Disjunctive Datalog with Existential Quantifiers: Semantics, Decidability, and Complexity Issues

MARIO ALVIANO, WOLFGANG FABER, NICOLA LEONE, MARCO MANNA∗
Department of Mathematics, University of Calabria, Italy
(e-mail: {alviano, faber, leone, manna}@mat.unical.it)

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

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∃ 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∃, called Datalog∃∨, 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∃∨, and provide a notion of instantiation, which we prove to be adequate for Datalog∃∨. A main issue of Datalog∃ and hence also of Datalog∃∨ 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).

∗ Marco Manna’s work was supported by the European Commission through the European Social Fund and by Calabria Region.
Datalog$^\lor$ \cite{Eiter1997}, an extension of Datalog in which rule heads may be disjunctions of atoms, proved to be especially rewarding in the context of AI, as it allows for the representation of concepts like incomplete knowledge and non-deterministic effects in a simple and intuitive way. Examples for the use of Datalog$^\lor$ span from planning \cite{Eiter2004}, to data-integration \cite{Leone2005}, to reasoning with ontologies \cite{Hustadt2004}.

Concerning ontologies, we observe that the field of ontology-based Query Answering (QA) is thriving in data and knowledge management \cite{Calvanese2007, Cali2009, Kollia2011, Cali2011}, 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 $\Sigma$. More specifically, $\Sigma$ describes rules and constraints for inferring intensional knowledge from the data stored in $D$ \cite{Johnson1984}. 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 $\Sigma$. To this end, Datalog$^\pm$, a family of extensions of Datalog proposed by Cali et al. \cite{Cali2009} for tractable QA over ontologies, has recently gained increasing interest \cite{Mugnier2011}. This family generalizes well-known ontology specification languages, and is mainly based on Datalog$^3$, 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$^{3,\lor}$. 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$^{3,\lor}$ rules (on the left-hand side) or in equivalent ontological terms (on the right-hand side):

\[
\begin{align*}
\text{carnivore}(X) \lor \text{herbivore}(X) & \leftarrow \text{animal}(X) \\
\exists Y \ preys(X,Y) & \leftarrow \text{carnivore}(X) \\
\text{animal}(Y) & \leftarrow \text{preys}(X,Y)
\end{align*}
\]

In general, Datalog$^{3,\lor}$ 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$^{3,\lor}$ programs. Since QA over Datalog$^{3,\lor}$ is undecidable in the general case (as it is undecidable already on its subclass Datalog$^3$), we identify a number of Datalog$^{3,\lor}$ fragments where QA is decidable, lifting to the disjunctive case several decidability results proved by Cali et al. \cite{Cali2009}. Moreover, we analyze the complexity of QA in Datalog$^{3,\lor}$ by varying different parameters. More specifically, our main contributions are the following:

- We define the novel language Datalog$^{3,\lor}$, extending both Datalog$^3$ and Datalog$^\lor$, 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. ▶ We define the new concept of instantiation inst(P) of a Datalog\(^{∃,∨}\) program P, and show that it is adequate for QA. The finiteness of 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 inst(P) in this case. We design a procedure for computing 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\(^{∃,∨}\) programs. We show that: (i) they extend the corresponding classes of Datalog\(^{∃}\) 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\(^{∃,∨}\) 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\(^3\) fragments. The decidable subclasses of Datalog\(^3\) 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\(^3\) 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\(^3\) programs. Finally, Shy, the newest among the syntactic Datalog\(^3\) families, offers a good balance between expressivity and complexity. This class significantly extends both the class of Datalog and linear Datalog\(^3\) 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.
2 The Disjunctive Datalog$^3$ Language

In this section we introduce syntax and semantics of Datalog$^{3,\lor}$ programs and formally define the query answering problem.

2.1 Preliminaries

The following notation will be used throughout the paper. We always denote by $\Delta_C$, $\Delta_N$ and $\Delta_V$, countably infinite domains of terms called constants, nulls and variables, respectively; by $\Delta$, the union of these three domains; by $\varphi$, a null; by $x$ and $y$, variables; by $X$ and $Y$, sets of variables; by $\Pi$ an alphabet of predicate symbols each of which, say $p$, has a fixed nonnegative arity; by $a$, $b$ and $c$, atoms being expressions of the form $p(t_1, \ldots, t_k)$, where $p$ is a predicate symbol, and $t_1, \ldots, t_k$ is a tuple of terms. For an atom $a$, we denote by $\text{pred}(a)$ the predicate symbol of $a$.

For a formal structure $\varsigma$ containing atoms, $\text{atoms}(\varsigma)$ denotes the set of atoms in $\varsigma$, and $\text{terms}(\varsigma)$ denotes the set of terms occurring in $\text{atoms}(\varsigma)$. If $X$ is the set of variables in $\varsigma$, i.e., $X = \text{terms}(\varsigma) \cap \Delta_V$, then $\varsigma$ is also denoted by $\varsigma|_X$. A structure $\varsigma|_\emptyset$ is called ground. If $T \subseteq \Delta$ and $T \neq \emptyset$, then $\text{base}(T)$ denotes the set of all atoms that can be formed with predicate symbols in $\Pi$ 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 $\Delta$. The application of $\mu$ to $T$, denoted by $\mu(T)$, is the set $\{\mu(t) \mid t \in T\}$. The restriction of $\mu$ to $T$, denoted by $\mu|_T$, is the mapping $\mu'$ 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 $\mu$ is an extension of $\mu'$, denoted by $\mu \supseteq \mu'$.

