Epistemic Disjunctive Datalog for Querying Knowledge Bases

Gianluca Cima, Marco Console, Maurizio Lenzerini, Antonella Poggi

Sapienza University of Rome
{cima, console, lenzerini, poggi}@diag.uniroma1.it

Abstract

The Datalog query language can express several powerful recursive properties, often crucial in real-world scenarios. While answering such queries is feasible over relational databases, the picture changes dramatically when data is enriched with intensional knowledge. It is indeed well-known that answering Datalog queries is undecidable already over lightweight knowledge bases (KBs) of the DL-Lite family. To overcome this issue, we propose a new query language based on Disjunctive Datalog rules combined with a modal epistemic operator. Rules in this language interact with the queried KB exclusively via the epistemic operator, thus extracting only the information true in every model of the KB. This form of interaction is crucial for not falling into undecidability. The contribution provided by this paper is threefold. First, we illustrate the syntax and the semantics of the novel query language. Second, we study the expressive power of different fragments of our new language and compare it with Disjunctive Datalog and its variants. Third, we outline the precise data complexity of answering queries in our new language over KBs expressed in various well-known formalisms.

Introduction

The query language that has received the most attention for querying knowledge bases (KBs) is *(unions of) conjunctive queries* ((U)CQ), i.e., the existential positive fragment of first-order logic. There are, however, several interesting scenarios where UCQs fail to express relevant information needs. For example, having some form of negation, including inequality, is of great importance in the context of data quality checking (Console and Lenzerini 2020). Another notable example is navigational queries, i.e., queries based on singling out paths in a graph or characterized by some form of recursion, as in Datalog. In the literature, there are several studies on adding such features to UCQs. Unfortunately, the results show that, as soon as we try to extend the UCQ class with either unlimited recursion or uncontrolled use of inequalities/negation, we get undecidability of query answering already for lightweight KBs of the DL-Lite family (Levy and Rousset 1998; Calvanese and Rosati 2003; Gutiérrez-Basulto et al. 2015). Other studies tried to explore limited versions of recursion and negation to be added to UCQ, with the hope of recovering decidability, if not tractability. Unions of conjunctive two-way regular path queries (Bienvenu, Ortiz, and Simkus 2015), UCQ with limited inequalities (Cima, Lenzerini, and Poggi 2020) and EQL-Lite (Calvanese et al. 2007a) are successful examples of such attempts. Each of them, however, lacks features of the others.

The goal of this work is to present a novel query language for KBs, called DataKlog, whose queries can use recursion, inequalities, negation, and an epistemic operator on top of UCQ without falling into undecidability. This new language allows for rules of the forms used in the Disjunctive Datalog family (Eiter, Gottlob, and Mannilla 1997), limiting their interaction with KBs through the use of a modal epistemic operator $\mathbf{K}$. In this way, only facts that are known to be true in every model of the queried KB interact with the recursive part of the query, thus overcoming the drawbacks of less restricted languages. Although simple, this idea defines a query language for KBs with several interesting features:

1. DataKlog can define a powerful form of navigational queries even when the (typically SPARQL) end-point used to access the KB does not explicitly allow them. In KBs structured as knowledge graphs (Hogan et al. 2021), this feature can be crucial for computing the transitive closure of relevant relations (e.g., *subset-of*), thus enabling meta-querying, i.e., queries mixing instance-based and schema-based traversal of paths in the graph.
2. By using an epistemic operator, our language can distinguish between what is known to be true for some elements and what is known to be true for some *known* elements. As we show later, this seemingly simple mechanism enables to express queries that are provably not expressible in Disjunctive Datalog and its variants.
3. Combining the epistemic operator with the use of negation, our language can capture expressive classes of queries without falling into undecidability. For example, adding a stratified form of negation to positive rules is sufficient to express various types of non-monotonic queries and the whole class of first-order queries over the certain facts extracted from the knowledge base.

We now illustrate some of the basic characteristics of DataKlog via an example. Let a portion of a KB (in DL notation) supporting health care personnel be:

$\exists \mathsf{hasInfected} \sqsubseteq \mathsf{Pat}, \quad \exists \mathsf{hasInfected}^- \sqsubseteq \mathsf{Pat},$

$\mathsf{Pat} \sqsubseteq \exists\mathsf{hasVariant}, \quad \mathsf{Pat0} \sqsubseteq \mathsf{Pat}, \quad \mathsf{Pat0(John)}$

stating that (*i*) every individual who has infected someone or was infected by someone is a patient, (*ii*) every individual who is a patient has a Covid variant, (*iii*) every patient zero is a patient, and (*iv*) John is a patient zero. Now, suppose that the healthcare personnel aims at identifying all patients whose variant is *known*. This can be done by computing the answers to the following DataKlog query:

$$\forall x.\exists v.\mathbf{K}(\mathsf{hasVariant}(x, v)) \rightarrow Ans(x),$$

where the formula $\mathbf{K}\phi$ is read as "ϕ is known to hold (in the KB)". We point out that this is different from the query obtained by removing the epistemic operator $\mathbf{K}$, which asks for all patients who have a variant. In fact, once posed over the KB above, the former would return an empty answer, while the latter would return John, since we know that John has a variant although we do not know which one.

Now, to appreciate the expressive power of our language, suppose that the healthcare personnel aims at checking whether it is possible to visit exactly once all patients whose variant is known, following the order in which they were infected, i.e., starting from a patient zero and testing a before b only if a has infected b. This can be done by checking the answers to the following DataKlog query:

$$\forall x.\exists v.\mathbf{K}(\mathsf{hasVariant}(x, v)) \rightarrow N(x) \quad (1)$$
$$\forall x, y.\mathbf{K}(\mathsf{hasInfected}(x, y) \rightarrow E(x, y) \quad (2)$$
$$\forall x.\mathbf{K}(\mathsf{Pat0}(x)) \rightarrow S(x) \quad (3)$$
$$\forall x.S(x) \land N(x) \rightarrow R(x) \quad (4)$$
$$\forall x, y.R(x) \land E(x, y) \land N(y) \rightarrow InP(x, y) \lor OutP(x, y) \quad (5)$$
$$\forall x, y.InP(x, y) \rightarrow R(y) \quad (6)$$
$$\forall x, y, z.InP(x, y) \land InP(x, z) \land y \neq z \rightarrow Ans() \quad (7)$$
$$\forall x, y, z.InP(x, y) \land InP(z, y) \land x \neq z \rightarrow Ans() \quad (8)$$
$$\forall x.N(x) \land \neg R(x) \rightarrow Ans() \quad (9)$$

The query defines the predicates N as the set of known patients whose variant is known, E as the set of known pairs of patients such that the former has infected the latter, and S as the set consisting of the known patients zero. Then, the query exploits non-determinism, recursion, and (stratified) negation within rules (4)–(9), to obtain the answer by means of the predicate Ans. Indeed, this predicate will be false in at least one *stable model* (Gelfond and Lifschitz 1988) if and only if there exists an Hamiltonian path in the graph defined by N and E and starting from a node in S (the patient zero John in the case of the KB).

Contributions The main contributions provided by this paper can be summarized as follows.

We first define the syntax and the semantics of the DataKlog($\mathcal{Q}$) family of query languages, where the parameter $\mathcal{Q}$, called the *embedded query language*, denotes the query language that can be used to access the KB. Over the certain answers obtained by querying the KB through queries in $\mathcal{Q}$, DataKlog($\mathcal{Q}$) allows using typical constructs of Disjunctive Datalog. The various members of the family are characterized by suitable restrictions on the use of negation, disjunction, and inequalities.

We then investigate the expressive power of various members of the DataKlog(CQ) family and compare them with other relevant languages.

We prove that, for any query language $\mathcal{Q}$ and KB language $\mathcal{L}$, if answering $\mathcal{Q}$ queries over $\mathcal{L}$ KBs is decidable and answers to queries return only constants occurring in the queried KB (a condition that most combinations of query languages $\mathcal{Q}$ and KB languages $\mathcal{L}$ satisfy), then answering the most general member of DataKlog($\mathcal{Q}$) queries over $\mathcal{L}$ KBs is decidable as well. Moreover, we sharpen the study of the data complexity of query answering in (several members of) DataKlog($\mathcal{Q}$), by focusing on various combinations of embedded query ($\mathcal{Q}$) and KB ($\mathcal{L}$) languages.

