of $\mathsf{inst}(P \cup \overleftarrow{\mathbf{a}})$ s.t. $M^* \models q$ implies $M \models q$. Initially, C^* and M^* coincide with C_0 and M_0, respectively. Subsequently, for each node $n \in \mathsf{nodes}(T) \setminus \mathsf{nodes}(S)$ s.t. both $\mathsf{parent}(n) \in C^*$ and $\mathsf{body}(\mathsf{label}(n)) \subseteq M^*$, C^* is augmented by n and M^* is augmented by the set $\{\mathbf{b} \in \mathsf{head}(\mathsf{label}(n)) \mid h(\mathbf{a}) \in M^*\}$ where h is the isomorphism between n and $f(n)$.

We now prove that QA can be performed by only considering rules in the stem.

Lemma 7
It holds that $\mathsf{inst}(P \cup \overleftarrow{\mathbf{a}}) \models q$ if and only if $\mathsf{sinst}(P \cup \overleftarrow{\mathbf{a}}) \models q$.

Proof
($\Leftarrow$) Since $\mathsf{sinst}(P \cup \overleftarrow{\mathbf{a}}) \subseteq \mathsf{inst}(P \cup \overleftarrow{\mathbf{a}})$, each model of $\mathsf{inst}(P \cup \overleftarrow{\mathbf{a}})$ is also a model of $\mathsf{sinst}(P \cup \overleftarrow{\mathbf{a}})$.

($\Rightarrow$) Let us assume that $\mathsf{inst}(P \cup \overleftarrow{\mathbf{a}}) \models q$ holds. Let M be a model of $\mathsf{sinst}(P \cup \overleftarrow{\mathbf{a}})$. Since, by construction, M^* is a model of $\mathsf{inst}(P \cup \overleftarrow{\mathbf{a}})$, and since $M^* \models q$ by hypothesis, then $M \models q$ holds. □

Tractability of atomic QA over *Linear-Datalog*$^{\exists,\vee}$ can now be established.

Theorem 9
Data complexity of atomic QA over *Linear-Datalog*$^{\exists,\vee}$ programs is in **LOGSPACE**.

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

Proof

Armed with Lemma 7, a logspace procedure iterates the database atoms looking for an atom $\mathbf{a} \in D$ s.t. $\mathsf{sinst}(P \cup \overleftarrow{\mathbf{a}}) \models q$. In fact, for each $n \in \mathsf{stem}(P \cup \overleftarrow{\mathbf{a}})$, $\mathsf{depth}(n) < |\pi| \cdot (2w)^w$, where w is the maximum arity over all predicate symbols in P, and π is the number of predicate symbols occurring in P. Therefore, cardinality of the ground program $\mathsf{sinst}(P \cup \overleftarrow{\mathbf{a}})$ does not depend on D and neither does the number of its minimal models, which are sufficient for QA. □

6.4 Discussion

Table 1 provides a comprehensive 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 (Calì *et al.* 2008; Calì *et al.* 2009), results for *Weakly-Guarded-Datalog*$^{\vee}$ (last two results in column $\{\vee\}$) follow from Eiter *et al.* (1997), since this class coincides with *Datalog*$^{\vee}$. All the remaining **coNP**-completeness results follow from Theorem 7 in Section 6.1, the remaining **EXP**-completeness results follow from Theorem 8 in Section 6.2, and the **LOGSPACE** upper bounds follow from Theorem 9 in Section 6.3.

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.

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 **LOGSPACE**	in **LOGSPACE**
	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

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

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.

In future work, we intend to investigate on the exact data complexity of atomic QA over (*Monadic-*)*Linear-Datalog*$^{\exists,\vee}$ programs, in particular whether it is in $\mathbf{AC}_0$ or not. We also intend to study the impact of disjunction on other tractable fragments of *Datalog*$^{\exists}$ based on different paradigms, for example *stickiness* (Calì *et al.* 2010a), *shyness* (Leone *et al.* 2012) and *weak-acyclicity* (Fagin *et al.* 2005). 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, for example in DLV$^{\exists}$ (Leone *et al.* 2012), is on our agenda.

Acknowledgements

The authors want to thank Georg Gottlob, Michael Morak, and Andreas Pieris for useful discussions on the problem. The work was partially supported by MIUR under the PON projects FRAME and TETRIS.

References

Andréka, H., Németi, I. and Van Benthem, J. 1998. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic 27*, 217–274.

Barany, V., Gottlob, G. and Otto, M. 2010. Querying the guarded fragment. In *Proceedings of the 25th Annual IEEE Symposium on LICS*. 1–10.

Calì, A., Gottlob, G. and Kifer, M. 2008. Taming the infinite chase: Query answering under expressive relational constraints. In *Proceedings of the 11th KR International Conference*. 70–80. Revised version: `http://dbai.tuwien.ac.at/staff/gottlob/CGK.pdf`.