For an atom $a = p(t_1, \ldots, t_k)$, we denote by $\mu(a)$ the atom $p(\mu(t_1), \ldots, \mu(t_k))$. For a formal structure $\varsigma$ containing atoms, we denote by $\mu(\varsigma)$ the structure obtained by replacing each atom $a$ of $\varsigma$ with $\mu(a)$. The composition of a mapping $\mu_1$ with a mapping $\mu_2$, denoted by $\mu_2 \circ \mu_1$, is the mapping associating each $t \in \Delta$ to $\mu_2(\mu_1(t))$.

Let $\varsigma_1$ and $\varsigma_2$ be two formal structures containing atoms. A homomorphism from $\varsigma_1$ to $\varsigma_2$ is a mapping $h$ s.t. $h(\varsigma_1)$ is a substructure of $\varsigma_2$ (for example, if $\varsigma_1$ and $\varsigma_2$ are sets of atoms, $h(\varsigma_1) \subseteq \varsigma_2$). An isomorphism between $\varsigma_1$ and $\varsigma_2$ is a bijective homomorphism $f$ from $\varsigma_1$ to $\varsigma_2$. If such an isomorphism exists, $\varsigma_1$ and $\varsigma_2$ are isomorphic, denoted by $\varsigma_1 \simeq \varsigma_2$. A substitution is a mapping $\sigma$ 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$^{3,\lor}$ rule $r$ is a finite expression of the form:

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

where (i) $X$ and $Y$ are disjoint sets of variables (next called $\forall$-variables and $\exists$-variables, respectively); (ii) $X' \subseteq X$; (iii) $\text{disj}_{[X \cup Y]}$ is a nonempty disjunction of
models of $P$ w.r.t. a program $P$. The same consideration also applies to the set of substitutions $\sigma$ s.t. $\sigma$ rules in $M|P$. For a set of models $\mathcal{M}|P$, some $\exists C \sigma X (BCQ)$ is a query of the form (2) s.t. Boolean CQ existentially, or fact according to whether $r$ contains some $\exists$-variable or not, respectively.

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

A conjunctive query (CQ) $q$, also denoted by $q(X)$, is of the form:

$$\exists Y \text{conj}_{[X \cup Y]}.$$ (2)

where $X$ and $Y$ are disjoint sets of variables, and $\text{conj}_{[X \cup Y]}$ is a conjunction of atoms from base($X \cup Y \cup \Delta_C$). Variables in $X$ 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. $X$ is empty. An atomic query is a CQ of the form (2) s.t. $\text{conj}_{[X \cup Y]}$ consists of just one atom.

2.3 Semantics

Let $M \subseteq \text{base}(\Delta_C \cup \Delta_N)$. $M$ is a model of a rule $r$ of the form (1), denoted by $M \models r$, if for each substitution $\sigma$ s.t. $\sigma(\text{body}(r)) \subseteq M$, there is a substitution $\sigma' \supseteq \sigma|_X$ s.t. $\sigma'(\text{head}(r)) \cap M \neq \emptyset$. $M$ is a model of a Datalog$_{\exists,\forall}$ program $P$, denoted by $M \models P$, if $M \models r$ for each $r \in P$. Let $\text{mods}(P)$ denote the set of all the models of $P$. Two programs $P, P'$ are called FO-equivalent if $\text{mods}(P) = \text{mods}(P')$.

A BCQ $q$ is true w.r.t. a model $M$, denoted by $M \models q$, if there is a substitution $\sigma$ s.t. $\sigma(\text{atoms}(q)) \subseteq M$. 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 $\text{mods}(P) \models q$.

The answer of a CQ $q(X)$ w.r.t. a set of models $\mathcal{M}$, denoted by $\text{ans}(q, \mathcal{M})$, is the set of substitutions $\sigma|_X$ s.t. $M \models \sigma|_X(q)$ for each $M \in \mathcal{M}$. The answer of $q(X)$ w.r.t. a program $P$, denoted by $\text{ans}_P(q)$, is the set $\text{ans}(q, \text{mods}(P))$. Note that for a BCQ $q$, either $\text{ans}_P(q) = \emptyset$ (if $P \not\models q$) or $\text{ans}_P(q) = \{\sigma|_\emptyset\}$ (if $P \models q$; $\sigma|_\emptyset$ is the identity mapping). The same consideration also applies to $\text{ans}(q, \mathcal{M})$.

2.4 The Query Answering Problem

Let $\mathcal{C}$ be a class of Datalog$_{\exists,\forall}$ programs whose terms belong to $\Delta_C \cup \Delta_N$. 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 we will call class $\mathcal{C}$ QA-decidable if QA over $\mathcal{C}$ is decidable.

We observe that computing $\text{ans}_P(q)$ for a CQ $q(X)$ of the form (2) is Turing-reducible to QA as defined above. In fact, $\text{ans}_P(q)$ is defined as the set of substitutions $\sigma|_X$ s.t. the BCQ $\sigma|_X(q)$ is true w.r.t. $P$. Since $\sigma|_X \in \text{ans}_P(q)$ implies $\sigma|_X(\Delta_N) \subseteq \text{terms}(P) \cap \Delta_C$, only finitely many substitutions have to be considered.
3 Universal Model Sets for Datalog$^{3,\lor}$ Programs

In this section we generalize the notion of universal model widely used in the context of QA over Datalog$^3$ programs. Intuitively, a universal model $M$ of a Datalog$^3$ program $P$ is such that each model of $P$ is homomorphic to a subset of $M$.