Related Work The idea of enriching rule-based formalisms à-la Datalog with epistemic capabilities have been explored in the context of relational database queries (Gelfond 1991; Fandinno, Faber, and Gelfond 2022). Usually, rules of these languages can use the epistemic operator freely to define conditions. In contrast, DataKlog($\mathcal{Q}$) defines rule-based KB queries where the epistemic operator regulates the otherwise problematic interaction with the KBs. To the best of our knowledge, ours is the first attempt of this kind.

DataKlog($\mathcal{Q}$) shares similarities also with other languages in the literature. A first attempt to define KB queries with epistemic capabilities was done with the EQL-Lite($\mathcal{Q}$) family (Calvanese et al. 2007a) of which DataKlog($\mathcal{Q}$) is a significant extension. A similar extension is used in (Cima et al. 2022) to define *monotonic* query abstractions. Besides basic computational properties, however, not much was known of the resulting language. The results presented in this paper shed some light on its expressive power and complexity. Moreover, DataKlog($\mathcal{Q}$) shares some similarities also with HEX-programs (Eiter et al. 2005) and their precursors DL-programs (Eiter et al. 2008b). Specifically, all these languages consist of rule-based queries that can access external data sources via a special set of rules. However, DL-programs define external sources using only a very limited query language (essentially, Description Logic expressions) while HEX-programs use them as black boxes and impose no definability condition. In contrast, the embedded query language $\mathcal{Q}$ of DataKlog($\mathcal{Q}$) is a parameter of the framework and can be formally analyzed within it.

Finally, besides what already mentioned, there have been several attempts at integrating rules with ontological reasoning. Related to our work are the so-called *hybrid* formalisms (Cadoli, Palopoli, and Lenzerini 1997; Donini et al. 1998; Motik, Sattler, and Studer 2005) of which the MKNF$^+$ framework (Motik and Rosati 2010) is a notable example. MKNF$^+$ KBs consist of two components: one based on DLs, the other based on epistemic Datalog rules. Semantics of these KBs is defined via the MKNF logic (Lifschitz 1991). In spite of their similarities, MKNF$^+$ is conceived to represent intensional knowledge while DataKlog($\mathcal{Q}$) is a query language. The fine-grained analysis presented here requires KBs and queries expressions to be separated.

Preliminaries

Computational Complexity We make use of standard computational complexity classes, such as P (polynomial time), NP, coNP, $\Delta_2^p = \mathrm{P}^{\mathrm{NP}}$, $\Pi_2^p = \mathrm{coNP}^{\mathrm{NP}}$, and EXPTIME. For a complexity class $\mathcal{C}$, we denote by $\mathrm{P}_{||}^{\mathcal{C}}$ the complexity class containing those decision problems recognizable by a Turing machine that runs in polynomial time to which it is allowed a *constant number of rounds of parallel queries* to an oracle for a decision problem in $\mathcal{C}$. By a round of parallel queries, we mean that the Turing machine can ask for polynomially many *non-adaptive* queries to the $\mathcal{C}$-oracle. While $\mathrm{P}_{||}^{\mathcal{C}} = \mathcal{C}$ for both $\mathcal{C} = \mathrm{P}$ and $\mathcal{C} = \textsc{ExpTime}$, from results in (Buss and Hay 1991), it is known that $\mathrm{P}_{||}^{\mathrm{NP}} = \mathrm{P}_{||}^{\mathrm{coNP}}$ coincides with the complexity class Θ_2^p (see (Wagner 1990) for further characterizations of this class).

Knowledge bases We refer to a *signature* Σ, constituted by three pairwise disjoint and countably infinite sets of symbols $\Sigma_\mathcal{R}$, $\Sigma_\mathcal{C}$, and $\Sigma_\mathcal{V}$ for predicates, constants (a.k.a. individuals), and variables, respectively. An $\mathcal{L}$ *knowledge base (KB)* $\mathcal{K}$ over Σ is a pair $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is called the TBox and $\mathcal{A}$ is called the ABox. The TBox $\mathcal{T}$ represents the intensional knowledge of a modeled domain formulated in $\mathcal{L}$ (this latter also referred as a *KB language*), i.e., it is a finite set of assertions allowed in the language $\mathcal{L}$ using symbols from $\Sigma_\mathcal{R}$ as predicates. The ABox $\mathcal{A}$ represents the extensional knowledge of the domain, i.e., a finite set of facts (a.k.a. ground assertions) of the form $R(\bar{c})$, where R is an n-ary predicate in $\Sigma_\mathcal{R}$ and $\bar{c}$ is an n-tuple of constants occurring in $\Sigma_\mathcal{C}$. We denote by $dom(\mathcal{K})$ the *active domain* of $\mathcal{K}$, i.e., the subset $dom(\mathcal{K}) \subseteq \Sigma_\mathcal{C}$ of the constants mentioned in the ABox $\mathcal{A}$. A KB $\mathcal{K} = \langle \emptyset, \mathcal{A} \rangle$ without assertions in the TBox is called *a trivial KB*, and we let $\mathcal{L}_{triv}$ be the KB language allowing only for trivial KBs.

In this paper, the semantics of a KB with signature Σ is given in terms of FOL interpretations over Σ following the *standard name assumption* (Levesque and Lakemeyer 2001) (which implies the *unique name assumption*), i.e., interpretations $\mathcal{I} = \langle \Sigma_\mathcal{C}, \cdot^\mathcal{I} \rangle$ with domain $\Sigma_\mathcal{C}$ and *interpretation function* $\cdot^\mathcal{I}$ assigning to each n-ary predicate $R \in \Sigma_\mathcal{R}$ an n-ary relation $R^\mathcal{I} \subseteq \Sigma_\mathcal{C}^n$ and to each constant itself, i.e. $c^\mathcal{I} = c$ for each constant $c \in \Sigma_\mathcal{C}$. Abusing notation, we will also denote by $R(\bar{c}) \in \mathcal{I}$ the fact that $\bar{c} \in R^\mathcal{I}$. An interpretation $\mathcal{I}$ is a *model* of $\mathcal{K}$ if $\mathcal{I}$ satisfies all the assertions in $\mathcal{K}$. We denote by $mod(\mathcal{K})$ the set of models of a KB $\mathcal{K}$ and say that a KB $\mathcal{K}$ is *satisfiable* if $mod(\mathcal{K}) \neq \emptyset$.

Queries An n-ary query q over a signature Σ is a function that, given either an interpretation over Σ or a KB over Σ, returns a subset of $\Sigma_\mathcal{C}^n$. When q is applied to an interpretation $\mathcal{I}$ over Σ, the result of evaluating q is denoted by $q(\mathcal{I})$. When q is applied to a KB $\mathcal{K}$ over Σ, the result of evaluating q is denoted by $ans(q, \mathcal{K})$, called the *answers to q w.r.t.* $\mathcal{K}$. We impose $ans(q, \mathcal{K}) = ans(q, \mathcal{K}')$ if $mod(\mathcal{K}) = mod(\mathcal{K}')$.

A query language is a formalism for defining queries and we assume familiarity with the *first-order with equality* (or simply, FOL) query language and the Disjunctive Datalog family of query languages (Eiter, Gottlob, and Mannilla 1997). An n-ary FOL query is an expression of the form $\{\bar{x} \mid \phi(\bar{x})\}$, where $\bar{x}$ is an n-tuple of variables in $\Sigma_\mathcal{V}$ and $\phi(\bar{x})$ is a first-order formula using $\Sigma_\mathcal{R} \cup \{=\}$ for predicate symbols and $\Sigma_\mathcal{V} \cup \Sigma_\mathcal{C}$ for term symbols. An equality atom $=(t_1, t_2)$ will be written as $t_1 = t_2$, and the negation of an equality atom $t_1 = t_2$ will be written as $t_1 \neq t_2$ and will be called inequality atom. Note that the characterization of the evaluation of a FOL query q over an interpretation $\mathcal{I}$ is the standard one (Abiteboul, Hull, and Vianu 1995). We are also interested in the following fragments of FOL: *atomic queries* (AQ), i.e. queries of the form $\{\bar{x} \mid R(\bar{x})\}$, *(unions of) conjunctive queries* ((U)CQ), *unions of conjunctive queries with inequalities* ($\mathsf{UCQ}^{\neq}$), and *unions of conjunctive queries with negated atoms* ($\mathsf{UCQ}^{\neg}$). We refer to (Eiter, Gottlob, and Mannilla 1997) for the Disjunctive Datalog family of query languages. Here, we only mention that, given a query q in some language of this family and an interpretation $\mathcal{I}$, $q(\mathcal{I})$ is defined according to the *stable model semantics* (Gelfond and Lifschitz 1988).