Calì, A., Gottlob, G. and Lukasiewicz, T. 2009. A general datalog-based framework for tractable query answering over ontologies. In *Proceedings of the 28th PODS Symposium*. 77–86.

Calì, A., Gottlob, G. and Pieris, A. 2010a. Advanced processing for ontological queries. *PVLDB 3*, 1, 554–565.

Calì, A., Gottlob, G. and Pieris, A. 2010b. Query answering under non-guarded rules in Datalog$^{\pm}$. In *Proceedings of the 4th RR International Conferences,* vol. 6333. 1–17.

Calì, A., Gottlob, G. and Pieris, A. 2011. New expressive languages for ontological query answering. In *Proceedings of the 25th AAAI Conference on AI*. 1541–1546.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M. and Rosati, R. 2009. Ontologies and databases: The *DL-Lite* approach. In *Reasoning Web*. LNCS, vol. 5689. Springer, 255–356.

Calvanese, D., Giacomo, G., Lembo, D., Lenzerini, M. and Rosati, R. 2007. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automatic Reasoning 39*, 385–429.

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press

Chekuri, C. and Rajaraman, A. 2000. Conjunctive query containment revisited. *Theoretical Computer Science 239*, 2, 211–229.

De Moor, O., Gottlob, G., Furche, T. and Sellers, A., Eds. 2011. *Datalog Reloaded. First International Workshop, Datalog 2010. Revised Selected Papers.* LNCS, vol. 6702. Springer Verlag.

Deutsch, A., Nash, A. and Remmel, J. 2008. The chase revisited. In *Proceedings of the 27th PODS Symposium* 149–158.

Eiter, T., Faber, W., Leone, N., Pfeifer, G. and Polleres, A. 2004. A logic programming approach to knowledge-state planning: Semantics and complexity. *ACM TOCL 5*, 2, 206–263.

Eiter, T., Gottlob, G. and Mannila, H. 1997. Disjunctive datalog. *ACM TODS 22*, 3, 364–418.

Fagin, R., Kolaitis, P. G., Miller, R. J. and Popa, L. 2005. Data exchange: Semantics and query answering. *TCS 336*, 1, 89–124.

Ferraris, P., Lee, J. and Lifschitz, V. 2011. Stable models and circumscription. *Artificial Intelligence 175*, 1, 236–263.

Gottlob, G., Leone, N. and Scarcello, F. 1999. Hypertree decompositions and tractable queries. In *Proceedings of the 18th PODS Symp.* 21–32.

Grädel, E. 1999. On the restraining power of guards. *The Journal of Symbolic Logic 64*, 4, 1719–1742.

Greco, S., Spezzano, F. and Trubitsyna, I. 2011. Stratification criteria and rewriting techniques for checking chase termination. *PVLDB 4*, 11, 1158–1168.

Hustadt, U., Motik, B. and Sattler, U. 2004. Reducing SHIQ- Descrption logic to disjunctive datalog programs. In *Proceedings of the 9th KR International Conference.* 152–162.

Johnson, D. and Klug, A. 1984. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences 28*, 1, 167–189.

Kollia, I., Glimm, B. and Horrocks, I. 2011. SPARQL query answering over OWL ontologies. In *Proceedings of the 24th DL International Workshop.* LNCS, vol. 6643. Springer, 382–396.

Leone, N., Gottlob, G., Rosati, R., Eiter, T., Faber, W., Fink, M., Greco, G., Ianni, G., Kałka, E., Lembo, D., Lenzerini, M., Lio, V., Nowicki, B., Ruzzi, M., Staniszkis, W. and Terracina, G. 2005. The INFOMIX system for advanced integration of incomplete and inconsistent data. In *Proceedings of the 24th ACM SIGMOD International Conference on Management of Data.* 915–917.

Leone, N., Manna, M., Terracina, G. and Veltri, P. 2012. Efficiently computable datalog$^{\exists}$ programs. In *Proceedings of the 13th KR International Conference*, Forthcoming. Long version: `www.mat.unical.it/kr2012/shy.pdf`.

Maier, D., Mendelzon, A. O. and Sagiv, Y. 1979. Testing implications of data dependencies. *ACM TODS 4*, 4, 455–469.

Marnette, B. 2009. Generalized schema-mappings: from termination to tractability. In *Proceedings of the 28th PODS Symposium.* 13–22.

Meier, M., Schmidt, M. and Lausen, G. 2009. On chase termination beyond stratification. *PVLDB 2*, 1, 970–981.

Mugnier, M.-L. 2011. Ontological query answering with existential rules. In *Proceedings of the 5th RR International Conference.* 2–23.

https://doi.org/10.1017/S1471068412000257 Published online by Cambridge University Press