**Definition 1**

Let $P \in \text{Datalog}^{3,\lor}$. A set $M \subseteq \text{mods}(P)$ is a universal model set for $P$ if for each $M \in \text{mods}(P)$ there is $M' \in M$ and a homomorphism $h$ s.t. $h(M') \subseteq M$.

Universal model sets are sufficient for QA over Datalog$^{3,\lor}$ programs.

**Theorem 1**

If $M$ is a universal model set for $P$, then $P \models q$ iff $M \models q$ for each BCQ $q$.

**Proof**

$(\Rightarrow)$ Immediate because $M \subseteq \text{mods}(P)$ by Definition 1.

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

We now design a strategy for identifying a universal model set for a Datalog$^{3,\lor}$ program $P$. First, we introduce the notion of fires of a rule $r \in P$ on a set $R$ of Datalog$^{3,\lor}$ ground rules. Next, we define an instantiation procedure for computing a ground program $\text{inst}(P)$, 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 $\sigma$ s.t. $\sigma = \sigma|X$ and $\sigma(\text{body}(r)) \subseteq \text{heads}(R)$. The firing of $r$ on $R'$ w.r.t. $\sigma$ yields a ground rule $\hat{\sigma}(r)$, where $\hat{\sigma}$ is obtained by extending $\sigma|X$ as follows: $\exists$-variables in $Y$ are assigned to the least $|Y|$ nulls not occurring in $R \cup R'$. (We assume a fixed well-ordering of $\Delta_N$ and that variables in $Y$ 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 $\text{inst}(P)$.

**Procedure 1:** PROGRAM-INSTANTIATION

**Input:** A Datalog$^{3,\lor}$ program $P$  

**Output:** The ground program $\text{inst}(P)$

1. $R := \emptyset$;
2. repeat
3. $R' := \emptyset$;
4. foreach $r \in P$ and foreach unspent firing substitution $\sigma$ for $r$ w.r.t. $R$ do
5.   $R' := R' \cup \{\hat{\sigma}(r)\}$;
6. $R := R \cup R'$;
7. until $R' = \emptyset$;
8. return $R$;
Example 1
Let $\varphi_1 < \varphi_2 < \cdots$ be a well-ordering of $\Delta_N$. A run of Procedure 1 on the following program (obtained from the one given in the introduction by predicate renaming):

$$
\begin{align*}
    r_1 &: c(X) \lor h(X) \leftarrow a(X) \\
    r_2 &: \exists Y \ p(X,Y) \leftarrow c(X) \\
    r_3 &: a(Y) \leftarrow p(X,Y)
\end{align*}
$$

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 $\sigma_1$ s.t. $\sigma_1 = \sigma_1|_{\{X\}}$ and $\sigma_1(X) = \text{lion}$. The fire of $\sigma_1$ yields $p(\text{lion},\varphi_1) \leftarrow c(\text{lion})$, which is added into $R'$, and then moved into $R$. Now there is a firing substitution for $r_3$, namely $\sigma_2$ s.t. $\sigma_2 = \sigma_2|_{\{X,Y\}}$, $\sigma_2(X) = \text{lion}$ and $\sigma_2(Y) = \varphi_1$, whose fire yields $a(\varphi_1) \leftarrow p(\text{lion},\varphi_1)$. After adding this rule into $R'$, and then moving it into $R$, there is a new firing substitution for $r_1$, namely $\sigma_3$ s.t. $\varphi_3 = \varphi_3|_{\{X\}}$ and $\varphi_3(X) = \varphi_1$. The fire of $\varphi_3$ yields $c(\varphi_1) \lor h(\varphi_1) \leftarrow a(\varphi_1)$, which is added into $R'$, and then moved into $R$. Now there is a new firing substitution for $r_2$, namely $\sigma_4$ s.t. $\sigma_4 = \sigma_4|_{\{X\}}$ and $\sigma_4(X) = \varphi_1$, whose fire yields $p(\varphi_1,\varphi_2) \leftarrow c(\varphi_1)$. The procedure thus go on, indefinitely. Let $I = \{c(\text{lion}), p(\text{lion},\varphi_1), a(\varphi_1)\}$. Subset-minimal models of $\text{inst}(P)$ have the following forms:

$$
\begin{align*}
    &\bigcup_{i \in [1..k]} \{c(\varphi_i), p(\varphi_i,\varphi_{i+1}), a(\varphi_{i+1})\} \cup I \cup \{h(\varphi_{k+1})\}, \quad \forall k \geq 1; \\
    &\bigcup_{i \geq 1} \{c(\varphi_i), p(\varphi_i,\varphi_{i+1}), a(\varphi_{i+1})\} \cup I.
\end{align*}
$$

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

Lemma 1
Let $P$ be a Datalog$^{\exists,\forall}$ program and $P' = \text{inst}(P)$. For each $M \in \text{mods}(P)$ there exist $M' \in \text{mods}(P')$ and a homomorphism $h$ s.t.: (i) $M' \subseteq \text{heads}(P')$; (ii) $h(M') \subseteq M$; and (iii) $h = h|_{\text{terms}(P')}$.

Proof
Let $M \in \text{mods}(P)$ 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 $i \geq 0$, there exist $M_i \in \text{mods}(P_i)$ and a homomorphism $h_i$ s.t.: $M_i \subseteq \text{heads}(P_i)$; $h_i(M_i) \subseteq M$; and $h_i = h_i|_{\text{terms}(P_i)}$.