As for the case when an n-ary query q expressed either as a FOL query or in one language of the Disjunctive Datalog family is applied to a KB $\mathcal{K}$, as usually done in the context of KR, $ans(q, \mathcal{K})$ is defined according to the *certain answers* semantics, i.e., $ans(q, \mathcal{K}) = \{\bar{c} \in \Sigma_\mathcal{C}^n \mid \bar{c} \in q(\mathcal{I})$ for each $\mathcal{I} \in mod(\mathcal{K})\}$.

Finally, for a query q, a KB language $\mathcal{L}$, and a query language $\mathcal{Q}$ which is a sublanguge of FOL or Disjunctive Datalog, we say that q is $\mathcal{Q}$*-rewritable in* $\mathcal{L}$ if, for each TBox $\mathcal{T}$ formulated in $\mathcal{L}$, there is a query $q_\mathcal{T} \in \mathcal{Q}$ such that $ans(q, \mathcal{K}) = q_\mathcal{T}(\mathcal{I}_\mathcal{A})$ holds for any KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{I}_\mathcal{A}$ is the interpretation such that $R(\bar{c}) \in \mathcal{I}_\mathcal{A}$ iff $R(\bar{c}) \in \mathcal{A}$.

Epistemic queries We recall the basis of EQL (Calvanese et al. 2007a), a first-order modal language with equality and with a modal operator $\mathbf{K}$ used to formalize the *epistemic state* of a KB. A query $\psi(\bar{x})$ in EQL for a KB $\mathcal{K}$ over the signature Σ is built according to the following syntax:

$$\psi(\bar{x}) ::= \mathbf{K}\varrho \mid \psi_1 \wedge \psi_2 \mid \exists z.\psi \mid w_1 \neq w_2 \mid \neg\psi,$$

where $\bar{x}$ are the free variables of the query, z denotes a fresh existentially quantified variable, w_1 (resp. w_2) denotes either a free variable $x_i \in \bar{x}$ or a freshly introduced existential variable, and ϱ is a query expression in $\mathcal{Q}$ using $\Sigma_\mathcal{R} \cup \{=\}$ as predicate symbols. The semantics is based on *epistemic interpretation* for $\mathcal{K}$, i.e., pairs $\langle \mathcal{W}, \mathcal{I} \rangle$, where $\mathcal{W}$ is a set of FOL interpretations over Σ and $\mathcal{I} \in \mathcal{W}$. An EQL sentence ψ is true in $\langle \mathcal{W}, \mathcal{I} \rangle$, ($\langle \mathcal{W}, \mathcal{I} \rangle \models \psi$), if the following holds:

$$\begin{array}{lll}
\langle \mathcal{W}, \mathcal{I} \rangle \models c_1 = c_2 & \text{iff} & c_1 = c_2 \\
\langle \mathcal{W}, \mathcal{I} \rangle \models P(\vec{c}) & \text{iff} & \mathcal{I} \models P(\vec{c}) \\
\langle \mathcal{W}, \mathcal{I} \rangle \models \psi_1 \wedge \psi_2 & \text{iff} & \langle \mathcal{W}, \mathcal{I} \rangle \models \psi_1 \text{ and } \langle \mathcal{W}, \mathcal{I} \rangle \models \psi_2 \\
\langle \mathcal{W}, \mathcal{I} \rangle \models \neg\psi & \text{iff} & \langle \mathcal{W}, \mathcal{I} \rangle \not\models \psi \\
\langle \mathcal{W}, \mathcal{I} \rangle \models \exists z.\psi & \text{iff} & \langle \mathcal{W}, \mathcal{I} \rangle \models \psi_c^z \text{ for some } c \in \Sigma_\mathcal{C} \\
\langle \mathcal{W}, \mathcal{I} \rangle \models \mathbf{K}\psi & \text{iff} & \langle \mathcal{W}, \mathcal{I}' \rangle \models \psi \text{ for every } \mathcal{I}' \in \mathcal{W},
\end{array}$$

where ψ, z, and ψ_c^z denote an arbitrary EQL sentence, a variable, and the EQL sentence obtained from ψ by replacing the variable z with the constant c, respectively. For an EQL sentence ψ and a set $\mathcal{W}$ of interpretations, we denote by $\mathcal{W} \models \psi$ whenever $\langle \mathcal{W}, \mathcal{I} \rangle \models \psi$ for each $\mathcal{I} \in \mathcal{W}$.

The Query Language $\mathsf{DataKlog}(\mathcal{Q})$

Syntax We assume to have a countably infinite set of predicate symbols, called IDB, which is pairwise disjoint with $\Sigma_\mathcal{R}$, $\Sigma_\mathcal{C}$, and $\Sigma_\mathcal{V}$, and contains a special n-ary predicate Ans used to define the certain answers to n-ary queries.

In what follows, we refer to an EDB (resp. IDB) atom as an atom whose predicate symbol occurs in $\Sigma_\mathcal{R}$ (resp. IDB) and its arguments are either constants in $\Sigma_{\mathcal{C}}$ or variables in $\Sigma_{\mathcal{V}}$, and to an EDB (resp. IDB) literal as an EDB (resp. IDB) atom or a negated EDB (resp. IDB) atom.

A $\mathsf{DataKlog}(\mathcal{Q})$ query posed to a KB $\mathcal{K}$ is a program that is parametric with respect to the *embedded query language* $\mathcal{Q}$, with $\mathcal{Q} \subseteq \mathsf{FOL}$, and can use two different types of rules. The first type, denoted by $(1)_{\mathcal{Q}}$, has the following form:

$$\forall \bar{x}.\psi(\bar{x}) \rightarrow h(\bar{x}),$$

where $\bar{x} = (x_1, \ldots, x_n)$ are the variables occurring in both sides of the implication, $h(\bar{x})$ is an IDB atom using the variables in $\bar{x}$, and $\psi(\bar{x})$ is an open EQL formula whose free variables are the variables in $\bar{x}$. Note that the syntax of EQL implies that all occurrences of the $\Sigma_{\mathcal{R}}$ predicates in ψ are in the scope of the modal epistemic operator $\mathbf{K}$, and therefore the rule allows extracting only what is *known to be true* in *all* the possible models of $\mathcal{K}$. An example of rule of this type is rule (1) in the introduction, where $\psi(x) = \exists v.\mathbf{K}(\mathsf{hasVariant}(x, v))$, and $h(\bar{x}) = N(x)$.

In what follows, for a tuple of constants $\bar{c} = (c_1, \ldots, c_n)$, we denote by $\psi^{\bar{x}}_{\bar{c}}$ (resp. $h^{\bar{x}}_{\bar{c}}$) the EQL sentence (resp. the IDB atom) obtained from the EQL formula $\psi(\bar{x})$ (resp. IDB atom $h(\bar{x})$) by replacing each variable $x_i \in \bar{x}$ with the constant $c_i \in \bar{c}$. We also denote by q_ψ the FOL query $q_\psi = \{\bar{x} \mid \textsc{body}_{\mathsf{FOL}}(\psi)\}$, where $\textsc{body}_{\mathsf{FOL}}(\psi)$ is obtained from $\psi(\bar{x})$ by replacing each epistemic atom $\mathbf{K}\varrho_i$ with a fresh atom $R_i(\bar{w_i})$, where $\bar{w_i}$ is the tuple of the free variables used in ϱ_i and R_i is a fresh predicate symbol of the same arity as $\bar{w_i}$.

The second type of rules, denoted by (2), allows for rules of the typical form of Disjunctive Datalog:

$$\forall \bar{x}.b_1 \wedge \ldots \wedge b_n \rightarrow h_1(\bar{x_1}) \vee \ldots \vee h_m(\bar{x_m}),$$

where each b_j is either an IDB literal or an inequality atom, for each $j = 1, \ldots, n$, $\bar{x}$ is the tuple of variables used in the various atoms b_j's, and $h_j(\bar{x_j})$ is a single IDB atom using variables from $\bar{x_j} \subseteq \bar{x}$, for each $j = 1, \ldots, m$.

Starting from the language defined above, we obtain several sublanguages, based on syntactic restrictions concerning *inequalities in the body (indicated by I)*, *negation in the body (N)*, and *disjunction in the head (D)*. The set of such sublanguages form the $\mathsf{DataKlog}(\mathcal{Q})$ family, each of whose member is denoted by $\mathsf{DataKlog}(\mathcal{Q})^{I,N,D}$, where I can be either void or $\neq$, D can be either void or $\vee$, and N can take the value void or be set to *semi-positive negation* $(\neg_0)$, *stratified negation* $(\neg_s)$, or *full negation* $(\neg)$. The meaning of I, D, N should be obvious: (*i*) if I is void, then inequalities in both rules are disallowed; (*ii*) if D is void, then the rules of the form (2) are such that $m = 1$; (*iii*) if N is void, then negation is disallowed, both in rules of type $(1)_{\mathcal{Q}}$ and in rules of type (2); (*iv*) if N is $\neg_0$, then only negation in the rules of the form (2) is disallowed; (*v*) if N is $\neg_s$, then the set of rules of the form (2) must form a stratified program (Eiter, Gottlob, and Mannilla 1997).

For any value taken by I and N as above, we will denote by $\mathsf{Data}\mathbf{K}_0(\mathcal{Q})^{I,N}$ the fragment of $\mathsf{DataKlog}(\mathcal{Q})^{I,N}$ allowing only for rules of the form $(1)_{\mathcal{Q}}$. As will be clear later, the expressive power of $\mathsf{EQL\text{-}Lite}(\mathcal{Q})$ (Calvanese et al. 2007a) is equivalent to that of $\mathsf{Data}\mathbf{K}_0(\mathcal{Q})^{\neq,\neg_0}$.

Observe that the query $\forall x.\exists v.\mathbf{K}(\mathsf{hasVariant}(x, v)) \rightarrow Ans(x)$ illustrated in the introduction is a $\mathsf{Data}\mathbf{K}_0(\mathsf{AQ})$ query, whereas the set of rules (1)-(9) in the introduction forms a $\mathsf{DataKlog}(\mathsf{AQ})^{\neq,\neg_s,\vee}$ query.

Adopting the same syntactic restrictions, we can also define the $\mathsf{Datalog}^{I,N,D}$ family of query languages (Eiter, Gottlob, and Mannilla 1997), where, however, rules of the form $(1)_{\mathcal{Q}}$ are replaced by rules of the form $\forall \bar{x}.b_1 \wedge \ldots b_n \rightarrow h(\bar{x_h})$, where b_j is either an EDB literal or an inequality atom, for each $j = 1, \ldots, n$, $\bar{x}$ is the tuple of variables occurring in the various atoms b_j's, and $h(\bar{x_h})$ is a single IDB atom using variables from $\bar{x_h} \subseteq \bar{x}$.

As a shorthand, we will denote $\mathsf{DataKlog}(\mathcal{Q})^{\neq,\neg,\vee}$ and $\mathsf{Datalog}^{\neq,\neg,\vee}$ simply by $\mathsf{DataKlog}(\mathcal{Q})^{\top}$ and $\mathsf{Datalog}^{\top}$, respectively.

As usually done when dealing with various forms of negation, we impose standard syntactic *safeness conditions* that ensures *domain independence* of queries in both the $\mathsf{DataKlog}(\mathcal{Q})$ and the $\mathsf{Datalog}$ family. Specifically, from now on, we implicitly assume that rules of the type (2) are such that each variable occurring in an inequality atom or in some negated atom also occur in some non-negated atom (Ceri, Gottlob, and Tanca 1989). As for rules of type $(1)_{\mathcal{Q}}$, from now on we deal only with rules $\forall \bar{x}.\psi(\bar{x}) \rightarrow h(\bar{x})$ such that (*i*) q_ψ is a *safe-range* FOL query (Abiteboul, Hull, and Vianu 1995), and (*ii*) each epistemic atom $\mathbf{K}\varrho_i$ occurring in ψ is such that $\{\bar{w_i} \mid \varrho_i\}$ is a safe-range FOL query, where $\bar{w_i}$ is the tuple of the free variables used in ϱ_i[1]. Also, in what follows, for the sake of simplicity, we assume to deal only with knowledge bases $\mathcal{K}$ such that $mod(\mathcal{K}) \neq \emptyset$.

Semantics The semantics of $\mathsf{DataKlog}(\mathcal{Q})$ is based on the classical notion of *cautious reasoning* over *stable models* for Disjunctive Datalog queries. Note that, in principle, other choices are possible, but the stable model semantics is the basis of many popular paradigms, including Answer Set Programming (Lifschitz 2019).

Let q be a $\mathsf{DataKlog}(\mathsf{FOL})^{\top}$ query over a KB $\mathcal{K}$ with signature Σ. An interpretation for q is a pair $I = \langle mod(\mathcal{K}), \mathcal{F} \rangle$, where $\mathcal{F}$ is a FOL interpretation for IDB with domain $\Sigma_{\mathcal{C}}$, i.e. $\mathcal{F} = \langle \Sigma_{\mathcal{C}}, \cdot^{\mathcal{F}} \rangle$ such that $\cdot^{\mathcal{F}}$ assigns to each n-ary predicate $P \in$ IDB an n-ary relation $P^{\mathcal{F}} \subseteq \Sigma^n_{\mathcal{C}}$. We now define when an interpretation I for q satisfies a rule $r \in q$:

1. I satisfies a rule $r = \forall \bar{x}.\psi(\bar{x}) \rightarrow h(\bar{x})$ of the form $(1)_{\mathsf{FOL}}$ if, for each possible tuple of constants $\bar{c}$ occurring in $\Sigma_{\mathcal{C}}$ such that $mod(\mathcal{K}) \models \psi^{\bar{x}}_{\bar{c}}$, we have that $h^{\bar{x}}_{\bar{c}} \in \mathcal{F}$;
2. I satisfies a rule $r = \forall \bar{x}.b_1 \wedge \ldots \wedge b_n \rightarrow h_1(\bar{x_1}) \vee \ldots \vee h_m(\bar{x_m})$ of the form (2) if $\mathcal{F} \models r$, i.e. if the FOL sentence r is true in $\mathcal{F}$.

We then say that an interpretation I for q is *a model of* q if I satisfies every rule $r \in q$.

Following the popular approach to deal with negation in logic programming, we now define the notion of stable model for $\mathsf{DataKlog}(\mathsf{FOL})^{\top}$ queries, which in turn relies on the notion of *reduct* (Gelfond 1994; Fandinno, Faber, and Gelfond 2022). We use reducts in order to assign the

[1]Note that restricting queries to be safe-range is the common safeness condition imposed on FOL queries.

truth value to EQL sentences on the left-hand side of *ground rules* of the form $(1)_{\mathcal{Q}}$. The *grounding* of a rule r, either of the form $(1)_{\mathcal{Q}}$ or of the form (2), is the set $gr(r)$ of rules obtained by instantiating all the universally quantified variables with the constants from $\Sigma_{\mathcal{C}}$ in all possible ways. We then define the grounding of a $\mathsf{DataKlog}(\mathcal{Q})^{\top}$ query q as $gr(q) = \bigcup_{r \in q} gr(r)$. The *reduct* of a $\mathsf{DataKlog}(\mathcal{Q})^{\top}$ query q w.r.t. a KB $\mathcal{K}$, denoted by $q^{\mathcal{K}}$, is the ground $\mathsf{Datalog}^{\top}$ query obtained from $gr(q)$ by replacing each ground rule $\psi_{\bar{c}}^{\bar{x}} \rightarrow h_{\bar{c}}^{\bar{x}}$ of the form $(1)_{\mathsf{FOL}}$ with $\mathtt{true} \rightarrow h_{\bar{c}}^{\bar{x}}$ if $\mathcal{W} \models \psi_{\bar{c}}^{\bar{x}}$; and with $\mathtt{false} \rightarrow h_{\bar{c}}^{\bar{x}}$ otherwise. We then say that an interpretation I for q of the form $\langle mod(\mathcal{K}), \mathcal{F}\rangle$ is a *stable model of* q if $\mathcal{F}$ is a stable model of the $\mathsf{Datalog}^{\top}$ query $q^{\mathcal{K}}$. In what follows, we denote by $SM(q, \mathcal{K})$ the set of stable models of q. It is easy to verify that each member of $SM(q, \mathcal{K})$ is a model of q while the converse does not hold in general.

We now define the notion of answers of queries in our language, based on the skeptical reasoning approach.

Definition 1. *The set of answers of a* $\mathsf{DataKlog}(\mathsf{FOL})^{\top}$ *query* q *w.r.t. a KB* $\mathcal{K}$ *is* $ans(q, \mathcal{K}) = \{\bar{c} \in \Sigma_{\mathcal{C}}^{n} \mid Ans(\bar{c}) \in \mathcal{F}$ *for each* $\langle mod(\mathcal{K}), \mathcal{F}\rangle \in SM(q, \mathcal{K})\}$.