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 $i \geq 0$ 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 $h_i = h_i|_{\text{terms}(P_i)}$ by the induction hypothesis. If $h_i \circ \hat{\sigma}(\text{body}(r)) \subseteq M_i$, there is a substitution $\sigma' \supseteq (h_i \circ \hat{\sigma})|_{X}$ s.t. $\sigma'(\text{head}(r)) \cap M \neq \emptyset$ (because $M$ is a model of $P$ by assumption). Otherwise, if $h_i \circ \hat{\sigma}(\text{body}(r)) \not\subseteq M_i$, let $\sigma' = h_i \circ \hat{\sigma}$. Let $h_{i+1}$ be the homomorphism s.t. $t \in \hat{\sigma}(Y)$ implies $h_{i+1}(t) = \sigma'(t)$, and $t \notin \hat{\sigma}(Y)$ implies $h_{i+1}(t) = h_i(t)$. Let $M_{i+1}$ be the following set of atoms: $M_i \cup \hat{\sigma}(\{a \in \text{atoms}(r) \mid \sigma'(a) \in M\})$.

The following properties hold by construction: $M_{i+1} \subseteq \text{heads}(P_{i+1})$; $h_{i+1}(M_{i+1}) \subseteq $
\(M\); and \(h_{i+1} = h_{i+1}|_{\text{term}(P_{i+1})}\). 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 \(\text{mods}(\text{inst}(P))\), which allows for answering queries on \(P\) by performing the reasoning on \(\text{inst}(P)\).

**Theorem 2**

Let \(P\) be a Datalog\(^{3,\forall}\) program and \(P' = \text{inst}(P)\). Model set \(M = \{M \in \text{mods}(P') \mid M \subseteq \text{heads}(P')\}\) is universal for \(P\).

**Proof**

By Lemma 1, for each \(M \in \text{mods}(P)\) there is \(M' \in \mathcal{M}\) and a homomorphism \(h\) s.t. \(h(M') \subseteq M\). It remains to show that \(\mathcal{M} \subseteq \text{mods}(P)\), i.e., \(M \in \text{mods}(P')\) s.t. \(M \subseteq \text{heads}(P')\) implies \(M \in \text{mods}(P)\). Let \(r \in P\) and \(\sigma\) be a substitution s.t. \(\sigma(\text{body}(r)) \subseteq M\), so \(\sigma\) is a firing substitution for \(P'\). Let \(\hat{\sigma}(r)\) be the rule of \(P'\) obtained by the firing of \(r\). Thus, \(\text{head}(\hat{\sigma}(r)) \cap M \neq \emptyset\), 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\(^3\) program with a universal model. In fact, the oblivious chase procedure can be obtained from Procedure 1 by replacing line 5 with \(R' := R' \cup \hat{\sigma}(\text{head}(r))\), which is enough for Datalog\(^3\) programs.

**Corollary 1**

Let \(P\) be a Datalog\(^3\) program. Then, \(\{\text{heads}(\text{inst}(P))\}\) is universal for \(P\).

### 4 Extending guards-based classes to Datalog\(^{3,\forall}\)

We next define subclasses of Datalog\(^{3,\forall}\) relying on a well known paradigm, called guardedness, first introduced by Andrèka et al. (1998) in the definition of the guarded fragment of first-order logic and further revisited by Cali et al. (2008) for defining Datalog\(^3\) 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\(^{3,\forall}\) rule \(r\) is said to be guarded if it is of the form:

\[
\forall X \exists Y \ \text{disj}_{X \cup Y} \leftarrow \text{guard}_{X}, \ \text{s-conj}_{X''}, \quad (3)
\]

where \(X'\) and \(X''\) are subsets of \(X\), \(\text{guard}_{X}\) is an atom called guard and denoted by \(\text{guard}(r)\), \(\text{s-conj}_{X''}\) is a conjunction of atoms called sides and denoted by \(\text{sides}(r)\). Moreover, a guarded rule \(r\) is called: multi-linear if each side atom could be chosen as guard; linear if \(\text{sides}(r) = \emptyset\); monadic-linear if \(\text{sides}(r) = \emptyset\) and all head predicates are unary. Hereafter, a Datalog\(^{3,\forall}\) 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.

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\(^{3,\lor} \) program, \( a \) be an atom, and \( x \) a variable occurring in \( a \) at position \( i \). Position \( i \) of \( a \) is (inductively) marked as affected w.r.t. \( P \) if there is a rule \( r \in P \) with an atom \( b \in \text{head}(r) \) s.t. \( \text{pred}(b) = \text{pred}(a) \) and \( x \) is either an \( \exists \)-variable, or a \( \forall \)-variable s.t. \( x \) occurs in \( \text{body}(r) \) in affected positions only. A variable \( x \) 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\(^{3,\lor} \) program, and \( r \in P \) be a rule of the form:
\[
\forall X \exists Y \ \text{disj}_{[X \cup Y]} \leftarrow \text{wguard}_{[X'']} \cup \text{s-conj}_{[X'']} \quad \text{(4)}
\]
where \( X' \subseteq X = X'' \cup X''' \). Rule \( r \) is said to be weakly-guarded w.r.t. \( P \), if each variable in \( X''' \setminus X'' \) is unaffected in \( r \). Here, \( \text{guard}(r) \) and \( \text{sides}(r) \) still denote the (weak) guard and the side atoms of \( r \), respectively. In the following, \text{Weakly-Guarded-Datalog}\(^{3,\lor} \) will denote the set of Datalog\(^{3,\lor} \) programs where each rule either is weakly-guarded or has an empty body.

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

Proposition 1
Definitions 2 and 4 generalize the classes Guarded-Datalog\(^{3} \), Linear-Datalog\(^{3} \), and Weakly-Guarded-Datalog\(^{3} \) defined by Cali et al. (2008).