To ease the presentation, from now on we assume that queries do not mention constants. We point out, however, that all our results can be straightforwardly adapted to the case where also constants in queries are allowed.

Following (Calvanese et al. 2007a, Proposition 7), given an n-ary query q and a KB $\mathcal{K}$, we say that q is $\mathcal{K}$*-range-restricted* if $ans(q, \mathcal{K}) \subseteq dom(\mathcal{K})^{n}$. For a KB language $\mathcal{L}$, we say that a query language $\mathcal{Q}$ is $\mathcal{L}$*-range-restricted* if q is $\mathcal{K}$-range-restricted for any query $q \in \mathcal{Q}$ and any satisfiable $\mathcal{L}$ KB $\mathcal{K}$. It is not hard to see that, for any query language $\mathcal{Q} \subseteq \mathsf{FOL}$ and KB language $\mathcal{L}$, due to the implicit syntactic safeness conditions imposed on $\mathsf{DataKlog}(\mathcal{Q})^{\top}$ queries, if $\mathcal{Q}$ is $\mathcal{L}$-range-restricted, then $\mathsf{DataKlog}(\mathcal{Q})^{\top}$ is $\mathcal{L}$-range-restricted as well.

Expressiveness

In this section, we study the expressive power of $\mathsf{DataKlog}$. To this aim, we focus on the Boolean and constant-free fragments[2] of $\mathsf{DataKlog}(\mathsf{CQ})$, based on the fact that CQ is the most widely used query language for accessing KBs. Thus we start by investigating *what* the various variants of $\mathsf{DataKlog}(\mathsf{CQ})$ can express. Second, we show that the set of the syntactic features of the richest variant of $\mathsf{DataKlog}(\mathsf{CQ})$ is "minimal", in the sense that no feature among disjunction, negation, and recursion can be discarded without loosing expressivity. Then, we conclude the investigation of the expressivity of $\mathsf{DataKlog}(\mathsf{CQ})$ by comparing it with that of other notable classes of queries. In particular, we focus on queries expressed in languages most commonly used in OBDA settings, ranging from CQ to (Disjunctive) Datalog.

Let us first introduce few definitions. Given two queries q, q' and a KB $\mathcal{K}$, *q is equivalent to q' over $\mathcal{K}$*, written $q \sim_{\mathcal{K}} q'$, if $ans(q, \mathcal{K}) = ans(q', \mathcal{K})$. Also, given a KB language $\mathcal{L}$, q is equivalent to q' over $\mathcal{L}$, written $q \sim_{\mathcal{L}} q'$, if $q \sim_{\mathcal{K}} q'$ for every $\mathcal{L}$ KB $\mathcal{K}$. Then, given a KB language $\mathcal{L}$ and two query languages $\mathcal{Q}$ and $\mathcal{Q}'$, *$\mathcal{Q}$ expresses $\mathcal{Q}'$ over $\mathcal{L}$ KBs* if, for every $q' \in \mathcal{Q}'$, there exists $q \in \mathcal{Q}$ such that $q \sim_{\mathcal{L}} q'$. Furthermore, if $\mathcal{Q}$ expresses $\mathcal{Q}'$ over $\mathcal{L}$ KBs but not vice-versa, we say that *$\mathcal{Q}$ is strictly more expressive than $\mathcal{Q}'$ over $\mathcal{L}$ KBs*. Finally, $\mathcal{Q}$ and $\mathcal{Q}'$ are *equivalent over $\mathcal{L}$ KBs* if $\mathcal{Q}$ expresses $\mathcal{Q}'$ and vice-versa. Observe that, if $\mathcal{Q}$ does not express $\mathcal{Q}'$ over $\mathcal{L}$ KBs, then $\mathcal{Q}$ does not express $\mathcal{Q}'$ over every $\mathcal{L}'$ KBs such that $\mathcal{L}' \supseteq \mathcal{L}$. On the contrary, the fact that $\mathcal{Q}$ expresses $\mathcal{Q}'$ over $\mathcal{L}$ KBs does not imply that $\mathcal{Q}$ expresses $\mathcal{Q}'$ over $\mathcal{L}'$ KBs with $\mathcal{L}' \supseteq \mathcal{L}$. This is because $\mathcal{L}'$ may define theories with properties that queries in $\mathcal{Q}$ cannot recognize while queries in $\mathcal{Q}'$ can.

With these definitions in place, we start our investigation by showing that, given a KB language $\mathcal{L}$, $\mathsf{DataKlog}(\mathsf{CQ})$ expresses over $\mathcal{L}$ KBs classes of queries that are considered fundamental in OBDA query answering, i.e., queries enjoying the property of being rewritable in $\mathcal{L}$ in terms of UCQ, if we consider $\mathsf{DataK}_0(\mathsf{CQ})$, and in terms of $\mathsf{Datalog}^{I,N,D}$, if we consider $\mathsf{DataKlog}(\mathsf{CQ})^{I,N,D}$.

Proposition 1. *Let $\mathcal{L}$ be a KB language such that CQ is $\mathcal{L}$-range-restricted, and let q be a query.*

- *If q is UCQ-rewritable in $\mathcal{L}$, then for every $\mathcal{L}$ KB $\mathcal{K}$, there exists $q_{\mathcal{K}}$ in $\mathsf{DataK}_0(\mathsf{CQ})$ such that $q \sim_{\mathcal{K}} q_{\mathcal{K}}$.*
- *If q is $\mathsf{Datalog}^{I,N,D}$-rewritable in $\mathcal{L}$, then for every $\mathcal{L}$ KB $\mathcal{K}$, there exists $q_{\mathcal{K}}$ in $\mathsf{DataKlog}(\mathsf{CQ})^{I,N,D}$ such that $q \sim_{\mathcal{K}} q_{\mathcal{K}}$.*

We further investigate $\mathsf{DataKlog}(\mathsf{CQ})$ and check whether each of the features of its richest variant actually adds expressivity to the language. First, we study the impact of negation in the language.

Proposition 2. *$\mathsf{DataKlog}(\mathsf{CQ})^{\neq,\vee}$ does not express $\mathsf{DataK}_0(\mathsf{CQ})^{\neg_0}$ over $\mathcal{L}_{\text{triv}}$ KBs.*

Proposition 2 shows that $\mathsf{DataKlog}(\mathsf{CQ})$, equipped with disjunction and inequalities cannot express $\mathsf{DataK}_0(\mathsf{CQ})$ equipped with semi-positive negation. This follows from the fact that $\mathsf{DataKlog}(\mathsf{CQ})^{\neq,\vee}$ is monotonic[3] while $\mathsf{DataK}_0(\mathsf{CQ})^{\neg_0}$ is not.

Second, we study the impact of disjunction. Indeed, the following claim shows that, under reasonable complexity assumptions and in the presence of recursion, stratified negation cannot express disjunction.

Proposition 3. *Unless P = NP, $\mathsf{DataKlog}(\mathsf{CQ})^{\neq,\neg_s}$ does not express $\mathsf{DataKlog}(\mathsf{CQ})^{\vee}$ over $\mathcal{L}_{\text{triv}}$ KBs.*

In a nutshell, the previous proposition follows from the fact that, over $\mathcal{L}_{triv}$ KBs, the complexity of query answering for $\mathsf{DataKlog}(\mathsf{CQ})^{\neq,\neg_s}$ is P-complete while the complexity of query answering for $\mathsf{DataKlog}(\mathsf{CQ})^{\vee}$ is coNP-complete (see Proposition 7 below).

Finally, we turn our attention to the impact of adding recursion to $\mathsf{DataK}_0(\mathsf{CQ})$, i.e., we compare the expressive power of $\mathsf{DataKlog}(\mathsf{CQ})$ with that of $\mathsf{DataK}_0(\mathsf{CQ})$.

[2]Similar results can be obtained relaxing these limitation at the expense of a heavier notation.

[3]In the context of KBs, a query language is monotonic if, given two KBs $\mathcal{K}$ and $\mathcal{K}'$ and a query q in the language, we have that $ans(q, \mathcal{K}) \subseteq ans(q, \mathcal{K}')$ whenever $mod(\mathcal{K}) \supseteq mod(\mathcal{K}')$.

Unsurprisingly, allowing recursion increases the expressive power of $\mathsf{Data}\mathbf{K}_0(\mathsf{CQ})$ even in the case of trivial KBs.