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

Theorem 3
Checking whether a program belongs to Guarded-Datalog\(^{3,\lor} \), Linear-Datalog\(^{3,\lor} \), or Weakly-Guarded-Datalog\(^{3,\lor} \) 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\(^{3,\lor} \) 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 guarded fragment of first-order logic (Andréka et al. 1998; Grädel 1999), here denoted by Guarded-FOL and inductively defined as follows: (i) \( \text{base}(\Delta_C \cup \Delta \lor) \subset \)
Guarded-FOL: (ii) if \( \psi_1, \psi_2 \in \text{Guarded-FOL} \), then \( \neg \psi_1, \psi_1 \lor \psi_2, \psi_1 \land \psi_2 \), and \( \psi_1 \leftarrow \psi_2 \) also belong to Guarded-FOL; and (iii) if \( a_{X \cup Y} \in \text{base}(<C \cup D> \cup E \land F) \), \( \psi(X' \cup Y') \in \text{Guarded-FOL} \), and the (free) variables of \( a \) include all the free variables \( X' \cup Y' \) of \( \psi \), then \( \exists Y(a_{X \cup Y} \land \psi(X' \cup Y')) \) and \( \forall X(\psi(X' \cup Y') \leftarrow a_{X \cup Y}) \) are also in Guarded-FOL.

Any Guarded-Datalog\(^{3,4} \) program can be viewed as a Guarded-FOL formula. 

**Proposition 2**
There is a logarithmic space transducer associating each Guarded-Datalog\(^{3,4} \) program with a FO-equivalent Guarded-FOL formula.

**Proof**
For a guarded Datalog\(^{3,4} \) rule \( r \) of the form \( \mathcal{B} \), let \( h_{i[X \cup Y]} \) be the \( i \)-th atom in \( \text{disj}_{X \cup Y} \), with \( i \in [1..k] \), \( X_i \subseteq X' \), and \( Y_i \subseteq Y \). Rule \( r \) is translated into the following FO-equivalent formula:

\[
\forall X(\exists Y_i h_{i[X_i \cup Y_i]} \lor \cdots \lor \exists Y_k h_{k[X_k \cup Y_k]} \lor \neg s\text{-conj}_{X'} \leftarrow \text{guard}_{|X|}).
\]

The whole disjunction is an expression \( \psi(X' \cup X'') \) in Guarded-FOL because each \( \exists Y_i h_{i[X_i \cup Y_i]} \) is equivalent to \( \exists Y_i (h_{i[X_i \cup Y_i]} \land h_{i[X_i \cup Y_i]}) \in \text{Guarded-FOL} \), and since \( \neg s\text{-conj}_{X'} \) trivially belongs to Guarded-FOL. Moreover, the expression \( \forall X(\psi(X' \cup X'') \leftarrow \text{guard}_{|X|}) \) is in Guarded-FOL since \( X' \cup X'' \subseteq X \). Finally, a similar construction applies to rules having empty bodies. \( \square \)

QA-decidability of Guarded-Datalog\(^{3,4} \) and its subclasses can now be established.

**Theorem 4**
Conjunctive QA is decidable under Guarded, Multi-Linear and Linear Datalog\(^{3,4} \).

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

In order to prove that Weakly-Guarded-Datalog\(^{3,4} \) is QA-decidable as well, we first introduce the notion of weak instantiation.

**Definition 5**
Let \( P \in \text{Weakly-Guarded-Datalog}^{3,4} \). For each \( r \in P \), let \( \text{winst}(r) \) denote the set of partially ground rules associated to \( r \) and consisting of the set \( \{ r \} \) or of the set \( \{ \sigma(r) \mid \sigma \text{ is a substitution from } X'' \setminus X' \text{ to terms}(P) \cap C \} \) according to whether rule \( r \) has an empty body or is of the form \( \mathcal{B} \), respectively. The weak instantiation of \( P \), denoted by \( \text{winst}(P) \), is defined as the union of \( \text{winst}(r) \) for each \( r \in P \). \( \square \)

The above definition transforms any Weakly-Guarded-Datalog\(^{3,4} \) program into a FO-equivalent Guarded-Datalog\(^{3,4} \) program.

**Lemma 2**
Let \( P \) be a Weakly-Guarded-Datalog\(^{3,4} \) program and \( P' = \text{winst}(P) \). Then, both \( P' \in \text{Guarded-Datalog}^{3,4} \) and \( \text{inst}(P) \simeq \text{inst}(P') \) hold.
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 $\sigma$ for a rule $r \in P$, there are $\sigma_1, \sigma_2$ s.t. $\sigma = \sigma_2 \circ \sigma_1$, where $\sigma_1$ is a substitution from $X'' \setminus X'$ to $\text{terms}(P) \cap \Delta_C$. Let $r' = \sigma_1(r)$. Therefore, $r' \in P'$ and $\sigma_2$ is a firing substitution for $r'$. Consider now the other direction. Let $\sigma'$ be a firing substitution for $r' \in P'$. Let $r' = \sigma(r)$, where $r \in P$ and $\sigma$ is a substitution from $X''' \setminus X''$ to $\text{terms}(P) \cap \Delta_C$. 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 \textit{Weakly-Guarded-Datalog}$_{\exists,\lor}$ is QA-decidable.

\textbf{Theorem 5}
Conjunctive QA is decidable under \textit{Weakly-Guarded-Datalog}$_{\exists,\lor}$.

\textbf{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 \textit{Datalog}$_{\exists,\lor}$ and queries. As usual in this setting, we assume that a \textit{Datalog}$_{\exists,\lor}$ program $P$ is paired with a (finite) database $D \subseteq \text{base}(\Delta_C)$. The set of ground facts \{\textit{a} $\leftarrow$ | a $\in$ D\} is denoted by $\overleftarrow{D}$. Similarly, $\overrightarrow{a}$ denotes the singleton \{a $\leftarrow$\} for some atom a $\in$ D. Finally, whenever $P$ contains a rule $r$ of the form $\text{disj} \leftarrow$ (even if $|\text{disj}| = 1$), we replace it in $P$ by $\text{disj} \leftarrow \text{edb}$ and we add to $D$ the extra (propositional) atom $\text{edb}$ of arity zero. Hereafter, we assume $D = \{a_1, \ldots, a_n\}$.

6.1 \textit{Guarded-Datalog}$_{\exists,\lor}$
We start by providing an upper bound for QA over \textit{Guarded-Datalog}$_{\exists,\lor}$.

\textbf{Theorem 6}
Data complexity of QA over \textit{Guarded-Datalog}$_{\exists,\lor}$ programs is in $\text{coNP}$.

\textbf{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 \textit{Guarded-FOL} formula is in $\text{coNP}$. The claim therefore follows from Proposition 2.

We now pinpoint the complexity of QA over \textit{Guarded-Datalog}$_{\exists,\lor}$.

\textbf{Theorem 7}
Data complexity of QA over \textit{Guarded-Datalog}$_{\exists,\lor}$ programs is $\text{coNP}$-complete in general, and it is $\text{coNP}$-hard already in the following cases:

1. A \textit{Monadic-Linear-Datalog}$_{\lor}$ program under an acyclic CQ.
2. A \textit{Multi-Linear-Datalog}$_{\lor}$ program under an atomic query.
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 setTimeout rule containing two head atoms. This result follows from Theorem 6.4 (and its proof) of Calvanese et al. (2009): Let \( \phi \) 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 \( \text{lit}(x) \) for each propositional variable \( x \), and atoms \( p_1(c,x_1), p_2(c,x_2), n_1(c, x_3), n_2(c,x_4) \) for each clause \( x_1 \lor x_2 \lor \neg x_3 \lor \neg x_4 \) having \( c \) as identifier. Let \( P \) be a Monadic-Linear-Datalog setTimeout program consisting of the following rule: \( t(X) \lor f(X) \leftarrow \text{lit}(x) \), and \( q \) be the following acyclic CQ: \( \exists \ C, P_1, P_2, N_1, N_2 \ p_1(C, P_1), f(P_1), p_2(C, P_2), f(P_2), n_1(C, N_1), t(N_1), n_2(C, N_2), t(N_2). \)

Hence, \( \phi \) is unsatisfiable if and only if \( P \cup \overline{D} \models q \).

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

\[
\begin{align*}
\text{sel}(L_1, N_1) & \lor \text{sel}(L_2, N_2) \lor \text{sel}(L_3, N_3) \leftarrow \text{clause}(L_1, L_2, L_3, N_1, N_2, N_3). \\
\text{wrongAssignment} & \leftarrow \text{sel}(L, N), \text{sel}(N, L).
\end{align*}
\]

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

\[\square\]

6.2 Weakly-Guarded-Datalog\(^{3,\lor}\)

As in the disjunction-free case, the complexity of QA over Weakly-Guarded-Datalog\(^{3,\lor}\) is harder than QA over Guarded-Datalog\(^{3,\lor}\).

Theorem 8

Data complexity of QA over Weakly-Guarded-Datalog\(^{3,\lor}\) is EXP-complete in general, and it is EXP-hard already for atomic queries over Weakly-Guarded-Datalog\(^{3}\).

Proof

Hardness comes from the EXP-hardness of Weakly-Guarded-Datalog\(^{3}\) (Cali et al. 2008). As for the membership, let \( P \) be a Weakly-Guarded-Datalog\(^{3,\lor}\) program and \( P' = \text{winst}(P \cup \overline{D}) \) be the Guarded-Datalog\(^{3,\lor}\) program built according to Definition 5. By Lemma 2, \( P \cup \overline{D} \models q \) if and only if \( P' \models q \). Moreover, let \( k \) be the maximum number of unguarded (thus unaffected) variables appearing in some rule of \( P \), \( \gamma \) be the number of constants occurring in \( P \), and \( w \) be the maximum arity over all predicate symbols in \( P \cup \overline{D} \). We point out that \( |P'| \leq |D| + |P| \cdot (w \cdot |D| + \gamma)^k \). 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 \( P' \models q \). \[\square\]
6.3 Atomic Queries over Linear-Datalog$^{3,\forall}$

In the following, let $P$ be a Linear-Datalog$^{3,\forall}$ program and $q$ be Boolean atomic query. As before, $D = \{a_1, \ldots, a_n\}$ is the input database. We first introduce a decomposition property relying on the structure of $P$.

**Lemma 3**

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

**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 $P \cup \bar{D}$, we have to show that whenever for a rule $r \in P$ there exists a substitution $\sigma$ s.t. $\sigma(\text{body}(r)) \subseteq M$, then $M \models \sigma(\text{head}(r))$. Let us fix a pair $(r, \sigma)$ s.t. $\sigma(\text{body}(r)) \subseteq M$. Since $P$ is linear, there is $i \in [1..n]$ s.t. $\sigma(\text{body}(r)) \subseteq M_i$. But since $M_i$ is a model of $P \cup \bar{a}_i$, then $M_i \models \sigma(\text{head}(r))$. Finally, the implication holds since $M_i \subseteq M$.