Proposition 4. *$\mathsf{Data}\mathbf{K}\mathsf{log}(\mathsf{CQ})$ is strictly more expressive than $\mathsf{Data}\mathbf{K}_0(\mathsf{CQ})$ over $\mathcal{L}_{\text{triv}}$ KBs.*

The proof of the above proposition relies on three claims. First, in the case of $\mathcal{L}_{triv}$ KBs, every query $q \in \mathsf{Data}\mathbf{K}_0(\mathsf{CQ})$ is equivalent to a UCQ. Second, for every $\mathsf{Datalog}$ query q and every database D, one can build a trivial KB $\mathcal{K} = \langle \emptyset, \mathcal{A} \rangle$ and a $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathsf{CQ})$ query q' over $\mathcal{K}$ such that the evaluation of q over D returns true if and only if the answer to q' over $\mathcal{K}$ is true. Third, UCQ cannot express $\mathsf{Datalog}$ due to locality properties. While one can easily show the former two, the latter is a well-known result from (Libkin 2003).

Next, we conclude the section by comparing the expressivity of $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathsf{CQ})$ with that of other languages commonly used to pose queries in OBDA settings. Specifically, we investigate whether some variant of $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathsf{CQ})$ can express languages ranging from CQ to Disjunctive Datalog. Clearly, the expressivity of a query language over a KB depends on the language of the KB. In particular, in the following we consider two KB languages: *DL-Lite*$_{core}$ and *DL-Lite*$_{RDFS}$. The reason for choosing these two specific languages is that on one hand they are both very simple, on the other hand they differ for a crucial aspect: *DL-Lite*$_{core}$ allows to express incomplete information while *DL-Lite*$_{RDFS}$ does not. As we will see, this significantly changes the expressivity of $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathsf{CQ})$.

Specifically, we first show that, over *DL-Lite*$_{RDFS}$ KBs, the epistemic operator has no impact on the expressive power of the monotonic fragments of our language.

Proposition 5. *Over DL-Lite$_{RDFS}$ KBs, it holds that:*

- *$\mathsf{Data}\mathbf{K}_0(\mathsf{CQ})$ is equivalent to UCQ;*
- *$\mathsf{Data}\mathbf{K}\mathsf{log}(\mathsf{CQ})^{I,D}$ is equivalent to $\mathsf{Datalog}^{I,D}$, for each possible I and D.*

Results similar to Proposition 5 can be obtained for KBs expressed in other languages not allowing to express incomplete information, such as *full-tgds* (Abiteboul, Hull, and Vianu 1995). Intuitively, the latter admit a single model, called *canonical*, that is representative of all models of the KB with respect to the answers to UCQ and that does not include any variable. Note that Proposition 5 cannot be extended to variants of $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathsf{CQ})$ with negation, because while $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathsf{CQ})$ becomes non monotonic as soon as we add negation, $\mathsf{Datalog}$ with negation over KBs is monotonic under the stable model semantics (Eiter, Gottlob, and Mannilla 1997).

We now switch to *DL-Lite*$_{core}$ and show that over *DL-Lite*$_{core}$ KBs even the powerful $\mathsf{Datalog}^{\top}$ does not express $\mathsf{Data}\mathbf{K}_0(\mathsf{CQ})$. To this end, we need to introduce some additional technical tools.

Let $\mathcal{I} = \langle \Sigma_{\mathcal{C}}, \cdot^{\mathcal{I}} \rangle$, $\mathcal{J} = \langle \Sigma_{\mathcal{C}}, \cdot^{\mathcal{J}} \rangle$ be two FOL interpretations. An isomorphism from $\mathcal{I}$ to $\mathcal{J}$ is a bijection $i : \Sigma_{\mathcal{C}} \to \Sigma_{\mathcal{C}}$ such that $i(\bar{c}) \in R^{\mathcal{J}}$ if and only if $\bar{c} \in R^{\mathcal{I}}$, for every R in $\Sigma_{\mathcal{R}}$. Assume two KBs A, B. We say that A *is isomorphically-similar to* B (written $A \rightarrow_{iso} B$) if for every model $\mathcal{I}_B \in mod(B)$ there exists a model $\mathcal{I}_A \in mod(A)$ such that there exists an isomorphism from $\mathcal{I}_A$ to $\mathcal{I}_B$. A Boolean query q is said to be *preserved in isomorphically-similar KBs* if $A \rightarrow_{iso} B$ implies $ans(q, A) \subseteq ans(q, B)$.

One can show that $\mathsf{Datalog}^{\top}$ queries are preserved in isomorphically-similar KBs while $\mathsf{Data}\mathbf{K}_0(\mathsf{CQ})$ queries are not. Thus, $\mathsf{Datalog}^{\top}$ does not express $\mathsf{Data}\mathbf{K}_0(\mathsf{CQ})$ already over lightweight KBs such as *DL-Lite*$_{core}$ (Calvanese et al. 2007b), which leads to the following proposition.

Proposition 6. *$\mathsf{Datalog}^{\top}$ does not express $\mathsf{Data}\mathbf{K}_0(\mathsf{CQ})$ over DL-Lite$_{core}$ KBs.*

Intuitively, Proposition 6 tells us that over *DL-Lite*$_{core}$ KBs, the epistemic operator increases the expressive power of UCQ, and the resulting $\mathsf{Data}\mathbf{K}_0(\mathsf{CQ})$ language cannot be expressed even if we allow the recursive and disjunctive constructs of $\mathsf{Datalog}^{\top}$.

Corollary 1. *Let $\mathcal{L} \supseteq$ DL-Lite$_{core}$ be a KB language. Over $\mathcal{L}$ KBs, $\mathsf{Data}\mathbf{K}_0(\mathsf{CQ})$ is strictly more expressive than CQ.*

Query Answering

We now focus on the problem of answering $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathcal{Q})$ queries over $\mathcal{L}$ KBs. More precisely, in this section, we implicitly refer to the associated *recognition problem*, i.e., given a satisfiable $\mathcal{L}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, a query q in $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathcal{Q})$, and a tuple of constants $\bar{c}$, check whether $\bar{c} \in ans(q, \mathcal{K})$. Also, we conduct our study by implicitly investigating the *data complexity* (Vardi 1982) of this problem, i.e., the complexity where only the ABox $\mathcal{A}$ is regarded as the input and the other components are assumed to be fixed.

We start by showing that query answering is decidable and provide an upper bound for its complexity under general assumptions for $\mathcal{L}$ and the embedded query language $\mathcal{Q}$.

Theorem 1. *Let $\mathcal{Q} \subseteq \mathsf{FOL}$ and $\mathcal{L}$ be a query language and a KB language, respectively, such that (i) $\mathcal{Q}$ is $\mathcal{L}$-range-restricted and (ii) answering $\mathcal{Q}$ queries over $\mathcal{L}$ KBs is in $\mathcal{C}$, for a complexity class $\mathcal{C}$. Then, the following holds:*

1) answering $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathcal{Q})^{\neq,\neg_s}$ queries over $\mathcal{L}$ KBs is in $P_{||}^{\mathcal{C}}$;

2.1) if either NP $\subseteq \mathcal{C}$ or coNP $\subseteq \mathcal{C}$, then answering $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathcal{Q})^{\neq,\neg}$ queries and $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathcal{Q})^{\neq,\neg_0,\vee}$ queries over $\mathcal{L}$ KBs are in $P_{||}^{\mathcal{C}}$;

2.2) if $\mathcal{C} \subseteq P$, then answering $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathcal{Q})^{\neq,\neg}$ queries and $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathcal{Q})^{\neq,\neg_0,\vee}$ queries over $\mathcal{L}$ KBs are in coNP;

3.1) if either $\Sigma_2^p \subseteq \mathcal{C}$ or $\Pi_2^p \subseteq \mathcal{C}$, then answering $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathcal{Q})^{\top}$ queries over $\mathcal{L}$ KBs is in $P_{||}^{\mathcal{C}}$;

3.2) if $\mathcal{C} \subseteq \Delta_2^p$, then answering $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathcal{Q})^{\top}$ queries over $\mathcal{L}$ KBs is in Π_2^p.