$(\supseteq)$ Let $M$ be a model of $P \cup \bar{D}$. For each $i \in [1..n]$, $M$ is also a model of $P \cup \bar{a}_i$. Consequently, the $n$-tuple $\langle M_1, \ldots, M \rangle$ belongs to $C$, entailing that $M \in M$.

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

**Lemma 4**

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

**Proof**

$(\Rightarrow)$ We prove the contrapositive. Let us assume that $\forall i \in [1..n] \ P \cup \bar{a}_i \not\models q$. 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 $P \cup \bar{D} \not\models q$.

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

Lemma 3 allows for focusing the analysis on a single database atom, say $a \in D$. The instantiation-tree for $P \cup \bar{a}$ is the directed acyclic graph $T = \text{tree}(P \cup \bar{a})$ inductively constructed as follows: (i) the root of $T$ is a node labeled with $\bar{a}$; (ii) for each node $m$ of $T$ and for each rule $r \in \text{inst}(P \cup \bar{a})$ s.t. $\text{body}(r)$ 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 $\text{nodes}(T)$ and $\text{arcs}(T)$ denote the nodes and arcs of $T$, respectively; $\text{label}(n)$ denotes the ground rule used as label for $n$; $n \in T$ is short for $n \in \text{nodes}(T)$; $\text{subtree}(n)$ is the tree below $n$; finally, $\text{depth}(n)$ 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 $P \cup \bar{a}$, denoted by $\text{stem}(P \cup \bar{a})$, is the maximal subtree that can be obtained starting from the root of $\text{tree}(P \cup \bar{a})$ in such a way that each path contains no nodes labelled with rules with isomorphic bodies. Finally, $\text{inst}(P \cup \bar{a})$ denotes the set $\{\text{label}(n) \mid n \in \text{stem}(P \cup \bar{a})\}$. 


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

\[
\exists Y \ a(Y) \leftarrow d(X) \\
d(X) \leftarrow b(X) \\
b(X) \lor c(X) \leftarrow a(X) \\
e(X) \leftarrow d(X)
\]

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

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

Proof
The statement holds if $m = \text{root}(T)$ or $n = \text{root}(T)$ 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), $\text{body}(\text{label}(m)) \subseteq \text{head}(\text{label}(m_p))$ and $\text{body}(\text{label}(n)) \subseteq \text{head}(\text{label}(n_p))$. Let $\text{label}(n) = \hat{\sigma}(r)$, where $r$ is a rule and $\hat{\sigma}$ is a substitution. Let $h$ be the isomorphism between $\text{body}(\text{label}(m))$ and $\text{body}(\text{label}(n))$. Thus, there is a child $m'$ of $m$ s.t. $\text{label}(m') = h \circ \sigma(r)$, which in turn implies $\text{label}(m') \simeq \text{label}(n)$. We now use induction. Let $n, n_1, \ldots, n_k$ and $m', m'_1, \ldots, m'_k$ ($k \geq 0$) be two isomorphic paths in $\text{subtree}(n)$ and $\text{subtree}(m')$, respectively. Still by construction, there is a one-to-one mapping $\mu$ 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 $\text{label}(n_{k+1}) \simeq \text{label}(\mu(n_{k+1}))$. □

Given a model $M$ of $\text{sinst}(P \cup \overline{\text{a}})$, we shall show how to build a model $M^*$ of $\text{inst}(P \cup \overline{\text{a}})$ s.t. $M^* \models q$ implies $M \models q$. Let $S = \text{stem}(P \cup \overline{\text{a}})$, and $C_0$ be the smallest subset of $\text{nodes}(S)$ satisfying the following properties: (i) $\text{root}(S) \in C_0$; (ii) $n \in C_0$ whenever its parent belongs to $C_0$ and $\text{body}(\text{label}(n)) \subseteq M$ holds. We can thus restrict model $M$ as follows: $M_0 = \{ b \in M \mid b \in \text{head}(\text{label}(n)) \land n \in C_0 \}$.
Example 3
Consider again the instantiation-tree reported in Fig. 1. Let \( M = \{ h(0) \} \cup \{ s(\varphi_i) \mid i \geq 1 \} \cup \{ b(\varphi_i) \mid i \geq 1 \} \cup \{ c(\varphi_i) \mid i \geq 2 \} \cup \{ d(\varphi_i) \mid i \geq 1 \} \cup \{ e(\varphi_i) \mid i \geq 1 \} \). Nodes in \( C_0 \) are those colored in gray, and \( M_0 = M \setminus \{ c(\varphi_i) \mid i \geq 2 \} \). Note that \( M_0 \) is still a model of the program, as formally established by the next lemma. \( \Box \)

Lemma 6
If \( M \) is a model of \( \text{sinst}(P \cup \bar{\alpha}) \), then also \( M_0 \) is.