The above theorem shows that decidability of answering $\mathcal{Q}$ queries over $\mathcal{L}$ KBs plus $\mathcal{L}$-range-restrictedness of $\mathcal{Q}$ queries implies decidability of answering $\mathsf{Data}\mathbf{K}\mathsf{log}(\mathcal{Q})^{\top}$ queries over $\mathcal{L}$ KBs. Note that this must be contrasted with the fact that as soon as we try to extend UCQ with either the recursion of plain $\mathsf{Datalog}$ or with inequalities/negation, we get undecidability of query answering already for lightweight ontologies. Indeed, from results of (Levy and Rousset 1998; Calvanese and Rosati 2003) and of (Gutiérrez-Basulto et al. 2015), we know that answering $\mathsf{Datalog}$ queries and $\mathsf{UCQ}^{\neq}$ and $\mathsf{UCQ}^{\neg}$ queries, respectively, already over *DL-Lite*$_{core}$ KBs is undecidable.

Moreover, the above theorem shows that the complexity of $\mathsf{DataKlog}(\mathcal{Q})$ query answering over $\mathcal{L}$ KBs, crucially depends on the complexity of answering $\mathcal{Q}$ queries over $\mathcal{L}$ KBs. Thus, in the following, we provide several tight data complexity results for combinations of KB languages $\mathcal{L}$ and embedded query languages $\mathcal{Q}$, for which the complexity of answering $\mathcal{Q}$ queries over $\mathcal{L}$ KBs is, respectively, in P, in EXPTIME, and in coNP. Importantly, by combining such results with results of the previous section, one can see that for some combinations of $\mathcal{Q}$ and $\mathcal{L}$ it is possible to pose provably more expressive queries in the $\mathsf{DataKlog}(\mathcal{Q})$ family (rather than be limited to $\mathcal{Q}$ queries) to $\mathcal{L}$ KBs at no additional computational cost (regarding data complexity). For example, our results imply that it is possible to pose strictly more expressive $\mathsf{DataKlog}(\mathsf{UCQ})^{\neq,\neg_s}$ queries (rather than be limited to UCQ queries) to Horn-$\mathcal{SHIQ}$ KBs while remaining in P in data complexity.

Let us first consider the case of combinations of $\mathcal{L}$ and $\mathcal{Q}$ for which query answering is in P. Note that this case covers several well-known cases in which the embedded query language $\mathcal{Q}$ is UCQ. Among them, we mention the cases where the KB language $\mathcal{L}$ is one of the following: *DLR-Lite*$_{\mathcal{A},\sqcap}$ (and thus in particular *DL-Lite*$_{core}$ and *DL-Lite*$_{RDFS}$) (Calvanese et al. 2013), *sticky-join tgds* (Calì, Gottlob, and Pieris 2012), Horn-$\mathcal{SHIQ}$ (Eiter et al. 2008a), *(frontier-)guarded tgds* (*(F)GTGDs*) (Baget et al. 2011; Calì, Gottlob, and Lukasiewicz 2012). Notably, for this important case, we get the following tight complexity results.

Proposition 7. *Let $\mathcal{Q}$ and $\mathcal{L}$ be a query language and a KB language, respectively, such that (*i*) $\mathsf{AQ} \subseteq \mathcal{Q} \subseteq \mathsf{FOL}$, (*ii*) $\mathcal{Q}$ is $\mathcal{L}$-range-restricted, and (*iii*) answering $\mathcal{Q}$ queries over $\mathcal{L}$ KBs is in P. Then, the following holds:*

1. *answering $\mathsf{DataKlog}(\mathcal{Q})^{\neq,\neg_s}$ queries over $\mathcal{L}$ KBs is P-complete. The hardness holds for $\mathsf{DataKlog}(\mathcal{Q})$;*
2. *answering $\mathsf{DataKlog}(\mathcal{Q})^{\neq,\neg}$ and $\mathsf{DataKlog}(\mathcal{Q})^{\neq,\neg_0,\vee}$ queries over $\mathcal{L}$ KBs is coNP-complete. The hardness already holds for $\mathsf{DataKlog}(\mathcal{Q})^{\neg}$ and $\mathsf{DataKlog}(\mathcal{Q})^{\vee}$;*
3. *answering $\mathsf{DataKlog}(\mathcal{Q})^{\top}$ queries over $\mathcal{L}$ KBs is Π_2^p-complete. The hardness holds for $\mathsf{DataKlog}(\mathcal{Q})^{\neg_s,\vee}$.*

The above proposition can be shown by combining Theorem 1 with the following observations. On one hand, answering $\mathsf{Datalog}$ queries, $\mathsf{Datalog}^{\vee}$ and $\mathsf{Datalog}^{\neg}$ queries, and $\mathsf{Datalog}^{\neg_s,\vee}$ queries is P-hard (Vardi 1982), coNP-hard (Eiter, Gottlob, and Mannilla 1997), and Π_2^p-hard (Eiter and Gottlob 1995), respectively, already over relational databases. On the other hand, a $\mathsf{Datalog}^{I,N,D}$ query q over a relational database can be simulated with a $\mathsf{DataKlog}(\mathcal{Q})^{I,N,D}$ query over a $\mathcal{L}_{triv}$ KB whose ABox coincides with the database, by choosing AQ as embedded query language $\mathcal{Q}$, where, obviously, $\mathsf{AQ} \subseteq \mathcal{Q} \subseteq \mathsf{FOL}$.

Let us now consider the case where $\mathcal{L}$ and $\mathcal{Q}$ are such that answering $\mathcal{Q}$ queries over $\mathcal{L}$ KBs is in EXPTIME. We point out that, if $\mathcal{Q}$ is UCQ, then notable cases of $\mathcal{L}$ that fall into this case are all languages between *weakly-guarded tgds* (*WGTGDs*) and *weakly-frontier-guarded disjunctive tgds* (*WFGDTGDs*). Indeed, for any such language, answering UCQ queries over $\mathcal{L}$ KBs is EXPTIME-complete (Baget et al. 2011; Bourhis, Morak, and Pieris 2013).

Proposition 8. *Let $\mathcal{Q} \subseteq \mathsf{FOL}$ and $\mathcal{L}$ be a query language and a KB language, respectively, such that (*i*) $\mathcal{Q}$ is $\mathcal{L}$-range-restricted and (*ii*) answering $\mathcal{Q}$ queries over $\mathcal{L}$ KBs is $\mathcal{C}$-complete, where EXPTIME $\subseteq \mathcal{C}$. Then, answering $\mathsf{DataKlog}(\mathcal{Q})^{\neq,\neg,\vee}$ queries over $\mathcal{L}$ KBs is $\mathcal{C}$-complete.*

The above proposition can be derived from Theorem 1 and the fact that, for any query language $\mathcal{Q} \subseteq \mathsf{FOL}$, already $\mathsf{DataK}_0(\mathcal{Q})$ is able to express $\mathcal{Q}$ queries.

Finally, we focus on combinations of embedded query languages $\mathcal{Q} \subseteq \mathsf{FOL}$ and KB languages $\mathcal{L}$ for which (*i*) answering $\mathcal{Q}$ queries over $\mathcal{L}$ KBs is in coNP and (*ii*) $\mathcal{Q}$ is $\mathcal{L}$-range-restricted. From Theorem 1 we know that answering $\mathsf{DataKlog}(\mathcal{Q})^{\neq,\neg}$ queries and $\mathsf{DataKlog}(\mathcal{Q})^{\neq,\neg_0,\vee}$ queries over $\mathcal{L}$ KBs is in $\mathrm{P}_{||}^{\mathrm{coNP}} = \Theta_2^p$. If $\mathcal{Q}$ is UCQ, examples of KB languages that fall into this case are $\mathcal{SHIQ}$ (Glimm et al. 2008) and *frontier-guarded disjunctive tgds* (*FGDTGDs*) (Bourhis, Morak, and Pieris 2013).

First, we prove Θ_2^p matching lower bounds already for very simple combinations of embedded query languages and KB languages. More specifically, consider the TBoxes $\mathcal{T}_{\sqcup} = \{\forall x.(A_1(x) \to A_2(x) \vee A_3(x))\}$, $\mathcal{T}_{\neg} = \{\forall x.(\neg A_1(x) \to A_2(x))\}$, and $\mathcal{T}_{\forall} = \{\forall x.(\forall y.(P(x,y) \to A_1(y)) \to A_2(x))\}$. These three simple TBoxes are expressive enough to make conjunctive query answering coNP-hard (Calvanese et al. 2013). We now show that, for any KB language $\mathcal{L}$ allowing for one of these TBoxes, answering already $\mathsf{DataKlog}(\mathsf{CQ})^{\neg_0}$ queries over $\mathcal{L}$ KBs is Θ_2^p-hard.