Proof
Let \( n \in \text{nodes}(S) \setminus C_0 \), \( m \) be the parent of \( n \), \( b \) be the unique atom in \( \text{body}(\text{label}(n)) \), and \( b \in M_0 \). We claim that \( \text{head}(\text{label}(n)) \cap M_0 \neq \emptyset \). By Procedure 1, since \( b \in M_0 \), then \( b \in \text{head}(\text{label}(m)) \). 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. \( b \in \text{head}(\text{label}(m')) \). But this means, since \( b \in M_0 \), that there is a child \( n' \) of \( m' \) s.t. \( n' \in C_0 \) and \( \text{label}(n') = \text{label}(n) \). However, since by construction the head of each node in \( C_0 \) has a nonempty intersection with \( M \), then \( \text{head}(\text{label}(n')) = \text{head}(\text{label}(n)) \) has a nonempty intersection with \( M_0 \). \( \Box \)

From \( T = \text{tree}(P \cup \bar{\alpha}) \), we define a total function \( f : \text{nodes}(T) \rightarrow \text{nodes}(T) \) as follows: For each node \( n \in S = \text{stem}(P \cup \bar{\alpha}) \), \( f(n) = n \). For the remaining nodes, let \( n \in \text{nodes}(T) \setminus \text{nodes}(S) \) s.t. its parent belongs to \( S \). Let \( m \) be the (unique) node in the path from \( \text{root}(T) \) to \( n \) s.t. \( \text{body}(\text{label}(m)) \simeq \text{body}(\text{label}(n)) \). Let \( m' \) be either \( m \) or one of its siblings according to whether \( \text{label}(m') \simeq \text{label}(n) \). Function \( f \) thus maps \( \text{subtree}(n) \) into \( \text{subtree}(m') \); 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 \( \text{sinst}(P \cup \bar{\alpha}) \) 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 \( n \in \text{nodes}(T) \setminus \text{nodes}(S) \) s.t. both \( \text{parent}(n) \in C^* \) and \( \text{body}(\text{label}(n)) \subseteq M^* \), \( C^* \) is augmented by \( n \) and \( M^* \) is augmented by the set \( \{ b \in \text{head}(\text{label}(n)) \mid h(a) \in M^* \} \) 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 \( \text{sinst}(P \cup \bar{\alpha}) \models q \) if and only if \( \text{sinst}(P \cup \bar{\alpha}) \models q \).

Proof
\((\Leftarrow)\) Since \( \text{sinst}(P \cup \bar{\alpha}) \subseteq \text{sinst}(P \cup \bar{\alpha}) \), each model of \( \text{sinst}(P \cup \bar{\alpha}) \) is also a model of \( \text{sinst}(P \cup \bar{\alpha}) \).

\((\Rightarrow)\) Let us assume that \( \text{sinst}(P \cup \bar{\alpha}) \models q \) holds. Let \( M \) be a model of \( \text{sinst}(P \cup \bar{\alpha}) \). Since, by construction, \( M^* \) is a model of \( \text{sinst}(P \cup \bar{\alpha}) \), and since \( M^* \models q \) by hypothesis, then \( M \models q \) holds. \( \Box \)

Tractability of atomic QA over \( \text{Linear-Datalog}^{\exists \forall} \) can now be established.

Theorem 9
Data complexity of atomic QA over \( \text{Linear-Datalog}^{\exists \forall} \) programs is in \( \text{LOGSPACE} \).
Proof
Armed with Lemma 7, a logspace procedure iterates the database atoms looking
for an atom $a \in D$ s.t. $\text{sin}(P \cup \overline{\text{a}}) \models q$. In fact, for each $n \in \text{stem}(P \cup \overline{\text{a}})$, $\text{depth}(n) < |\pi| \cdot (2w)^w$, where $w$ is the maximum arity over all predicate symbols in $P$, and $\pi$ is the number of predicate symbols occurring in $P$. Therefore, cardinality of the ground program $\text{sin}(P \cup \overline{\text{a}})$ 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 6 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 cell in column $\{\vee\}$) follow from Eiter et al. (1997), since this class coincides with Datalog\$\vee$. All the remaining $\text{coNP}$-completeness results follow from Theorem 7 in Section 6.1, the remaining $\text{EXP}$-completeness results follow from Theorem 8 in Section 6.2 and the $\text{LOGSPACE}$ 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 $\text{AC}_0$ to $\text{coNP}$-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$^{\exists, \vee}$.

<table>
<thead>
<tr>
<th>Datalog Restrictions</th>
<th>Query Structure</th>
<th>\text{Datalog Extensions}</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Monadic-)Linear</td>
<td>AQ</td>
<td>in AC$_0$ in LOGSPACE in LOGSPACE</td>
</tr>
<tr>
<td></td>
<td>ACQ/ACQ/CQ</td>
<td>in AC$_0$ coNP-complete coNP-complete</td>
</tr>
<tr>
<td>Multi-Linear</td>
<td>AQ/ACQ/CQ</td>
<td>in AC$_0$ coNP-complete coNP-complete</td>
</tr>
<tr>
<td>Guarded</td>
<td>AQ/ACQ/CQ</td>
<td>P-complete coNP-complete coNP-complete</td>
</tr>
<tr>
<td>Weakly-Guarded</td>
<td>AQ/ACQ/CQ</td>
<td>EXP-complete coNP-complete EXP-complete</td>
</tr>
</tbody>
</table>
Let us now discuss the impact of adding existential quantification in the presence of disjunction in rule heads, i.e. columns \( \{ \lor \} \) versus \( \{ \exists, \lor \} \). 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 \( \text{coNP} \)-completeness to \( \text{EXP} \)-completeness and thus provable intractability.

In future work, we intend to investigate on the exact data complexity of atomic QA over (Monadic-)Linear-Datalog\(^3,\lor\) programs, in particular whether it is in \( \text{AC}_0 \) or not. We also intend to study the impact of disjunction on other tractable fragments of Datalog\(^3\) based on different paradigms, for example stickiness (Calì et al. 2010a), shyness (Leone et al. 2012) and weak-acyclicity (Pagin 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\(^3\) (Leone et al. 2012), is on our agenda.

7 Acknowledgments

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