Lemma 1. *Let $\mathcal{L}$ be any KB language allowing for a TBox that is logically equivalent to one among $\mathcal{T}_{\sqcup}$, $\mathcal{T}_{\neg}$, and $\mathcal{T}_{\forall}$. Then, answering $\mathsf{DataKlog}(\mathsf{CQ})^{\neg_0}$ queries over $\mathcal{L}$ KBs is Θ_2^p-hard, and therefore Θ_2^p-complete.*

We now prove that, unless $\Theta_2^p = $ coNP, the semi-positive negation is essential for the above hardness result to hold.

Proposition 9. *Let $\mathcal{Q} \subseteq \mathsf{FOL}$ and $\mathcal{L}$ be a query language and a KB language, respectively, such that (*i*) $\mathcal{Q}$ is $\mathcal{L}$-range-restricted and (*ii*) answering $\mathcal{Q}$ queries over $\mathcal{L}$ KBs is coNP-complete. Then, answering $\mathsf{DataKlog}(\mathcal{Q})^{\neq,\vee}$ queries over $\mathcal{L}$ KBs is coNP-complete.*

We thus observe that, quite surprisingly and differently from previous cases, allowing negation (or not) actually makes a difference for combinations of $\mathcal{L}$ and $\mathcal{Q}$ for which query answering is in coNP.

Conclusion and Future Work

We investigated the expressive power and the computational characteristics of a novel family of query languages for KBs that can express powerful recursive and non-monotonic queries over the epistemic state of KBs, thus overcoming the undecidability of well-known query languages such as $\mathsf{Datalog}$. As future work, we plan to study (fragments of) $\mathsf{DataKlog}(\mathcal{Q})^{\top}$ with embedded query languages $\mathcal{Q}$ beyond FOL, and to investigate different semantics for the language, e.g., the well-founded and the perfect model semantics.

Acknowledgements

This work has been partially supported by the ANR AI Chair INTENDED (ANR-19-CHIA-0014), by MUR under the PRIN 2017 project "HOPE" (prot. 2017MMJJRE), by the EU under the H2020-EU.2.1.1 project TAILOR, grant id. 952215, and by MUR under the PNRR project PE0000013-FAIR.

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison Wesley Publishing Company.

Baget, J.-F.; Mugnier, M.-L.; Rudolph, S.; and Thomazo, M. 2011. Walking the Complexity Lines for Generalized Guarded Existential Rules. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 712–717.

Bienvenu, M.; Ortiz, M.; and Simkus, M. 2015. Regular Path Queries in Lightweight Description Logics: Complexity and Algorithms. *Journal of Artificial Intelligence Research*, 53: 315–374.

Bourhis, P.; Morak, M.; and Pieris, A. 2013. The Impact of Disjunction on Query Answering Under Guarded-Based Existential Rules. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 796–802.

Buss, S. R.; and Hay, L. 1991. On Truth-Table Reducibility to SAT. *Information and Computation*, 91(1): 86–102.

Cadoli, M.; Palopoli, L.; and Lenzerini, M. 1997. Datalog and Description Logics: Expressive Power. In *Proceedings of the Sixth International Workshop on Database Programming Languages (DBPL 1997)*, volume 1369 of *Lecture Notes in Computer Science*, 281–298.

Calì, A.; Gottlob, G.; and Lukasiewicz, T. 2012. A General Datalog-Based Framework for Tractable Query Answering Over Ontologies. *Journal of Web Semantics*, 14: 57–83.

Calì, A.; Gottlob, G.; and Pieris, A. 2012. Towards More Expressive Ontology Languages: The Query Answering Problem. *Artificial Intelligence*, 193: 87–128.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007a. EQL-Lite: Effective First-Order Query Processing in Description Logics. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 274–279.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007b. Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. *Journal of Automated Reasoning*, 39(3): 385–429.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2013. Data Complexity of Query Answering in Description Logics. *Artificial Intelligence*, 195: 335–360.

Calvanese, D.; and Rosati, R. 2003. Anwering recursive queries under keys and foreign keys is undecidable. In *Proceedings of the Tenth International Workshop on Knowledge Representation meets Databases (KRDB 2003)*, volume 79 of *CEUR Electronic Workshop Proceedings*.

Ceri, S.; Gottlob, G.; and Tanca, L. 1989. What you Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1): 146–166.

Cima, G.; Console, M.; Lenzerini, M.; and Poggi, A. 2022. Monotone Abstractions in Ontology-Based Data Management. In *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2022)*, 5556–5563.

Cima, G.; Lenzerini, M.; and Poggi, A. 2020. Answering Conjunctive Queries with Inequalities in *DL-Lite*$_{\mathcal{R}}$. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, 2782–2789.

Console, M.; and Lenzerini, M. 2020. Epistemic Integrity Constraints for Ontology-Based Data Management. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, 2790–2797.

Donini, F. M.; Lenzerini, M.; Nardi, D.; and Schaerf, A. 1998. $\mathcal{AL}$-log: Integrating Datalog and Description Logics. *Journal of Intelligent Information Systems*, 10(3): 227–252.

Eiter, T.; and Gottlob, G. 1995. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence*, 15(3-4): 289–323.

Eiter, T.; Gottlob, G.; and Mannilla, H. 1997. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3): 364–418.

Eiter, T.; Gottlob, G.; Ortiz, M.; and Šimkus, M. 2008a. Query Answering in the Description Logic Horn-$\mathcal{SHIQ}$. In *Proceedings of the Eleventh European Conference on Logics in Artificial Intelligence (JELIA 2008)*, 166–179.

Eiter, T.; Ianni, G.; Lukasiewicz, T.; Schindlauer, R.; and Tompits, H. 2008b. Combining answer set programming with description logics for the Semantic Web. *Artificial Intelligence*, 172(12-13): 1495–1539.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2005. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, 90–96.

Fandinno, J.; Faber, W.; and Gelfond, M. 2022. Thirty years of Epistemic Specifications. *Theory and Practice of Logic Programming*, 22(6): 1043–1083.

Gelfond, M. 1991. Strong Introspection. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI 1991)*, 386–391.

Gelfond, M. 1994. Logic Programming and Reasoning with Incomplete Information. *Annals of Mathematics and Artificial Intelligence*, 12(1-2): 89–116.

Gelfond, M.; and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming (ICLP/SLP 1998)*, 1070–1080.

Glimm, B.; Lutz, C.; Horrocks, I.; and Sattler, U. 2008. Conjunctive Query Answering for the Description Logic $\mathcal{SHIQ}$. *Journal of Artificial Intelligence Research*, 31: 151–198.

Gutiérrez-Basulto, V.; Ibáñez-García, Y. A.; Kontchakov, R.; and Kostylev, E. V. 2015. Queries with negation and inequalities over lightweight ontologies. *Journal of Web Semantics*, 35: 184–202.

Hogan, A.; Blomqvist, E.; Cochez, M.; d'Amato, C.; de Melo, G.; Gutiérrez, C.; Kirrane, S.; Gayo, J. E. L.; Navigli, R.; Neumaier, S.; Ngomo, A. N.; Polleres, A.; Rashid, S. M.; Rula, A.; Schmelzeisen, L.; Sequeda, J.; Staab, S.; and Zimmermann, A. 2021. *Knowledge Graphs*. Synthesis Lectures on Data, Semantics, and Knowledge. Morgan & Claypool Publishers.

Levesque, H. J.; and Lakemeyer, G. 2001. *The Logic of Knowledge Bases*. The MIT Press.

Levy, A. Y.; and Rousset, M.-C. 1998. Combining Horn Rules and Description Logics in CARIN. *Artificial Intelligence*, 104(1–2): 165–209.

Libkin, L. 2003. Expressive power of SQL. *Theoretical Computer Science*, 296(3): 379–404.

Lifschitz, V. 1991. Nonmonotonic Databases and Epistemic Queries. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI 1991)*, 381–386.

Lifschitz, V. 2019. *Answer Set Programming*. Springer.

Motik, B.; and Rosati, R. 2010. Reconciling Description Logics and Rules. *Journal of the ACM*, 57(5): 30:1–30:62.

Motik, B.; Sattler, U.; and Studer, R. 2005. Query Answering for OWL-DL with Rules. *Journal of Web Semantics*, 3(1): 41–60.

Vardi, M. Y. 1982. The Complexity of Relational Query Languages (Extended Abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC 1982)*, 137–146.

Wagner, K. W. 1990. Bounded Query Classes. *SIAM Journal on Computing*, 19(5): 833–846.